Thèse n° 8046

Graph Embedding for Retrieval

Présentée le 11 mars 2022

Faculté informatique et communications
Laboratoire de systèmes d'information répartis
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Chi Thang DUONG

Acceptée sur proposition du jury

Prof. J.-Y. Le Boudec, président du jury
Prof. K. Aberer, directeur de thèse
Prof. B. Yang, rapporteur
Prof. Z. Miklos, rapporteur
Prof. A.-M. Kermarrec, rapporteuse

École
polytechnique
fédérale
de Lausanne

2022

To D.C.V

Acknowledgements

First and foremost, I would like to thank my advisor *Prof. Karl Aberer*. He gave me the chance to do my bachelor thesis at LSIR 10 years ago and the rest is history. His constant support allows me to focus on doing research. I will never forget the research freedom I have during my PhD.

A big thank to my thesis committee: *Prof. Anne-Marie Kermarrec*, *Prof. Bin Yang*, *Prof. Zoltan Miklos* and *Prof. Jean-Yves Le Boudec* for agreeing to be in the committee and the insightful comments and discussions.

A special thank to *Chantal* for your help with all the paperwork and the administrative stuff. I will never understand how you could manage these things in such an efficient way.

I also would like to thank three PhD students in my lab who I consider as my good friends. *Panayiotis Smeros*, thank you for all your helps and discussions in and outside our work, I now know who to call when I need to move again. *Jérémie Rappaz*, thank you for introducing me to Sat and all the chats we have and for all the French translation. *Tugrulcan Elmas*, I still sometimes don't understand you but thank you for going to the concert with me.

I was lucky to have a great mentor which is *Prof. Quoc Viet Hung Nguyen*. His guidance and advices were invaluable. Without his help, my PhD would be extremely difficult. A great gratitude to *Dr. Thanh Tam Nguyen* who is an expert in Latex, EasyChair, CMT and all the conference management systems out there.

I also would like to thank my coauthors: *Prof. Hongzhi Yin, Prof. Matthias Weidlich, Trung-Dung Hoang*. Without your contributions and discussions, I wouldn't have a paper accepted. Thank you a lot for your help.

I would like to thank *Hien Dang, Thanh Phan, Son Do, Anh Tran* for being there when we needed them the most.

During my PhD, I spent a lot of time playing Dota 2. I would like to thank my teammates: *Shiawase na Buta-chan, lola, Nikasa* for all the ups and downs, wins and losses. Without you, I would be stressed to death. Smile because it happened.

My friends in Lausanne and elsewhere, thank you your support and the joyful time together. Without you, my life in Lausanne would be dull and boring.

I could never be here without my family. My parents and my sister who have always got my back and kept reminding me that I can always go home when things get difficult. Last but not least, my wife - thank you for choosing me and being here with me. Without you, I would be “sad to death”.

Abstract

Information retrieval (IR) systems such as search engines are important for people to find what they need among the tremendous amount of data available in their organization or on the Internet. These IR systems enable users to search in a large data collection by specifying queries that describe their information needs. Traditionally, the data elements in these collections are text documents that have no explicit relationships between them. As conventional IR systems are designed to handle text documents, the queries are limited to multisets of textual keywords. However, with the advance of social media, the data collection has become heterogenous in terms of modality as it has become easier for users to share not only texts but also images, audios and videos. In addition, data collections have evolved to contain also the relationships between data elements. For instance, in social networks, the relationships between users are as important as the users themselves.

Given these changes in the data collection, conventional bags of keywords queries become underwhelming as they are unimodal which cannot handle the heterogeneity the data elements. Moreover, they ignore the relationships between the query terms as they consider them to be independent. In this thesis, we show how to support *context-rich queries* both in terms of heterogeneity and interconnectivity by exploiting a common underlying graph model for the data collection. Our approach follows the vector space retrieval model where we design *graph embedding* techniques to represent each data element and each query as a vector i.e. an embedding. Our embedding model is designed to capture both the *heterogeneity* and the *connectivity* available in a data collection or a query in an elegant manner. As the data collection is usually large, this leads to a very large graph model which can not be handled by traditional graph embedding techniques. In this thesis, we also propose an approach to make graph embedding *scalable* to large graphs.

Regarding heterogeneity, we propose to construct a *heterogeneous information network* to capture the availability of different modalities in the same data element as well as relations between different data elements. This enables the construction of a graph embedding model to produce an embedding for each data element. We then propose a query embedding model that incorporates different modalities in the query based on the constructed graph embedding model. By taking into account different modalities available in the data collection, our model can return more relevant results in comparison with a unimodal model.

Regarding connectivity, we aim to tackle the problem of subgraph isomorphism search where queries are small graphs of connected query terms while the data collection is a large data graph. We propose a graph embedding model that captures the connection pattern of the data graph to create the query embeddings. To speed up retrieval, we also propose a cache mechanism based on the query and subgraph embedding to reuse past retrieval results. Our proposed retrieval model based on embedding is significantly faster than a structure-based approach in several orders of magnitude.

Regarding scalability, existing graph embedding techniques require a long training time for very large graphs. To scale these techniques, we propose a divide-and-conquer approach where a large graph is divided into smaller subgraphs where graph embedding is constructed independently before merging them together. Our implementation on Spark is able to handle graphs of billion scale while maintaining better speedup than traditional approaches.

Keywords: *information retrieval, graph embedding, scalability, multimodal retrieval, heterogeneity, subgraph retrieval, scalable graph embedding*

Résumé

Les systèmes de recherche d'information, tels que les moteurs de recherche, permettent aux gens de trouver ce dont ils ont besoin parmi l'énorme quantité de données disponible au sein de leur organisation ou sur internet. Ces systèmes permettent aux utilisateurs de rechercher dans de vastes jeux de données en spécifiant des requêtes décrivant leurs besoins en information. Traditionnellement, les éléments de données constituant ces collections sont des documents textes qui n'ont pas de relations explicites entre eux. Puisque les systèmes de recherche conventionnels sont conçus pour traiter des documents textes, les requêtes sont limitées à des mots-clés. Cependant, depuis l'avènement des médias sociaux, les données sont devenues plus hétérogènes en termes de modalités car il est devenu plus facile pour les utilisateurs de partager, non seulement du texte, mais aussi des images, du son ou de la vidéo. De plus, les jeux de données contiennent désormais les relations entre les éléments de données. Par exemple, sur les réseaux sociaux, les relations entre les utilisateurs sont aussi importantes que les utilisateurs eux-mêmes. Compte tenu de ces changements dans la nature des jeux de données, les requêtes par "sac de mots" classiques sont devenues insuffisantes car elles sont unimodales et ne peuvent pas gérer l'hétérogénéité des éléments de données. De plus, elles ignorent les relations entre les termes de la requête, qu'elles considèrent indépendants. Dans cette thèse, nous montrons comment prendre en compte des requêtes riches en contexte, à la fois en termes d'hétérogénéité et d'inter-connectivité, en exploitant un modèle de graphe. Notre approche utilise des techniques de plongement de graphes pour représenter chaque élément de donnée et chaque requête comme un vecteur. Notre modèle de plongement est conçu pour capturer à la fois l'hétérogénéité et l'inter-connectivité d'une requête ou d'un jeu de données. Les jeux de données utilisés pour ce type de tâches sont souvent très volumineux, ce qui implique des modèles de graphe très larges qui ne peuvent pas être gérés par les techniques de plongement de graphes traditionnelles. Dans cette thèse, nous proposons également une approche qui permet le plongement de grands graphes.

Concernant l'hétérogénéité, nous proposons de construire un réseau d'information hétérogène afin de capturer les différentes modalités d'un élément de donnée, ainsi que les relations des éléments de données entre eux. Cela permet la

construction d'un modèle de plongement de graphes qui produit un plongement pour chaque élément de donnée. Nous proposons ensuite un modèle de plongement de requêtes qui intègre différentes modalités dans la requête. En considérant les différentes modalités disponibles dans le jeu de données, notre modèle peut renvoyer des résultats plus pertinents, comparé à un modèle unimodal.

En ce qui concerne l'inter-connectivité, nous visons à résoudre le problème de la recherche d'isomorphismes de sous-graphes : les requêtes sont de petits graphes de mots-clés interconnectés qui permettent de chercher dans un grand graphe de données. Nous proposons un modèle de plongement de graphes qui capture la connectivité du graphe de données pour créer un plongement des requêtes. Pour accélérer la recherche, nous proposons également un mécanisme de cache qui enregistre les requêtes ainsi que les plongements de sous-graphes, et réutilise les résultats des recherches passés. Le modèle de recherche d'information proposé, basé sur le plongement, augmente la vitesse de recherche de plusieurs ordres de grandeur, comparé à une approche basée sur la structure.

En ce qui concerne l'extensibilité, les techniques de plongements de graphes existantes nécessitent un long temps d'apprentissage pour de très grands graphes. Pour y remédier, nous proposons une approche "diviser pour régner", où un grand graphe est divisé en sous-graphes, plus petits, et où les plongements de graphes sont construits indépendamment avant d'être fusionnés. Notre implémentation, basée sur Spark, est capable de gérer des graphes comprenant des milliards de nœuds, tout en maintenant une meilleure accélération d'apprentissage comparé aux approches traditionnelles.

Mots-clés : *recherche d'information, plongement de graphe, extensibilité, multi-modalité, hétérogénéité, recherche de sous-graphes*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The amount of data available to us is both a blessing and a curse. It is a blessing as all the information we need are always within our reach while it is a curse as the ocean of data makes it difficult to find where they are. Information retrieval (IR) is the field of study that helps people to cope with this curse. It has become the dominant way of accessing information, replacing traditional database searches where one would already know exactly what to find [SMR08]. Given a *query* which is a bag of query terms, an information retrieval system would go through all the data elements in its *data collection* checking for elements relevant to the query based on the query terms before returning them to the user.

1.1.1 The Need for Context-rich Query

Traditional IR systems are designed for document retrieval where a query contains a *multiset* of keywords and the data collection is a list of textual documents [BYRN11]. While these systems are able to handle traditional textual queries extremely well, they are tailored for queries which are *bags/multisets* of *textual* terms. These restrictions make it difficult for users to express their information needs. As such, users would abandon their search or they need to reformulate the queries such that they can capture their needs better. These additional queries would not only incur unnecessary processing but also compromise user experience. For example, according to a Web search engine that supports user relevance feedback, 70% of users only look at the first page of results without further continuing the search. However, over 66% of the time, a query reformulation based on user feedback would see their results improved [SMR08, BYRN11]. In other words, if the queries are rich enough semantically that they can capture the users' intention, fewer reformulated queries would be required. In other study, similar results are also found such as 50% of users modify their queries in a search session at least once while 33% of queries are modified more than 3 times [JSP05]. Moreover, in many cases, users express the desire to specify their queries in more detail but traditional IR systems are limited to only textual keywords [Bil00, PH97].

In addition to better capture users' information need, changes in the data collections also call for context-rich queries. With the proliferation of social media and the ubiquity of social networks such as Flickr, Facebook, Youtube and TikTok, users are able to share not only texts but also images, audios or videos [soc, LNC+18]. This makes the data collections to go from textual only to multimodal. As data become heterogenous, unimodal queries become ineffective as they cannot cover users' information needs for other modalities. Another change in the data collection is the need to store relations between data elements. There are several applications where the connections between data elements in the data collection is imperative. For instance, social networks need to not only store information about users but also their relationships. A search for a user based on knowledge about her friends is not possible with traditional queries as we need to specify the relationships between the search terms. Another well-known example is information retrieval on the Web where the links between the webpages are equally important as the contents of the webpages themselves [SRNC+00, PBMW99]. A retrieval system that models the connections and the contents independently would lose a lot of crucial information. In this thesis, we aim to handle these problems by supporting context-rich queries. These queries are rich as they support multimodal query terms and the relationships between them.

1.1.2 Graph for Data Model

We have two different settings for the data collections that we want to handle. First, a data collection can be multimodal where its data elements are of different modalities. Second, the data collection is interconnected where there are relationships between the data elements. Note that these configurations are not mutually exclusive as a data collection can both be interconnected and multimodal. In our setting, we need a common data model that can handle both the heterogeneity and interconnectivity of the data collections and, hence, the context-rich queries as well. Note that this covers the vanilla case where the query is unimodal and the query terms are not connected.

To this end, we propose to use graph as the underlying data model. First, graphs are ubiquitous as they arise naturally when there are connections among data elements such as social networks, information networks, biological networks or even molecular networks. In our setting, as our data collections are interconnected, graph would be a natural choice. Second, in the heterogeneous case, a graph can combine data elements from different modalities in an effortless way. For instance, given two data elements from different modalities, we can always integrate them by creating an edge between them. This connection can be based on any property that is shared between them. This is the power of graph as new data can always be integrated with existing data by creating edges connecting related data elements. As such, graph provides a unifying model to combine different data elements and modalities. Third, graph is a simple but powerful data structure as it can represent an enormous amount of data using only two fundamental elements: nodes and edges. It is also user-friendly as it supports visualization naturally.

1.2 General Approach

Before we proceed with the overview of our approach, we first describe the vector space retrieval model which is the framework that we build our approach upon.

1.2.1 Vector Space Model

In the vector space model, data elements and queries are both represented as vectors in the same d-dimensional vector space. The degree of similarity between a query and a data element, which is a proxy for the relevance of the data element, is measured by the distance between the vector representation of the query and that of the data element. By measuring the similarity between a query and every data element in the collection, we can return a ranked list of data elements ordered by relevancy.

The core component of the vector space model is a function to embed a data element or a query to its vector representation. There are two important requirements for the embedding function. First, both the query vectors and the data element vectors need to be in the same vector space. This requirement ensures that we can measure the distance between two vectors in a meaningful way. This requirement is usually guaranteed if the same mapping model is used for both the queries and the data elements. Second, the vectors need to capture the similarity between the data elements. In particular, two vectors should be close in the vector space if the data elements they represent are similar. This requirement makes sure that the distance in the vector space correlates with the similarity of the data elements for the retrieval to work.

Traditionally, the vector space model is used in document retrieval where the embedding function is the tf-idf vectorization [SMR08, BYRN11]. The vector dimensions represent the words in the vocabulary and the value at a specific dimension of a vector captures the importance of the word according to the document/query and the whole document collection. The importance of a word is measured based on the number of times it appears in the document and its uniqueness according to the whole data collection. As such, the mapping function satisfies the second requirement as similar documents should have similar word distribution, which makes their vectors to be similar as well.

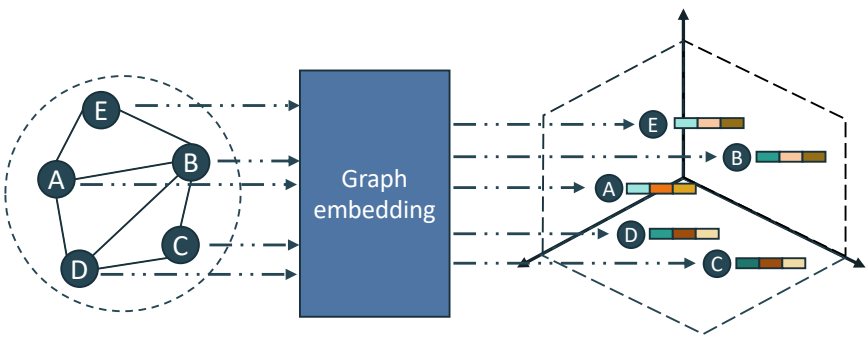


Figure 1.1: Graph embedding

1.2.2 Graph Embedding for Retrieval

To apply the vector space model in our setting, we need to construct an embedding function that satisfies the above conditions. As we use graph as the underlying data model, we follow the graph representation learning approach to create the embedding function. Graph embedding aims to create an embedding of every node in a graph such that the node embeddings satisfy some predefined requirements. A common requirement, for instance, is that the node embeddings should capture the closeness of the nodes in the graph. Node closeness can be defined in several ways such as based on their neighborhood or their cooccurrence on a random walk. An illustration of a graph embedding technique is shown in Figure 1.1. In this figure, the node embeddings are created such that neighboring nodes have close node embeddings. In our setting, we propose two graph embedding techniques: one to handle heterogeneous graphs, the other for connected queries. While there are several existing heterogeneous graph embedding techniques, they are not designed for information retrieval. On the other hand, our proposed techniques, which are based on the message-passing neural network, consider both the construction of node embeddings and query embeddings in a common framework. This enables clear reasoning of node embedding computation and extendability.

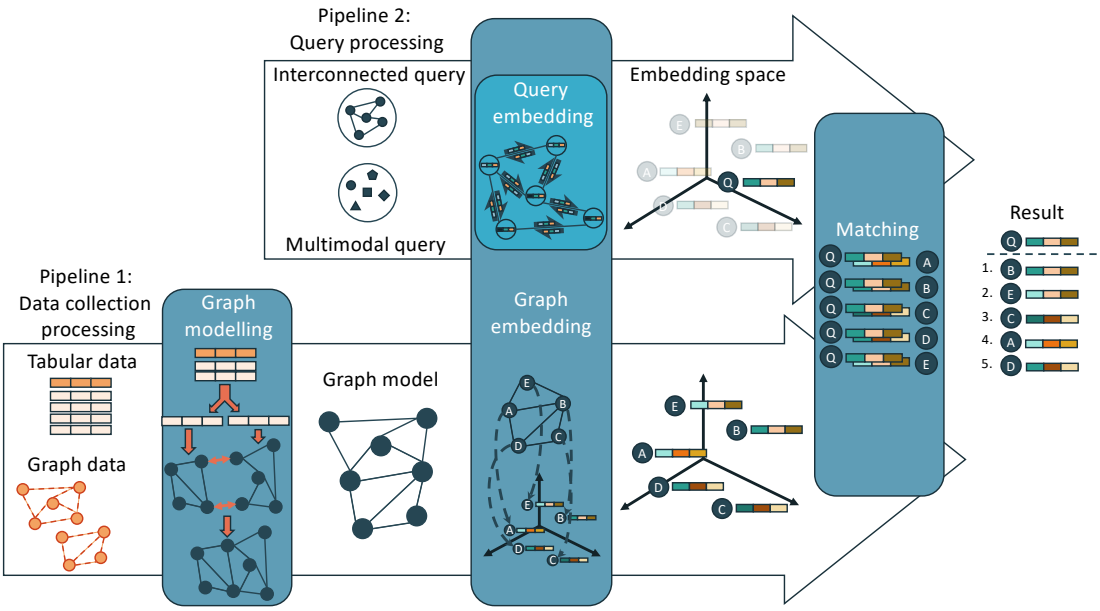


Figure 1.2: Architecture

Our general framework for graph embedding for retrieval is illustrated in Figure 1.2. There are two pipelines in our system. In the first pipeline which is usually offline, data collections are processed into graphs where graph embedding models are built. The second pipeline, which is an online process, involves using the graph embedding models to construct the query embeddings. The input to this pipeline are the queries which can be multimodal or interconnected. The first pipeline takes the data collections which can be in tabular or graph format. In the tabular case, a graph can be created by connecting

common data elements. This graph is used to construct a graph embedding model. The graph embedding model is built such that the embeddings of similar data elements have close embeddings. In the second pipeline, the query embeddings are created based on a query embedding model. This model ensures that the query embeddings and the node embeddings in the first pipeline are in the same embedding space. When a new query arrives, its embedding is created and matched with the node embeddings. This creates a ranked list of results which we return as answers to the query.

1.3 Research Questions

In this thesis, we aim to design an IR system that supports context-rich queries. We focus on two types of contextual information which are the *modalities* of the query terms and the *relations* between them. The foundation of such IR system is an underlying graph model that captures the connections between data elements in the data collection and different modalities in a data element. To design such system, we need to answer research questions in three aspects: *Heterogeneity*, *Connectivity* and *Scalability* as shown in Figure 1.3.

Heterogeneity. As data become heterogenous, queries for these data need to be heterogenous for better retrieval result. We first aim to support queries where the query terms can be of different modalities. We propose to use a graph to integrate different data elements and modalities in a principled manner. In addition, as we follow the vector space model, we need to develop graph embedding technique that can handle heterogeneous graphs and queries. This would allow us to create query and node embeddings for retrieval. To achieve these goals, we need to answer the following research questions:

- *How to create the heterogenous graph?* There are several ways a graph can be created from a multimodal data collection. Choices regarding what constitutes a node, how to connect two nodes or what to include as node feature can affect the quality of the embeddings and the training time. As a result, identifying what to include in the graph and how to structure it is highly important. In principle, the graph needs to capture the data elements, their modalities and their connections.

- *How to construct graph embedding for retrieval on heterogenous graphs?* The presence of different modalities make traditional graph embedding techniques not applicable. While several graph embedding techniques for heterogeneous graphs have been proposed recently, they are not designed for the information retrieval setting. We need a heterogeneous graph embedding technique that also supports the creation of a query embedding model in a structured and principled way.

- *How to build query embeddings that consider all the query terms and their modalities?* Since our queries are multimodal as well, a query embedding mechanism based on the above heterogenous graph embedding technique is required. This would make the query embeddings and the node embeddings of the heterogenous graph to be in the same embedding space for retrieval.

Connectivity. For data collections where the relationships between data elements are first-class citizens, queries for these data need to consider relationships between query terms as first-class citizens as well. A query in this setting can be modelled as a small graph where the nodes are the terms and the edges capture the term relationships. To answer these queries, it is akin to answer subgraph query search in the subgraph isomorphism problem. In this problem, we are given a large data graph and a small query graph and we need to identify subgraphs in the data graph which are isomorphic to the query graph. By constructing subgraph and query embeddings, we can answer subgraph search based on the embeddings instead of comparing them structurally. To solve this problem, we need to answer the following questions:

- *How to leverage graph embedding to speedup traditional subgraph isomorphism search techniques?* Traditional solutions to subgraph isomorphism search involve enumerating substructures in both the data graph and query graph. These substructures are used as indices to find candidate matching regions in the data graph. However, substructure enumeration and comparison is time-consuming. With subgraph and query graph embedding, we expect to speedup the comparison by comparing structures based on embedding distance.

- *How to construct query and subgraph embeddings that support subgraph retrieval?* To compare a subgraph and a query graph based on their embeddings, we need a model that can create close embeddings for structurally-similar subgraphs. This model needs to learn specific patterns belonging to the data graph as the subgraph embeddings need to capture these patterns as well. The model can then be used to create query embeddings for retrieval following the vector space model.

- *How to make subgraph retrieval fast even the problem is NP-Hard?* Even with embeddings, subgraph isomorphism search may suffer from long retrieval time. To overcome this problem, we propose to use a cache to store past results to answer similar queries. This requires a cache management strategy involving cache admission and cache eviction as well as a way to compare if two queries are similar.

Scalability. The third aspect we need to consider is the scalability of our graph embedding model. As the constructed heterogenous graph is built based on the data collection, it can contain million number of nodes and billion number of edges. To make our retrieval system practical, we need a way to scale our embedding technique beyond single machine. We propose to divide the graph and "conquer" each subgraph independently before merging them together. This requires us to answer the following questions:

- *How to divide the graph into subgraphs in a meaningful way?* Splitting the graph into smaller subgraphs enables independent computation of embeddings. However, the decomposition need to be properly designed such that it can support merging the embedding spaces created independently. This can be done by making the subgraphs share some common nodes. The subgraphs should have similar size

as well to prevent stragglers on commodity hardware. Finally, as the graph is large, we need a graph decomposition algorithm that can handle large graphs while satisfying the above requirements.

- *How to reconcile the embeddings learned from each subgraph?* As the embeddings are learned independently, they may belong to different embedding space. As a result, the embeddings need to be merged into one final embedding in a common embedding space to be used in a downstream task. The merging shall be done by leveraging the overlapping nodes between subgraphs for reconciliation.

- *How to make scaling graph embedding technique user-friendly and fault-tolerant?* Traditional techniques to scale graph embedding techniques are brittle and prone to errors. An error occurred during training is extremely costly as the whole training needs to restart from scratch. These techniques are also hard to use as it involves careful monitoring from users. As such, a more robust and user-friendly framework is required to allow any user to obtain graph embedding for large graphs.

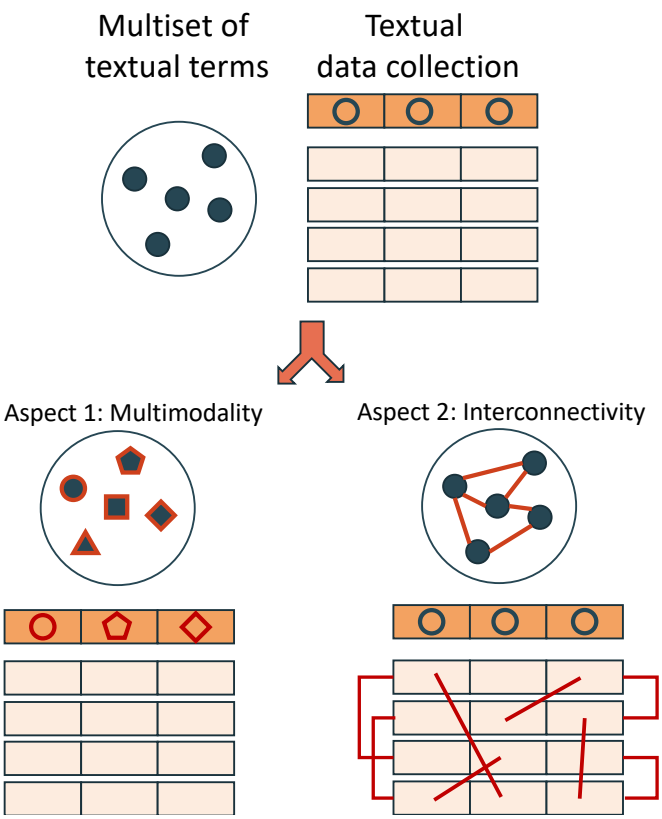


Figure 1.3: Context-rich query

1.4 Thesis Statement and Contribuions

This thesis contributes to the quest of achieving better retrieval quality by supporting richer contexts in the query.

Thesis Statement

Traditional information retrieval systems focus on supporting queries that contain independent query terms of textual format. To better capture users' intention and information needs, IR systems should support queries that are multimodal and interconnected.

We now present the main contributions of this thesis. Each of these contributions addresses one of the above research questions.

Heterogeneity. In Chapter 3, we solve the problem of heterogenous query retrieval. That is, given a query specified using different modalities, we need to find the data elements that satisfy this query. In particular:

- We propose a model to represent heterogenous data based on Heterogenous Information Networks (HINs) that can capture the semantic relations between data of different modalities. We construct a small HIN for each data element before combining them together to obtain the HIN for the whole data collection.
- We propose a heterogenous graph embedding model based on the message passing framework that can take into account the heterogeneity of the HINs. The message passing framework enables easy reasoning of the node embedding computation. This is the foundation for the construction of the query embeddings.
- We introduce a query embedding model that allow users to incorporate different modalities in the query. As the query embedding model is based on the heterogenous graph embedding model, it ensures that the query embeddings and the node embeddings are in the same space.

Connectivity. In Chapter 4, we handle the connectivity available in the query. We tackle the subgraph isomorphism problem based on graph embedding.

- We propose a graph embedding model to construct the embeddings of both subgraphs and queries. As the embedding model is constructed on the data graph, it can capture its specific patterns. As we use this model to construct both subgraph and query embeddings, we ensure that similar subgraphs/queries should have close embeddings.
- We improve traditional subgraph isomorphism algorithms by leveraging embeddings to filter matching candidates as the graph embedding model also creates node and edge embeddings which we can use as filtering criteria. The node and edge embeddings are better than traditional structural indices as they are fast to compute and compare.
- We present a cache management strategy based on embeddings that achieve higher level of cache hits. On the other hand, our cache management strategy is made possible by the embeddings as they enable fast comparison between a cached query and a new subgraph query.

Scalability. Training graph embedding models take a lot of time as the embedding techniques are mainly designed for single machine. In Chapter 5, we solve the problem of scaling any graph embedding technique in a distributed setting. We propose to divide the graph into subgraphs before performing the embedding of each subgraph independently. Finally, a reconciliation of subgraph embeddings is done to obtain the final embedding.

- We propose a node-centric graph decomposition algorithm to divide the graph into overlapping subgraphs. The node-centric algorithm enables distributed computation which allows us to scale into large graphs. We also propose a strategy to select shared nodes between subgraphs such that we can achieve better results.
- We propose a MapReduce-based framework to scale any graph embedding technique using Spark. Spark is a managed system which makes it easy to use for users. This also makes our approach to scaling graph embedding user-friendly.
- We present a reconciliation technique to merge the embeddings constructed independently. The shared nodes are considered as landmarks where the embeddings obtained from different subgraphs can be reconciled.
- We propose several optimization strategy regarding storage, computation to speed up the training process. We enable combination of Spark and an automatic differentiation framework to support training graph embeddings on GPU. This helps speedup the training process significantly.

The remainder of this thesis is organized as follows. Chapter 2 provides the background and the literature related to research problems discussed in this thesis. Chapter 6 concludes our thesis and discusses some possible future works.

1.5 Selected Publications

This thesis is based on the following research papers:

- **Duong, C. T.**, Tam Thanh Nguyen, Hongzhi Yin, Matthias Weidlich, Son Mai, Karl Aberer, Quoc Viet Hung Nguyen, "Efficient and Effective Multi-Modal Queries through Heterogeneous Network Embedding." In: **TKDE** (2021)
- **Duong, C. T.**, Hongzhi Yin, Dung Hoang, Minh Hung Nguyen, Matthias Weidlich, Quoc Viet Hung Nguyen, Karl Aberer, "Graph Embeddings for One-pass Processing of Heterogeneous Queries." In: **ICDE** (2020), pp.1994-1997
- **Duong, C. T.**, Dung Hoang, Hongzhi Yin, Matthias Weidlich, Quoc Viet Hung Nguyen, Karl Aberer, "Efficient Streaming Subgraph Isomorphism with Graph Neural Networks." In: **VLDB** (2021) pp. 730-742
- **Duong, C. T.**, Dung Hoang, Hongzhi Yin, Matthias Weidlich, Quoc Viet Hung Nguyen, Karl Aberer, "Scalable Robust Graph Embeddings with Spark." In **VLDB** (2022), pp. 914 - 922

The ideas for these research papers originated from the first author. The first author also contributed in majority to the texts, figures and experiments. The second author of the connectivity and scalability papers helped with running the experiments of these papers. The remaining authors, who had advisory roles, helped with proofreading, text editing of small parts of the above papers.

Despite the aforementioned papers, during my doctoral studies, I have also contributed to the following publications:

- Huynh, T. T., **Duong, C. T.**, Nguyen, T. T., Van Tong, V., Sattar, A., Yin, H., & Nguyen, Q. V. H. (2021). "Network Alignment with Holistic Embeddings". In **TKDE** (2021)
- Trung, H. T., Toan, N. T., Van Vinh, T., Dat, H. T., **Duong, C. T.**, Hung, N. Q. V., & Sattar, A. (2020). "A comparative study on network alignment techniques". **Expert Systems with Applications**, 140, 112883.
- Huynh, T. T., **Duong, C. T.**, Quyet, T. H., Nguyen, Q. V. H., & Sattar, A. (2019, August). "Network alignment by representation learning on structure and attribute". In **PRICAI**(pp. 698-711). Springer, Cham.
- **Duong, C. T.**, Dung Hoang, Quoc Viet Hung Nguyen, Ha The Hien Dang & Karl Aberer (2019). "Intrinsic Evaluation of Unsupervised Node Embedding". In **NeurIPS Workshop on Graph Representation Learning**, 2019.
- **Duong, C. T.**, Thanh Dat Hoang, Ha The Hien Dang, Quoc Viet Hung Nguyen & Karl Aberer (2019). "On Node Features for Graph Neural Networks". In **NeurIPS Workshop on Graph Representation Learning**, 2019.
- **Duong, C. T.**, Quoc Viet Hung Nguyen & Karl Aberer (2019). "Interpretable node embeddings with mincut loss". In **ICML Workshop on Learning and Reasoning with Graph-Structured Representations**, 2019.
- **Duong, C. T.**, Lebret, R., & Aberer, K. (2017). "Multimodal classification for analysing social media". arXiv preprint arXiv:1708.02099.
- **Duong, C. T.**, Nguyen, Q. V. H., Wang, S., & Stantic, B. (2017, September). "Provenance-based rumor detection". In **Australasian Database Conference** (pp. 125-137). Springer, Cham.

Chapter 2

Background

In this chapter, we provide several background topics and notations that are going to be used throughout this thesis. Additional background will be introduced in the later chapters if necessary. We begin by discussing the definition of graph and embedding in Section 2.1 before providing the background on graph embedding in Section 2.2. We continue with a discussion on scalable graph embedding in Section 2.3 before closing this chapter with a discussion on retrieval techniques in Section 2.4.

2.1 Preliminaries

Graph. We define a graph $G = \{V, E\}$ by its set of nodes V, set of edges $E \subseteq V \times V$. The graph can also be attributed where each node is associated with a set of features. For instance, if a node represents a user, its feature would be the information regarding the user such as date of birth, age, education. In other case where a node represents an image or an audio, the node is associated with the feature vector constructed from a feature extractor. The node features can be represented as a feature matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ where $n = |V|$ and p is the node feature size. Mathematically, the graph structure can be represented by its adjacency matrix $\boldsymbol{A}$ of size $n \times n$, where each row and column represents a node in G.

Embedding. Embeddings are vectors to represent some concepts in some numeric space. Yet, this representation shall be such that semantically related concepts have close representations, i.e., their geometric relation in the embedding space encodes their semantic relation. Compared to symbolic representations that consider each concept as independent, embeddings enable conclusions on the relation of the concepts based on their representations. Moreover, embeddings are succinct in the sense that with a d-dimensional embedding space (d is called the embedding size) where the domain of each dimension has size k, k^d concepts can uniquely be described. Figure 2.1 illustrates this schematically. The embeddings of the upper three graphs are assigned vectors that are close, while the one of the lower subgraph is more distant.

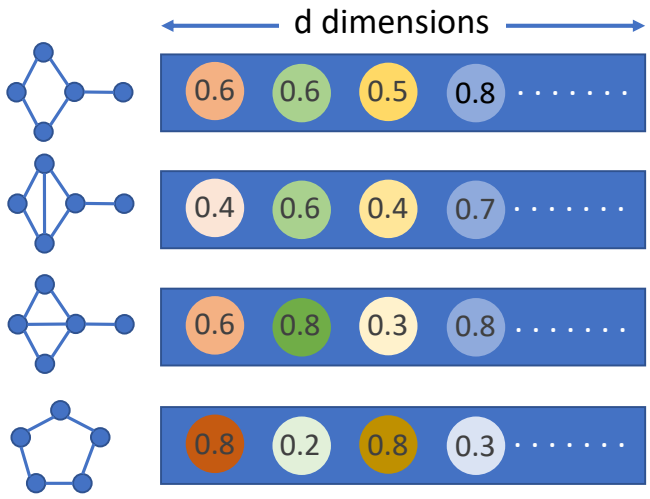


Figure 2.1: Distributed representation of graphs

2.2 Graph embedding

A vectorized representation of the nodes in a graph may be derived by graph embedding, which is realized by an *encoder* and a *decoder* [HYL17b]. The former is a function $h_\theta : V \to \mathbb{R}^d$ that constructs a d-dimensional vector (where $d \ll |V|$), aka embedding, for the node. The latter is a function that maps these vectors to domain-specific quality metrics. In case of a graph-based model, the decoder is typically a similarity metric that reflects the proximity of two embeddings in the vector space. The accuracy of the representation of the proximity between nodes by the similarity of their embeddings is captured by a *loss function*, denoted by L. When learning an embedding model, one tries to minimize this loss function by finding an optimal set of parameters for the encoder and decoder. The node embeddings then can be used in various downstream tasks such as node classification, link prediction.

The decoder and the loss function are used to incorporate our requirements for the node embeddings. For instance, if we have labels for the nodes, we may want embeddings of nodes of the same label to be close in the embedding space. On the other hand, if the labels are not available, we may want to encode the node closeness on the graph in the embedding space. There are several ways to measure node closeness either by their shortest path distance or the number of cooccurrences on a random walk.

Categorization. There are several ways to classify graph embedding techniques. Regarding model inference, graph embedding can be classified into two categories [HYL17b]: transductive and inductive models. Transductive models can only generate node embeddings for a node if this node has been seen during training. On the other hand, inductive models allow to compute node embeddings even for unseen nodes.

On the other hand, if we consider model architecture, graph embedding techniques can be classified into shallow models or deep models such as graph neural networks [ZCH+20, HYL17b]. The difference is in how the graph structure is incorporated into the model.

For shallow models, the graph structure is captured in the loss function while graph neural networks consider the structure explicitly in their computation.

Deep model methods can be classified into two categories: spectral and spatial graph neural networks [ZCH+20]. Spectral approaches extend the idea of the convolution operator in CNN to graphs. By constructing the spectral representation of the graph, the node embeddings can be learned. On the hand, spatial methods perform graph convolution directly on the graph structure. They can be modelled by a message passing framework where where nodes communicate with each others to compute the node embeddings. As such, these models are also called message passing neural network [GSR+17]. In the following, we first discuss transductive models before turning to inductive models.

2.2.1 Transductive models

Shallow models. Shallow models [PARS14, GL16] compute for each node a unique embedding directly. Shallow models are usually transductive as it needs to "see" a node to create its embedding. A notable example is DeepWalk [PARS14] and its node2vec variant [GL16]. In DeepWalk, random walks are performed on the graph from each node with a specific walk length. Then, these random walks can be considered as sentences while nodes are considered as words in a word2vec [MCCD13] model. This allows DeepWalk to create a node embedding for every node. As a result, DeepWalk implicitly models node proximity as two nodes are considered close if they appear in the same random walk. Other approaches to capture the network structure consider two nodes to be similar, if their neighbourhoods are highly overlapping [CLX15, OCP+16], which is commonly referred to as second-order node similarity. While shallow models are typically fast to train and can be used without initial node features, they require all nodes to be seen during training which limits their application in our setting. In our case, we would like to create embeddings for queries that are usually unseen.

Spectral graph neural network. Spectral methods operate on the spectral domain of graphs. A graph signal is transformed into the spectral domain based on the graph Fourier transform. The convolution operator is applied on the transformed graph signal in this spectral domain before they are transformed back using the inverse graph Fourier transform. The most popular method in spectral GNN is Graph Convolutional Network (GCN) [KW17]. In this method, the node embeddings are constructed from the node features and adjacency matrix as follows:

$$\boldsymbol{H}^{(l+1)} = \sigma(\hat{\boldsymbol{D}}^{-0.5}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-0.5}\boldsymbol{H}^{(l)}\boldsymbol{W}^{(l)})$$

where $\hat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$, $\hat{\boldsymbol{D}}$ is the degree matrix of $\hat{\boldsymbol{A}}$, $\boldsymbol{H}$ is the node embeddings obtained at the l-th layer and $\boldsymbol{W}$ is the parameter matrix at the l-th layer. As the adjacency matrix $\boldsymbol{A}$ is required to compute the node embeddings, spectral methods such as GCN require access to the whole graph structure. This is the main problem with transductive models that they can not be used to generate embeddings for unseen graphs.

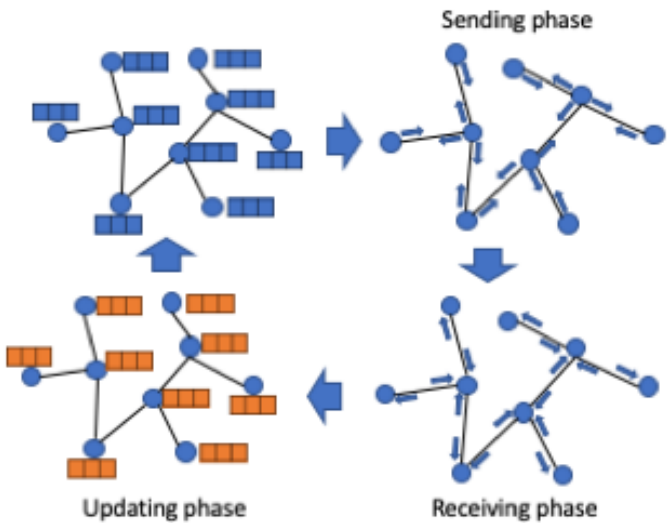


Figure 2.2: Phases in message passing neural network.

2.2.2 Inductive models

Before discussing different inductive techniques, we discuss the Message Passing Neural Network (MPNN) framework [GSR+17], which we used as the framework for our embedding techniques.

Message Passing Neural Network. In (MPNN) [GSR+17], a node representation is created by combining the representation of its own properties with those of its neighbors, through message-based interactions. A message sent from one node to its neighbors is constructed based on the node's current representation. Since messages are exchanged only between nodes connected by edges, the graph structure is incorporated. Message passing happens in several rounds, each involving three steps [DYH+20]:

Sending: A node u constructs a message in the i-th round based on its representation $z_u^{(i)}$. The node sends the message to a set of *selected* neighbors using a parameterized function $f_s^{(i)}$:

$$m_u^{(i)} = f_s^{(i)}(z_u^{(i)}).$$

Receiving: Once a node v received messages from all of its neighbors, denoted by $N(v)$, in a round, it aggregates them using a parameterized function $f_a^{(i)}$:

$$z_{N(v)}^{(i)} = f_a^{(i)}(\{m_u^{(i)}, \forall\ u \in N(v)\}).$$

Updating: A node updates its representation, combining its current representation with the aggregated messages:

$$z_v^{(i+1)} = f_u^{(i)}(z_{N(v)}^{(i)}, z_u^{(i)})$$

An MPNN can be formulated as a function $f(g, l)$, where g is a graph and l denotes the feature of the nodes. The function f represents the combination of the above functions used in the sending, receiving, and updating steps of all rounds and returns a set of node embeddings $\{z_u\}$ for each node in the graph. Note that the parameters of f need to be learned before the model can be used, though.

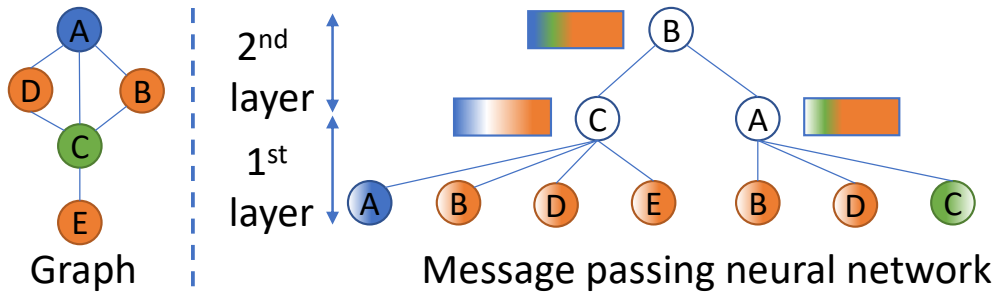


Figure 2.3: An MPNN with 2 layers

Example 1. *Given the graph on the left of Figure 2.3, an embedding for node B is created using a 1-layer MPNN as:* $z_B^{(1)} = f_u^{(0)}(z_{N(B)}^{(0)}, z_B^{(0)})$ *where* $z_{N(B)}^{(0)} = f_a^{(0)}(\{f_s^{(0)}(z_C^{(0)}), f_s^{(0)}(z_A^{(0)})\})$ *is the embedding of the neighborhood of B. In its basic form, function f_s of the sending step is parameterized by a matrix W_s, i.e., $f_s(z) = W_s z$. The aggregation function in the receiving step derives the mean of the node embeddings, i.e., $f_a(N(v)) = 1/|N(v)| \sum_{u \in N(v)} z_u$. Function f_u of the updating step is parameterized by a matrix W_a, before applying a non-linearity, i.e., $f_u(z_{N(v)}, z_v) = \sigma\big((z_{N(v)} + z_v)W_a\big)$. Based thereon, the node embedding of B is computed:*

$$z_B^{(1)} = f_u(\{m_u^{(0)} \mid u \in N(B)\}, z_B^{(0)}) = \sigma\big(((z_A^{(0)} W_s + z_C^{(0)} W_s)/2 + z_B^{(0)})W_a\big) \quad (2.1)$$

An MPNN may also be viewed from a node's perspective. Then, the operations to compute the embedding for a node u induce a k-layer tree, rooted at u. The embedding of u is based on the nodes at the i-th layer of the tree, which are neighbors of u within distance i in the graph. Information at the leaves of the tree is given by the features of the respective nodes, which is then aggregated to the root: The i-th layer employs the parameterized function f_i to aggregate the results of the i+1-th layer. Node u and its neighbors within distance k induce a subgraph, called the *receptive field* of u. The higher the number of rounds of message passing, the larger the receptive field. The embedding of u represents a summarization of its receptive field, in terms of both structure (as message passing follows the graph structure) and node features (as messages are constructed initially from node features).

Example 2. *The node-centric view is illustrated in Figure 2.3. Given the graph on the left and using a 2-layer MPNN, the embedding of node B is constructed by aggregating the embeddings of nodes C and A in the first layer. These embeddings are, in turn, constructed from the embeddings of their neighbors. This process is captured by a 2-layer tree rooted at B.*

Techniques. There are several GNN techniques that can be expressed by the message-passing framework. GraphSAGE [HYL17a] is one example of traditional MPNN where the sending, receiving and updating functions are the identity function, the mean function and the concatenation function with a linear transformation, respectively. One

Figure 2.4: Two non-isomorphic graphs

drawback with traditional MPNN is that it considers all neighbors of a node to be equally important. Graph Attention Network (GAT) [VCC+17] approaches this problem by incorporating a weighting scheme into the functions to differentiate between neighbors of a node. The weighting scheme is embeded into the receiving function of the MPNN.

For unsupervised graph embedding, another way to improve MPNN is by making the loss function more expressive. Deep Graph Infomax (DGI) [VFH+18] tackles this problem by creating node embeddings such that the mutual information between the local node embeddings and the global graph embedding is maximized.

On the other hand, there are a lot of research aiming to make GNNs to be able to distinguish different structures. For instance, traditional GNNs can not count triangles in a graph which makes it difficult to apply in social network setting where such a motif is widely present. For instance, in Figure 2.4, two non-isomorphic subgraphs would be consider as isomorphic as traditional GNNs based on message passing can not distinguish between them [MRF+19, BFW+21]. Several solutions for this problem have been proposed [MRF+19, BFW+21]. The general idea of these methods is to leverage higher-order structure such as subgraphs instead of nodes. These methods improve the discriminative power of GNNs but they are more computationally expensive.

Recently, there are several methods aiming to adapt GNN for heterogeneous graphs. Several techniques [WJS+19, FZMK20] are based on metapaths [SHY+11] which are random walks following some predefined walking strategies based on node types. By using metapaths, these techniques can essentially map heterogeneous graphs to homogeneous graphs where traditional approaches can be used [ZCH+20]. Our approach differs from these techniques as we can learn node embeddings for heterogenous graphs directly without constructing the metapaths. The metapath construction phase is usually time-consuming as well. Techniques [ZSH+19, HDWS20, SKB+18] that do not depend on metapaths use different weight matrices to encode embeddings from sampled neighbor nodes. Some methods [SKB+18] leverage a type-based sampling function to select nodes that are considered as neighbors of a vertice. Our proposed approach is similar to these techniques as we also use different matrices for each node type. However, our method differs from these approaches as we follow the message-passing framework. This enables clear reasoning of the node embedding computation, which leads to better extendability. In addition, our retrieval model based on graph embedding requires us to create query embedding for every query. However, it is not clear how a graph embedding model constructed on a graph can be used to create the subgraph/query embeddings embeddings.

In this thesis, we propose truncated message passing as a framework to construct query and subgraph embeddings based on the message-passing framework.

2.2.3 From node to graph embedding

Graph embedding techniques first and foremost are designed to create node embeddings. From these node embeddings, a (whole) graph embedding can be created using a *readout* function [XHLJ18]. This function aggregates all the node embeddings into a single embedding for the whole graph.

Global Pooling. Global pooling is the vanilla version of graph pooling methods. From the node embeddings, the graph embedding can be constructed by either summing or averaging over all the node embeddings. For example, given the node embeddings $\{z_v\}$ of a graph G, the graph embedding can be constructed with global mean pooling as follows:

$$z_G = \frac{1}{|G|} \sum_{v \in G} z_v$$

While global pooling is the most common method, they consider all the nodes to be equally important, which hinders their performance.

Differential Pooling. One method that aims to fix this problem is differential pooling or DiffPool [YYM+18]. DiffPool follows the compositional structure of CNN where it pools nodes into subgraphs before pooling subgraphs into graph. Given a graph with an adjacency matrix $\boldsymbol{A}$, DiffPool aims to learn a binary matrix $\boldsymbol{S} \in \{0,1\}^{n \times k}$ that combines "related" nodes into subgraphs: $\boldsymbol{A}' = \boldsymbol{S}^T \boldsymbol{A} \boldsymbol{S}$ where n is the number of nodes in $\boldsymbol{A}$, k is the number of subgraphs or nodes in the new graph and $\boldsymbol{A}'$ is the adjacency matrix of the new graph. In this new graph, each node represents a subgraph in the old graph. As such, sequential applications of DiffPool can be used to obtain the final graph embedding. The matrix $\boldsymbol{S}$ can be learned in an end-to-end manner from a downstream task such as graph classification.

While pooling methods such as DiffPool can be used to create subgraph embeddings, they are not applicable in our setting. First, which subgraphs are selected to create subgraph embeddings are not controllable as they are learned in an end-to-end manner. Second, the subgraph embeddings are created as the results of the learning process. It is not possible to create subgraph embeddings without learning. In our setting, we would like to create subgraph embeddings for subgraphs of our choice. Our proposed method which is based on truncated message passing is able to create subgraph embeddings for any subgraph without a separate learning process.

2.3 Scalable graph embedding

Approaches to scale graph embedding techniques to large graphs are classified as *centralised* or *distributed*. Centralised approaches, e.g., SIGN [RFC+20], Cluster-GCN [CLS+19], SGN [WJZ+19], MILE [LGP18], rely on 'simpler' models to achieve scalability. The embedding models of SGN [WJZ+19] and SIGN [RFC+20] consist of several layers of

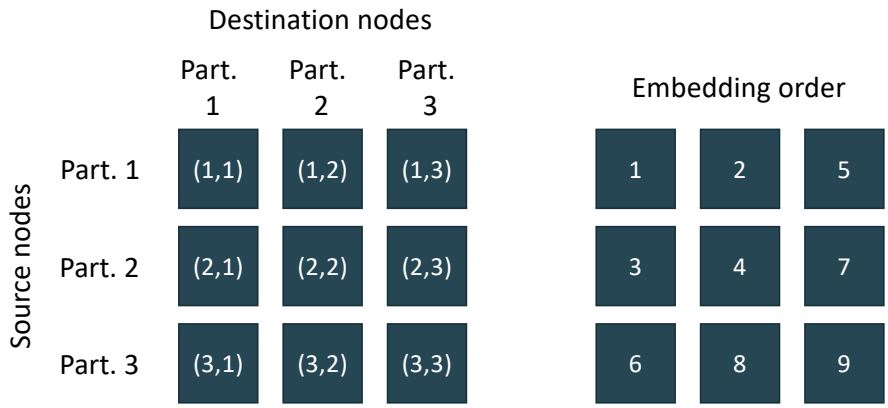


Figure 2.5: PBG embedding order (adapted from Figure 1 [LWS+19])

matrix multiplication on the node features and graph adjacency matrix. In particular, SIGN [RFC+20] constructs the node embeddings as follows:

$$\boldsymbol{Z} = \sigma([\boldsymbol{X}\boldsymbol{\Omega_0}, \boldsymbol{A_1}\boldsymbol{X}\boldsymbol{\Omega_1}, \cdots, \boldsymbol{A_r}\boldsymbol{X}\boldsymbol{\Omega_r}])$$

where $\boldsymbol{A}_i$ are modified versions of the adjacency matrix $\boldsymbol{A}$ and $\boldsymbol{\Omega_i}$ are the parameters of the model. The node embeddings $\boldsymbol{Z}$ can then be fed to a node classifier such that the whole model can be learned in an end-to-end manner. SGN [WJZ+19] approaches this problem in a similar manner where the node embeddings can be computed as follows:

$$\boldsymbol{Z} = \boldsymbol{S} \cdots \boldsymbol{S}\boldsymbol{X}\boldsymbol{\Omega_1} \cdots \boldsymbol{\Omega_r}$$

where $\boldsymbol{S} = \hat{\boldsymbol{D}}^{-\frac{1}{2}}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-\frac{1}{2}}$, $\hat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}$ and $\hat{\boldsymbol{D}}$ is the degree matrix of $\hat{\boldsymbol{A}}$. As these models are composition of matrix multiplication, they are scalable as these computations can be scaled to many GPUs. On the other hand, MILE [LGP18] first abstracts the graph into a smaller one to perform the embedding. It consists of three phases: graph coarsening, graph embedding and embedding refining. In the graph coarsening phase, connected nodes in the graph are merged into supernodes, which reduces the size of the graph significantly. This phase can be done repeatedly to obtain a graph that can be handle by a graph embedding technique. In the graph embedding phase, the supernode embeddings of the coarsened graph is obtained by any graph embedding technique. In the final phase, the embeddings are refined based on the matching between the supernodes and the original nodes. As these centralised approaches use only a single machine, they are inherently limited by the machine's capacity.

Distributed approaches leverage a cluster of compute nodes. They are complementary in the sense that they may rely on centralised techniques for graph embedding on each individual compute node. To date, there are two distributed graph embedding frameworks, PBG [LWS+19] and DGL [ZMW+20]. PBG is tailored to scale link-prediction-based shallow embedding techniques that use negative sampling. It has been proposed for compute nodes with shared storage that also communicate during training. The main idea of PBG is to partition the graph into several subgraphs such that the

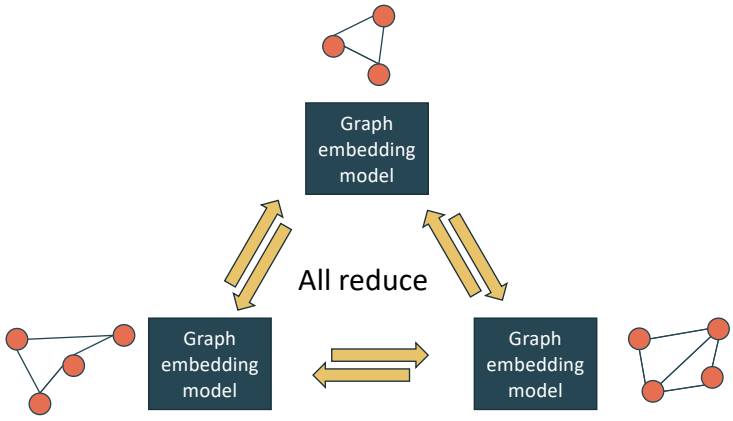


Figure 2.6: All reduce approach

training can be done independently in different compute nodes and each subgraph can fit into a compute node's memory. As shown in Figure 2.5, PBG divides the graph into several partitions. This also divides the edges into several buckets of intra-subgraph edges (edges between nodes in the same partition) and inter-subgraph edges (edges between nodes in two different partitions). To make the embeddings be on the same space, the training is actually done in a semi-sequential manner. For instance, the embeddings of edges between partition 1 and 2 can only start if the embeddings of nodes in partition 1 have been obtained. As such, the embeddings of nodes in partition 2 have fewer degree of freedom as they are dependent on the node embeddings of partition 1.

DGL also partitions the graph into several subgraphs such that each compute node can handle a subgraph. The graph embedding process is done on all compute nodes in a synchronous manner. After each training epoch, the gradients need to be synchronized across all machines. This can be done either by a parameter server in which all compute nodes send their gradient updates to the server. The parameter server then combines the updates and sends the combined gradients to all compute nodes. Another method is all-reduce where the compute nodes send the gradients to other nodes such that at the end of this process, every compute node receives the gradient updates from all the others.

In summary, both of these techniques require constant communication between compute nodes or to the parameter server. As such, they incur a lot of communication overhead. In addition, as these approaches are synchronous, worker nodes need to wait for others nodes to finish computation before synchronization. This is problematic in the presence of stragglers where all worker nodes need to wait for a long running task. In the case of the parameter server, it becomes a critical point of failure as if the server fails, the whole training process could be lost.

In this thesis, we propose a fully parallelization method based on the distributed model where worker nodes work independently and embeddings are "synchronized" by a reconciliation step. As such, our work, PBG and DGL are similar as we strive for

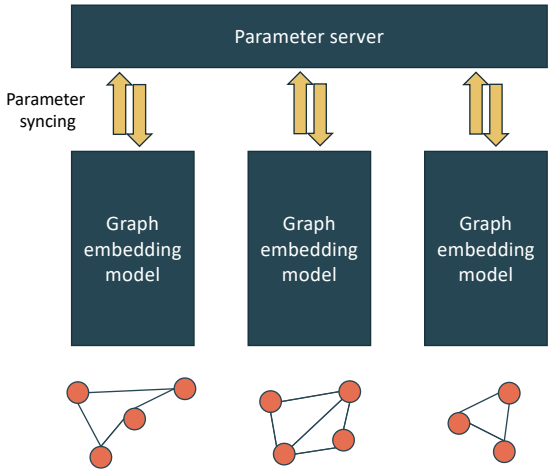


Figure 2.7: Parameter server

scaling embedding techniques by distributing computation across compute nodes. Yet, there are several differences which enable our approach to have better speedup and faster training time with less overhead. First, our focus is on a shared-nothing infrastructure, as commonly encountered in compute clusters. Both DGL and PBG require continuous communication between nodes, which slows down the training process. Second, PBG performs random partitioning of the graph. DGL proposes a centralised approach to graph partitioning, which cannot handle extremely large graphs that exceed a single node's capacity. Our approach includes a distributed algorithm for graph decomposition. Third, DGL and PBG are susceptible to node failure, even though the probability of node failure increases with the cluster size. Our approach enables a fault tolerant implementation, due to the reduced inter-node communication and the proposed checkpointing approach.

2.4 Information retrieval

Information retrieval aims to provide users with easy access to information. Users specify their information need in the form of *queries* in which is a set of query terms. The role of an IR system is to retrieve all the relevant documents according to the query [BYRN11]. An IR system typically consists of a *data collection* which is usually a set of documents indexed to speed up retrieval [SMR08, BYRN11]. When a user specifies a query, the query could be expanded with additional information such as synonyms, spellings before it is compared with the indexed documents to return a subset of results to the users. The results are usually ranked according to their perceived relevance to the user's information need. This is usually done by comparing the query representation with the document representation according to some ranking function [BYRN11].

Techniques in information retrieval differ in how they create the query, document representation and the ranking function. In general, they can be classified mainly into three categories: *set-based*, *vector-based* and *probability-based* [BYRN11, SMR08]. In *set-based*

models, the document and the query are represented as sets of keywords. A document is relevant to a query if the query terms are available in the document. For instance, $q = k_1 \wedge (k_2 \vee \neg k_3)$ is a query in the set-based models in which k_1, k_2, k_3 are the keywords. Every document and query are then represented in their disjunctive normal form (DNF). For instance, this query then can be converted as follows: $[(1,1,1) \vee (1,1,0) \vee (1,0,0)]$. By comparing the DNFs of the query and the document, we can figure out whether a document is relevant to the query or not. As such, documents can only be relevant or irrelevant to the query, which is one drawback of the set-based model [BYRN11]. Another drawback is that all query terms or keywords in the documents are considered to be equivalent [BYRN11]. On the other hand, the model is simple as it allows for combination of query terms using Boolean operators.

Vector-based models overcome these drawbacks by representing both the queries and the documents as vectors. The ranking function is the distance between the vectors. For textual data, the document and query vectors can be created based on the tf-idf or the latent semantic indexing model. In the tf-idf model [SMR08], given a data collection D and a vocabulary V, the value of a term in a document is computed based on the frequency of the term in the document and the uniqueness of the term in the whole collection. More precisely, the tf-idf value of a term t according to a document d is computed as follows:

$$tf - idf(t,d) = n_{t,d} \times log \frac{|D|}{|\{d \in D : t \in d\}|}$$

where $n_{t,d}$ is the number of times the term t appears in the document d, D is the data collection. For a document, we can compute the tfidf values of every term in the vocabulary. These values can then be used to construct the document vector.

Latent semantic indexing [SMR08, BYRN11] (LSI) approaches the problem from a different angle. It first constructs the term-document matrix $\boldsymbol{M}$ where each element is the frequency of the term in the document or a derivation of this frequency. This matrix is highly sparse as a term may not be contained in all documents. LSI aims to approximate this matrix using singular value decomposition. Let $\boldsymbol{M} = U\Sigma V^T$ be the singular value decomposition of $\boldsymbol{M}$. A low-rank approximation of the matrix $\boldsymbol{M}$ can be constructed by selecting the top-k largest singular values and their corresponding vectors. A document vector can be constructed from its corresponding singular vector and singular value. Note that the dimensions of the vectors are considered to represent some latent topics. The vector-based model is the most common as it is the most robust ranking strategy as it considers keyword importance and provides a ranked list of results [BYRN11, SMR08].

Probability-based models first need to construct languages models based on the documents in the data collection. A language model would capture how the documents are created. Based on the language model, he relevancy of a query to a document can be calculated based on the likelihood that both are created from the same language model. The main drawback of probability-based models is the need to construct language models

which require careful parameter tuning. In addition, probability-based models usually have longer construction and retrieval time [BYRN11].

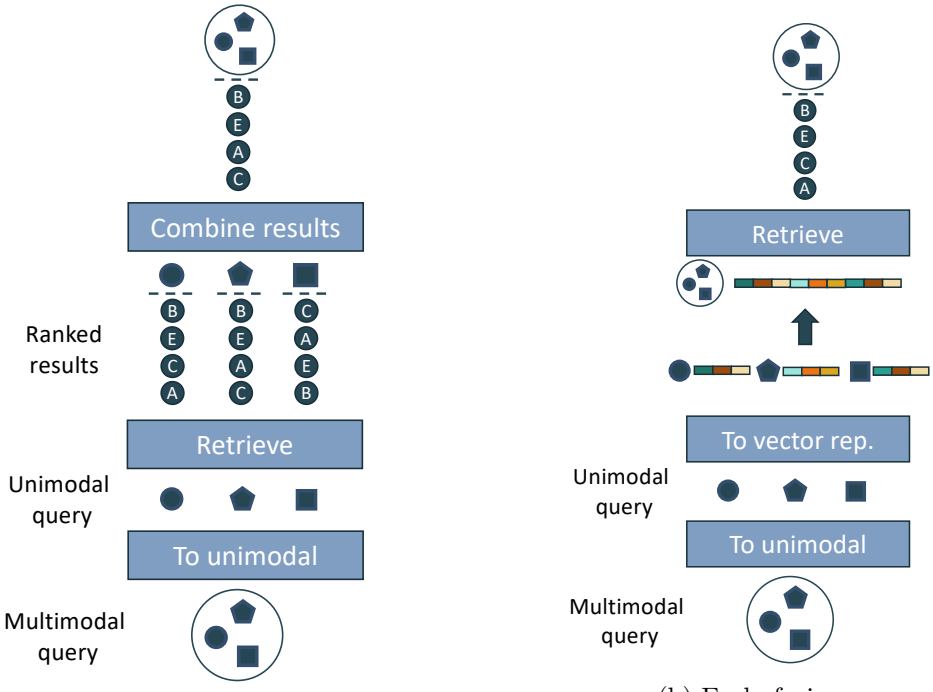
In our setting, we follow the vector-based retrieval approach for its robustness and applicability [BYRN11]. However, traditional vector-based models are designed for text retrieval where the queries and the documents are of textual modality, which is limited when the documents become multimodal. In addition, the query terms in a query are considered to be independent which limits the expressiveness of the query. In the following, we first discuss techniques that support multimodal queries before turning to techniques that consider query term relations i.e. subgraph retrieval.

2.4.1 Multi-modal query

Multi-modal IR systems can process homogeneous queries or heterogeneous queries. While systems to support homogeneous queries are easier to implement and have been employed in first-generation commercial applications [WYO+16, WOY+14], heterogeneous queries become more popular due to extending information needs of users. Yet, systems designed for homogeneous queries cannot directly handle heterogeneous data. It was therefore suggested to create a common representation of heterogeneous data [WYO+16, CLW+16, DSV+18], before transforming a homogeneous query to this common space and determining the relevant results via nearest-neighbour search. To provide more direct support for querying heterogeneous data, cross-modal retrieval systems, such as [DSV+18, CLWL17], have been proposed.

Moreover, systems that support queries with multiple modalities have been proposed [YKY+18, DSV+18, CLWL17]. These systems answer a multi-modal query through a combination of several uni-modal queries. There are mainly two ways to combine uni-modal queries: late fusion and early fusion as shown in Figure 2.8. *Late fusion* [EHSM08, MMM15, SSH14] combines the results obtained for each modality in isolation while *early fusion* [XHS+18, BBZ17, SY17] embeds uni-modal queries based on different feature vectors that are combined with an aggregation function. Late fusion systems [EHSM08, MMM15, SSH14] answer queries for each modality separately and then merge the respective results. For instance, in [EHSM08], results for images and texts are obtained separately and then combined. Due to the need to pass over the data multiple times, these systems show performance issues. Early fusion systems such as [XHS+18, BBZ17, SY17] computes different representations for the modalities and embeds the multi-modal queries into these vector spaces. Closest to our work is [HBZ+18] where the authors propose a technique to construct query embeddings for multimodal queries based on mappings between node types. However, the technique is tailored for conjunctive logical queries on knowledge graphs while we focus on multiset of query terms on heterogeneous information network. In general, knowledge graphs are more complex than HIN as there can be multiple edges between two nodes while the number of node and edge types could be higher. However, this complexity is not desirable in our setting as they can lead to model overfitting and longer training time. Going beyond the state-of-the-art, we process multi-modal queries with a single pass over the data, while incorporating the

relations between modalities. This is achieved by using a heterogenous graph embedding model that can construct the query embedding in one pass.



(a) Late fusion

(b) Early fusion

Figure 2.8: Late vs. early fusion

2.4.2 Subgraph retrieval

The problem of answering queries where query terms are connected is akin to the subgraph isomorphism problem. In this problem, a query is a small graph and the problem is to find matching subgraphs in a larger data graph. Techniques to solve this problem can be classified into two categories: single-query and multiple query retrieval.

		#data graphs	
		Single graph	Multiple graphs
#queries	Single query	VF2 [CFSV04], TurboISO [HLL13] BoostISO [RW15], WaSQ [LZ19]	TurboFlux [KSH+18] [ZYP07, ZQYC19]
	Multiple queries	MQO [RW16], Ours	n/a

Table 2.1: Categorization of different subgraph isomorphism techniques

Single-query subgraph isomorphism. Many approaches have been developed to answer a single subgraph isomorphism query, by leveraging structural equivalence between the query graph and the data graph. The mapping is usually constructed iteratively, preserving the nodes' connectivity. Notable representatives are VF2 [CFSV04], TurboISO [HLL13], BoostISO [RW15], WaSQ [LZ19], CPI [BCL+16]. VF2 [CFSV04] is the

traditional search algorithm. It identifies matching candidates between the subgraph and the data graph by comparing the node labels and the nodes' degrees. TurboISO [HLL13] improves the mapping construction by an additional data structure, called matching regions. During the subgraph search, only nodes in these regions are considered, which reduces the runtime significantly. BoostISO [RW15] follows a different direction to reduce the search time. It adapts the data graph by merging nodes that are structurally similar, which yields a hypergraph as the basis for the search. This enables finding hyper-mappings from the query graphs to the hypergraph, which can be expanded to obtain normal mappings. Moreover, WaSQ [LZ19] performs rewriting of the query graph to match structures for which partial mappings are known already. Based thereon, the final mapping is derived from the partial one.

While our focus has been on streaming subgraph isomorphism, our approach may also be used to answer a single query. The presented embeddings of nodes can be seen as an additional data structure to support subgraph search. They serve a similar role as node labels or node degrees employed by existing techniques to filter the set of candidate nodes in the construction of a mapping.

Another way to speed up subgraph isomorphism is to identify features that are representative of subgraphs for comparison [BFG+10, KKM11, BCL+16, KNT15]. These features can be considered as subgraph indices similar to our subgraph embeddings. CPI [BCL+16] constructs a data structure called compact path index, which is similar to a spanning tree, to support subgraph search. Compact path index can be considered as another way to compress the data graph by first considering nodes on the compact path index, which is similar to a spanning tree. GGSX [BFG+10] considers paths with bounded lengths as features. These paths are stored in a suffix tree to speed up the search. In addition to paths, CTIndex [KKM11] also uses cycles to create graph fingerprints. There are several problems with structure-based indices, though. First, they are not efficient as identifying structure or motifs in graphs is often a subgraph isomorphism problem in itself. Second, they often require users to identify which structures are relevant to the subgraph isomorphism problem. This is a tedious task as these structures are typically specific to a graph. Our embedding-based index avoids these issues as the embeddings can be learned efficiently and capture the graph structure in an automatic manner.

Multiple-query subgraph isomorphism. Recently, the problem of answering multiple subgraph isomorphism queries at the same time has attracted attention. The multi-query setting enables the identification of common structures among the queries, which provides an angle for optimization as exemplified for SPAQRL queries over RDF graphs in [LKDL12]. This approach divides queries into groups based on their edge labels, Jaccard similarity, and benefits for batch optimization. Then, a common subgraph pattern is extracted per group and the queries are rewritten to comprise the pattern and optional constraints. A query engine that supports such optional constraints is used to answer the original queries. MQO [RW16] tackles the multi-query subgraph isomorphism search for general graphs. It processes queries in batches. For the queries in the

same batch, common structures are identified. As a matching subgraph for a common structure can be used for all queries in the batch, this reduces the set of candidate nodes. However, MQO organizes them in a containment tree, called pattern containment map (PCM), in which a directed edge connects queries, where one is a subgraph of the other. Queries are then answered in a top-down manner with respect to the PCM. This allows MQO to use mappings of parent queries to derive answers for their child queries. While MQO is close to our work in terms of striving for reuse in subgraph isomorphism search, there are several important differences. First, MQO is cache-oblivious, whereas our approach heavily relies on caching. Caching makes it possible to reuse not only immediate results (e.g., in the same batch of queries), but the results from all past queries can, potentially, be reused. Second, MQO is a batch processing algorithm that processes one set of queries after another one. We presented a proper stream processing algorithm. Applying MQO over streams would require to partition the stream into batches, which is not practical. Third, MQO exploits the structure of query graphs in the same batch to identify similar queries, which is inefficient time-consuming. By leveraging subgraph embeddings, our approach handles query graphs much more efficiently.

Orthogonal to our work is TurboFlux, a subgraph isomorphism system for handling a streaming data graph [KSH+18]. Here, a standing query is posed against a data graph for which the structure changes over time. This is the mirrored case of our setting, in which the data graph is static and a continuous stream of queries needs to be evaluated. Another related problem is multi-data-graph subgraph isomorphism search [ZYP07, ZQYC19]. This problem is about finding mappings of a query, not for one, but several data graphs. We foresee that our approach can be extended to this setting by learning an embedding for each data graph.

Chapter 3

Heterogeneity - Graph Embedding for Heterogenous Data

Graph Embeddings for One-pass Processing of Heterogeneous Queries

ICDE 2020

Efficient and Effective Multi-Modal Queries through Heterogeneous Network Embedding

TKDE 2021

The heterogeneity of today's Web sources requires information retrieval (IR) systems to handle multi-modal queries. Such queries define a user's information needs by different data modalities, such as keywords, hashtags, user profiles, and other media. Recent IR systems answer such a multi-modal query by considering it as a set of separate uni-modal queries. However, depending on the chosen operationalisation, such an approach is inefficient or ineffective. It either requires multiple passes over the data or leads to inaccuracies since the relations between data modalities are neglected in the relevance assessment. To mitigate these challenges, we present an IR system that has been designed to answer genuine multi-modal queries. It relies on a heterogeneous network embedding, so that features from diverse modalities can be incorporated when representing both, a query and the data over which it shall be evaluated. By embedding a query and the data in the same vector space, the relations across modalities are made explicit and exploited for more accurate query evaluation. At the same time, multi-modal queries are answered with a single pass over the data. An experimental evaluation using diverse real-world and synthetic datasets illustrates that our approach returns twice the amount of relevant information compared to baseline techniques, while scaling to large multi-modal databases.

3.1 Introduction

The Web is built from heterogeneous representations of information, including texts, images, and videos. Recently, the rise of social media led to further rich and complex data modalities gaining increasing importance, among them user profiles, hashtags, and keywords. Given the rich semantics of heterogeneous data and the large data volumes faced in practice, retrieval of relevant information remains a challenging task for data on the Web [WOY+14].

A multi-modal information retrieval (IR) system takes as input a set of multi-modal queries and returns a ranked list of elements selected from a corpus of multi-modal data. It generalizes the model for uni-modal queries over textual corpora of web-pages and documents, which has been the primary concern of IR systems for several decades [WYO+16]. Various systems that support non-textual retrieval have been proposed recently [WYO+16, DSV+18]. Here, a user searches for information represented as images or audio based on queries that include several modalities beyond textual keywords.

Existing IR systems deal with heterogeneity of the queried data by constructing a vectorized representation of it. However, these systems lack support for multi-modal queries and focus on textual queries. This has negative implications on the retrieval accuracy due to the inherent textual ambiguity. In addition, textual queries limit users in effectively formulating their information needs [SY18]. Consider a scenario in which a user searches for a song in a music library. Instead of relying only on keywords without any semantics, a multi-modal query supports the explicit specification of a composer or album of the song. By incorporating these modalities, retrieval becomes more effective.

Multi-modal queries as outlined above are different from faceted search. The latter classifies data into semantic categories (facets) to filter results *after* retrieval [VAFK17]. Multi-modal queries, in turn, incorporate modalities in the identification of the relevant results that shall be retrieved.

Recently, first IR systems started to support the specification of multi-modal queries [DSV+18, CLWL17]. Yet, they do not genuinely evaluate multi-modal queries, but consider them as a combination of several uni-modal queries, each of which covering a different modality. Then, the retrieval results are constructed by combining the answers to different uni-modal queries via *fusion*. To this end, *late fusion* [EHSM08, MMM15, SSH14] combines the results obtained for each modality in isolation, which is inefficient as it requires several passes over the data. Another approach is using *early fusion* [XHS+18, BBZ17, SY17], which embeds uni-modal queries based on different feature vectors that are combined with an aggregation function. While such an approach requires only a single pass over the data, it ignores the relations between modalities. Hence, it breaks the semantic structure of the data, so that features of retrieved elements may not belong to the same data entity.

In this chapter, we provide a novel angle for multi-modal IR based on a vectorized representation of a *heterogeneous information network* (HIN). The HIN model captures

the rich semantics of both, multi-modal data and queries. Our contributions are summarized as follows.

- *A model for heterogeneous data:* We introduce a representation of heterogeneous data based on HINs to capture the semantic relations between data of different modalities of the same entity [SH13, SLZ+16]. This includes a graph/network embedding model to produce a feature vector for each node in the HIN based on its modality.
- *Multi-modal query model:* We propose a query model that enables users to incorporate different modalities. We construct a query embedding based on the HIN embedding of the queried data through novel mapping and combination operators. This way, we ensure that query and data embeddings are defined over the same space, which facilitates an accurate relevance assessment.

We evaluated the proposed approach with a set of diverse real-world and synthetic datasets. Our techniques turn out to be both efficient, scaling linearly to hundreds of thousands of data elements, and effective, retrieving twice the amount of relevant tuples compared to baseline techniques. The remainder of the chapter is organized as follows. Next, we motivate and formulate the addressed problem in Section 3.2. Then, Section 3.3 gives an overview of our approach. The embedding of HINs is discussed in Section 3.4, while query embedding is explained in Section 3.5. Experimental results are presented in Section 3.6. We conclude in Section 3.7.

3.2 Problem Formulation

Below, we first present a motivating example (Section 3.2.1) and a formal model (Section 3.2.2), before we formulate the problem addressed in this work (Section 3.2.3).

3.2.1 Motivation

Running example. Let us consider the following setting, in which a user searches for a song.

Example 3. *A user searches for the song 'Jenny of Oldstones', but does not remember its title. The user recalls that it is a song from the soundtrack of 'Game of Thrones'. Yet, using this as a keyword is not effective, since the retrieved results are mostly related to the movie itself. Specifying a query that explicitly includes a modality 'music album' with the value being 'Game of Thrones' is more precise, but still leads to many songs that have been included in the soundtrack. A user may then try to identify the song by querying for other modalities. For instance, the user may recall some terms of the lyrics, such as 'winter' or provide a small audio sample of the song. Also, the user may search for songs of specific singers, e.g., 'Florence', or composers, such as 'Dan Weiss'. Yet, considering each of these queries in isolation, long lists of songs will be returned, as each modality on its own does not enable a precise identification of a specific song.*

The above is a common example of a retrieval task in which a user needs to specify multiple queries in several modalities. In the end, a user has to identify the relevant results in large collections of retrieved elements, with the risk of not finding the requested element at all. Note that we use a music database as an example for multi-modal information retrieval. Yet, our techniques are independent of a specific domain since they adopt a generic graph-based representation of heterogeneous data.

Limitations of traditional approaches. Traditional IR systems fail to satisfy information requirements such as illustrated above, as they support search over one modality only. Consequently, they potentially miss out on important semantics. The limitation is not bound to modalities that are textual, but is also observed for other types of data, such as videos and images. Again, limiting a user to formulate a single, unimodal query may make it impossible to capture the users' true intention. In practice, this limitation is often addressed by incremental algorithms, such as query refinement and user feedback [JFLW17]. Since the user intent cannot be specific precisely, multiple passes over the data are needed to refine a search result.

Against this background, it was suggested to combine the results of several uni-modal queries [Abb09]. Yet, fusing search results of different queries neglects that the relevance ranking obtained for different modalities might be incompatible. In addition, such an approach does not scale well to large data, since it requires multiple passes over the data to answer the uni-modal queries. A different angle is followed by IR systems with 'querying by example' functionality over multimedia databases. Here, a user may conduct retrieval by specifying a data sample [MTX13]. Yet, the respective systems are highly domain-specific and are not applicable in the absence of such a sample, as in Example 3.

3.2.2 A Multi-Modal Query Model

We design a model for multi-modal queries as follows.

Data model. We capture heterogeneous data by a set of tuples $D = \{d_1, \ldots, d_n\}$ and a set of attributes $\mathcal{A} = \{A_1, \ldots, A_m\}$. Each attribute is considered as a modality, which defines a set of possible attribute values $A_i = \{t_{i1}, t_{i2}, \ldots\}$. Each tuple is defined as a vector $d = (a_1, \ldots, a_m)$ where a_i is a subset of A_i. This model is generic and applicable for various types of data [LGZ+18]. For instance, a set of relational tables can be flatten into a single table and domain-specific features, such as the audio embedding of a song, may also be represented through a concatenation of features in the tuple vector [JSN+17].

Example 4. *A song is modelled as a tuple of* (title, album, lyrics, singer, composer, audio)*. The song described in Example 3 is then represented as* ({*Jenny of Oldstones*}, {*Game of Thrones*}, {*winter, etc.*}, {*Florence*}, {*Dan Weiss, etc.*}, {*wav-file*})*.*

Query model. A user's query over multi-modal data is captured by a set of uni-modal queries $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$. Each of them relates to a different modality and specifies a subset of its domain, $Q_i \subseteq A_i$. With all queries except one being empty, the

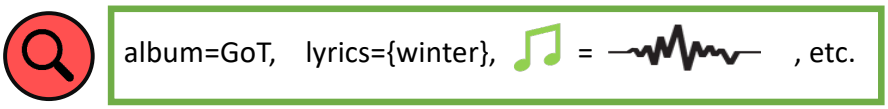


Figure 3.1: An example of a multi-modal query.

model captures the setting of traditional uni-modal information retrieval. Multi-modal information retrieval happens, if $\mathcal{Q}$ contains at least two non-empty queries. We denote the set of all values of all modalities as $\Omega = \cup_{A \in \mathcal{A}} A$. Similarly, the set of all queried values is $q = \cup_{Q \in \mathcal{Q}} Q$, which is a subset of Ω.

Note that, in addition to user-specified queries, this model also supports the paradigm of 'querying by example'. In that case, the queried values are derived directly from a selected sample of the database.

Example 5. *The user's information need in Example 3 can be captured by a multi-modal query that comprises several uni-modal queries:* $Q_{lyrics} = \{winter\}$, $Q_{album} = \{Game\ of\ Thrones\}$, $Q_{singer} = \{Florence\}$, *and* $Q_{composer} = \{Dan\ Weiss\}$. *A user could even provide an audio sample, so that query* $Q_{audio} = \{wav\text{-}file\}$ *would point to an audio file, if available. Figure 3.1 illustrates an example of a multi-modal query.*

3.2.3 Problem Statement

Given the above model, we consider the problem of identifying tuples that are most relevant for a multi-modal query. Formally, we capture the underlying notion of relevance by a function $f^* : \mathcal{D} \times \Omega \to \mathbb{R}$ that assigns a relevance score to each tuple $r \in D$ and the set of queried values q. Based thereon, the top-k tuples may be identified in terms of their relevance to the given query (i.e., the set of queried values).

Let $R = (r_1, \ldots, r_k)$ be a relevance-ordered sequence of k-tuples, i.e., it holds that $r_i \in D$ and $i < j$ implies that $f^*(r_i, q) \leq f^*(r_j, q)$. The sequence is referred to as a top-k result, denoted by $R^* = (r_1^*, \ldots, r_k^*)$, if for any other relevance-ordered sequence of k-tuples $R' = (r_1', \ldots, r_k')$ it holds that $\sum_{i=1}^{k} f^*(r_i', q) \leq \sum_{i=1}^{k} f^*(r_i^*, q)$. We phrase the problem of identifying a top-k result as follows:

Problem 1 (Multi-Modal Information Retrieval)**.** *Given a multi-modal query* $\mathcal{Q}$ *and a set of tuples* D*, the problem of* multi-modal information retrieval *is to retrieve a top-k result, i.e., a relevance-ordered sequence of k-tuples* $R^* = (r_1^*, \ldots, r_k^*)$.

The relevance function f^* is commonly unknown. Hence, any IR system will employ its own relevance function f that aims to approximate f^*. The quality of the employed relevance assessment may then be quantified based on some ground truth information, i.e., a top-k result for a specific multi-modal query. To this end, let $R = (r_1, \ldots, r_k)$ be a relevance-ordered sequence of k-tuples returned for a multi-modal query by an IR system that employs a relevance function f. Then, we assess the quality of the result retrieved

through this relevance function by the normalised discounted cumulative gain [JK02]:

$$nDCG(f) = \frac{\sum_{i=1}^{k} \frac{f(r_i,q)}{\log_2(i+1)}}{\sum_{i=1}^{k} \frac{f(r_i^*,q)}{\log_2(i+1)}}. \tag{3.1}$$

Note that $nDCG(.) \in [0,1]$, while a value of 1.0 indicates perfect result quality. Note that while we chose nDCG to measure the retrieval quality, our system is not limited to this metric. Other common retrieval metrics such as MRR, Precision@k, MAP can be used in our system as well. In the experiments, we evaluate our framework on both nDCG and Precision@k.

3.3 Approach Overview

In this work, we propose an approach for multi-modal information retrieval that is generally applicable in diverse domains. In Section 3.3.1, we first explain the design principles for our approach. Then, in Section 3.3.2, we discuss several core concepts and representations, before Section 3.3.3 summarises our overall approach to multi-modal information retrieval.

3.3.1 Design Principles

A generic approach to multi-modal information retrieval shall respect the following principles:

(DP1) *Domain-independence:* An IR approach should unify the relevance computation of tuples regardless of the modalities that are considered in a particular domain. Traditional systems for uni-modal IR learn an embedding for one modality and define relevance based on a distance between the resulting embeddings [JSN+17]. However, the respective embedding model is not applicable for another modality and the choice of a distance function also depends on the application in question.

(DP2) *Fusion-independence:* State-of-the-art approaches for multi-modal IR treat each modality separately, which requires several passes over the data. Moreover, the final step of fusing the results of several uni-modal queries is challenging since the rankings obtained for different modalities are incomparable [SY18]. Multi-modal IR should, therefore, avoid any dependency implied by the need to fuse the results of different modalities.

(DP3) *Embedding-independence:* Vectorization of diverse modalities leads to different embedding spaces, so that additional reconciliation of vector-spaces is needed [WLLZ18]. This, however, introduces a potential source of errors, so that multi-modal IR should not depend on such reconciliation. Moreover, the chosen embedding technique shall be invariant to certain transformations of heterogeneous data, such as rotation and illumination for images or watermarking and time-scale modification for audio and video data [Abb09, Abb07, LWSP14].

3.3.2 Core Concepts and Representations

Our approach to multi-modal IR is based on the notion of a heterogeneous information network, which we summarize below before turning to the definition of its schema and its vectorized representation.

Heterogeneous Information Networks (HIN). A HIN is a undirected graph $G = (V, E)$ with typed nodes V and edges $E \subseteq [V]^2$ between them [SHY+11], where $[V]^2$ are all subsets of V of size two. A type function $\phi : V \to \mathcal{A}$ maps each node to a modality $\phi(v)$. A node represents a value of its assigned type. Specifically, the value may be a vector of features of its modality (aka intrinsic characteristics of the node), such as the audio representation of a song.

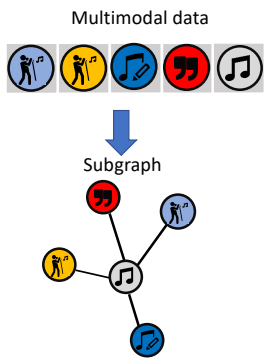


Figure 3.2: A HIN.

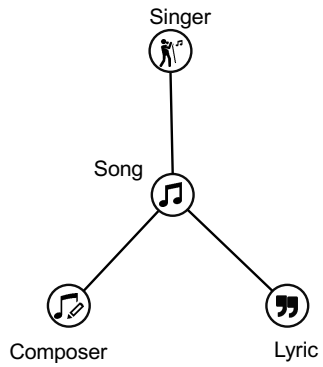


Figure 3.3: A HIN schema.

Example 6. *The construction of a HIN from a multi-modal tuple is illustrated in Figure 3.2. Here, node types include a title, a singer, lyrics, and an audio representation of a song. The edges capture the relations between the nodes. For instance, an edge represents the fact that the respective singer has recorded the song.*

HIN schema. To capture the characteristics of the entities represented by nodes types in a HIN, a *HIN schema* specifies multi-modal constraints by restricting the types of nodes that can be connected by edges. A HIN schema is a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V} \subseteq \mathcal{A}$ are modalities and $\mathcal{E} \subseteq [\mathcal{V}]^2$ are edges between them. A HIN $G = (V, E)$ is an instance of the HIN schema $\mathcal{G}$, if for all nodes node $v \in V$ it holds that $\phi(v) \in \mathcal{V}$ and for all edges $\{v_1, v_2\} \in E$ it holds that $\{\phi(v_1), \phi(v_2)\} \in \mathcal{E}$. The definition of a HIN schema is akin to the indexing of database attributes in IR frameworks [SLZ+16].

Example 7. *For our running example, a HIN schema as shown in Figure 3.3. It comprises four modalities,* singer, song, composer, *and* lyric. *The previous HIN in Figure 3.2 is an instance of this schema.*

Table 3.1 gives an overview of the most important notations used throughout the chapter.

Table 3.1: Overview of important notations.

Notation	Explanation
$\mathcal{D} = \{d_1, \ldots, d_n\}$	A multi-modal database of tuples
$\mathcal{A} = \{A_1, \ldots, A_m\}$	A set of modalities
$\mathcal{Q} = \{Q_1, \ldots, Q_m\}$	A multi-modal query: A set of uni-modal queries
$\Omega = \cup_{A \in \mathcal{A}} A$	All possible values of modalities
$q = \cup_{Q \in \mathcal{Q}} Q$	The set of queried values
$G = (V, E)$	A heterogeneous information network (HIN)
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	A HIN schema

3.3.3 Multi-Modal IR based on Graph Embedding

Using the above concepts and representations, the problem of multi-modal information retrieval as phrased in Problem 1 is lifted to HINs. That is, retrieval aims at the extraction of a top-k result, but not in terms of tuples, but in terms of nodes of a target modality:

Problem 2 (HIN Information Retrieval)**.** *Given a multi-modal query $\mathcal{Q}$ and a HIN G, the problem of* HIN information retrieval *with target modality A is to retrieve a relevance-ordered sequence of k nodes $R_v^* = (v_1^*, \ldots, v_k^*)$ with $\phi(v_i) = A$.*

Once the respective nodes have been retrieved by solving Problem 2, the result for Problem 1 can be derived: For each retrieved node, a tuple is constructed by concatenating its value with those of neighbouring nodes in the HIN.

Following the idea to solve the problem of multi-modal IR through HIN IR, Figure 3.4 gives an overview of our approach. In the first step, tuples are transformed into a single HIN to represent their relations and modalities. Based thereon, a graph embedding is constructed for each node of the HIN. This embedding captures both the intrinsic characteristics of the nodes as well as their relations in the HIN. This construction, detailed in Section 3.4, is done once and the results are indexed to speed up retrieval.

Given the queried values of a multi-modal query, we construct a query embedding in the same space. Such integration of different modalities at the query level enables query processing with a single pass over the data. The embedding of the queried values is explained in Section 3.5.

The retrieval of tuples that are relevant to the query is then grounded in the respective embeddings. Given the embedding of the query, z_q, and the set of embeddings of nodes of the HIN, Z, we identify the k nodes of the HIN that have embeddings $z_v \in Z$ closest to z_q, according to some similarity function $s(z_q, z_v)$ (e.g., Euclidean similarity).

Turning to the design principles of Section 3.3.1, the universal nature of the HIN model enables us to achieve the required independence properties: Relevance computation becomes independent of the modalities and similarity measures employed in a specific domain (DP1). Moreover, our approach does not depend on fusion of retrieval

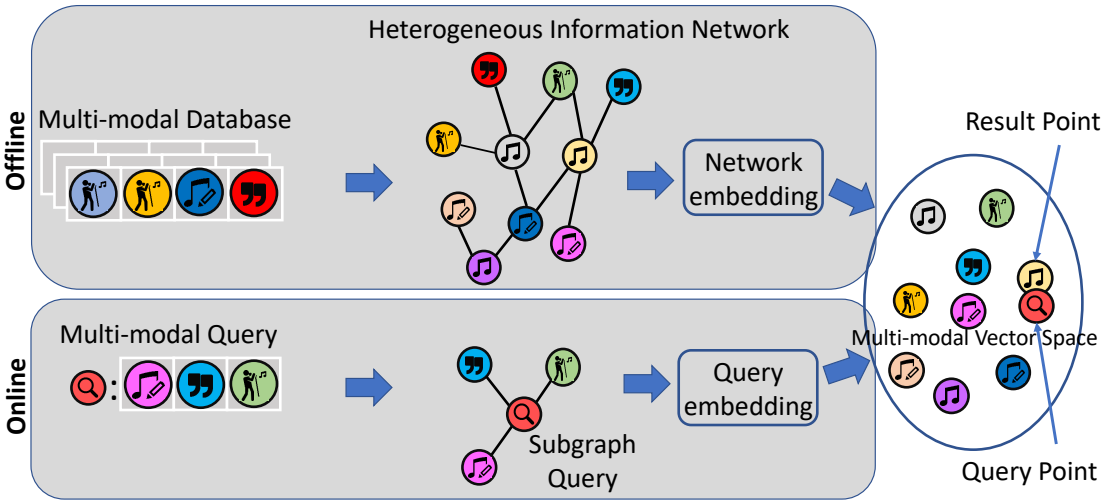


Figure 3.4: The proposed approach to multi-modal information retrieval.

results obtained for separate modalities (DP2). Finally, by constructing query embeddings in the embedding space of the data, we avoid the need for potentially erroneous reconciliation (DP3).

3.4 Heterogeneous Graph Embedding

This section introduces our process of embedding a multi-modal database. We first show how to construct a HIN from the database (Section 3.4.1). Then, we propose a novel model for graph heterogeneous embedding based on message-passing (Section 3.4.2).

The notion of a HIN was first proposed in [SHY+11], for the setting of similarity search. Since then, HINs have been used in various applications [SH13, SLZ+16], such as clustering [SYH09, LPW14], link prediction [CKP14], or recommendation [SZK+12]. Our approach is the first to apply a HIN in an information retrieval setting [SLZ+16]. Unlike the aforementioned techniques that are based on an assessment of meta-paths, we follow the paradigm of representation learning. That is, we represent the nodes of a HIN as embeddings in order to facilitate retrieval. The universal nature of our HIN model enables us to apply it for diverse retrieval settings that involve multiple modalities.

3.4.1 HIN Construction

Given a HIN schema, the construction of HIN for a multi-modal database D is done in a bottom-up manner. First, for each tuple $d \in D$, we construct a subgraph. The nodes are the respective attribute values of the tuple, i.e., $V_d = \{a_i \in d\}$, and while edges are defined between all pairs of values for which the HIN schema defines an edge between the respective attributes. Figure 3.2 illustrates this construction. The subgraphs created for all tuples are then connected through their shared nodes (of the same type) to construct the final HIN. Figure 3.5 gives an example where three subgraphs are connected through nodes that represent the same singer and the same composer.

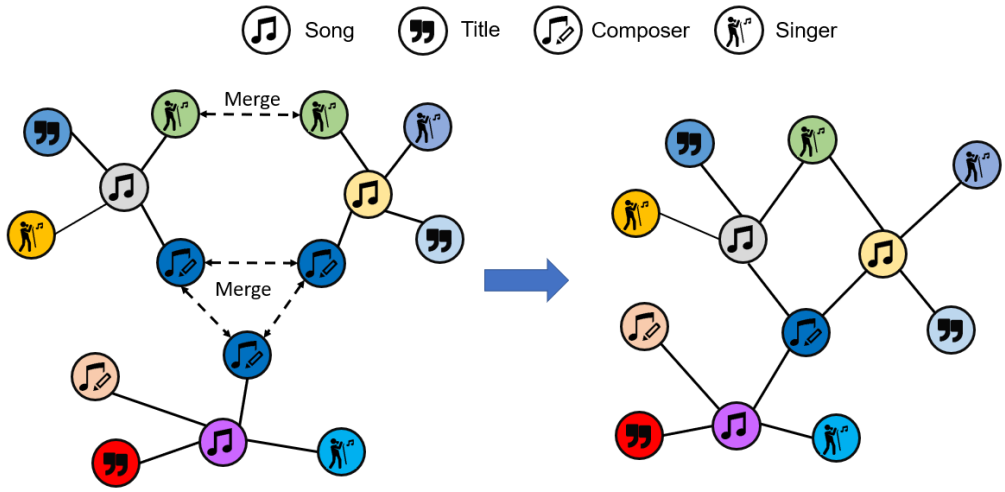


Figure 3.5: A HIN for music information retrieval (right) is created by merging shared nodes of different subgraphs of multimodal data tuples (left).

Example 8. *Consider two tuples representing songs: 'Paint it, Black' and 'Sympathy for the Devil'. To construct a HIN, we derive a subgraph for the first tuple with the respective song node; a singer node 'The Rolling Stones'; composer nodes 'Mick Jagger' and 'Keith Richards'; and a lyrics node. The nodes are connected according to the HIN schema in Figure 3.3. Given the subgraph for the second tuple, the two subgraphs are connected since they share the composer nodes for 'Mick Jagger' and 'Keith Richards'.*

3.4.2 HIN Embedding with Message-Passing

Requirements. A model for graph embedding that shall be applied for HINs should capture the following aspects:

1. *Connectivity*: Embeddings for nodes shall capture their relations. Following the homophily principle, the distance of nodes in the graph should be reflected by the distance of their embeddings in the embedding space.

2. *Node types*: The different types of nodes shall be incorporated in the embeddings.

3. *Node values*: The values of nodes, potentially involving multiple features that capture the nodes' characteristics, shall be represented in the embeddings.

Based on these requirements, we propose heterogenous GNN, or h-GNN for short. h-GNN is an instantiation of the message-passing neural network discussed in Chapter 2 that can take the node types into account.

h-GNN. The model adapts the message-passing procedure and incorporates node types (i.e., the modalities of the HIN) in the *sending* and *receiving* phases.

Sending: The sent message now depends on the type $s = \phi(v)$ of the sending node v and the type $t = \phi(u)$ of the receiving node u:

$$m_{v \to u}^{(l)} = M_{s,t}^{(l)} z_v^{(l)} \tag{3.2}$$

where $M_{s,t}^{(l)}$ is a separate matrix for each pair of types (s,t).

Receiving: Instead of using the maximum as an aggregation function, we sum up the messages from neighbours of a node v to avoid the loss of modality information.

$$agg_l(\{m_{u\to v}^{(l)}, \forall u \in N(v)\}) = \sum_{u \in N(v)} \sigma(W_{agg}^{(l)} m_{u\to v}^{(l)} + b^{(l)}) \tag{3.3}$$

The idea being that neighbouring nodes potentially belong to different modalities, so that the sum retains information about all of them. An aggregation based on the maximum, as in a traditional GNN, in turn, would only keep the information of one modality. Moreover, it is known that an aggregation based on the maximum, in some cases, cannot distinguish between two different neighbourhoods [XHLJ18].

Algorithm. We describe the procedure to embed a HIN with a h-GNN in Algorithm 3.1. The algorithm corresponds to the forward pass in a deep learning setting, in which the parameters of the h-GNN $(M_{s,t}^{(l)}, W_{agg}^{(l)}, b^{(l)}, W_{concat}^{(l)})$ have already been derived. Initially, these parameters are assigned randomly, while, later, they are learned gradually using Stochastic Gradient Descent (SGD), taking into account a loss function.

Algorithm 3.1: HIN embedding with h-GNN.

input : HIN $G = (V, E)$; input features $\{x_v, \forall\, v \in V\}$;
number of iteration L; weights $M_{s,t}^{(l)}, W_{agg}^{(l)}, b^{(l)}, W_{concat}^{(l)}$
output: Embedding z_v for all nodes $v \in V$

1 $z_v^0 \leftarrow x_v, \forall\, v \in V$;
2 **for** $l = 1 \ldots L$ **do**
3 **for** $v \in V$ **do**
// Sending
4 **for** $u \in N(v)$ **do** $m_{v\to u}^{(l)} = M_{\phi(v),\phi(u)}^{(l)} z_v^{(l)}$;
// Receiving
5 $I_v^{(l)} = \{m_{u\to v}^{(l)}, \forall u \in N(v)\}$;
6 $z_{N(v)}^{(l)} = \sum_{m_{u\to v}^{(l)} \in I_v^{(l)}} \sigma(W_{agg}^{(l)} m_{u\to v}^{(l)} + b^{(l)})$;
// Updating
7 $z_v^{(l+1)} = \sigma(W_{concat}^{(l)} concat(z_{N(v)}^{(l)}, z_v^{(l)}))$;
8 **return** $\{z_v^L, \forall\, v \in V\}$

Algorithm 3.1 starts by assigning an embedding to each node based on the node's features. Then, in each iteration l, the following steps are performed for each node: sending, receiving, and updating. For each node v in the HIN, a message is sent to its neighbour u (line 4). The message is constructed based on the node types $\phi(v)$ and $\phi(u)$. In the receiving step, given the messages that a node v received from its neighbours $I_v^{(l)}$, the function agg_l is applied to obtain the community embedding of node v, i.e., $z_{N(v)}^{(l)}$ (lines 5-6). In the updating step, the new embedding of v at iteration l+1 is obtained by applying the function $combine_l$ to the community embedding $z_{N(v)}^{(l)}$ and its embedding of

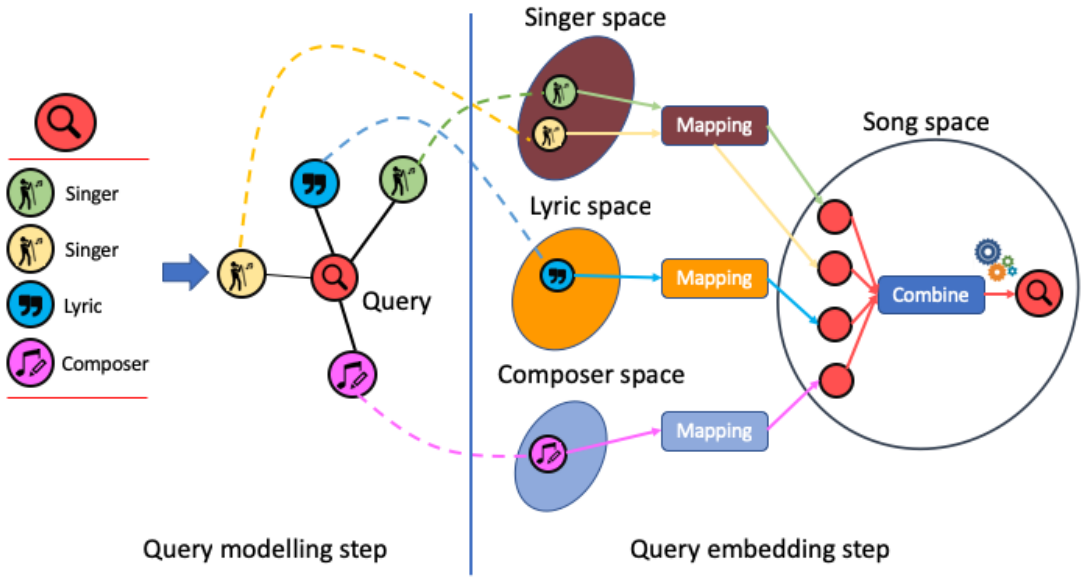


Figure 3.6: From a multi-modal query to a graph model to a unified embedding.

the previous iteration $z_v^{(l)}$. Finally, after L iterations, we return the obtained embeddings (line 8).

Parameter Learning. We learn the model parameters, i.e., the message matrix M and parameters of functions *agg* and *combine*, using SGD over the following loss function:

$$L(z_v) = -\log(\sigma(z_v^T z_u)) - Q.E_{u_n \sim P_n(v)} \log(\sigma(-z_v^T z_{u_n})) \tag{3.4}$$

This loss function minimizes (maximizes) the distance between embeddings of nodes that are close (distant) in the original HIN. Here, the exact definition of which nodes are deemed close and distant depends on a chosen notion of proximity (e.g., 1-hop or 2-hop). That is, $P_n(v)$ is a sampling function to select distant nodes (since their number is commonly much larger than the number of close nodes) and Q is the number of samples.

3.5 Embedding Multi-Modal Queries

In this section, we show how to embed a multi-modal query, in the same space as the embedding of the HIN. Below, we first discuss how to transform a multi-modal query into a subgraph query (Section 3.5.1), before turning to its embedding (Section 3.5.2) and the question of how to learn the parameters for the embedding model (Section 3.5.3).

3.5.1 From Multi-modal Queries to Subgraph Queries

Recall that a multi-modal query is represented by a set of queried values $q \subseteq \Omega$ (see Section 3.2.2). Since a database is modelled using a HIN $G = \{V, E\}$, a query q corresponds to a subgraph of G with a set of nodes $q \subset V$. Also, according to Problem 2, each query q is associated with a modality (type) of interest $t \in \mathcal{A}$, denoted by a query subscript, q_t. Hence, the result for a query q_t is a set of nodes of type t.

Example 9. *Figure 3.6 illustrates a query of a user looking for a song by specifying two singers, keywords in the lyrics, and a composer. In a first step, this query is converted into a subgraph.*

3.5.2 Multi-modal Query Embedding

To facilitate retrieval, an embedding of a query q needs to be in the same space as the embedding of the database, i.e., the HIN. In addition, the query embedding must be constructed based on *all* information available. Since q is a set of queried values and each value is captured by a node ($q \subset V$) for which an embedding is available already, the problem becomes the combination of embeddings of queried values to represent the whole query.

A straight-forward approach to query embedding is to take the average over the respective embeddings of nodes. That is, the embedding of q, denoted by z_q, is computed as $z_q = (1/|\{v \in q\}|) \sum_{v \in q} z_v$. By computing a linear combination of node embeddings, the resulting query embedding is in the same space as the HIN embedding. However, this approach neglects the multi-modal nature of the query, since the types of queried values are not distinguished. Below, we address this shortcoming.

Multi-modal space mapping. To incorporate the different types of queried values, we adopt the following view when constructing a query embedding: The nodes of each type are mapped into an embedding space that is specific to that type. For instance, we cannot expect the representation of a singer to be in the same space as the representation of a song title. However, the embeddings of a set of singers will be in the same space.

Adopting this view, we construct the embedding for a query q_t by mapping the embeddings of the queried values (and, thus nodes) of all types to the embedding space of t, the type of interest. Subsequently, the mapped embeddings are combined to obtain the unified embedding for the whole query. To this end, we define two embedding operators, which, following [HBZ+18], are referred to as mapping P and combination C of the embeddings of nodes in the query. While operator P maps an embedding from one space for a type to another one, operator C combines information from several embeddings in the space of a single type.

Mapping operator: The mapping operator P takes as input an embedding z_v of a node v of type s and a destination type t. It returns a new embedding z'_v, i.e., $z'_v = P(z_v, s, t)$. The new embedding z'_v can be considered as the embedding of v in the space of type t. Formally, P is defined as:

$$z' = M(z, s, t) = M_{s,t} z \tag{3.5}$$

where $M_{s,t} \in R^{d \times d}$ is a mapping matrix from type s to t. The matrix $M_{s,t}$ needs to be learned during a training phase.

Combination operator: This operator combines different embeddings in the same space. As there is no ordering of the queried values (i.e., the nodes) in query q_t, the operator shall be invariant to input permutation. For a set of embeddings $Z = z_1, \ldots, z_m$ of type t, it returns an embedding z':

$$z' = C(\{z_1, \ldots, z_m\}) = M_t \sigma(\phi(Z)) \tag{3.6}$$

where σ is a non-linear function, such as the Rectified Linear Unit (ReLU), ϕ is a permutation-invariant function that can be applied on set, and M_t is a matrix that is trained for type t. In its simplest form, the operator is defined with ϕ being the summation and M_t as the identity matrix:

$$z' = \sigma(\sum_i z_i).$$

Algorithm. Given the operators P and C, Algorithm 3.2 captures the process of constructing an embedding for a query q_t.

Algorithm 3.2: Construction of a query embedding.

input : node embeddings $\{z_v\}$; query $q_t = \{v\}$;
mapping operator M; combination operator C;
output: embedding for $q_t : z_q$

1 $Q = \emptyset$;
2 **for** $v \in q_t$ **do** // 1. Mapping step
3 $\quad z'_i \leftarrow M(z_v, \phi(v), t)$;
4 $\quad Q \leftarrow Q \cup \{z'_i\}$;
5 $z_q \leftarrow C(Q)$; // 2. Combination step
6 **return** z_q;

The algorithm starts with the embedding of each node v of query q_t. It converts the embedding of each node to the space of type t using the mapping operator (line 3), which yields a set of mapped embeddings Q (line 4). Finally, we apply the combination operator C to the set Q to obtain the query embedding z_q (line 5).

Example 10. *Taking up Example 9, after obtaining the graph of the query, the mapping operator converts the singer, lyric, and composer embeddings to the embedding space of a song, see Figure 3.6. Note that the respective embeddings of the singers, lyric, and composer are available from the construction of an embedding for the HIN. Then, the mapped embeddings are combined to obtain the query embedding.*

A major advantage of our framework is its flexibility, as the training of the query embedding process is independent of the HIN embedding. Hence, we learn the parameters of operators P and C using the available HIN embedding.

3.5.3 Parameter Learning

To learn the parameters of the operators for the query embedding process, we need to define (i) a loss function that captures our objective, and (ii) training samples used to optimize the parameters. From Problem 1, we derive that training data shall contain pairs of a query and a top-k result, (q, R^*). This would allow to optimize the parameters to obtain a retrieval result R for query q, such that R approximates R^*. However, such training data is commonly not available and their manual construction is infeasible, as a large amount of queries would be required.

Generate sample queries. Against this background, we follow a training approach that employs top-1 queries. Such an approach has the advantage that training is efficient, since the result for each query is a single node. The generated sample queries are then used to define our loss function.

Given a type t, we define a training set of top-1 queries that have as a result a single node of type t. We consider queries of varying levels of precision based on the number of neighbouring nodes of the result node that are part of the query. The most precise queries are defined as follows:

$$\mathbb{Q}_t^{(1)} = \{(q, R)\} = \{(N(v), (v)), \forall\ v \in V : \phi(v) = t\}.$$

That is, if the query contains all neighbours of a node v, v is the best retrieval result. We generalize this idea to the set of queries with ρ-precision. It includes all queries that

remove at most ρ-1 neighbours of a node v, when querying for v. The set of respective queries is defined recursively:

$$\mathbb{Q}_t^{\rho} = \mathbb{Q}_t^{(\rho-1)} \cup \{(N', (v)), \forall\ v \in V, N' \subset N(v) : \phi(v) = t \wedge |N(v) \setminus N'| = \rho - 1\}.$$

Note that, with larger ρ, queries become less precise and may not have a unique answer any more. As such, relatively small values should be used for parameter ρ.

Loss function. In our setting of vector-based retrieval, a query embedding shall be close to the embeddings of the queries constructed above. That is, given the training queries, we expect $dist(z_q, z_v)$ to be small where q and v is a pair of a training query and a node of the result of a top-1 query with ρ-precision, whereas the distance shall be large for all other nodes. We therefore define a loss function based on triplet loss [SKP15] as follows:

$$L(z_q, z^+, z^-) = \frac{1}{2} \max(0, m + dist(z_q, z^+) - dist(z_q, z^-)) \tag{3.7}$$

where z_q, z^+, z^- are the embeddings of the query, a node of the query result (i.e., a *positive sample*), and node that is not part of the query result (i.e., a *negative sample*), respectively, and m is a margin. This function is minimal, when $dist(z_q, z^-) - dist(z_q, z^+) > m$, i.e., when the distance between embeddings of the positive sample and the query is small, and the distance between the query embedding and the negative sample is large.

3.6 Experimental Results

This section reports on an experimental evaluation of the proposed framework using a diverse set of real-world and synthetic datasets. Our experimental setup is discussed in (Section 3.6.1), before the following aspects are evaluated:

- The general efficiency of our approach (Section 3.6.2).
- The effectiveness of HIN embedding (Section 3.6.3).
- The effectiveness of query embedding (Section 3.6.4).
- The efficiency and effectiveness of information retrieval in comparison to baseline techniques (Section 3.6.5).

3.6.1 Setup

Datasets. We use 4 real-world datasets as in Table 3.2.

Table 3.2: Statistics for real-world datasets.

Dataset	flickr	dbis	fma	citation
#Nodes	19'471	150'858	244'360	1'511'035
#Edges	105'425	708'502	426'296	2'084'019

Real-world datasets: The *flickr* dataset includes several modalities, including images, image groups, terms, and users [SHY+11]. Relations link images and users, images and terms, and images and groups. The *dbis* dataset [DCS17] includes modalities such as papers, authors, venues, and terms in the research domain of databases and information

systems. Relations link papers and authors, papers and venues, and papers and terms. The *fma dataset* [DBVB17] contains songs and related information such as short audio segments, albums, artists, and genres. We enrich the fma dataset with album arts, further increasing its heterogeneity. Relations link songs with the other types, except album arts, which are linked to albums.

For each of these datasets, we generate 100 multi-modal queries of varying sizes, ranging from 1 to 10. The values for a query are selected using truncated random walks around a node in the respective HIN. This way, we ensure that the queries are semantically meaningful. For each set of queries of the same size, we ensure that the query contains different modalities (1-3 for flickr/dbis and 2-4 for fma). The retrieved result is judged by three hired experts, who score the relevance of top-20 results on a scale of 0 to 5.

Synthetic datasets: In addition to the real-world datasets, we also rely on synthetic data for sensitivity analysis. In particular, we study the effects of dataset size and the number of modalities, which includes several control parameters: m, the number of tuples; α, the HIN density (#tuples/#nodes); n, the number of nodes; H, the structure of the HIN schema; l, the heterogeneity degree (#attributes); and δ, the distribution controlling how many attributes are HIN nodes. We first generate n nodes and then divide the nodes into l clusters according to δ. Then, we construct m tuples by selecting from each attribute a set of nodes according to a HIN schema. Given the tuples and the schema H, we can construct the HIN following the process described in Section 3.4. If not stated otherwise, we use the following parameter configurations: $n = 1M$, $m = 500K$, $l = 5$, δ is a uniform distribution for l-1 attributes while the last attribute contains m nodes, and H is a star schema.

Evaluation Metrics. Four main metrics are used:

Runtime: We evaluate efficiency in terms of the retrieval time needed to answer a multi-modal query, and the training time required to construct embeddings.

Normalized Discounted Cumulative Gain (nDCG): The effectiveness is evaluated using nDCG (see also Section 3.2.3), a state-of-the-art IR metric that incorporates the relevance and ranking position of results.

Precision@k: Another common metric for evaluating retrieval result is Precision@k which is the precision of the result considering only the top-k answers.

Hyperparameters. Unless stated otherwise, we use the parameters suggested in [HYL17a] for graph embedding. The learning rate is set to 0.0001 for all experiments. The batch size for experiments with real-world data is 256, and 16 for the synthetic data. As for the loss function (Equation 3.4), for each node, we use 50 random walks with a length of 5. All models use a Rectified Linear Unit as a non-linear function.

Regarding the GNN, we found that by training the node embeddings at iteration 0 together with the model, we achieve better results. Following [HYL17a], for the aggregation and combine functions, we use *max* and *concat*, respectively. Regarding the query embedding, the margin for the loss function in Equation 5.1 is set to 0.1, as suggested in [SKP15].

Environment. All experiments have been obtained on an Intel Core i7-6700K server with a NVIDIA GTX 1080Ti and 32GB RAM. For k-NN search, we leverage nmslib for indexing, which facilitates fast retrieval.

3.6.2 General Efficiency

We analyse the general efficiency of our approach in terms of its runtime, using the synthetic dataset.

Model for query embedding. We assess the training time needed to learn the parameters for query embedding (see Section 3.5.3), relative to the HIN's density level and its number of nodes. Figure 3.7 depicts the runtime (in minutes) for different configurations.

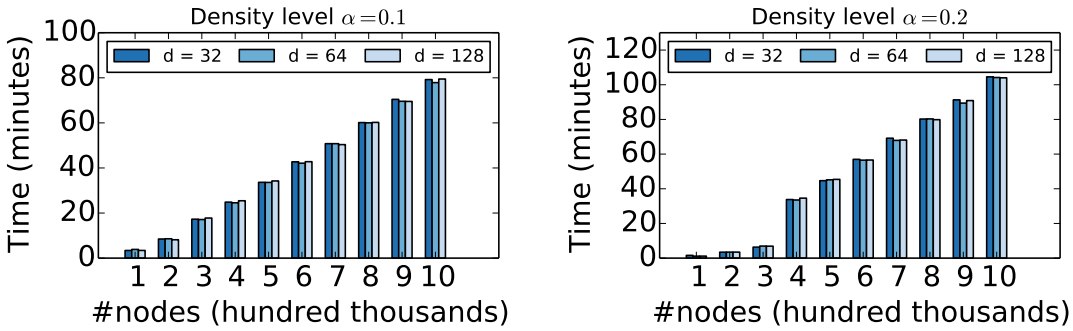


Figure 3.7: Training time for the model for query embedding vs. #nodes.

In general, the training time increases linearly with the number of nodes in the HIN. This is expected as the number of training queries increases linearly w.r.t the number of nodes. For example, the training time is 3min when the network size is 100k, but increases to 79min when the network size is 1M. On the other hand, the embedding size has little effect on the training time.

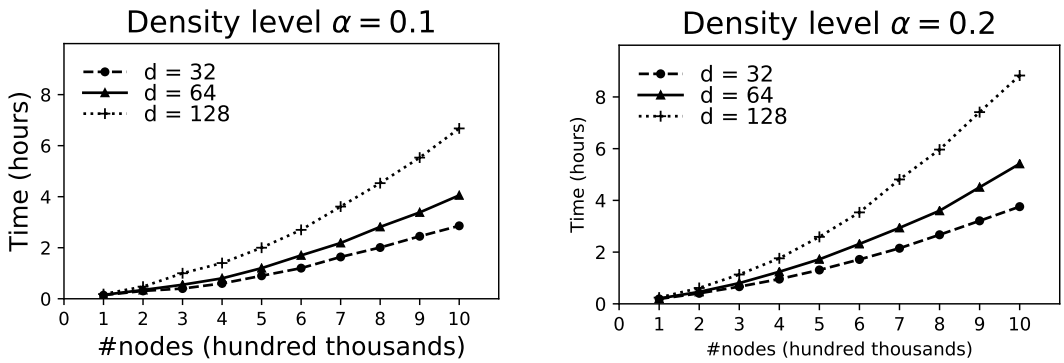


Figure 3.8: Training time of the whole framework vs. #node.

End-to-end embedding co-training. We further analyse the training time of the end-to-end process, incorporating both models for query and graph embedding, see Figure 3.8. The measured training time is higher than solely for the model for query embedding (Figure 3.7), due to a higher overall number of parameters. Yet, it is significantly lower than the sum of training times if the models for query and graph embedding are learned separately. For instance, for a graph with 500k nodes, the end-to-end process for co-training takes ∼2h, whereas isolated training would require 3h for graph embedding and 0.5h for query embedding.

Retrieval time. Finally, we evaluate the time required to answer a query using our proposed approach, relative to the query size and the number of nodes in the HIN. Figure 3.9 illustrates that, unlike training times, retrieval times are extremely short. For

instance, for a query of size nine and a network with 1M nodes, retrieval is done in 23ms. Also, the query size has little effect on the retrieval time. Also, for common queries of rather small size, there is virtually no difference in retrieval time. For example, increasing the query size from three to nine, for a network of 200k nodes, the retrieval time remains stable at 4ms. This comes from the fact that the query embedding computation involves mostly matrix multiplication which can be extremely fast given leveraging GPU. However, we expect the query size to have effects on retrieval time for a query with large amount of query terms, which is an unrealistic setting.

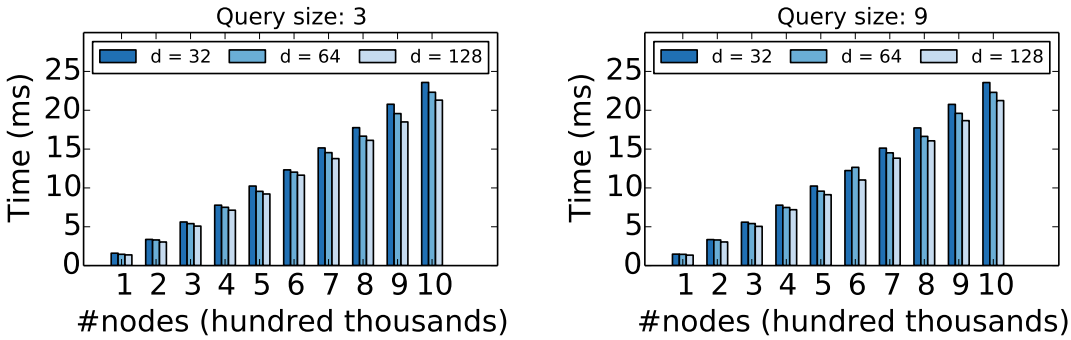


Figure 3.9: Effects of query size on retrieval time.

A key observation is that the embedding size does not affect the retrieval time. This is explained by the fact that we need to consider solely the relative difference between the embeddings. By indexing these differences beforehand, we remove any effect of the embedding size on retrieval time. Another observation is that the retrieval time scales linearly with the graph size. We observe increment (from 2ms to 22ms) as we increase the graph size from 100k to 1M nodes with an embedding size of 64. This is expected as increasing the graph size also increases the retrieval space. We can reduce retrieval time further by streaming IR, but is left as future work.

3.6.3 Effectiveness of HIN Embedding

Quantitative evaluation. This experiment compares our embedding model, h-GNN as presented in Section 3.4.2, against two baselines, i.e., traditional GNN [HYL17a] and node2vec, in terms of the normalized Discounted Cumulative Gain (nDCG). The embedding of a query is generated by averaging the embeddings of all nodes in the query. Moreover, we employ the loss function defined in Equation 3.4.

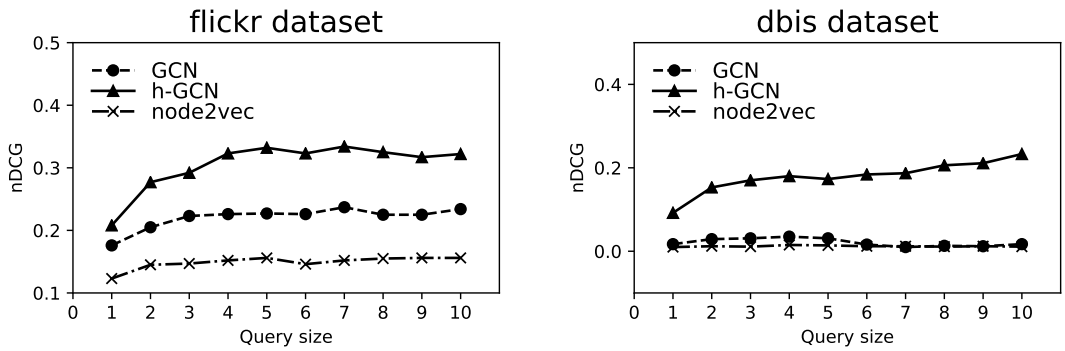


Figure 3.10 shows the nDCG (the higher, the better) related to the query size. Our model outperforms all baselines by a large margin. For example, in dbis dataset, h-GNN achieves an nDCG of 0.184, which is 11.5× better than the best baseline, when

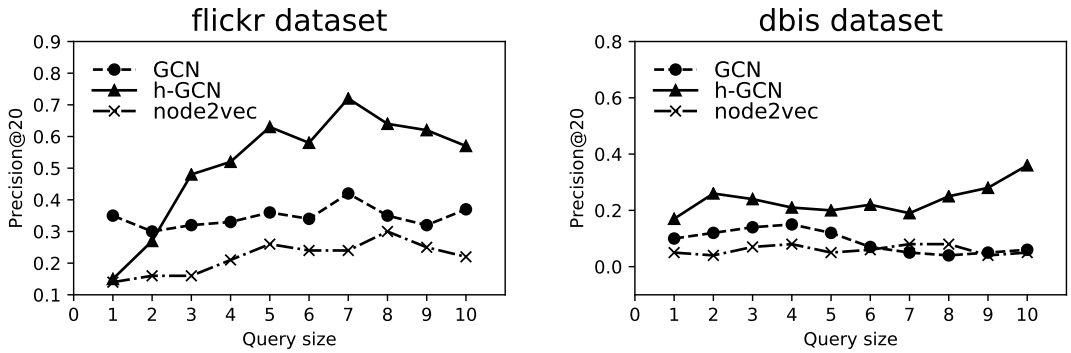


Figure 3.10: Effectiveness of HIN embedding.

the query size is 5. The worst results are obtained with node2vec, since it neglects intrinsic characteristics of nodes. We observe the same trend with Precision@20 where our method outperforms the baselines significantly.

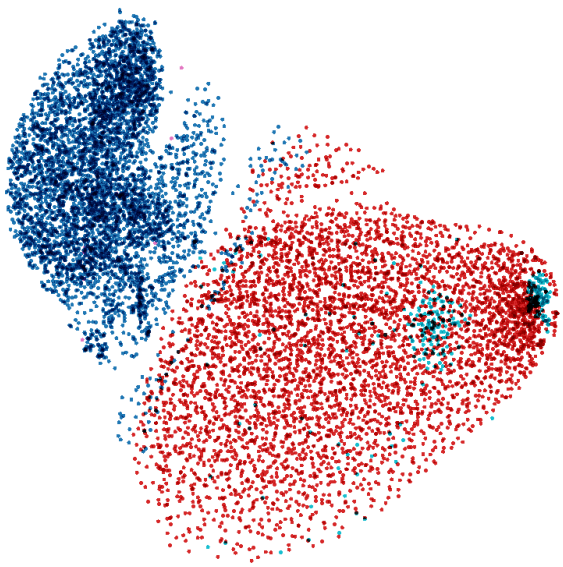


Figure 3.11: Visualization of the embeddings.

Qualitative evaluation. Figure 3.11 shows the t-SNE visualization of node embeddings for the flickr dataset. It illustrates three clusters: terms (red), images (blue) and users (light blue). This confirms our hypothesis that the terms, images, users belong to different embedding spaces.

3.6.4 Effectiveness of Query Embedding

To evaluate the effectiveness of our approach to query embedding, we first construct the graph embedding for the HIN using traditional GCN. Based on the node embeddings, we then learn the query embedding following Algorithm 3.2. Our model is compared with a baseline which computes the query embedding by averaging the query nodes. The comparison is performed with different query sizes.

Figure 3.12 illustrates that the approach proposed in this thesis outperforms the baseline technique. For the flickr dataset, the nDCG values are 0.3 and 0.22, respectively, when the query size is six. Our approach achieves this improvement by incorporating

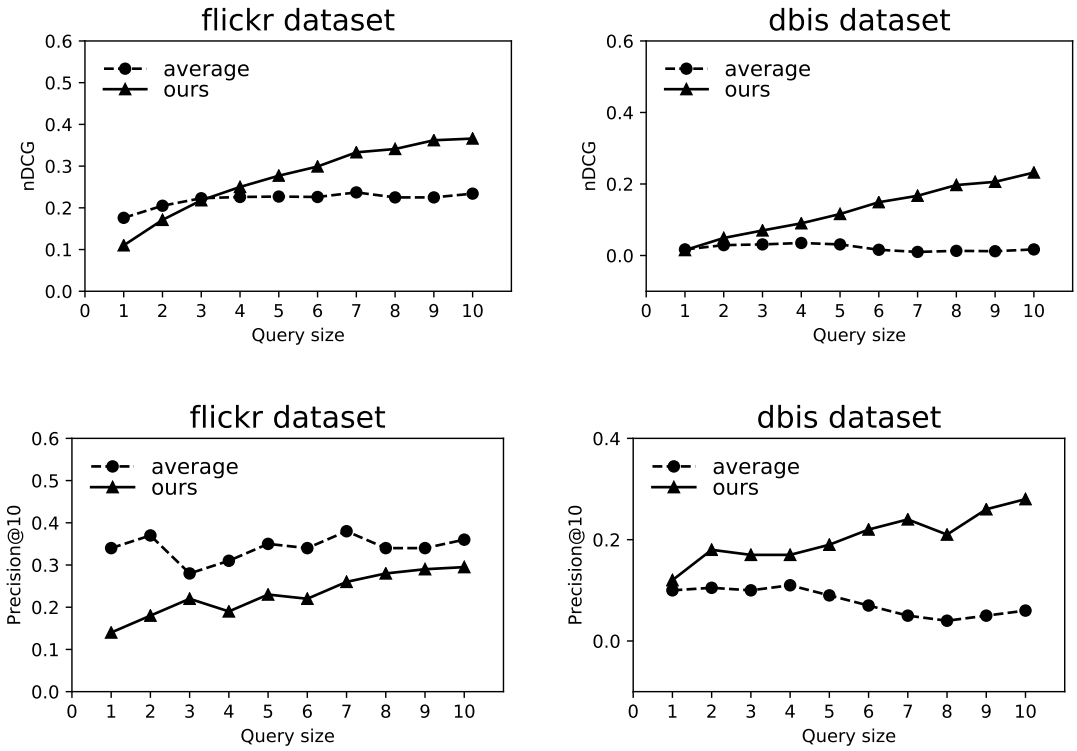


Figure 3.12: Comparison of different query embedding methods.

the heterogeneous information available in the query to retrieve more relevant results. Regarding the precision metric, we observe the same situation where our method is better than the baseline across different query size. This clearly shows the effectiveness of our query embedding approach.

3.6.5 End-to-end Comparison with SOTA

Comparison with single-pass baselines. We compare our approach with several traditional retrieval methods, such as tfidf [LRU14], CCA [DSV+18], MSAE [WOY+14]. To make tfidf work in a multi-modal setting, we represent other modalities by their respective textual descriptions. For CCA and MSAE, which are cross-modal retrieval methods, we compare the obtained results with our method by performing retrieval from text to image on the flickr dataset and text to audio on the fma dataset. For the dbis dataset, CCA is ignored as it does not support text-to-text retrieval. We further vary the query heterogeneity, i.e., the number of considered modalities.

Table 3.3: Comparison with baselines in terms of nDCG.

Dataset	**flickr**			**dbis**			**fma**		
Heterogeneity	1	2	3	1	2	3	2	3	4
tfidf	0.36	0.37	0.35	**0.47**	0.45	0.49	0.42	0.47	0.52
cca	0.42	0.45	0.44	N/A	N/A	N/A	0.52	0.54	0.57
msae	0.52	0.51	0.55	0.36	0.37	0.33	0.57	0.62	N/A
Ours	**0.55**	**0.63**	**0.87**	0.41	**0.77**	**0.93**	**0.83**	**0.84**	**0.87**

Table 3.3 shows that our method generally outperforms the baseline techniques. It achieves an nDCG value of 0.63, whereas CCA reaches 0.45, MSAE reaches 0.31, and tf-idf reaches 0.37 on the flickr dataset with a query heterogeneity of two. Our technique yields better results than CCA and MSAE, since these techniques handle image/audio/text as separate modalities. Both of them try to map different modalities to a pair of embeddings that are close. Yet, the embeddings for different modalities are learned separately, so that relations between them are not directly captured.

Also, as the query becomes more heterogeneous, the differences between our approach and the baseline techniques becomes larger. This is due to the baselines not leveraging the query heterogeneity in their models.

Moreover, tfidf yields the best results for the dbis dataset with a query heterogeneity of one, since tfidf is optimized for text-to-text retrieval. Yet, our method performs better than tfidf when the query becomes more heterogeneous.

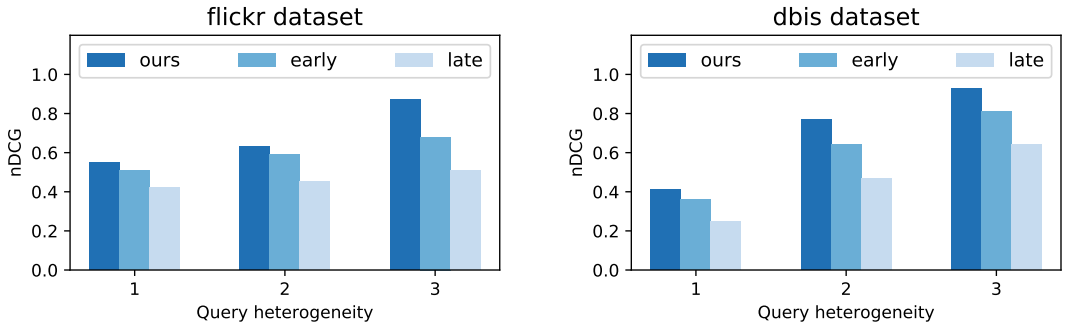


Figure 3.13: Our technique for one pass retrieval vs techniques for multi-pass retrieval - Retrieval measure.

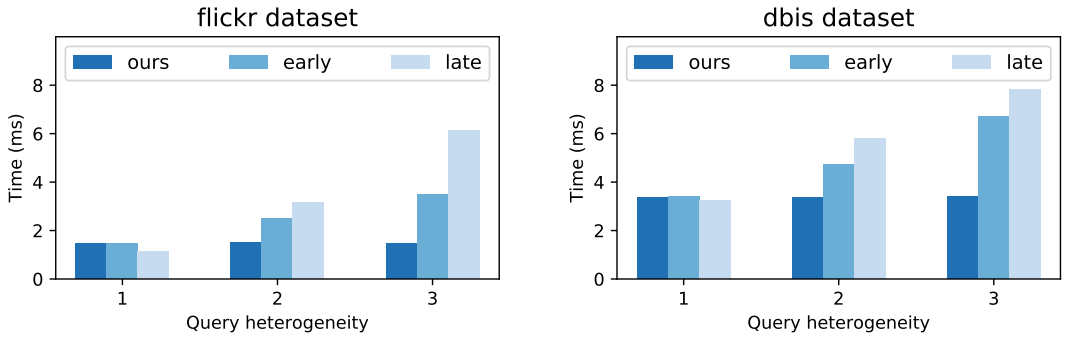


Figure 3.14: Our technique for one pass retrieval vs techniques for multi-pass retrieval - Retrieval time.

Comparison with multi-pass baselines. Next, we compare our approach that requires only a single pass over the data with baseline techniques that combine the results from several passes over the data, one per queried modality. Specifically, we use the tfidf baseline to retrieve results using each attribute separately and then combine their rankings. We also construct another baseline based on tfidf, which computes the representation for each attribute separately and generates the query embedding by averaging the attribute representations. The two baselines represent early and late fusion approaches in heterogeneous retrieval [XHS+18, MMM15], respectively.

The effectiveness and efficiency of the considered approaches is visualised in Figure 3.14. Here, our technique clearly dominates in terms of retrieval quality. While all techniques have a similar retrieval time for a query heterogeneity of one, our technique is able to retrieve answers faster than fusion-based approaches for heterogeneous queries. This trend stems from late fusion requiring several passes over the data, while early fusion requires several passes over the query to construct the embedding.

3.6.6 Ablation study

We analyze the effectiveness of different components of our framework in this experiment. As our framework involves 2 components: HIN embedding and query embedding, we vary the methods in each components to demonstrate the benefits of our proposed approaches. For HIN embedding, we analyze two baselines which are GNN and node2vec in comparison with our h-GNN approach. For query embedding, we compare our query embedding approach using message passing (MP) with a baseline using averaging of node embeddings. We evaluate the components using both nDCG and Prec@10 metrics.

Table 3.4: Ablation study

		flickr		dbis	
		nDCG	Prec@10	nDCG	Prec@10
HIN embedding	node2vec	0.156	0.26	0.01	0.05
	GNN	0.227	0.36	0.03	0.12
	h-GNN	*0.332*	*0.63*	*0.19*	*0.2*
Query embedding	Avg.	0.227	0.23	0.031	0.09
	MP	*0.277*	*0.35*	*0.116*	*0.19*
hGNN + MP (Ours)		**0.43**	**0.73**	**0.24**	**0.37**

The experimental results shown in Table 3.4 confirm the benefits of our proposed approaches. Our h-GNN HIN embedding achieves better nDCG and Prec@10 for both datasets in comparison with the baselines. Note that for analyzing the effects of HIN embeddings, we fix the query embedding to be averaging. Therefore, the difference in results can only be explained by better model. We obtain similar results for our proposed query embedding approach. Our approach of constructing the query embedding by passing messages on the query graph retrieve more relevant results in comparison with averaging. Finally, we observe that by combining hGNN for HIN embedding and MP for query embedding, we achieve better results than all combinations. This clearly shows the effectiveness of our components and their combination.

3.7 Summary

In this chapter, we presented a new direction for multi-modal IR that relies on an embedding of a heterogeneous information network. Such a HIN enables us to capture the rich semantics of both, multi-modal data and queries. Based thereon, we proposed a novel graph embedding model to obtain a vectorized representation of a HIN and a technique to construct a query embedding based on the HIN embedding through mapping and combination operators. As a result, we obtain embeddings that are defined over the

same space to achieve accurate relevance assessment. Our experimental results for real-world and synthetic datasets illustrate the efficiency and effectiveness of our approach. In future work, we will explore the dynamic evolution of a network to enable the retrieval of data that has not been considered in the training phase. This way, we can support applications based on streaming data.

Chapter 4

Connectivity - Graph Embedding for Streaming Subgraph Retrieval

Efficient Streaming Subgraph Isomorphism with Graph Neural Networks

VLDB 2021

Queries to detect isomorphic subgraphs are important in graph-based data management. While the problem of subgraph isomorphism search has received considerable attention for the static setting of a single query, or a batch thereof, existing approaches do not scale to a dynamic setting of a continuous stream of queries. In this chapter, we address the scalability challenges induced by a stream of subgraph isomorphism queries by caching and re-use of previous results. We first present a novel subgraph index based on graph embeddings that serves as the foundation for efficient stream processing. It enables not only effective caching and re-use of results, but also speeds-up traditional algorithms for subgraph isomorphism in case of cache misses. Moreover, we propose cache management policies that incorporate notions of reusability of query results. Experiments using real-world datasets demonstrate the effectiveness of our approach in handling isomorphic subgraph search for streams of queries.

4.1 Introduction

Graphs are a natural representation of relations between entities in complex systems, such as social networks, chemical compounds, or biological structures [TVVT+20, DYH+20, TTVV+20, HDQ+19, DHD+19]. Hence, efficient management of graph-structured data is of crucial importance in diverse domains and subgraph isomorphism queries are an important means to detect patterns in larger graphs [RW16, Ull76, ZLY09]. Specifically, given a query graph q (i.e., the pattern) and a data graph g, such a query returns all mappings of nodes of q to nodes of g that preserve the respective edges. Answering subgraph isomorphism queries is useful, for instance, to analyze propagation patterns in social networks or to query protein connections in protein interaction networks.

Since the problem of subgraph isomorphism search is NP-Hard, various heuristics to speed up the search have been proposed [CFSV04, ZLY09, HLL13, Ull76]. These algorithms have in common that they are based on measures of node similarity and

subgraph similarity. The former enables conclusions on nodes of the data graph that cannot be mapped to nodes of the query graph and are, therefore, filtered. Measuring subgraph similarity, in turn, is the first step of verifying whether a subgraph of the data graph is isomorphic to the query graph.

In domains such as social networks, chemistry, or biology, subgraph isomorphism queries occur frequently. They are issued concurrently by many users and systems. For instance, ChemSpider is a search engine with an API that answers subgraph isomorphism queries for molecular structures in a database of more than 77 million molecules [**?**]. Once a stream of queries is considered, the aforementioned algorithms to subgraph isomorphism search become infeasible. They employ notions of similarity for nodes and subgraphs that are based on the actual structure of the graphs. Since the respective structural comparison has a worst-case runtime complexity of $O(N! \cdot N^2)$ in the size of the graphs [CFSV04] for large query graphs, or $O(N^k)$ for small query graphs with k nodes [MlP14], traditional approaches do not scale to a streaming setting.

For other data models, techniques to process a continuous stream of queries are commonly addressed using caching strategies. Caching is possible in these cases as the queries show a large overlap, which enables re-use of previous results. Examples include techniques to evaluate queries in web search engines [AAB+19, OAU11, LGZ04] and to answer resource requests in web applications [GMA+08, APTP03, GY13]. In either case, cached query results are re-used when answering subsequent queries. However, this principle cannot be adopted directly for subgraph isomorphism queries, since it was shown empirically that most existing techniques for structural indexing have an exponential runtime [KNT15]. Hence, it is infeasible to index the data graph, or parts thereof, as it would be required for efficient caching and re-use of query results.

In this chapter, we propose to use embeddings as a foundation for the evaluation of subgraph isomorphism queries. An embedding maps a graph to a numerical space, such that structurally-similar nodes and subgraphs are close to each other [NCC+16, PARS14, HYL17a]. Embeddings support indexing naturally. Nodes and subgraphs are points in a high-dimensional space, so that indices for space partitioning, e.g., R-tree [Gut84] or kd-tree [DBCVKO08], may be leveraged. Based thereon, similarity computation or nearest neighbor search are realized efficiently.

Using embeddings as the basis for subgraph isomorphism further has the advantage that it enables cache management based on diversity considerations. A diversified cache is more effective since query graphs in a stream are more likely to be similar to those cached already. However, diversity-based caching for subgraph isomorphism is infeasible for traditional structural indexing due to the computational overhead induced by a structural comparison of query graphs and cached graphs. Yet, using embeddings, the graph comparisons becomes fast and accurate, so that our work is the first to propose such diversity-based cache management for subgraph isomorphism.

To realize the above vision, we define the problem of *streaming subgraph isomorphism* and propose a framework to solve it (Section 4.2). We then instantiate this framework, making the following contributions:

- *Graph indexing using embeddings* (Section 4.3). As the basis for our work, we propose an indexing mechanism based on node, edge, and subgraph embeddings. While we incorporate state-of-the-art techniques for graph representation learning, we provide a theoretical justification for our mechanism by showing that the embedding process is similar to Weisfeiler-Lehman isomorphism testing [MRF+19].
- *Query stream processing with a cache* (Section 4.4). Using the indices, we show how to answer a stream of subgraph isomorphism queries while exploiting cached

results. Specifically, upon the arrival of a query, similar past queries are identified to re-used their results. In the case of a cache miss, subgraph embeddings are exploited to speed up traditional algorithms for subgraph isomorphism (e.g., TurboISO [HLL13]). In case of a cache hit, we assess the overlap of the current query with the cached ones and derive an answer from the cached results.

- *Cache management* (Section 4.5). As the size of a cache is limited, we need to control cache admission and eviction. To this end, we propose a policy that minimizes the number of cache misses. Compared to traditional policies (LRU [You08] or GreedyDual [You08]), it assesses the utility of a query result not only based on processing time, but includes a notion of diversity.

We evaluate our approach using several real-world datasets in Section 4.6. We show that our embedding-based index outperforms structural indices by two orders of magnitude. When answering subgraph isomorphism queries, our approach based on caching and re-use of results leads to runtime improvements of at least 100% over state-of-the-art algorithms such as MQO [RW16] and TurboISO [HLL13]. We conclude in Section 4.7.

4.2 Model and Approach

4.2.1 Model

We target the problem of subgraph isomorphism search for undirected, labelled graphs. Let $g = (V, E)$ be a *graph* with a set of nodes V and a set of edges $E \subseteq V \times V$. It is associated with a labeling function $l : V \rightarrow \Sigma$ that captures intrinsic properties of its nodes. If the alphabet of labels Σ is defined as $\mathbb{R}^k$, i.e., labels are k-dimensional real vectors, we refer to (g, l) as an attributed graph.

Two attributed graphs (g_1, l_1) and (g_2, l_2) are *isomorphic*, if there exists an edge-preserving bijective function $f : V_1 \rightarrow V_2$ such that:
(1) $\forall\ v \in V_1 : l_1(v) = l_2(f(v))$, and
(2) $\forall\ (v_1, v_2) \in E_1 : (f(v_1), f(v_2)) \in E_2$.

If g_1 is isomorphic to an induced subgraph g_2' of g_2, g_1 is *subgraph isomorphic* to g_2, written as $g_1 \preceq g_2$. We call the bijection between g_1 and g_2' a *mapping*, and g_1 is said to have a mapping in g_2. There may be several mappings of g_1 in g_2. We write $F(g_1, g_2) = \{f_1, f_2, \ldots, f_k\}$ for the set of all mappings. The *subgraph isomorphism* problem is to find all mappings $F(g_1, g_2)$ for a given pair of graphs.

In graph-based data management, a subgraph isomorphism query is defined through a *query graph* $q = (V', E')$ for which the subgraph isomorphism problem shall be solved regarding a *data graph* $g = (V, E)$. We target scenarios in which queries arrive continuously. We therefore define a *query stream* as a sequence of queries, $Q = \langle q_1, q_2, \ldots \rangle$, arriving one after another. Each query arrives at a particular point in time, denoted by $q_i.t$, and the stream is totally ordered by these time points, i.e., for any two queries q_i and q_j of the stream, if $i < j$ then $q_i.t < q_j.t$. We denote the finite prefix of stream Q until index k as $Q_{[k]} = \langle q_1, \ldots, q_k \rangle$. In our setting, the queries in the stream may overlap or repeat, so that results stored for previous queries may be re-used.

Query processing incurs a latency, i.e., the time between the arrival of a query and the time it is answered. Based thereon, we capture the problem addressed in this chapter, as follows:

Problem 3 (Streaming Subgraph Isomorphism)**.**
Given a data graph, the problem of streaming subgraph isomorphism *is to solve the*

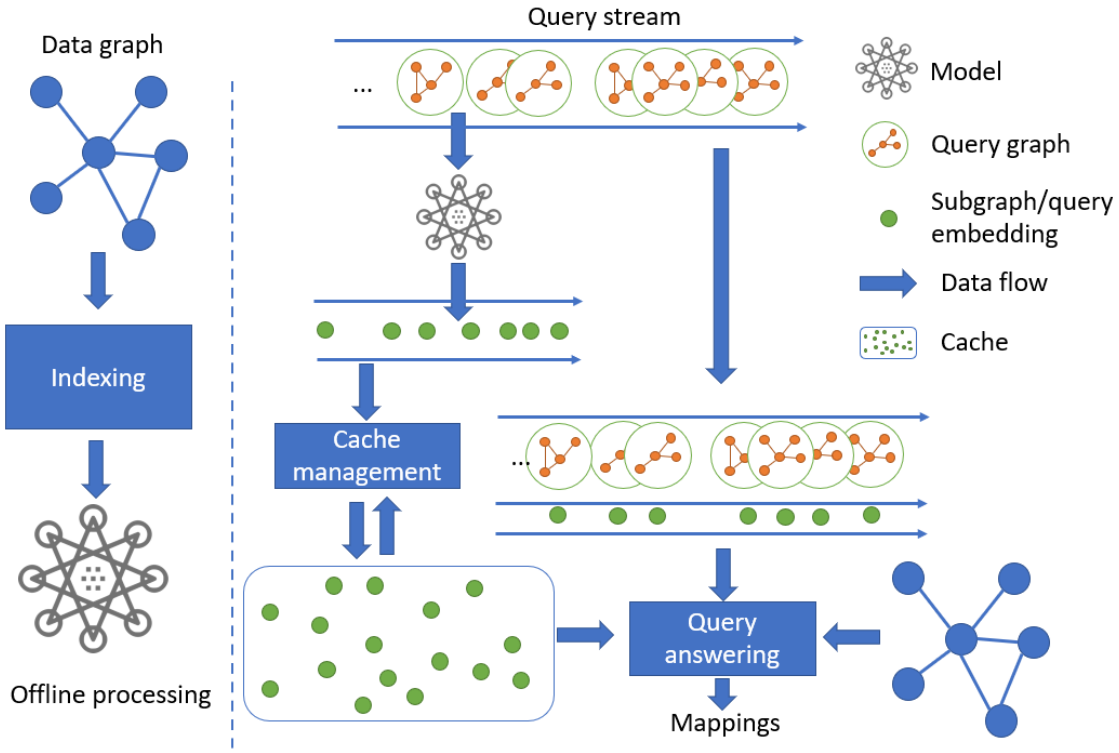


Figure 4.1: Framework for streaming subgraph isomorphism.

subgraph isomorphism problem for all queries of a query stream, while minimizing the processing latency.

In this problem, we consider the processing latency for the whole data stream.

4.2.2 Approach

To address the problem of streaming subgraph isomorphism, we propose a framework that exploits caching strategies. Our idea is to re-use query results for a large number of the queries in the query stream, thereby minimizing the processing latency. However, a realization of this idea raises several research questions: (Q1) *how to index* nodes, edges, and subgraphs for efficient caching and re-use of query results? (Q2) *how to answer* queries based on cached results? (Q3) *how to manage* the result cache? The interplay of these questions is shown in the illustration of our framework in Figure 4.1. Below, we summarize our techniques to instantiate this framework.

We present a novel method for graph indexing, which speeds up the search for isomorphic subgraphs. The index is based on embeddings of nodes, edges, and subgraphs, in which similar nodes, edges, and subgraphs have similar index values. While the subgraph index enables us to identify re-usable query results in a swift manner, the node and edge indices accelerate traditional algorithms for subgraph isomorphism by pruning the search space.

Our indices serve as a foundation for a novel evaluation algorithm for streaming subgraph isomorphism. It exploits cached results whenever possible. In case of a cache miss, our node and edge indices speed up *any* existing branch-and-bound algorithm used to solve the subgraph isomorphism problem.

In the light of a limited cache size, we further present policies for cache management.

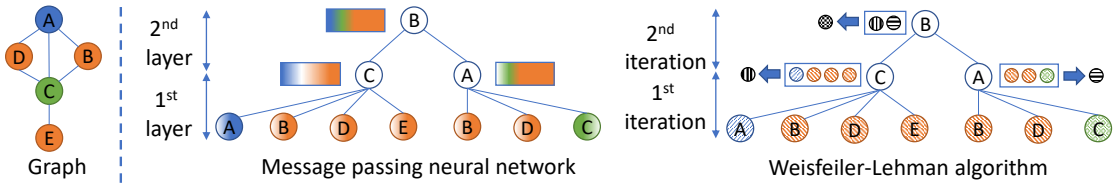


Figure 4.2: Message-passing neural network (color gradients represent embeddings) vs. Weisfeiler-Lehman algorithm (patterns illustrate symbolic representations).

Specifically, we propose to store only a fixed amount of results per query to enable uniform retrieval. To guide cache admission and eviction, we adapt the Landlord algorithm to the setting of streams of subgraph isomorphism queries, which results in a high ratio of cache hits.

4.3 Graph Indexing

This section introduces indices for nodes, edges, and subgraphs based on graph embeddings. We first introduce approaches to learn node and edge embeddings (Section 4.3.1) and subgraph embeddings (Section 4.3.2). Based there, we define the respective indices (Section 4.3.3).

4.3.1 Node and Edge Embeddings

When computing embeddings for nodes, there are two kinds of semantic information to consider: The labels assigned to nodes and their connections to other nodes in the graph. Assuming that semantically similar nodes are assigned similar labels, the respective representation can be incorporated directly in a node embedding. Yet, labels commonly capture external knowledge, not the graph structure. Therefore, we propose to follow the idea of message passing neural networks (MPNN) to enrich the node embeddings with structural information.

Note that we use the embeddings as a means to index nodes, edges, and subgraphs. This is different from traditional graph indexing [BFG+10, KKM11] that relies on subgraphs such as paths, triangles, and cliques as reference points in graph comparison. Our approach avoids the need to detect such subgraphs to construct the respective indices.

MPNN and isomorphism testing. The use of MPNN in our setting is theoretically well-grounded, as it is related to the Weisfeiler-Lehman (WL) isomorphism test [MRF+19]. The WL algorithm also proceeds in rounds and, in the k-th iteration, constructs a node labeling $l^k : V \to \Sigma$ by considering the labels assigned to nodes and their neighbors in the previous iteration. That is, the label of a node v at the k-th iteration is derived as:

$$l_v^{(k)} = h\big(\{l_u^{(k-1)} \mid u \in N(v)\}, l_v^{(k-1)}\big) \tag{4.1}$$

where h is a hash function that maps to a new label, not used in previous iterations. Running the above procedure on two graphs simultaneously, we can test if they are isomorphic: If in any iteration, the constructed node labels differs, the graphs are not isomorphic [MRF+19]. This process is illustrated in Figure 4.2.

Example 11. *In Figure 4.2, right side, labels are visualized by a pattern. In the first iteration, we construct the label of node C by hashing the set containing its own label and the labels of its neighbors. The same is done for node A. The results are used in the 2nd iteration to derive the label for node B.*

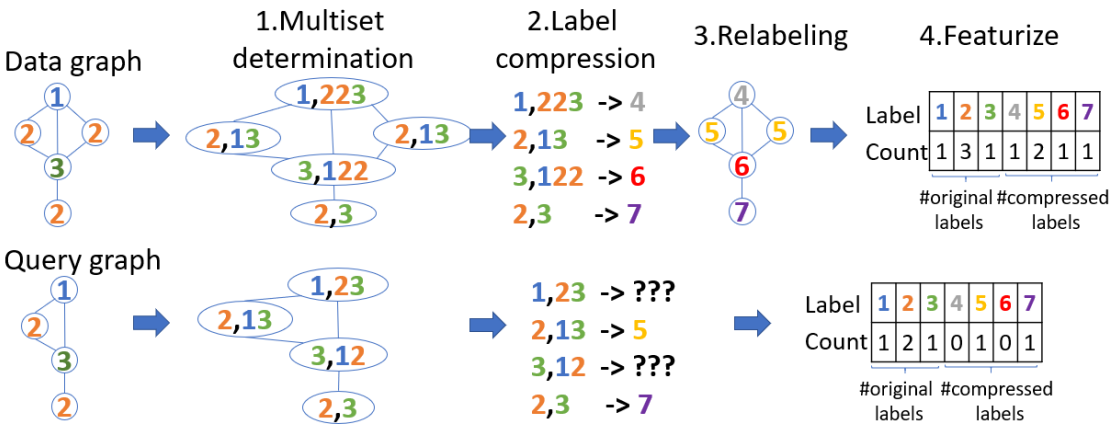


Figure 4.3: Embedding generation process by WL.

The formulations of the MPNN in Equation 2.1 and the WL algorithm in Equation 4.1 are similar. Their relationship is formalized as follows.

Theorem 1 (Weisfeiler-Lehman Testing & MPNN)**.** *Given a labeled graph (g, l), let $l^{(k)}$ be the node labeling obtained using the WL algorithm after k iterations and $z^{(k)}$ be the embeddings obtained by a k-layer MPNN. Then, with suitable initial embeddings $z^{(0)}$ and parameterized functions of the MPNN, for all nodes u, v of g, if $l^{(k)}(u) = l^{(k)}(v)$ then $z^{(k)}(u) = z^{(k)}(v)$.*

The above result follows directly from Theorem 1 in [MRF+19]. It shows that the MPNN-based formulation is as strong as the WL isomorphism test, which provides a theoretical basis for applying MPNN in our framework for streaming subgraph isomorphism. However, the WL algorithm and the MPNN differ in how they represent node labels. The labels derived with the WL algorithm are symbolic representations, i.e., unrelated symbols. The embeddings obtained with the MPNN capture semantic relations, so that an assessment of their similarity is meaningful. In Figure 4.2, we distinguish between both representation by color gradients and patterns, respectively.

WL vs. MPNN. Theorem 1 shows that WL and the MPNN have the same strength to detect graph isomorphism. However, WL requires defining a hash function to compress a multiset to a label as shown in Figure 4.3. This is problematic, once isomorphism shall be detected for graphs with unseen properties. Consider Figure 4.3, where a query graph comprises two multisets (1,23) and (3,12) that have not yet been encountered. WL cannot compress the label and, hence, cannot conduct the isomorphism test effectively. This issue could be addressed in three ways: 1) assuming knowledge of all graphs, a multi-graph WL algorithm is employed to construct all required hash functions; 2) new labels are assigned to new multisets, which are then added in the featurization; or 3) hash functions are removed and the comparison is performed directly on the multiset. While the first solution is not realistic, the second one incurs many zero values in the embeddings, so that comparison becomes imprecise. The third solution incurs significant overhead in terms of processing time to measure the similarity of multisets. The MPNN, in turn, can handle new query graphs seamlessly. Intuitively, it compares multisets, but relies on the embeddings as succinct representations of fixed size, which renders this comparison more efficient. We later confirm empirically that WL is computationally more expensive than the MPNN regarding new queries.

Parameter learning. To learn the parameters of the MPNN, a loss function needs to

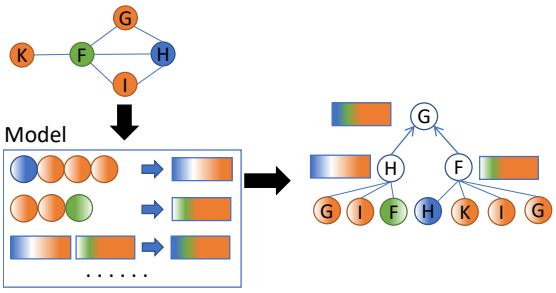


Figure 4.4: Different, but isomorphic, graph yields equivalent embeddings.

be defined. As mentioned, a node embedding represents a summarization of its receptive field. Hence, we define a loss function that rewards if similar embeddings are assigned to similar nodes, i.e., those that are close in the graph:

$$L(z_v) = -log(\sigma(z_v^T z_u)) - Q\mathbb{E}_{u_n \sim P_n(v)} log(\sigma(-z_v^T z_{u_n}))$$

where v is called a positive sample such as u's neighbor, u_n is a negative sample obtained from a negative sampling distribution P_n, and Q is the number of negative samples. The above function strives for similar representations for similar nodes u, v by maximizing $z_v^T z_u$, while minimizing $z_v^T z_{u_n}$ fosters different representations for dissimilar nodes v, u_n. We observe that adding a supervised loss function to reconstruct the node labels to the unsupervised loss can also improve the model's performance.

Edge embeddings. As usual, we construct edge embeddings by averaging the embeddings of the adjacent nodes. We later show that edge embeddings are more discriminative than node embeddings as they enable better pruning of candidates for subgraph isomorphism.

4.3.2 Subgraph Embeddings

For subgraph embeddings to be meaningful, similar subgraphs shall have close embeddings and the labels of nodes shall be incorporated. Moreover, when considering the problem of streaming subgraph isomorphism, we need to cater for large differences in the sizes of the assessed graphs. Given a small query graph, there are potentially very many isomorphic subgraphs in a large data graph [RW16]. An embedding shall support a test for isomorphism that is independent of the specific locations of these subgraphs.

Truncated message-passing for subgraph embedding. Our approach to embed subgraphs of a labeled graph (g, l) (the data graph, in our setting) builds on the function $Z = f(g, l)$ that returns embeddings for all nodes in g. This model, learned on the whole graph, captures the graph's structure in a comprehensive manner. Hence, for a labeled subgraph (s, l'), we can project the model on the respective nodes and their labels, which yields an embedding $Z' = f(s, l')$. Such a projection is akin to truncated message passing, in which solely the nodes in s send messages to neighboring nodes that are also in s. Note though that the parameters of the functions used for sending, receiving, and updating are taken from the MPNN learned to embed the individual nodes.

Example 12. *Figure 4.5 illustrates truncated message passing for a graph of four nodes, A-D, which is a subgraph of the one in Figure 4.2. Messages are exchanged only within the subgraph, but not with node E. Hence, the tree of operations, rooted at B, does not include E.*

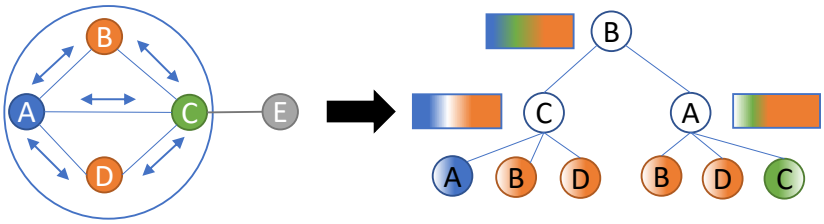


Figure 4.5: Illustration of truncated message passing.

The above process yields embeddings for all nodes in a subgraph. Since each embedding summarizes the node's receptive field, i.e., the subgraph, it is a candidate to represent the whole subgraph. Against this background, we follow a compositional approach and average the node embeddings to represent the subgraph.

In addition, we propose an approach to construct subgraph embeddings from the edge embeddings. Again, a compositional approach is adopted that averages the edge embeddings to represent the subgraph. This is equivalent to a degree-weighted combination of the node embeddings as shown in the following formula:

$$z_s = \frac{1}{|E_s|} \sum_{(u,v)\in E_s} z_{(u,v)} = \frac{2}{\sum_{u\in V_s} deg(u)} \sum_{(u,v)\in E_s} \frac{z_u + z_v}{2}$$
$$= \frac{2}{\sum_{u\in V_s} deg(u)} \sum_{u\in V_s} deg(u) z_u = \frac{\sum_{u\in V_s} deg(u) z_u}{\sum_{u\in V_s} deg(u)} \quad (4.2)$$

We later show empirically that edge embeddings lead to better subgraph embeddings, as cache management becomes more effective.

Transferring models. The model learned to embed the nodes (and, hence, subgraphs) of one graph, may also be transferred to another graph. In our context, it enables us to apply the model learned for the data graph g also to a query graph q. Specifically, we derive the node embeddings $Z_q = f(q, L_q)$ of the query graph, which are then aggregated to obtain an embedding for the whole query graph using the above process. This way, subgraph embeddings of the data graph and the embedding of the query graph are constructed using the same model. Hence, a subgraph of the data graph that is isomorphic to the query graph has an equivalent embedding.

Example 13. *Figure 4.4 illustrates the application of the earlier model to a new graph. Intuitively, the model defines 'rules' to combine embeddings at different layers. Applying the model to isomorphic graphs, see Figure 4.2 and 4.4, yields equivalent embeddings.*

4.3.3 Indexing Embeddings

Similarity computation. To assess the structural similarity of two nodes, or subgraphs, we compute the cosine similarity of their embeddings. This choice is motivated by the locality property of the cosine similarity: It emphasizes the immediate neighborhood of the nodes, independent of their global location in the graph [CLG18].

Indexing high-dimensional embeddings. In our context, the relative similarity of embeddings is more important than their absolute similarity. When answering streams of subgraph isomorphism queries, it is important to find nearest neighbors in the high-dimensional embedding space. Since we approach the problem of subgraph isomorphism

based on embeddings, we can rely on a large body of work on indexing for fast nearest neighbor search in numeric spaces. Specific examples include R-trees [Gut84] and kd-trees [DBCVKO08], which have been shown to be efficient and scalable.

Note that these indexing techniques can be applied to cosine similarity by normalizing each embedding to have a length of one. In this case, the cosine similarity corresponds to the dot product between two embeddings, which is negatively correlated with their Euclidean distance. Moreover, several variants of R-trees and kd-trees that can handle high-dimensional embeddings have been proposed [SAH08, ML09, KS97]. In our experiments, we later adopt an improved version of the kd-tree [SAH08] and also observe that a relatively small embedding size is sufficient to achieve good performance.

4.4 Query Stream Processing

This section introduces our approach to answer a stream of subgraph isomorphism queries. Our idea is to cache and re-use the results of past queries to derive a full or a partial answer to the current query. To identify suitable past queries, we leverage the subgraph indices introduced above. Specifically, for each answered query w, the embedding z_w is indexed. Given a new query q with embedding z_q, identify those past queries w, for which the distance between the embeddings z_w and z_q is below a threshold τ. Depending on whether at least one such query w is identified, we refer to the situation as a cache miss or a cache hit, respectively. For either case, we describe our approach in the remainder of this section.

4.4.1 Handling Cache Misses

In case of a cache miss, we resort to traditional algorithms for subgraph isomorphism. These algorithms follow a backtracking strategy, which explores solutions incrementally, abandoning those that turn out to be invalid. Algorithm 4.1 illustrates this generic process for a given query graph q and a data graph g. Here, a crucial step is to filter candidate structures to map to those of the query graph (line 5). In the worst case, filtering is invoked an exponential number of times, in the size of the data graph, since it relies on the current partial mapping. Hence, the filter step needs to be efficient.

Common subgraph isomorphism algorithms match a query graph and a data graph based on their nodes. In that case, Ω in line 2 contains the nodes of the data graph and s' is the data graph node that matches the query graph node s according to the filter strategy in line 5. Depending on the specific algorithm, the filter leverages a node's label and degree (Ullmann's algorithm [Ull76]) or its connections (VF2 [CFSV04] and QuickSI [SZLY08]). Yet, simple, efficient strategies based on a node's label or degree, tend to be of limited effectiveness.

Advanced strategies to filter candidates for subgraph isomorphism work on the level of subgraphs, not on the level of nodes. While they are commonly very selective, they also suffer from a high computational overhead. For instance, QuickSI [SZLY08] constructs minimum spanning trees and GADDI [ZLY09] is based on shortest path computation. These algorithms need to enumerate particular structures in both, the query and the data graph, which are then used for similarity computation [SZLY08, ZLY09]. This enumeration is expensive, as it essentially solves another graph isomorphism problem.

Embedding-based pruning. We propose to filter candidate structures using their embeddings. Specifically, using the model f, subgraph embeddings of structures of

Algorithm 4.1: Generic search for subgraph isomorphism.

Input : Query graph q; data graph g.
Output: All isomorphic subgraphs of q in g.

```
1 M = ∅;
2 Ω = EnumerateStructures(g);
3 while |M| <> |q| do
4     s ← GetNextStructure(q);
5     for s' ∈ FilterCandidates(s, M, Ω) do
6         M ← Combine(M, s');
7         if M is valid then  R ← R ∪ {M};
8         M ← Backtrack();
9 return R
```

Algorithm 4.2: Embedding-based candidate filtering.

Input : Query graph q; data graph structures Ω; model f; threshold k.
Output: Matching candidates for every structure in q.

```
1 for s' ∈ Ω do
2     z_s' ← f(s', L) ;                    // Conducted offline
3     Z ← Z ∪ {z_s'} ;
4 ℂ ← ∅;
5 for s ← GetNextStructure(q) do
6     z_s ← f(s, L');                      // Conducted online
7     C_s ← kNNSearch(z_s, Z, k);
8     ℂ ← ℂ ∪ {C_s}
9 return ℂ
```

interest in the data graph are created (Ω in Algorithm 4.2-line 2). Note that these embeddings are computed offline. For each structure s identified in the query graph (Algorithm 4.2-line 4), we also construct an embedding. Based thereon, we identify candidate structures in g for s by extracting k nearest neighbors of z_s. Algorithm 4.2 summarizes this idea. First, subgraph embeddings of g are computed offline (line 1-line 3). Then, for each structure of interest in q (line 5), the embedding z_s is computed online (line 6). It is used for a k-nearest neighbor search over the embeddings Z of subgraphs of g (line 7) to obtain candidate structures.

Selecting subgraphs. When searching for isomorphic subgraphs, pruning of candidate structures may be based on different kinds of subgraphs, e.g., nodes [CFSV04], trees [SZLY08], or paths [ZLY09]. Smaller and simpler subgraphs are easier to enumerate, whereas they are less selective. In our setting, we use 2-node subgraphs (i.e., edges) as the basis for a pruning strategy. Edges are easy to enumerate while being more discriminative than nodes. Although it is possible to use 3- or 4-node subgraphs, the exponential growth of the respective subgraphs, illustrated in Table 4.1 for the datasets later used in our experiments, induces severe computational challenges.

The above strategy based on embeddings is orthogonal to other filter mechanisms. Note that, we later show experimentally that, this way, adding our strategy to *any* subgraph isomorphism algorithm reduces the number of candidates to consider significantly.

Table 4.1: Number of subgraphs of different sizes

	1-node	2-node	3-node	4-node
yeast	3'101	12'519	487'696	34'508'857
human	4'674	86'282	7'106'210	>154'376'187
cora	2'708	5'278	59'707	2'270'091
citeseer	3'327	4'600	29'930	643'980
pubmed	19'717	44'324	818'753	28'779'383
wordnet	82'670	127'124	3'260'142	>155'289'760

Algorithm 4.3: Approach to query stream processing.

Input : Data graph g; query graph $q = (V_q, E_q)$ with embedding z_q; cached query embeddings Z; neighbors threshold k; overlap threshold ω.

Output: Node mapping M for g and q.

```
1  R ← kNNSearch(z_q, Z, k);
2  bestW, bestMap ← ∅;
3  for w ∈ R do
4      map ← mcs(w, q) ;                                  // Max common subgraph
5      if |map| = |V_q| = |w| then                        // Case 1
6          return projectMapping(q, w, map, g);
7      if |map| > |bestMap| then
8          bestW ← w;
9          bestMap ← map;
10 if |bestMap| = |V_q| < |w|) then                       // Case 2
11     return projectMapping(q, w, bestMap, g);
12 if |bestMap| ≥ |V_q| − ω then                          // Case 3
13     M ← projectMapping(q, bestW, bestMap, g);
14     return subgraphIsomorphismInitMap(q, g, M)
15 else return subgraphIsomorphism(q, g) ;                // Case 4
```

4.4.2 Handling Cache Hits

Whether the cached results of a past query can be reused for the current query depends on their overlap. Traditionally, the overlap between the query graph q and a cached query graph w is determined by their maximum common subgraph (MCS). The larger the MCS, the better can the results of w be reused for q [LZ19]. Yet, MCS algorithms run in exponential time in the size of the graphs [HLJ06]. Hence, the computation of the MCS of the query graph q with *every* cached query graph induces a significant performance penalty.

To speed up this process, we propose to limit the MCS computation to promising cached queries w, i.e., those that are structurally similar to q. To this end, we use the subgraph indices introduced in Section 4.3. Specifically, we employ the embeddings of the query graph q and the cached query graphs to find the k nearest neighbors to q. While the kNN search can be done efficiently using our index, the MCS problem needs to be solved solely for k pairs of graphs.

The above steps are the first ones in our general approach to query stream processing, as formalized in Algorithm 4.3. Once promising cached queries have been identified by kNN search (line 1), the MCS is computed for each of them and the query graph (line 4). Based on the MCS node mapping, we then assess the level of reusability of the cached results in terms of four cases:

Case 1: When there is an exact match of q and a cached query w, we return the mapping cached for w after projecting it from q to w and w to g, to obtain a mapping from q to g (line 5).

Case 2: If q is isomorphic with a *subgraph* of a cached query w (line 10), we proceed similarly to the first case and construct the result through projection of the cached mapping.

Case 3: A cached query w is said to have some overlap with the query $q = (V_q, E_q)$, if their MCS has at least size $|V_q| - \omega$, where ω is an overlap threshold (line 12). The threshold avoids the re-use of results with a very small overlap, which would not pay-off due to the implied overhead. If the overlap is sufficiently large, the cached mapping, after it has been projected from q to w and w to g, is only a subset of the mapping between q and g. However, this partial mapping is useful in the construction of the actual result. Since common algorithms for subgraph isomorphism construct a mapping by establishing correspondence between nodes, one at a time, they can incorporate the partial mapping derived from the cached result as a starting point for the search.

Case 4: If no promising cached queries can be identified (line 15), we observe a cache miss and resort to the procedure described in Section 4.4.1, i.e., Algorithm 4.1 with embedding-based filtering (Algorithm 4.2).

In our experiments, we observe that a small value of ω leads to minor differences between a cached query and the current query. In some cases, the difference is a single node, which is not meaningful. A large ω reduces the size of the overlap, which increases the time to detect subgraph isomorphism. In our experiments, we observed that setting ω to be 10% of $|V_q|$ strikes a good balance of this trade-off.

4.5 Cache Management

Since a cache has limited size, cache admission and eviction shall be managed such that the number of cache misses is reduced and there is a large overlap between the current and past queries. In this section, we first discuss our general approach to cache management, before we turn to the specific policy.

4.5.1 General Approach

Cache management in our context resembles the problem of online file caching [You08], defined by a cache of fixed size and a sequence of requests to files with assigned retrieval costs. If a file is not in the cache, it is retrieved for the assigned cost, while other files are evicted to make space for it. While such an approach seems useful also for the problem of streaming subgraph isomorphism, there are additional requirements that render existing solutions for online caching to perform poorly in our context.

Cache requirements for streaming subgraph isomorphism. Traditional online caching assumes that every request needs to be answered. However, we may shed queries from cache management, if the incurred delay becomes too large. This provides an additional degree of freedom for a caching policy. Also, in our context, a query may be

answered partially by a cached result, see Section 4.4.2. Hence, a cache management policy shall consider partial cache hits.

Existing online caching algorithms. While various algorithms for online caching have been proposed in the literature, most of which can be described in the framework of the *Landlord algorithm* [PIC19]. The algorithm assigns a credit to each query to denote the cost of answering it. Keeping the query in the cache, this answer cost is saved. Upon the arrival of a new query that does not match any query in the cache, the credit is decreased for all cached queries. Queries without any remaining credit are evicted. On the other hand, when a query is reused, its credit is increased.

The Landlord algorithm does not satisfy the above requirements: It requires *every* query to be put into cache and incorporates solely complete cache hits. Online caching algorithms such as Greedy-Dual and LRU are instances of the Landlord algorithm, so that they suffer from the same shortcomings [PIC19].

4.5.2 Query Utility

The above requirements motivate our design of a new policy for cache management, which we coin the *Screening Landlord* (SL) strategy. It adapts the Landlord algorithm and relies on the *utility* of queries to decide on cache admission or eviction. Here, the utility is both, *time-based*, to incorporate the effort to answer the query, and *diversity-based*, to reflect the query re-usability.

Time-based utility. Three aspects influence the time saved by keeping a query in the cache. First, the *answer time* is the time needed to answer the query and add its results to the cache, which corresponds to runtime for the fourth case in Algorithm 4.2. The answer time for a query q, denoted by $a(q)$, is saved in case of cache hit.

Second, the *reuse time* is the time needed to access and reuse the cached result for a query w to answer a query q. It is denoted by $r(q, w)$ and captures the overhead induced by the cache, i.e., the runtime of the first three cases in Algorithm 4.2. Note that the reuse time varies depending on the size of the overlap of the current query and a cached query. The larger the overlap, the smaller the reuse time.

Given a query stream Q, the answer time and reuse time are aggregated per cached query w as follows:

$$\mu_t(w) = \sum_{q \in Q, mcs(q,w) \geq |V_q| - \omega} (a(q) - r(q, w)) \tag{4.3}$$

Here, the condition $mcs(q, w) \geq |V_q| - \omega$ ensures that only the first three cases of Algorithm 4.2 are considered.

Diversity-based utility. Traditional instances of the Landlord algorithm such as LRU or Recache consider only time and frequency when determining the utility of cached results [You08, PIC19]. However, we strive for caching results of queries that can be reused for many other queries, so that we propose to incorporate a notion of reusability. Intuitively, it is not useful to have many 'similar' queries in the cache, as a diverse set of queries increases the chance that results may be reused for a new query. Therefore, we define a notion of diversity-based utility based on the average embedding distance between a query w and a set C of cached queries:

$$\mu_d(q) = \frac{1}{|C|} \sum_{w \in C} dist(z, z_w) \tag{4.4}$$

The above notion of utility, for the first time, incorporates the diversity of query graphs in cache management. Such an approach would be extremely hard to realize for traditional, structure-based indexing: Measuring dis/similarity between query graphs based on structural properties is computationally expensive and hence, not suited for cache management. Only once embeddings are used, cache management that is guided by the diversity considerations becomes feasible. We later demonstrate empirically that cache diversity is indeed beneficial. It increases the number of cache hits and reduces the answering time significantly.

Combined utility. We combine the above notions to define the overall utility of a query. While both aspects of utility are important, we have to acknowledge that they differ in their normalization factors. Hence, to obtain a meaningful combination, the overall utility is defined by their product, $\mu(q) = \mu_t(q)\mu_d(q)$.

4.5.3 Utility-based Cache Management

Using the above notions, we present the Screening Landlord algorithm for cache management. Unlike the traditional Landlord algorithm, it includes screening step that employs two bounds to decide whether a new query q shall be admitted to the cache.

First, in our proposed method, a user can specify a time upper bound. If processing q takes longer than this threshold, the query is not considered for admission to the cache. This way, the overhead of caching is limited for challenging queries.

Second, there is also a lower bound for the time to process q. It is derived dynamically from the minimum overall utility of cached queries and the distance-based utility of q. Specifically, the bound is the ratio of these values. Intuitively, it defines after which answer time of q, there is a break-even point, i.e., the overall utility of q is higher than the minimum utility of a query currently in the cache.

Having discussed the general intuition, Algorithm 4.4 formalizes our Screening Landlord algorithm. For a new query q, we compute its distance-based utility $\mu_d(q)$ based on the subgraph embeddings of q and the queries in the cache C (line 2). We then extract the minimum utility of cached queries (line 3), before determining the lower bound for the answer time of q (line 4). That is, the lower bound τ is set based on the incoming queries and the current cache. Next, query q is processed using Algorithm 4.3, while monitoring the runtime and aborting cache management based on a user-defined threshold (lines 5-8). If query processing finishes before the timeout, the query is admitted to the cache (line 10). This ensures that the lowest utility in the cache does not decrease as we process more queries.

When a query shall be added to the cache, space may need to be made by evicting the cached query with minimum utility (line 13). Following the Landlord algorithm, once a query is evicted, we decrease the utility of every cached query by the minimum utility value (line 14). This way, we ensure that the cache is not saturated. Without this mechanism, the utility would never decrease, which would prevent any new admission to the cache.

While processing the query q with Algorithm 4.3, we may reuse the result of a cached query w. In this case, we update the utility of w according to Equation 4.3 (line 17). To obtain the reuse cost of a cached query w regarding a query q, we measure the runtime to handle cached results, i.e., the first three cases (lines 4-15) of Algorithm 4.3.

Algorithm 4.4: The Screening Landlord algorithm.

```
1  Proc screeningLandlord(query q, cache C, threshold t):
2      μ_d(q) ← (1/|C|) Σ_{w∈C} dist(z_q, z_w);
3      Δ ← min_{w∈C} μ(w);
4      τ ← Δ / μ_d(q);
5      Try:
6          s ← current time;
7          process q using Algorithm 4.3;
8          a(q) ← current time − s;
9      Catch current time − s ≥ t:  return ;
10     if a(q) ≥ τ then  Add q to cache ;
11 Proc OnInsert(query q):
12     if Cache is full then
13         Evict w with minimum utility;
14         for w ∈ C do  μ(w) ← μ(w) − Δ ;
15     Add q to cache with utility μ(q);
16 Proc OnCacheHit(cached query w, query q):
17     μ(w) ← μ(w) + a(w) − r(q, w)
```

4.5.4 Further Considerations

Handling cache cold start. Initially, as the cache is empty, no results can be reused and query processing is slow due to the subgraph isomorphism search over the complete data graph. Therefore, we propose to populate the cache pro-actively in an offline phase with results from random subgraphs. That is, we randomly select k diverse nodes in the data graph that are far from each other. Starting with each of these node, we construct an ego-network which serves as a subgraph query for which the results are added to the cache. As the actual query stream is processed, we expect these surrogate queries to be evicted from the cache. We confirm the benefits of populating the cache in this manner with a dedicated experiment.

Minimizing distance computation overhead. To compute the distance from the new query to the cached queries (line 2 in Algorithm 4.4), all cached queries need to be traversed. To reduce the induced overhead, we limit this computation to the k nearest neighbors of the query. While this yields solely an approximation of the cache diversity, in practice, the estimates are sufficiently accurate to make correct decisions about cache eviction and admission.

Result cardinality. Since each query may have a different number of matching subgraphs, the time to store the query results varies between queries. We therefore propose to store solely the first k matching subgraphs, which is akin to displaying the first k results in information retrieval systems. After examining these initial results, a users may decide whether the complete query answer shall be derived. If so, the former results are leveraged, similar to the third case of our approach to query stream processing in Algorithm 4.3.

Table 4.2: Statistics of the datasets.

Dataset	$\|V\|$	$\|E\|$	#Labels	Avg. degree
Yeast	3'101	12'519	71	8.07
Human	4'674	86'282	90	36.92
Wordnet	82'670	127'124	5	3.08
Cora	2'708	5'278	7	3.90
Citeseer	3'327	4'600	6	2.77
Pubmed	19'717	44'324	3	4.5

4.6 Evaluation

In this section, we report on comprehensive evaluation experiments. We first clarify the experimental setup (Section 4.6.1), before turning to our approaches to construct embeddings for individual nodes (Section 4.6.2) and complete subgraphs (Section 4.6.3). We then evaluate the effectiveness of our policy for cache management (Section 4.6.5), before we close with an end-to-end comparison of our approach (Section 4.6.7).

4.6.1 Experimental setup

Datasets. We used six standard real-world benchmark datasets for subgraph isomorphism: Yeast, Human, Wordnet, Cora, Citeseer, and Pubmed. The first three datasets originate from [RW16, LHKL12]. Yeast is a protein-protein-interaction (PPI) network with a small average degree, but a large number of labels. Human is also a PPI network, but with a large node degree. Wordnet is a graph capturing relations between English words. It has a small number of labels and a small node degree. The last three datasets were used in [KW17, VFH+18] and denote citation networks. Nodes of these datasets are attributed. However, using the attributes alone is not enough to search for subgraph isomorphism. Statistics of the datasets are given in Table 4.2.

Baselines. We compare our approach against several baselines.

VF2[CFSV04]: is the traditional subgraph isomorphism search algorithm. It is an instance of the generic candidate filtering in Algorithm 4.1, using only labels and the nodes' degrees as filtering criteria.

TurboISO[HLL13]: is the state-of-the-art technique for single-query subgraph isomorphism search. It performs candidate search by constructing candidate regions which can be match with the query graphs. During the subgraph search, only candidates in the regions are considered, which reduces the running time significantly.

MQO[RW16]: is a state-of-the-art technique for multi-query subgraph isomorphism search. It processes queries in batches. For the queries in the same batch, common structures are identified. As a matching subgraph for a common structure can be used for all queries in the batch, this reduces the set of candidate nodes.

For graph indexing, we compare our embedding-based approach with structure-based indices such as GGSX [BFG+10] and CTIndex [KKM11]. GGSX uses paths with bounded length as features to compare subgraphs. CTIndex identifies both paths and cycles of interest to create graph fingerprints. Both methods can be seen as manual feature engineering based on the graph structure, whereas our embedding-based approach derives features automatically.

For cache management, we compare our policy with LRU [You08] and Recache [AKA17],

which are both instances of the Landlord algorithm. Recache manages a cache based on processing times, while LRU incorporates the last access time.

Query streams. Our setting of streaming subgraph isomorphism belongs to the class of multi-query subgraph problems. For such problems, it is a common evaluation strategy to generate queries randomly with parameters that control their overlap and repetition. Specifically, to generate query streams, we follow the generation process from [LHKL12, RW16]. Given a number of subgraph families m, we randomly select m nodes from the data graph of each dataset. For each node, a core subgraph containing n nodes is derived by a random walk. Note that our generation process creates larger queries than those reported in [LHKL12, RW16], as this process measured the graph size by the number of edges. Hence, we derive a more challenging query workload. We then create query streams by inserting, in each of them, a core subgraph, before iteratively adding other subgraphs. With probabilities a, b, c, we add a subgraph previously seen in the stream, in its original form (a), with nodes added (b), or with nodes removed (c). With probability d, we add a new core subgraph. This way, we simulate query streams of different characteristics.

Metrics. As our main metric, we measure the average processing time over the whole query stream. To compare approaches to cache management, we also assess the average hit rate. This metric is used to compare the performance of our cache management strategy.

Implementation and environment. We implemented our model for graph indexing in Python and used Pytorch for offline training. The online query evaluation was implemented in C++.

Our experiments were conducted on a workstation with a 2.4GHz CPU and 24GB RAM. We report average results over 20 experimental runs. Unless stated otherwise, we use a default setting of a query size of 10, a cache size of 20, a timeout of 100ms, a query stream of 1000 queries and an embedding size of eight.

In the following, we first analyze the effects of using embeddings in different components of our framework *independently*. In Section 4.6.2, we analyze the effects of embeddings in filtering candidate nodes and edges for subgraph isomorphism as discussed in Algorithm 4.2. Next, Section 4.6.3 aims to validate the correctness of our subgraph embeddings in different aspects. In Section 4.6.4, we focus on analyzing the effectiveness of the learned embedding function and its training process. Section 4.6.5 is dedicated to the cache management in isolation, while 4.6.7 compares our framework with other baselines in an end-to-end manner.

4.6.2 Effectiveness of Embeddings in Pruning

To evaluate the benefit of using embeddings in pruning matching candidates for subgraph isomorphism, we construct a subgraph of size 20 for every node in the data graph. Hence, for every node and edge of a subgraph, we know their correct mappings. Then, we construct embeddings for all the nodes and rank the nodes of the data graph by their distance to each node of the subgraph. We repeat the same process for the edges. We measure the percentage of the reduction of candidate nodes and edges achieved by filtering based on embeddings. Figure 4.6-A shows that using the embeddings, we are able to filter more than 70% of node candidates and 99% of edge candidates, which highlights the suitability of embeddings in this context. We also observe that edge embeddings are more discriminative than node embeddings, which shows the benefit of using subgraphs (even with only 2 nodes) as pruning criteria.

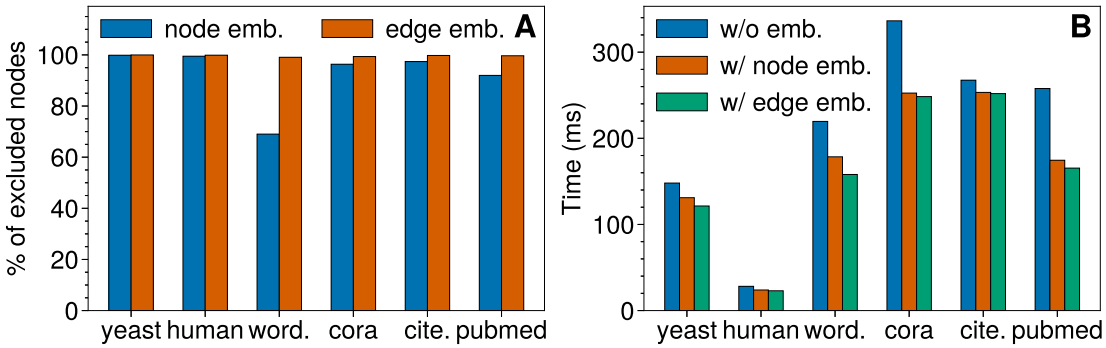


Figure 4.6: Effects of using embeddings.

We further enhance TurboISO with node and edge embeddings as a candidate filtering strategy (see Algorithm 4.1). In Algorithm 4.1, the more candidate structures are identified for each structure in the query graph, the more branches need to be considered in the search. We therefore evaluate the effectiveness of using embeddings by the total time required to find the matching subgraph in the data graph. Figure 4.6-B shows that by adding embeddings as a filter strategy, we reduce the answering time for all datasets, e.g., from 219ms to 178ms to 157ms for the Wordnet dataset by using node and edge embeddings respectively. The observed benefits are relatively small for the Human dataset. The reason being the high label diversity of the dataset. If filtering based on labels is already very effective, further filtering with embeddings becomes negligible. Given the effectiveness of edge embeddings, in the following experiments, we construct subgraph embeddings using edge embeddings.

4.6.3 Evaluation of Subgraph Embeddings

In this set of experiments, we evaluate the correctness of our subgraph embeddings.

Subgraph similarity vs. embedding distance. Next, we aim to evaluate whether the embeddings of structurally-similar subgraphs are indeed close in the embedding space. To measure subgraph similarity, we use two metrics which are the size of the maximum common subgraph (MCS) and the subgraph edit distance. We create a pair of subgraphs with edit distance k by first constructing a two-hop ego graph g from a randomly-selected node in the data graph. We then remove $1 \leq k \leq 7$ edges randomly, so that the subgraph is still connected, to obtain a subgraph g'. As for the size of the MCS, for each dataset, we randomly extract subgraphs of size 15 from the data graph. We then select 5000 pairs of subgraphs randomly and compute the size of its maximum common subgraph. For each pair, we then construct their subgraph embeddings to measure their embedding distance. Our hypothesis is that there is a correlation between the MCS size and edit distances and the embedding distances.

Figure 4.7 confirms this hypothesis. When the MCS size increases, the subgraph embedding distance increases as well. This observation is consistent over all datasets. We observe a clear linear increase for subgraph pairs with MCS larger than 4. The Pearson's correlation values in Table 4.3 confirm that there is a strong correlation between the MCS size and edit distances and the embedding distances. Hence, the subgraph embeddings indeed reflect the structural similarity.

Subgraph embedding visualization. To analyze the subgraph embeddings qualitatively, we extract subgraphs from the data graph of the Human dataset, such that

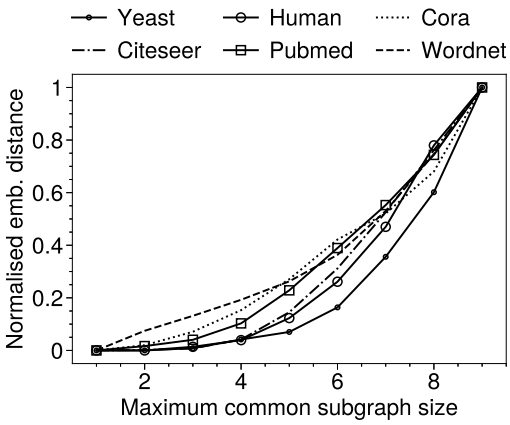


Figure 4.7: MCS size vs. embedding distance.

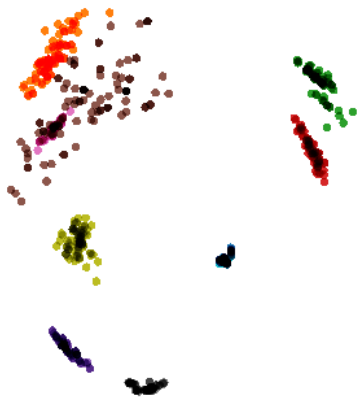


Figure 4.8: Visualization of subgraph embeddings.

Table 4.3: Pearson's correlation coefficients.

	Yeast	Human	Wordnet	Cora	Citeseer	Pubmed
MCS size	-0.95	-0.93	-0.96	-0.88	-0.87	-0.95
Edit distance	0.99	0.98	0.97	0.99	0.99	0.98

subgraphs belong to 10 families. Then, we construct the embeddings of these subgraphs and map them to two dimensions for visualization using PCA. Figure 4.8 shows the subgraph embeddings with color-coded families, highlighting a clear clustering into the families. This highlights that the structural similarities of graphs in a family is captured well by the embeddings.

Search performance. To analyze the benefits of using subgraph embeddings for searching similar subgraphs, for each data graph, we create 20 random families of 50 subgraphs. Then, for each query, we use one subgraph to search for the others. We compare our approach of using subgraph embeddings with a traditional approach to search for similar subgraphs based on the graph structure.

Figure 4.9 shows a large difference in the observed search times. There is a consistent

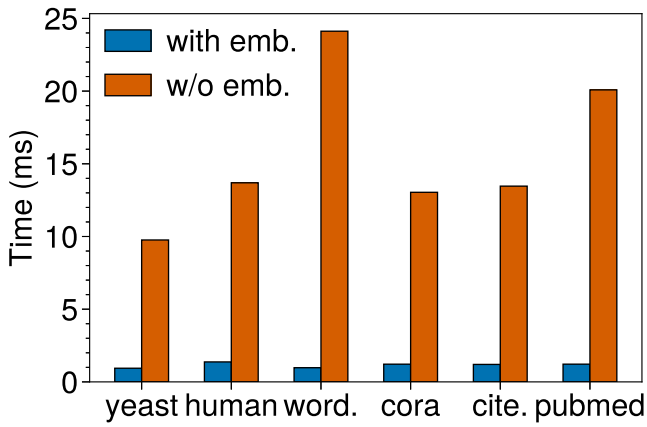


Figure 4.9: Search time.

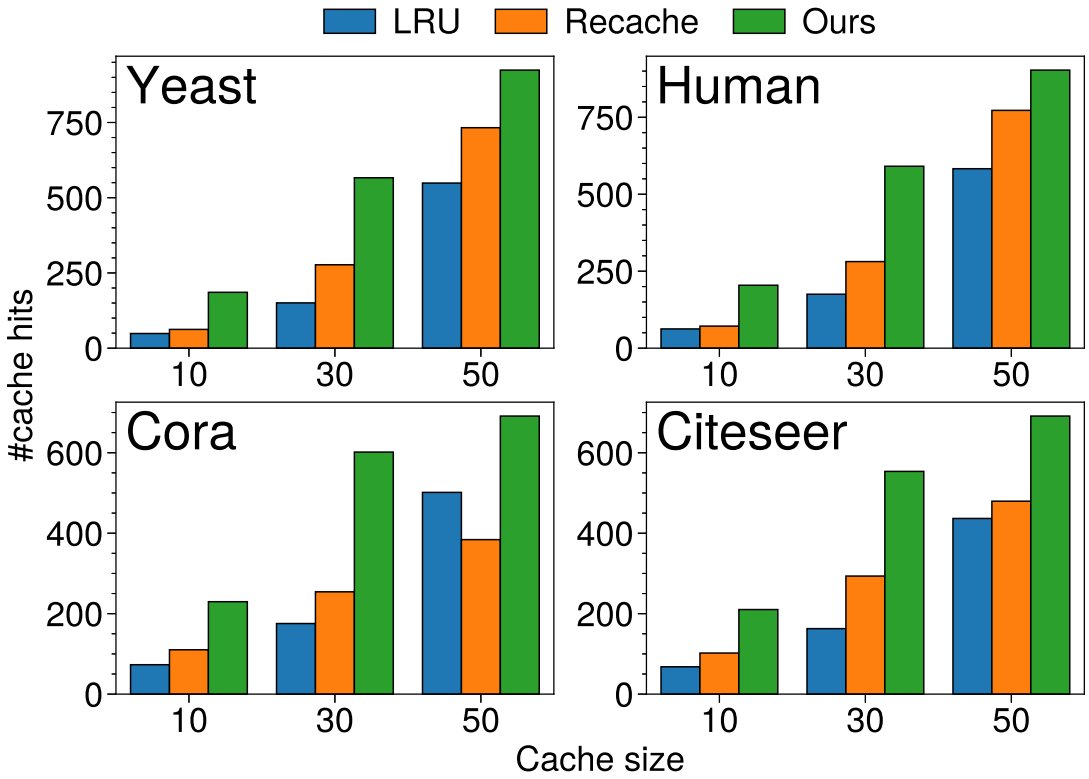


Figure 4.10: Cache size vs hits.

improvement, over all datasets, when using embeddings, with the difference being at least 8ms. For instance, with subgraph embeddings, we can achieve a speedup of up to 25× on the Wordnet dataset. The reason being that searching for similar subgraphs is significantly faster in the embedding space. Structure-based approaches, in turn, require a costly exploration of the actual graph structure. Note that caching was not used in this experiment.

Effectiveness of indexing. We also compare our embedding-based index to structural indices, such as CTIndex or GGSX in terms of time required to create an index for a subgraph. Note that for our approach, this is the time required to construct a subgraph embedding *after* we already train the model (training time will be investigated in Section 4.6.4). In this experiment, we set the subgraph size to 15. The experimental results are shown in Table 4.4. There is a remarkable difference in the efficiency of structure-based approaches and our embedding-based index. CTIndex requires at

Table 4.4: Comparison on indexing time (ms).

		Yeast	Human	Wordnet	Cora	Citeseer	Pubmed
CTIndex (0.5kB)		86.63	1235.45	17.48	59.45	76.68	48.5
GGSX	Time	3.085	528.44	0.395	1.867	3.307	1.142
	Space	14.01	232.8	0.25	1.18	1.46	0.57
Ours (0.5kB)		**0.021**	**0.0195**	**0.0219**	**0.022**	**0.022**	**0.023**

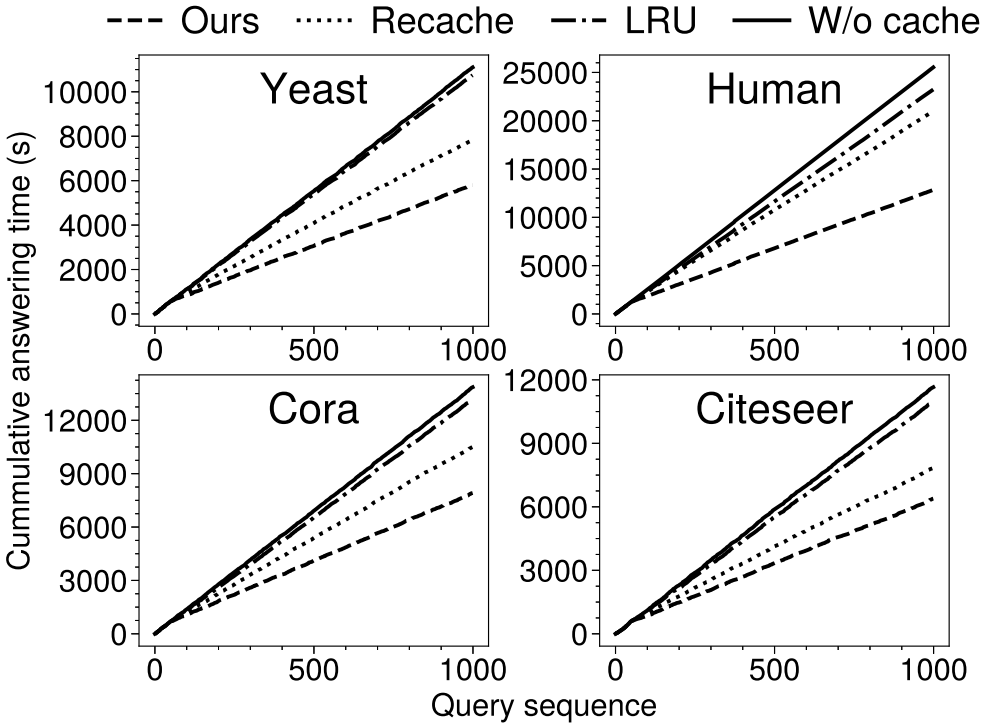


Figure 4.11: Caching strategy vs time.

least 17ms to create an index and on large datasets, such as Human, it would take 1.2s. GGSX performs significantly better than CTIndex with the indexing time staying below 4ms for most datasets. However, our method is an order of magnitude faster and constructs an index in around 0.02ms. These performance results illustrate another benefit of using embeddings in our context. Note that in this experiment, we evaluate the indexing component in isolation, without any caching. Hence, the observed differences stem exclusively from the use of embeddings.

Turning to space requirements, we first note that we used a fixed embedding size for both CTIndex and our embeddings with a similar index size for a fair comparison. As such, CTIndex and our index both require 0.5kB space. The index size of GGSX, in turn, depends on the data graph, see Table 4.4, and ranges from 0.25kB to 232.8kB. We conclude that our index has comparable size to other approaches, but can be constructed much quicker.

4.6.4 Evaluation of Parameterized Subgraph Isomorphism

WL vs. MPNN. We compare the quality of embeddings generated by WL and our MPNN by measuring the correlation between their embedding distance with the subgraph similarity. We randomly extract arbitrary subgraphs from the data graph for each dataset. This is likely to create subgraphs that are *not* observed in the data graph as these subgraphs may not be induced subgraphs. Then, for each pair from a randomly-selected set of subgraph pairs, we measure the size of their MCS. Figure 4.12 confirms our hypothesis that WL would perform poorly on subgraphs that are observed in the data graph. MPNN outperforms WL for all datasets, with large differences emerging

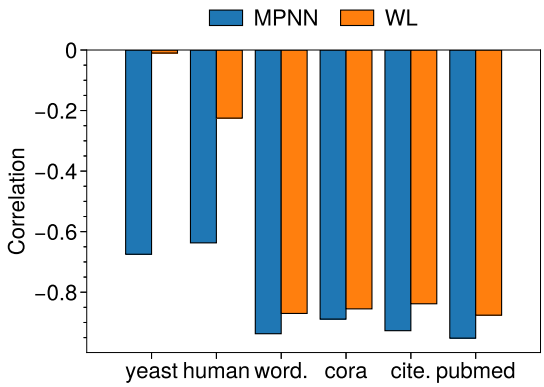


Figure 4.12: WL vs. MPNN (lower is better).

for the Yeast and Human datasets. This is attributed to a high label diversity in both datasets (see Table 4.2), so that the generated subgraphs are more likely to be different from ones in the data graphs. As WL relies on statically determined hash functions, unseen combinations of labels cannot be handled. For the other datasets, the labels are more homogeneous, so that most of the label combinations are already available in the data graphs. As a result, the performance of WL is only slightly worse than the one of MPNN.

To make WL able to handle unseen subgraphs, one may ignore the hash functions and measure subgraph similarity based on their multiset representations (see Section 4.3.1). We measure the total time required to compare the similarity for each pair of subgraphs generated as detailed above. Table 4.5 shows that comparing subgraphs based on MPNN embeddings is significantly faster for all datasets, though. Another interesting observation is that comparing using the MPNN embedding is robust against changes in the subgraph sizes. For the case of WL, larger subgraphs require more time for comparison. The reason is that MPNN embeddings have a fixed size, whereas WL uses a symbolic representation, in which the representation size increases as more combinations of labels are present.

Table 4.5: Time required to compare 1000 subgraph pairs (ms).

	Sub. size	Yeast	Human	Wordnet	Cora	Citeseer	Pubmed
WL	10 nodes	38.89	24.56	22.74	26.42	28.01	22.16
	20 nodes	85.56	48.54	28.05	45.95	52.76	37.62
MPNN		**1.02**	**1.02**	**1.03**	**1.01**	**1.01**	**1.02**

Training time. To measure the training time, we report the time to train one epoch of our model. The number of training epochs can be considered as normalized training time independent of any infrastructure. From Figure 4.13, we observe that the training time per epoch is very small. The longest training time is around 5s for the Wordnet dataset. We further observed that the loss converges after around 10 epochs. This means that the total training time to obtain a good model is at most 60s. Hence, even with short training time, we have already obtained a high-quality embedding model.

Sensitivity to number of parameters. Exploring the effect of the number of parameters in our model, we vary the number of parameters by changing the embedding size,

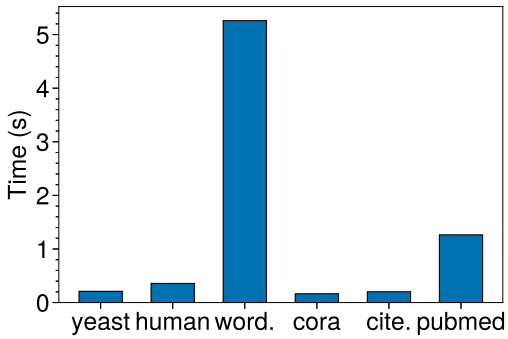


Figure 4.13: Training time.

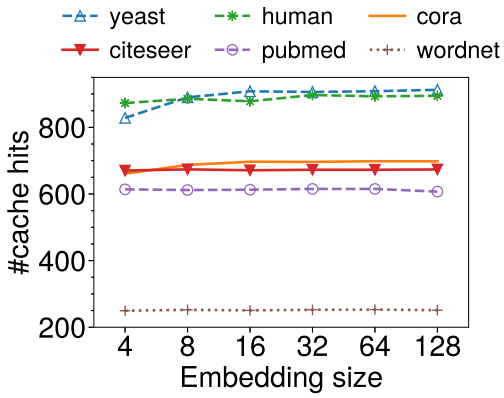


Figure 4.14: Robustness.

as they are closely related. We measure the model's performance by the number of cache hits. Figure 4.14 illustrates that a larger embedding size tends to lead to more cache hits. Yet, the improvement becomes small after an embedding size of 16. For instance, increasing embedding size from 16 to 32 leads to an increase of cache hit by around 2 for Citeseer, Wordnet and Pubmed. We conclude that our model is relatively robust against the number of parameters, while having a small embedding size is commonly sufficient to achieve good performance.

4.6.5 Effectiveness of Cache Management

Having evaluated the benefits of using embeddings without any caching, the next set of experiments consider our caching policy.

Effects of cache size. We measure the number of cache hits for different policies when varying the cache size from 10 to 50. Figure 4.10 indicates that, as the cache size increases, all methods are able to obtain more cache hits, as expected. In general, our cache management strategy outperforms both Recache and LRU, while Recache tends to yield better results than LRU. For instance, using the Human dataset with a cache size of 30, Recache improves over LRU with the difference being 200 hits, while our technique adds further 180 cache hits. Recache performs better than LRU as it considers differences in the queries' answer time. We conclude that our approach of making the cache diverse through a diversity-based notion of utility helps in achieving more cache hits.

Effects of timeout threshold. In this experiment, we analyze our caching policy when varying the timeout threshold for a query from 0.1 to 1 second. Similar to the above experiment, we compare our approach with LRU and Recache and measure the number of cache hits. The results in Figure 4.15 show that our method performs best. For instance, we observe an improvement of 14% and 31% on the Cora dataset over Recache and LRU, respectively. Hence, our method can outperform the baselines irrespective of the timeout threshold and datasets. An interesting observation for the Human dataset is that the number of cache hits remains constant as we increase the timeout threshold for all methods. This can again be attributed to the label diversity in the Human dataset.

Evaluation for query streams. Next, we assess the effectiveness of our caching policy when queries arrive as a stream. Figure 4.11 confirms the observations made for the case of a single query. That is, our policy consistently outperforms the baselines

over all datasets. Compared with using no caching at all, an LRU policy, and Recache, our strategy leads to improvements of 74%, 42%, and 19%, respectively, on the Human dataset at 900 queries. A key observation is that the gap between our strategy and the baselines widens as we process more queries. Hence, using our strategy becomes more beneficial over time, due to increasing cache effectiveness.

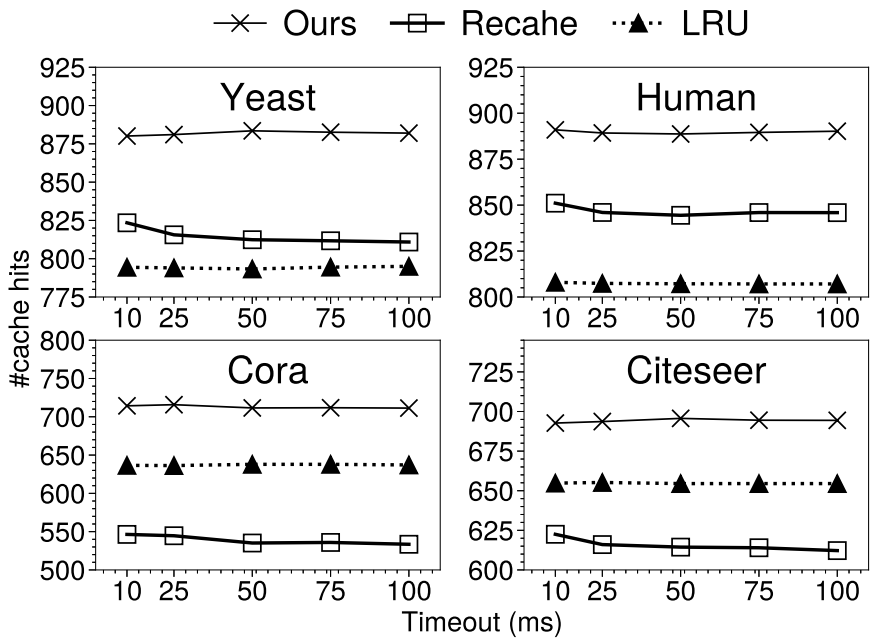


Figure 4.15: Caching strategy vs hits.

Cache initialization. Our technique to alleviate the cold-start problem is evaluated in Figure 4.16 for the Pubmed dataset. The cumulative answer time for the first 50 queries shows that, without using cache initialization, the answer time increases linearly. With our initialization strategy, we are able to reduce the total answer time by at least 20%. Note that our cache initialization strategy is enabled by subgraph embeddings, since those support fast cache search and the identification of partial matches. Cache initialization that would rely on structural similarity search, in turn, would imply a significant overhead, i.e., cache initialization may not pay off.

4.6.6 Workload Evaluation

The following experiments evaluate our framework with respect to different query stream workloads. This requires changing different parameters in our query stream generation, see Section 4.6.1. We also aim to show our simulation is flexible that it can cover different real-world settings. Specifically, we report results on varying the query overlap and repetition.

Effects of query overlapping. We control the overlap of queries by varying the probability of adding subgraphs with added nodes (parameter b in Section 4.6.1). This changes the percentage of overlapping queries in the query streams. The results in Figure 4.18B show that, as the number of overlapping queries increases, the average time required to answer a query decreases for all datasets. This is expected as more overlapping queries mean higher reusability, which leads to better cache hits and reduced answering time.

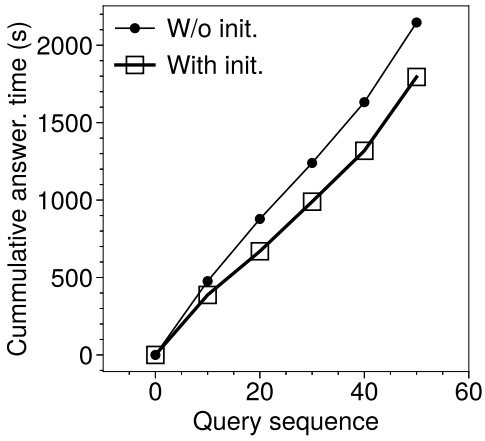


Figure 4.16: Cache init.

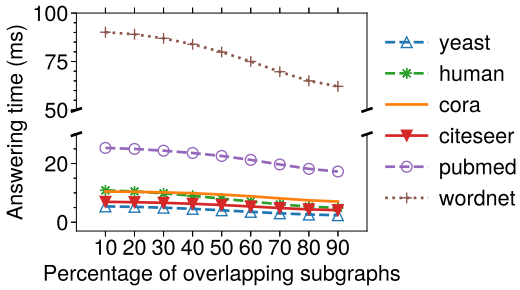


Figure 4.17: Query overlapping

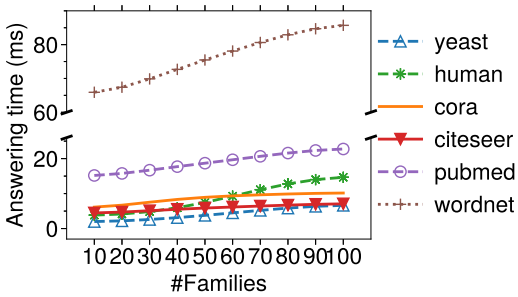


Figure 4.18: Query repetition

Effects of query repetition. We further vary the number of query families in the streams. The more families are present, the higher the diversity of the stream. Figure 4.18A shows that a query stream having more families leads to increased answer time. However, the rate with which it increases is small among all datasets, which shows a certain robustness of our approach to high stream diversity. Note that in this experiment, the cache size is fixed at 50. By increasing the cache size, we expect our method to perform better with more diverse streams.

4.6.7 End-to-end Comparison

After we evaluated the individual building blocks of our solution to streaming subgraph isomorphism, we analyze its end-to-end performance, also in comparison to other techniques.

Comparative analysis. We first compare our approach with traditional subgraph isomorphism techniques. Table 4.6 lists the overall processing times observed for the different datasets. VF2, which is the traditional approach to subgraph isomorphism, has the worst performance. TurboISO, which employs more advanced methods to filter candidate nodes leads to a smaller processing time. It often also performs better than MQO, which is a batch-processing technique. However, across all datasets, our technique leads to processing times that are significantly lower.

The results are interesting in particular in relation to MQO. The latter constructs

Table 4.6: Comparison of different subgraph isomorphism search techniques in terms of overall processing time.

	Yeast	Human	Wordnet	Cora	Citeseer	Pubmed
VF2	45.3	33.7	498.2	60.7	66.2	253.1
TurboISO	12.4	29.6	139.5	15.7	12.7	51.4
MQO	16.2	15.4	315.4	29.8	30.1	80.7
Ours	**3.7**	**7.5**	**114.1**	**9.6**	**7.1**	**31.1**

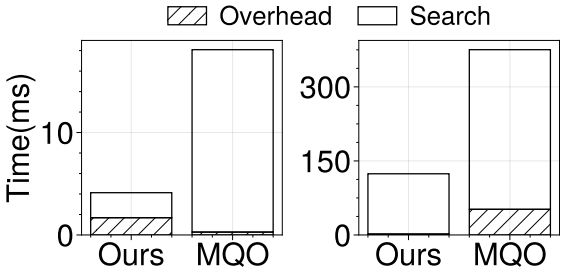


Figure 4.19: Time break-down for Yeast dataset (Left) and Wordnet (Right).

structural indices of queries in the same batch and conducts the search directly on the graph structure. As such, it is similar to our technique in terms of aiming at reuse through indexing. Yet, by relying on embeddings, our approach requires significantly less time to process a query on average.

To provide further insights to the differences between our approach and MQO, we measure the time required to construct the indices (i.e., the induced overhead) and the actual time used for graph search. Figure 4.19 shows that, on a simple dataset such as Yeast, our approach incurs more overhead compared to MQO, as the query embeddings need to be constructed and the cache is searched. Yet, the overhead pays off, as our approach spends less time in the actual search in comparison to MQO. For a complex dataset such as Wordnet, both, the overhead and the search time are lower for our approach. This is due to the inherent complexity of the structure-based approach of MQO.

Effects of query size. Finally, we analyze the impact of the query size on techniques for subgraph isomorphism search. We vary the query size from 10 to 25 and measure the average processing time. Here, our method shows better answering times than the baseline, see Figure 4.20. As expected, the response time increases as the query size increases. However, for our method, the respective rate is smaller or on par with the best baseline. In other words, our method is robust to changes in the query size.

4.7 Summary

In this chapter, we proposed an approach to handle subgraph isomorphism search for streams of queries. Based on advances in subgraph representation learning, we proposed a novel graph indexing technique. This index provides the foundation for our approach to streaming graph isomorphism that exploits caching and reuse of query results. Moreover, we presented a new policy for cache management that assesses the utility of a query not only based on processing time, but incorporates a notion of reusability. Experiments with

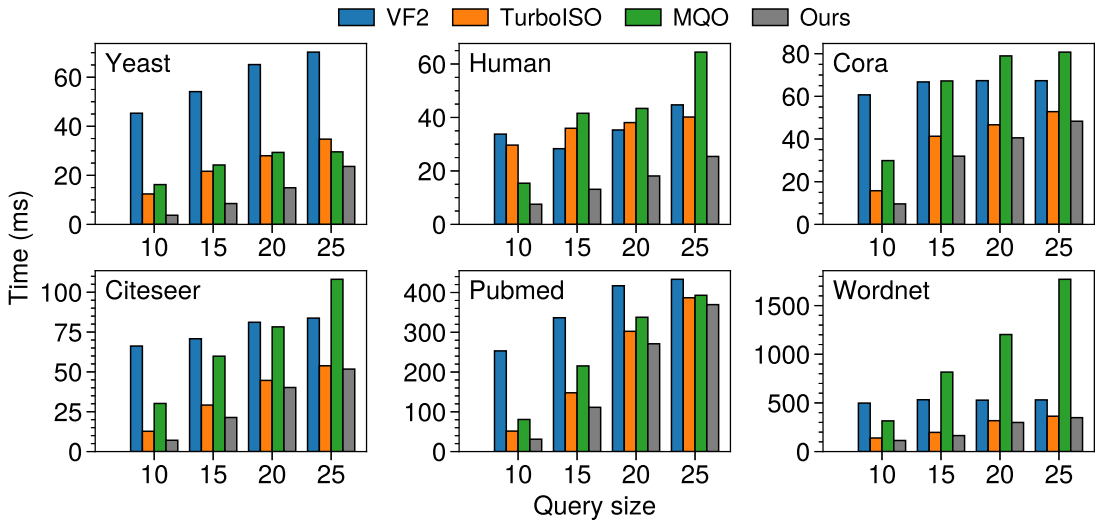


Figure 4.20: Effects of query size.

several real-world datasets confirm the efficiency of our approach and the effectiveness of our design choices.

Chapter 5

Scalability - Scalable Graph Embedding

Scalable Robust Graph
Embedding with Spark

VLDB 2022

Graph embedding aims at learning a vector-based representation of vertices that incorporates the structure of the graph. This representation then enables inference of graph properties. Existing graph embedding techniques, however, do not scale well to large graphs. While several techniques to scale graph embedding using compute clusters have been proposed, they require continuous communication between the compute nodes and cannot handle node failure. We therefore propose a framework for scalable and robust graph embedding based on the MapReduce model, which can distribute any existing embedding technique. Our method splits a graph into subgraphs to learn their embeddings in isolation and subsequently reconciles the embedding spaces derived for the subgraphs. We realize this idea through a novel distributed graph decomposition algorithm. In addition, we show how to implement our framework in Spark to enable efficient learning of effective embeddings. Experimental results illustrate that our approach scales well, while largely maintaining the embedding quality.

5.1 Introduction

Graphs represent relations between entities in complex systems, such as social networks or information networks. To enable inference on graphs, a *graph embedding* may be learned. It comprises *vertex embeddings*, each being a vector-based representation of a graph's vertex that incorporates its relations with other vertices [HYL17b]. Note that in this chapter, we use the term vertex for a node in a graph while we reserve the term node for compute nodes in a compute cluster. Inference tasks, such as vertex classification and link prediction, can then be based on the vertex embeddings rather than the original graph. Various techniques to learn a graph embedding have been proposed [PARS14, HYL17b, HYL17a]. Yet, aiming at high embedding quality at the expense of computational efficiency, they often do not scale to extremely large graphs with billions of vertices and trillions of edges [LWS+19]. Embedding a graph of such size may take weeks, which renders it practically infeasible, and has a large memory footprint. Keeping a graph with two billion vertices in main memory requires 1TB RAM [LWS+19], which exceeds the capacity of commodity servers.

Existing techniques for distributed graph embedding require *continuous* communication between the nodes of a compute cluster for model or gradient synchronisation [ZMW+20, LWS+19]. For instance, in DGL [ZMW+20], each compute node handles a separate subgraph. Yet, all nodes share the same embedding model, which requires synchronization: Each node needs to send gradient updates learned from its subgraph to *all* other compute nodes. As a result, these synchronous approaches suffer from large communication costs. In addition, they are highly susceptible to communication loss or node failure. Upon failure of a compute node, *all* other nodes need to restart from their latest checkpoint. This leads to longer training times and the need for manual intervention in case of failures.

Against this background, in this chapter, we propose a framework for scalable and robust graph embedding that is agnostic to the underlying technique to construct the embeddings. Our idea is to ground the construction of graph embeddings in the MapReduce model. That is, a graph is split into subgraphs, so that an embedding is learned from each subgraph on a separate compute node (map phase). Without a need for synchronisation between the nodes during this computation, *any* specific technique to construct embeddings is improved in terms of scalability and robustness. As learning is done independently and each compute node considers solely a subset of vertices, the results are vectors in different embedding spaces, though. Hence, reconciliation of these spaces is needed to obtain a meaningful graph embedding (reduce phase).

Realising the above vision is challenging, since graph partitioning is an NP-hard problem, which we need to solve in a distributed manner to handle extremely large graphs. In addition, the obtained subgraphs must share vertices, referred to as *landmarks*, to enable reconciliation of embedding spaces. This constraint calls for a new distributed graph decomposition algorithm that carefully chooses the landmarks and considers them in the partitioning process.

In the remainder, we first define the problem of distributed graph embedding and outline our general approach (Section 5.2). We then present the details of our framework, making the following contributions:

MapReduce-based graph embedding (Section 5.3): Our framework includes a map phase, in which each node embeds a subgraph, and a reduce phase to reconcile the embedding spaces. As such, we learn the map and reduce functions instead of defining them upfront.

Scalable graph decomposition (Section 5.4): To facilitate MapReduce-based graph embedding, we propose a scalable graph decomposition algorithm that incorporates landmarks. The algorithm follows the vertex-centric programming model, which enables distributed computation of graph algorithms.

Implementation in Spark (Section 5.5): We show how to implement our framework in Spark [ZXW+16], focusing on data locality, communication optimisation, and GPU integration. The choice of Spark is motivated by its computational model, as Spark workers can work independently on each subgraph before merging the results. Also, Spark provides communication and error handling, which helps in training large graphs as the cost of restarting in case of errors during training is extremely high. We also introduce an iterative refinement process to improve the embedding quality.

Comprehensive evaluation experiments with real-world data of billion-scale graphs illustrate the effectiveness and efficiency of our approach (Section 5.6). We show that our graph embedding framework is at least 2× faster than existing approaches, reduces communication cost by at least an order of magnitude, and still achieves better embedding quality than existing techniques. We conclude the chapter in Section 5.7.

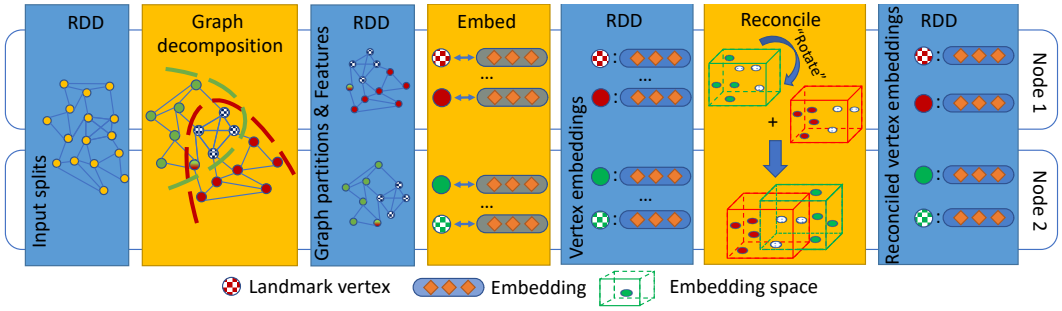


Figure 5.1: One round of computation in our framework on two compute nodes (yellow: operations; blue: input/output data).

5.2 Problem and approach

Below, we first formalize the addressed problem, before outlining our general approach.

5.2.1 Problem statement

Let $\mathcal{G} = (V, E)$ be an undirected graph with vertices V and edges $E \subseteq [V]^2$. A graph embedding technique aims to learn a mapping $f_\Theta : V \to \mathbb{R}^d$ from vertices V to an embedding in a low-dimensional space ($d \ll |V|$), such that 'similar' vertices are mapped to close vertex embeddings [HYL17b].

To learn embeddings for large graphs, we consider a cluster of n compute nodes. Given a graph and an embedding technique, the problem of *distributed graph embedding* is to leverage the n compute nodes for the efficient construction of the graph embedding. Here, efficiency is largely determined by the communication cost between the compute nodes, which shall be minimized.

5.2.2 Approach

Our approach to distributed graph embedding works in several rounds. In each round, as illustrated in Figure 5.1, the graph is decomposed into n subgraphs, so that the construction of embeddings can be distributed among n compute nodes. However, as the vertex embeddings are created independently, they belong to different spaces. To tackle this problem, embeddings from different spaces are reconciled based on vertices that are shared among the subgraphs, called *landmarks*. To obtain embeddings of high quality, the above computation is performed in several rounds, in which the models obtained in the previous round are used for further refinement in the next round.

Our approach can be formulated in the MapReduce model [DG08]. The construction of embeddings is akin to the map function, since a subgraph is mapped to several vertex embeddings. The reconciliation of embedding spaces denotes a reduce phase, in which a single reconciled embedding is derived. Following this model, our framework can be implemented in state-of-the-art engines for distributed data processing, as demonstrated later with Spark [ZXW+16].

We note that the decomposition of the original graph into subgraphs is a crucial part of our approach. However, there is an exponential number of possible decompositions of a graph into n partially overlapping subgraphs, and each split has different consequences for the quality of the final embedding. To be able to decompose the graph in a distributed manner and chose suitable landmarks, we design a landmark-aware decomposition algorithm based on the vertex-centric programming model [MAB+10]. Our

algorithm is based on the Label Propagation Algorithm (LPA) [MLLS17], in which we control the condition upon which a vertex may migrate from one partition to another.

5.3 MapReduce-based Embedding

To formulate distributed graph embedding in the MapReduce model, we first provide background on MapReduce (Section 5.3.1), before discussing the map (Section 5.3.2) and reduce (Section 5.3.3) functions.

5.3.1 Background on MapReduce

MapReduce [DG08] is an iterative computational model, where in each round, two types of functions are applied: *map* and *reduce*. A map function f_m is applied to a multiset of data elements of some type τ and yields a multiset of pairs of data keys of type k and data values of type v, i.e., $f_m : \tau \rightarrow \mathcal{B}(k \times v)$ with $\mathcal{B}(X)$ as the set of all multisets (or bags) over some set X. A reduce function f_r is applied to a multiset of key-value pairs of the same key and combines values of type v to return another value of type v, i.e., $f_r : \mathcal{B}(k \times v) \rightarrow v$. Grounding some computation in the MapReduce model enables highly scalable and fault-tolerant execution since a large number of *mappers* and *reducers*, i.e., tasks to evaluate the map and reduce functions, can be executed concurrently.

5.3.2 Learned Map Function

In our framework, the map function takes a subgraph $\mathcal{S}$ as input and returns its vertex embeddings $\boldsymbol{F}$. In other words, the map function is exactly the function $f_\Theta : V \rightarrow \mathbb{R}^d$ that a graph embedding technique aims to learn. Hence, unlike traditional map functions in MapReduce, our map function is not defined upfront. Rather, we define the *structure* of the function to construct embeddings and rely on learning framework such as Pytorch [PGM+19] to estimate its parameters by minimising a loss function. Then, each mapper learn its own function f_Θ based on the input subgraph and additional information, such as vertex features.

There are two approaches to construct graph embeddings and, hence, two ways to define the structure of the respective function: shallow graph embedding techniques and graph neural networks.

Shallow graph embedding. In shallow embedding techniques, the function f_Θ is just a mapping from the vertices to the embeddings, i.e., the vertex embeddings are learned directly. Hence, the parameters of function f_Θ are the vertex embeddings. That is, the function is defined as $f_\Theta \colon u \mapsto \boldsymbol{u}$, where $\{\boldsymbol{u} \mid \forall\, u \in V\} = \Theta$ are the parameters that we need to learn. In other words, the vertex embeddings are the model itself.

Graph neural network. A graph neural network (GNN) is a deep embedding model, where the final vertex embeddings are obtained by applying several transformation functions consecutively on the vertex features. Put it differently, function f_Θ is a composition of several transformation functions, such that each parametrised transformation function maps from one vertex embedding to another one. The initial embeddings are given directly by the vertex features. For instance, GCN [KW17] defines such transformation functions as $f_\Theta^{(k+1)}(\boldsymbol{E}^{(k)}, \hat{\boldsymbol{A}}) = \hat{\boldsymbol{D}}^{-1/2}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-1/2}\boldsymbol{E}^{(k)}\Theta^{(k)}$, each mapping an embedding from the previous layer $\boldsymbol{E}^{(k)}$ to the next layer using the parameters $\Theta^{(k)}$. Here, $\hat{\boldsymbol{D}}$ and $\hat{\boldsymbol{A}}$ are the normalised degree matrix and the normalised adjacency matrix, respectively.

Learning the parameters. The parameters of the functions to construct embeddings are learned by minimising a loss function. The loss function represents the objective that shall be captured by the vertex embeddings. There are two common types of loss functions: supervised and unsupervised. With a supervised function, vertices are labelled and the embeddings shall be able to predict these labels. In the unsupervised setting, vertices do not carry labels and the vertex embeddings shall capture the structure of the graph. We focus on the unsupervised setting as 1) it is more widely applicable and 2) it imposes more severe challenges in terms of scalability.

For vertex embeddings to reflect the graph structure, they shall capture the similarity between vertices. More precisely, 'similar' vertices should have close vertex embeddings, and vice versa. Traditionally, the similarity of vertices is measured by the number of times that they co-appear on a random walk [HYL17a, HYL17b]. Closeness of two vertex embeddings, in turn, is commonly measured by their cosine similarity. In this case, the loss function is defined as follows:

$$\mathcal{J}(\boldsymbol{u}) = -log(\sigma(\boldsymbol{u}^T\boldsymbol{v})) - Q\mathbb{E}_{v_n \sim P_n(v)} log(\sigma(-\boldsymbol{u}^T\boldsymbol{v}_n)) \tag{5.1}$$

where v is a vertex that co-occurs with vertex u on a random walk and $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{v}_n$ are the embeddings of vertex u, v, v_n respectively. Q denotes the number of negative samples, which are obtained from a negative sample distribution P_n.

The random walks are controlled by two hyperparameters, which are the walk length and the number of walks. The longer the walk length, the more likely that further vertices are considered as being similar to the current one. On the other hand, with more walks per vertex, there is a higher chance that more vertices are covered.

5.3.3 Landmark-based Reduce Function

Learned reduce function. The reduce function takes two embeddings $\boldsymbol{F}_1, \boldsymbol{F}_0$ as input and returns a reconciled embedding $\boldsymbol{F}$. Our idea is to reconcile embedding spaces based on landmarks. A landmark will be associated with different embeddings in different embedding spaces, even though it relates to the same entity, i.e., the same vertex in the original graph. Hence, landmarks tell us how to convert an embedding space into another one.

Figure 5.2 shows embedding spaces derived for two subgraphs, where embeddings of three landmarks are shown as stars, triangles, and squares. By 'rotating' one embedding space, such that the embeddings of landmarks in either spaces are close to each other, we are able to align the two spaces.

We realize this idea by learning a mapping function $h(\boldsymbol{F}_1)$ that takes a source embedding space as input and returns a mapped space, such that the embeddings of the landmarks are close in $\boldsymbol{F}_0$. In other words, we want to "rotate" the embedding space $\boldsymbol{F}_1$ such that their embeddings are aligned in the embedding space $\boldsymbol{F}_0$. We call the embedding space $\boldsymbol{F}_0$ the *anchor space*. This approach is inspired by techniques for network alignment [MSL+16] and cross-lingual dictionary building [CLR+18, ALA16]. However, there is an important difference: In network alignment, the vertex correspondences are pre-specified and used as input to train the model. In our setting, the landmarks can be chosen explicitly, which we later exploit with a dedicated selection strategy. The mapping function can be linear or a multilayer perceptron [RRK+90]. In any case, our objective is captured by the following loss function:

$$L(h, \boldsymbol{F}_1, \boldsymbol{F}_0, L) = \sum_{v \in L} ||h(\mathbf{z}_{i,v}) - \mathbf{z}_{0,v}||_F \tag{5.2}$$

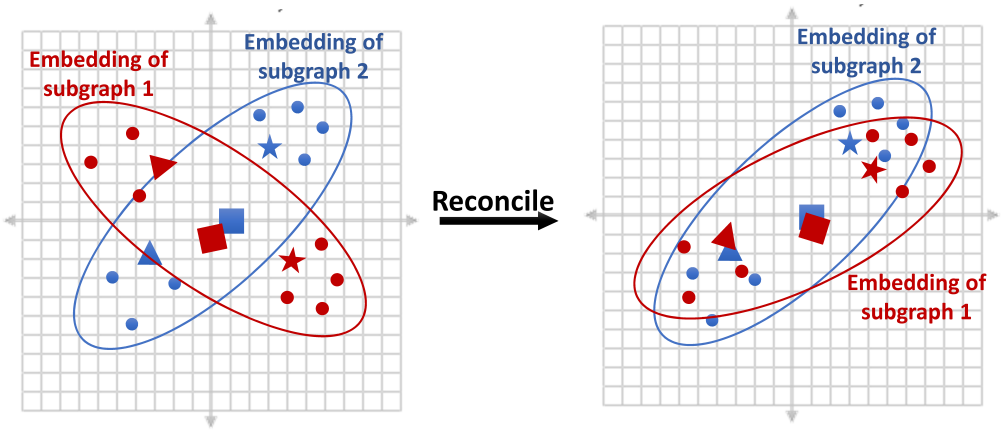


Figure 5.2: Reconciliation of embedding spaces.

where $||.||_F$ is the Frobenius norm, L is the set of landmarks, and $\mathbf{z}_{i,v}$ is the embedding of landmark v in $\boldsymbol{F}_1$. Since it was shown that a linear function is sufficient to obtain a good mapping [CLR+18, ALA16], we define the mapping function as $h(\boldsymbol{F}_1) = \boldsymbol{F}_1 \times \boldsymbol{W}$ where $\boldsymbol{W} \in \mathbb{R}^{d \times d}$ and d is the embedding dimensionality. The above equation is rewritten in its matrix form, if we denote the embedding matrices of the landmark nodes of $\boldsymbol{F}_1$ and $\boldsymbol{F}_0$ as $\boldsymbol{H}_1$ and $\boldsymbol{H}_0$:

$$L(\boldsymbol{H}_1, \boldsymbol{H}_0, \boldsymbol{W}) = ||\boldsymbol{H}_1\boldsymbol{W} - \boldsymbol{H}_0||_F \tag{5.3}$$

Also, it is known that a better mapping is obtained when enforcing orthogonality on $\boldsymbol{W}$ [CLR+18]. Under the orthogonality constraint, the mapping matrix $\boldsymbol{W}$ that minimises Equation 5.2 is found using singular value decomposition (SVD). Let $\boldsymbol{U\Sigma V}^T = \boldsymbol{H}_0\boldsymbol{H}_1^T$ be the SVD of the matrix $\boldsymbol{H}_0\boldsymbol{H}_1^T$. Then, $\boldsymbol{W}$ is computed as $\boldsymbol{W} = \boldsymbol{UV}^T$. The matrix $\boldsymbol{W}$ can be computed as discussed above since this is its closed form solution. In the general case, it can be found by minimzing the function in Equation 5.3.

While we focus on mapping the landmarks, the learned mapping function is applicable to the whole embedding space. Let $\boldsymbol{W}_1$ be the mapping matrix from $\boldsymbol{H}_1$ to $\boldsymbol{H}_0$. Then, the reconciled embedding space of $\boldsymbol{F}_1$ is $\boldsymbol{F}_1\boldsymbol{W}_1$. Combined with $\boldsymbol{F}_0$, we obtain the reconciled embedding space $[\boldsymbol{F}_0, \boldsymbol{F}_1\boldsymbol{W}_1]$ where $[\,.\,,\,.\,]$ is the concatenation operator. As such, the reduce function is given as:

$$r(\boldsymbol{F}_0, \boldsymbol{F}_1) = [\boldsymbol{F}_0, \boldsymbol{F}_1\boldsymbol{W}_1] \tag{5.4}$$

Reduction order. To support the parallel execution of reducers, the reduce function needs to be *commutative* and *associative*. In our case, these properties ensure that the order in which we reconcile the embedding spaces does not affect the final embeddings.

First, the reduce function learned as in Equation 5.4 is commutative. While the order of reducing would return either $[\boldsymbol{F}_0, \boldsymbol{F}_1\boldsymbol{W}_1]$ or $[\boldsymbol{F}_0\boldsymbol{W}_0, \boldsymbol{F}_1]$, both results have the same meaning in our setting as the relative positions of the embeddings are the same in both cases. More precisely, we consider two embedding spaces to be the same, if one can be obtained from the other under a rotation. The above embedding spaces are similar as $[\boldsymbol{F}_0, \boldsymbol{F}_1\boldsymbol{W}_1] = [\boldsymbol{F}_0\boldsymbol{W}_1^{-1}, \boldsymbol{F}_1] = [\boldsymbol{F}_0\boldsymbol{W}_0, \boldsymbol{F}_1]$ since $\boldsymbol{W}_0 = \boldsymbol{W}_1^{-1}$. The latter is derived from the fact that in Equation 5.3, depending on the order of application, we obtain either $\boldsymbol{W}_1$ or $\boldsymbol{W}_0$.

However, our reduce function is not guaranteed to be associative. For $r(r(\boldsymbol{F}_0, \boldsymbol{F}_1), \boldsymbol{F}_2)$ to be equal to $r(\boldsymbol{F}_2, r(\boldsymbol{F}_0, \boldsymbol{F}_1))$, $[\boldsymbol{F}_{01}, \boldsymbol{F}_2\boldsymbol{W}_2]$ needs to be equal to $[\boldsymbol{F}_{01}\boldsymbol{W}_{01}, \boldsymbol{F}_2]$ or

$\boldsymbol{W}_{01} = \boldsymbol{W}_2^{-1}$. The latter is only true, if the embedding space $\boldsymbol{F}_2$ shares the same landmarks with both $\boldsymbol{F}_1$ and $\boldsymbol{F}_0$. Put differently, for multiple applications of the reduce function to be associative, all the embedding spaces need to share the same landmarks.

From the above observation, we derive a requirement for our graph decomposition strategy: The subgraphs shall share a common set of landmarks. Note that further vertices shared by a pair of subgraphs, which are not landmarks, will not affect the resulting embedding space.

5.4 Scalable graph decomposition

This section introduces our approach to graph decomposition. We first propose a two-step algorithm based on message-passing, a vertex-centric computational paradigm (Section 5.4.1). We then define two decomposition strategies used by our approach, landmark-aware partitioning (Section 5.4.2) and complement graph partitioning (Section 5.4.3).

5.4.1 General Approach

Requirements. While graph decomposition is a well-studied problem, there are several requirements that pertain to our setting:

(1) The decomposition shall be able to handle large graphs and operate in a distributed setting. Centralised graph partition algorithms, such as METIS [KK95], can handle large graphs, but have a large memory footprint and, hence, are not applicable for compute clusters built from commodity hardware.

(2) The decomposition shall support constraints on the size of the subgraphs. This way, subgraph sizes can be determined based on the memory available on compute nodes, thereby enabling their optimal utilisation and preventing stragglers. Since we assume commodity hardware, we aim to have partitions of equal size. If nodes have the same amount of memory, the subgraph size is $\frac{n-n_l}{k} + n_l$, where n is the graph size, n_l is the landmark subgraph size, and k is the number of nodes.

(3) Subgraphs shall share the same set of connected landmark vertices to support reconciliation of embedding spaces. As vertex embeddings are learned based on their connectivity, connectedness ensures that landmark embeddings are meaningful, while having important vertices as landmarks ensures that landmarks have strong connections.

(4) Subgraphs shall have minimal overlap apart from the landmarks, as these boundary edges may be ignored, which could negatively affect the embedding quality.

Decomposition problem. We formulate the decomposition problem for a graph $\mathcal{G} = (V, E)$ as follows. Let $\mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_n$ denote n subgraphs of $\mathcal{G}$, where $\mathcal{S}_i = (V_i, E_i)$ for $1 \leq i \leq n$. Moreover, let $\mathcal{L} = (V_L, E_L)$ denote a landmark graph, i.e., a graph that is induced by a set of landmark vertices $V_L \subseteq V$. For a subgraph $\mathcal{S}_i$, we denote by $\overline{\mathcal{S}}_i$ the complement graph obtained from $\mathcal{S}_i$ after excluding the landmark graph $\mathcal{L}$.

Using the above notions, we define the landmark-aware graph decomposition problem. It is the decomposition of a graph $\mathcal{G}$ into n overlapping subgraphs $\mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_n$ such that

(1) the subgraphs share m landmarks $V_L \subseteq \cap_i V_i$ and $|V_L| = m$;

(2) the induced landmark graph $\mathcal{L}$ is connected;

(3) the landmarks are important vertices, where importance is measured by a function $\delta : V \to \mathbb{N}$;

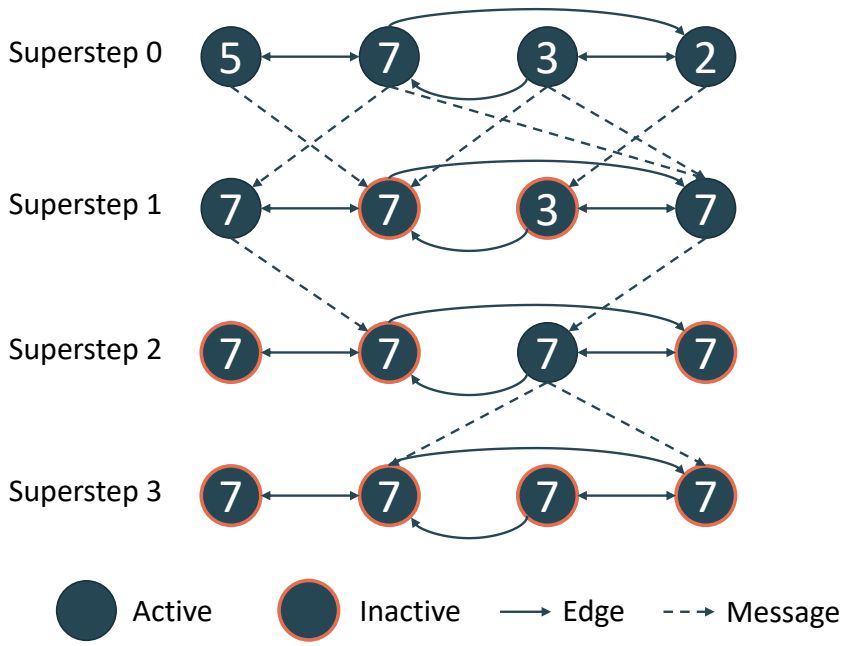


Figure 5.3: Maximum value computation by vertex-centric computation model

(4) the subgraphs are of predefined sizes $n_1, \cdots, n_k$, $|V_i| = n_i$;
(5) the number of edges of the original graph that connect vertices of different complement graphs $\overline{\mathcal{S}}_i, \overline{\mathcal{S}}_j$ is minimised.

Vertex-centric computational model. The above requirements suggest to rely on message-passing as a vertex-centric computational model [MAB+10, MLLS17]. It enables the efficient and robust realization of distributed graph algorithms. Also, reasoning about the algorithm becomes straightforward since one may focus on local operations on vertices, while the global semantics emerge from the combination of local operations. The model supports distribution, as there is no order of local operations as part of a so-called superstep, while all communication happens between these supersteps.

In this model, in each superstep, a user-defined function for each vertex is executed in a synchronous manner [MAB+10]. The model permits fourth types of operations that a vertex can perform. First, a vertex may read messages it received in the previous superstep. Second, a vertex may send messages to other vertices (usually its neighbours) that they will receive in the next superstep. Third, a vertex may change its internal state and modify its outgoing edges if necessary. Fourth, a vertex may vote to halt the computation. Based on these four primitive operations, various graph algorithms can be implemented, e.g., Pagerank [MAB+10], connected components [MAB+10]. Spark has an implementation of the vertex-centric computation model called GraphX, which we use to design our partitioning algorithm.

Figure 5.3 illustrates the computation of the maximum value of a strongly connected graph. In the first superstep, every vertex sends its known maximum value to its neighbors. Each vertex then updates its known maximum value and votes to halt. When a vertex receives a message, it becomes active, updates the known maximum value before sending the updated value to its neighbors again. This process ends when every node votes to halt.

Algorithm 5.1: Label propagation algorithm

input : Graph $\mathcal{G} = (V, E)$; label set $\mathbb{L} = \{l_1, \ldots, l_n\}$; compatibility function *comp*, termination condition Ω.
output: n subgraphs $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n$ induced by the vertex labelling.

```
   // Label initialisation - Vertex program
1  for v ∈ V do label(v) ← init_label(v);
   // Label propagation
2  while not Ω do
3      for v ∈ V do                                        Vertex program
4          best_label ← {};
5          best_score ← −∞;
           // Compute compatibility score
6          for l ∈ L do
7              if comp(v, l) > best_score then
8                  best_label(v) ← l;
9                  best_score ← comp(v, l);
10
       // Vertex migration
11     for v ∈ V do                                        Vertex program
12         label(v) ← migrate(label(v), best_label(v))
13
       // Statistics to support comp calculation
14     compute_statistics(G, label);
15 return S_1, S_2, ···, S_n where S_i = {v ∈ V ∧ label(v) = l_i};
```

Label propagation algorithm. Most algorithms using the vertex-centric computational model are instances of the Label Propagation Algorithm (LPA) [MAB+10, MLLS17], which is illustrated in Algorithm 5.1. LPA first assigns labels to vertices randomly (line 1). Then, it iteratively improves the results by reassigning vertex labels (lines 2-10). A vertex v will take the label l, if it is the most compatible one according to a compatibility function, $comp(v, l)$. Algorithms based on LPA differ in how they measure the compatibility between a vertex and a label. Moreover, each vertex can then choose to 'migrate' from one label to another one based on information obtained from its neighbours or itself from previous iterations (lines 11-12). After the migration, depending on the function *comp*, statistics are derived to support the evaluation of function *comp* (line 14). For instance, if *comp* involves constraints on subgraph sizes, the statistics would include the measured sizes. While both the compatibility scoring and migration are implemented as vertex-centric programs, for readability, we present LPA as an iterative algorithm.

Next, we show how LPA is instantiated in our setting to design an algorithm for landmark-aware graph decomposition.

A two-step approach. A solution to our problem of landmark-aware graph decomposition would be an algorithm based on n-way graph partitioning [KK98]. It would

decompose a graph into n subgraphs at the same time, such that all subgraph satisfy the above constraints. However, finding such a decomposition is difficult, as even in the simplest case of balanced graph partitioning, the problem is already NP-hard [FF15]. This is further complicated by the usage of the vertex-centric programming model.

We therefore propose a heuristic algorithm that works in two steps, each tackling a subset of the constraints. In the first step, we focus on constructing the connected landmark graph of vertices of high importance. In the second step, we aim to construct the complement graphs that satisfy the constraints on subgraph sizes and crossing edges. The algorithm is illustrated in Algorithm 5.2 and Figure 5.4.

Algorithm 5.2: Landmark-aware graph decomposition

input : Graph $\mathcal{G} = (V, E)$; number of subgraphs k; number of landmarks m; maximal subgraph sizes $\{n_1, \ldots, n_k\}$.

output: k subgraphs $\mathcal{S}_1, \ldots, \mathcal{S}_k$; landmark graph $\mathcal{L}$.

```
// Computing vertex centrality
```

1 $\delta \leftarrow centrality(G)$;

```
// Landmark-Complement graph bi-partition
```

2 $\mathcal{L}, \overline{S} \leftarrow LPA(\mathcal{G}, \{l_{\mathcal{L}}, l_{\overline{\mathcal{L}}}\}, Equation\ 5.7)$;

```
// Complement graph partition
```

3 $\overline{S}_1, \cdots, \overline{S}_k \leftarrow LPA(\overline{S}, \{l_1, \cdots, l_k\}, Equation\ 5.8)$;

4 **for** $i \in [1, k]$ **do** $\mathcal{S}_i \leftarrow \overline{S}_i + \mathcal{L}$;

5 **return** $\mathcal{S}_1, \ldots, \mathcal{S}_n, \mathcal{L}$;

We first measure the importance of each vertex in the graph based on a centrality score (line 1). Next, we decompose the graph into the landmark graph and a complement graph using the LPA with the compatibility function introduced later in Equation 5.7 (line 2).

After obtaining the landmark graph, we continue to split the complement graph following the same procedure (line 3). We use a different compatibility function for this step, as later introduced in Equation 5.8. For both steps, we need to enforce the size constraints on the subgraphs. Hence, in the LPA, the aforementioned statistics (line 14 of Algorithm 5.1) include the subgraph sizes. Finally, after splitting the complement graph into subgraphs, we merge each of them with the landmark graph to obtain the final decomposition (line 4).

5.4.2 Landmark-aware Partitioning

Next, we provide details on the first step of our approach, i.e., the instantiation of LPA to construct a connected landmark graph of a specific size that contains important vertices. This requires us to define a compatibility function that takes into account the graph size, its connectedness, and vertex importance. We first discuss how to incorporate the importance of vertices.

Importance-based compatibility. There are several ways to measure the importance of vertices. However, in our setting, we focus on techniques that measure vertex centrality in a distributed manner under the vertex-centric programming model. This limits our options to either the degree centrality or eigenvector centrality, of which PageRank is a particular instance. Note that some popular centrality measures are not applicable in

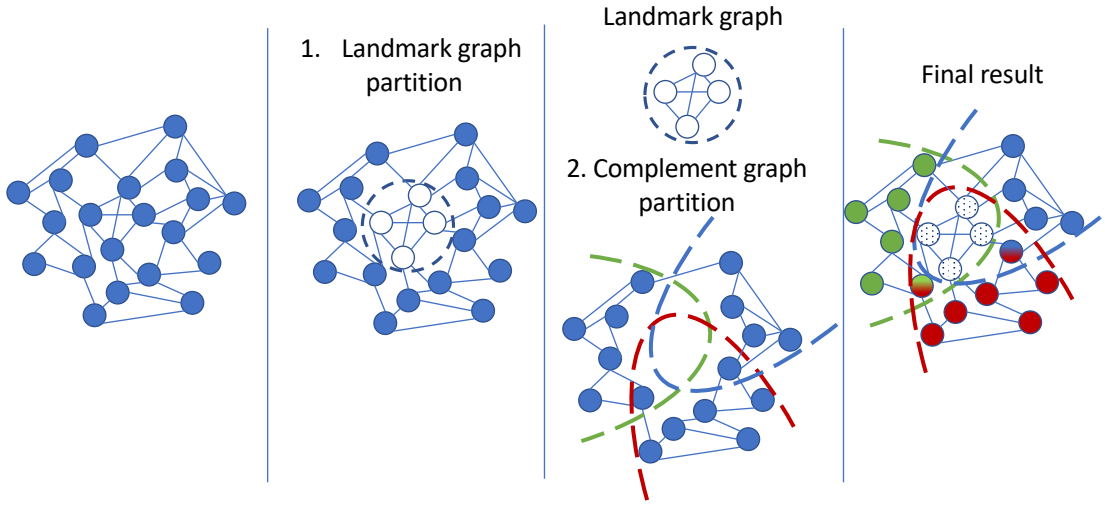


Figure 5.4: Landmark-aware graph decomposition.

our context due to the implied computational overhead. For instance, the betweenness measure requires computing all pairs shortest paths, which is intractable for large graphs. Spark provides several algorithms to compute vertex centrality in an efficient way.

Let δ be the function that measures the importance of each vertex. We define the compatibility between a vertex and a label as:

$$d(v, l) = 1_{l=0} \left(\frac{\delta(v)}{\delta_m} - 1 \right) \tag{5.5}$$

where δ_m is a parameter which signifies the smallest level importance we can tolerate. Here, $1_{l=0}$ ensures that we only focus on landmark vertices, i.e., the landmark graph is assumed to have label 0. The larger a vertex importance $\delta(v)$, the more likely it is compatible with the landmark graph. **For the complement graph, we fix the importance of a node to a constant as all nodes are equally important to the complement graph.**

Size penalty. Strictly enforcing the size constraint on the landmark graph can lead to slow convergence and instability. In practice, by allowing for a small difference [MLLS17], the algorithm can converge faster. To this end, we define a penalty score for each partition as a soft constraint to incorporate in the compatibility function.

Let n_l be the desired size of the partition having label l. We denote $C(l) = cn_l$ to be the maximum capacity of the partition with label l where $c > 1$ is a slack parameter. That is, we allow the partition with label l to exceed its size n_l by a factor of c. Let $c(l)$ be the number of vertices of partition l at an iteration. We define the size-based penalty as:

$$s(l) = \frac{c(l)}{C(l)}. \tag{5.6}$$

When the partition size is close to its capacity, the penalty is high. For two-way partitioning such as considered in this step, we need to define two penalty functions based on the landmark graph size m and the complement graph size $|V| - m$. As this formulation is not limited to 2-way partitioning, we can apply the same strategy to n-way partitioning by integrating several size penalties as above to the compatibility function. This is useful for our next step of partitioning the complement graph.

Finally, we can define the compatibility score by combining Equation 5.5 and Equation 5.6 as follows:

$$comp(v, l) = \sum_{u \in N(v)} 1_{label(u)=l}(\lambda_1 d(v, l) - \lambda_2 s(l)) \tag{5.7}$$

where $N(v)$ is the set of neighbours of vertex v. Here, the summation condition ensures that the landmark graph is connected as the more landmarks a vertex is connected to, the more compatible it is to the landmark graph. Furthermore, λ_1, λ_2 are hyperparameters to balance the size penalty and importance-based compatibility.

5.4.3 Complement Graph Partitioning

After the landmark-aware partitioning, we obtained a landmark graph and the complement graph. As the landmark graph is shared between subgraphs, to obtain the final decomposition, we need to split the complement graph. The aforementioned requirements request that subgraphs shall have particular sizes and the splits of the complement graph shall minimise the number of edges between subgraphs. For the former conditions, we rely on the size penalty as defined already in Equation 5.6. As for the latter, we aim to maximise the edge locality. That is, a vertex is more compatible with a label that is shared the most of its neighbours, which is captured as:

$$a(v, l) = \sum_{u \in N(v)} 1_{label(u),l}$$

Combining the above functions, we obtain the following compatibility function for complement graph partitioning:

$$comp(v, l) = a(v, l) - s(l). \tag{5.8}$$

By applying the above function as part of LPA, we split the complement graph into non-overlapping partitions. The subgraphs can then be combined with the landmark graph to obtain the required subgraphs, as illustrated in Figure 5.4.

5.5 Implementation & Optimisation

Having introduced our framework to scalable and robust graph embedding, we discuss how it is implemented in Spark [ZXW+16], a state-of-the-art engine for distributed data processing. We outline the general system design (Section 5.5.1), before proposing optimizations related to lazy reconciliation (Section 5.5.2) and iterative refinement of the constructed embeddings (Section 5.5.3).

5.5.1 System Design

Data storage. Our implementation handles three types of data, i.e., graphs, vertex features, and vertex embeddings, using a Distributed File System (DFS), the main memory, and the local file system (LFS) of a compute node. Spark manages data using Resilient Distributed Datasets (RDDs), multisets of data elements kept in main memory. Data on stored on a DFS and in an RDD can be accessed by all machines, in contrast to data stored on the LFS. However, accessing data on the LFS does not incur communication cost. Also, has generally bigger capacity than RDDs, which are limited by the size of

the main memory of compute nodes. Note that DataFrames and RDDs can be used interchangeably in Spark, with largely the same performance.

Storing a graph: Initially, the input graph is stored on the DFS to provide access for all compute nodes. The graph is then split randomly where each random subgraph is stored as an RDD, as hinted at already in Figure 5.1. The actual graph decomposition can then operate on these RDDs, while the subgraphs obtained by the decomposition algorithm are stored on the LFS of compute nodes. This leverages data locality as a mapper can directly access the partition without communicating with other nodes. As a storage format for the DFS and the LFS, we rely on edge lists, i.e., files in which each line represents a node and its neighbours.

Storing vertex features and embeddings: Vertex features and embeddings are stored in a similar manner, i.e., each line in a file represents a vertex and its features or its embeddings. As using a GNN as an embedding technique requires both vertex features and the subgraph for training, we co-locate them together at a compute node to reduce communication cost. The vertex embeddings obtained after the map phase and each reduce phase are stored as RDDs, which can be read by subsequent reducers.

Training with GPU. Our implementation supports the use of GPUs to speed up the training process. To this end, elements of an RDD are piped from Spark to an Automatic Differentiation Framework (ADF) that supports graph learning using Pytorch-Geometric [FL19]. As the training is based on the graph and the vertex features, we would need to pass these information, stored in RDDs, to an ADF. However, sending large subgraphs with vertex features would take a lot of time and communication. We therefore opted for storing only the path to a subgraph and its vertex features as an element in an RDD, which are then read directly by the ADF from the DFS. After training, the results, which are the node embeddings, are piped back to Spark and stored as RDDs.

Note that the overhead of combining Spark with an ADF is small. First, all communication happens locally on the same worker instance. Second, there is no inter-process communication between Pytorch and Spark during training. Hence, the overhead is only the initial I/O, where Pytorch reads subgraphs and vertex features, and the final I/O, where Spark reads the output from Pytorch. Since Pytorch reads the subgraphs and features directly from the LFS, the initial I/O overhead is small. Third, there is no preprocessing involved as the subgraphs, vertex features, and embeddings are stored in data formats that are natively supported by Pytorch-Geometric [FL19]. In our experiments, we have observed the total overhead to be around 30s for a graph of 1M nodes.

Fault tolerance. Spark can recover from node failure by reconstructing the input data from the stored data linage. However, if node failure occurs while the data is being processed, all the processed information is lost. While this problem is inherent to the design of Spark, we are able to alleviate this problem by checkpointing the processed data. More precisely, at specific epochs of the training process, we save the embedding model to distributed persistent storage, i.e., the DFS. When a node fails and another node is used to restart computation, it can continue training from the last checkpoint by loading the model from the DFS. Note that we use checkpointing only for the map phase as it incurs the highest processing times and, hence, the largest recovery cost.

5.5.2 Lazy Reconciliation

After learning the mapping matrix $\boldsymbol{W}$ to reconcile the embedding spaces $\boldsymbol{F}_0, \boldsymbol{F}_1$ in a reduction step (see Section 5.3.3), we can reconcile the embedding spaces immediately

based on Equation 5.4. The reconciled embedding space can then be stored for further reduction step, if needed. Yet, each reduction step requires inter-node communication to transfer *all* the vertex embeddings from the nodes at which they are stored to the node evaluating the reduce function. However, we note that to learn the mapping matrix $\boldsymbol{W}$, in Equation 5.3, only the embeddings of the landmarks are needed. Hence, to reduce communication cost, at each reduction step, we only need to fetch the embeddings of the landmarks for reconciliation. Only afterwards, when the reduction step finishes, we apply the stored mapping matrices to reconcile also the non-landmark vertices. This optimisation reduces the communication cost significantly, with only minor overhead caused by the need to store the mapping matrices.

5.5.3 Iterative Refinement

As another optimization, to further improve the embedding quality, our implementation iteratively refines the constructed embeddings. Here, our idea is to use different graph decompositions in each round of the general process (which includes the graph decomposition, the map phase, and the reduce phase) and to incorporate the embeddings learned in one round also in the subsequent round. This way, we ensure a certain diversity of the subgraphs used in the learning process. By combining the results obtained for these diverse subgraphs the final embedding quality is improved.

Diversity per round. We use a different set of landmarks, and hence subgraphs, in each round of the learning process. As a result, different sets of edges are considered, as vertices of the same subgraph in one round, may be part of different subgraphs in another round. Edges that are ignored in one round will be included in other rounds, so that, ideally, all edges are eventually incorporated in the learning process. To operationalize this idea, our landmark-aware decomposition algorithm is modified to assign multiple labels to each vertex, which then induce multiple decomposition results.

Multi-round refinement. Once a round has completed, we store the learned models in a *model bank* (on the DFS). In the next round, the models from this model bank are used as additional input in the learning process. The intuition is that the vertex embeddings constructed in a round should inherit the results from the previous rounds. Also, to maintain continuity, the models obtained in one round should not be vastly different from those of previous rounds. Depending on the graph embedding model, we propose different ways to leverage model bank to improve the embedding quality.

For *shallow graph embedding models*, the vertex embeddings directly correspond to the learned model. In the first round, the model bank is initialised with random vertex embeddings. Then, in each round, we use the vertex embeddings available in the model bank to initialise the embeddings. After each round, the model bank is updated with the newly constructed vertex embeddings. Hence, unlike a traditional setup that would initialise the vertex embeddings randomly in each round, in our proposal, the learning process takes advantage of the results from previous rounds.

For *graph neural networks*, the models stored in the model bank obtained from the previous round are further refined with different subgraphs. More precisely, the model $f^{(k)}(\mathcal{S}_i, \boldsymbol{F}_i)$ obtained at the k-th round by training on subgraph $\mathcal{S}_i$ and vertex features $\boldsymbol{F}_i$ will be trained on another subgraph $\mathcal{S}_j$ with its corresponding vertex features $\boldsymbol{F}_j$: $f^{(k+1)} = f^{(k)}(\mathcal{S}_j, \boldsymbol{F}_j)$. As part of that, we need to make sure, though, that the subgraphs $\mathcal{S}_i, \mathcal{S}_j$ are different, but still share some vertices to ensure model continuity.

5.6 Experiments

This section reports on an experimental evaluation of our approach to scalable robust graph embedding. We first outline the experimental setup (Section 5.6.1). Subsequently, we present results on the effectiveness of our algorithm to graph decomposition (Section 5.6.2), the impact of the reconciliation of embedding spaces (Section 5.6.1), the end-to-end performance of our approach (Section 5.6.4), and the effect of the optimisation based on iterative refinement (Section 5.6.5).

5.6.1 Experimental Setup

Datasets. We rely on five real-world standard benchmark datasets, see Table 5.1. Flickr and Youtube are social networks with a medium number of nodes but a large number of edges and originate from [TL09]. Note that the edge size is an important measure of the graph size as the models are more sensitive to the number of edges. Arxiv and Papers are two bibliographic networks connecting papers, while Products is a network of Amazon products linked by customer purchases. The last two datasets are from the graph benchmark [HFZ+20]; the Arxiv dataset has been introduced in [LKF07]. Also, vertices of Products and Papers are attributed.

Baselines. We compare against DGL [ZMW+20] and PBG [LWS+19], which are two frameworks for distributed graph embedding. While DGL is a general-purpose approach that can be used for any embedding technique, PBG is a scalable shallow embedding model. Both approaches are similar to our approach as they involve partitioning the original graph to scale out the learning process.

Also, we compare our landmark-aware decomposition against graph partitionings obtained with Spinner [MLLS17] and DGL [ZMW+20]. Spinner is a distributed graph partitioning algorithm based on Pregel [MAB+10] which can also be considered as an extension of the LPA. DGL uses a centralised approach that first abstracts the graph for partitioning and then refines the coarse-grained partitions to obtain the result.

Measures. To measure the embedding quality, we train a linear classifier using the embeddings as features. We then evaluate the classifier on a test set and measure its *accuracy* to obtain a metric for the embedding quality. Efficiency is measured by the *communication cost*, the data volume transferred per epoch of the training process, the *training time* per epoch which is the total training time divided by the number of epochs, and the *speedup*, the training time per epoch normalised by the number of compute nodes used.

Configuration. Unless stated otherwise, we use the following hyperparameters. We split the graph into 5 equal partitions with a landmark subgraph of size 0.01%. Due to the high cost of these experiments, we do not perform hyperparameter tuning and use the suggested default values. For node2vec, we use 10 walks per node with a walk length of 10, batch size 2000, embedding size 128, and learning rate 0.01. For GraphSAGE, we use 2 layers of GNN with 10 and 5 neighbours, respectively, a hidden size of 128, a dropout after the first layer with probability 0.5, batch size 2000, embedding size 128, and learning rate 0.03. We train these algorithms for 5 epochs and report the test accuracy of the last model. We use an AWS cluster of p2.xlarge instances (4 VCPU, 61GB RAM, 1 VGPU) except for experiments involving PBG. In these cases, we use m5a.4xlarge instances, as PBG cannot leverage a GPU. For experiments with the Papers dataset, we use an m5a.12xlarge cluster (48 VCPU, 192 GB memory) due to DGL's memory

requirements. We chose these configurations to demonstrate our framework's scalability on *commodity hardware*.

5.6.2 Effectiveness of Graph Decomposition

We first explore the effectiveness of our landmark-aware graph decomposition when splitting the graphs into give overlapping subgraphs. We set the landmark graph size to be 0.1% of the original graph and measure the vertex importance by their degree. The quality of the decomposition is assessed by two metrics: the average degree of a node in the landmark graph and the normalised number of edge cuts. We expect a good partition to have a large average degree and a small number of edge cuts.

Table 5.2 shows that our approach outperforms the baselines significantly on both metrics, over all datasets. For instance, on the Arxiv dataset, our approach returns a landmark graph with an average degree twice that of Spinner and 6 times that of DGL. In addition, our approach has a significantly lower number of edge cuts in comparison with the baselines. We also observe that only our technique, which follows a distributed approach, is able to handle billion-scale graphs, such as the Papers dataset. This experiment confirms that our approach is both scalable and effective in decomposing a graph for the subsequent learning process.

5.6.3 Effects of MapReduce-based Embedding

Effects of Reconciliation. First, we investigate the impact of the reconciliation on the resulting vertex embeddings. We compare the quality of vertex embeddings obtained with and without reconciliation based on accuracy. The results shown in Figure 5.5-A confirm that the reconciled embedding space has higher accuracy than the non-reconciled one across all datasets. For instance, on the Arxiv dataset, the accuracy of the non-reconciled embedding space is only 0.35 while after reconciliation, the accuracy is 0.47.

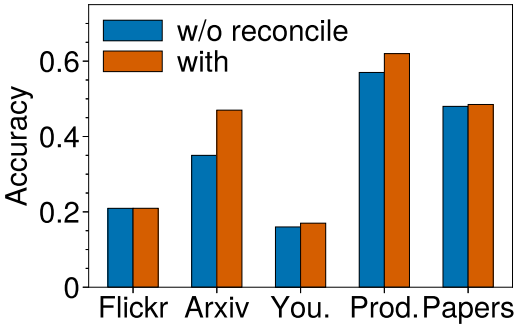


Figure 5.5: Reconciliation.

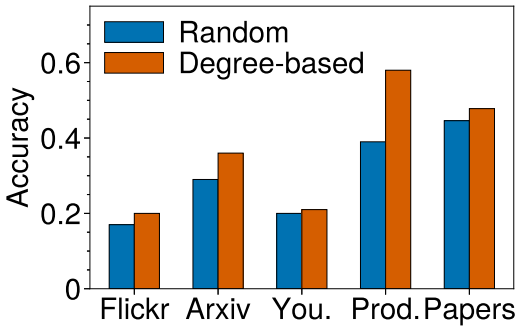


Figure 5.6: Degree vs. Random selection

Effects of landmark selection strategy. Next, we analyse the role of landmark by comparing of two landmark selection strategies: random and our proposed degree-based selection. We also measure the quality by comparing the accuracy of the resulting vertex embeddings. Figure 5.5-B shows that having important landmarks is key in improving the quality of vertex embeddings. The improvement is consistent across all datasets with the highest is 0.2 on Products.

Effects of landmark subgraph size. In this experiment, we vary the size of the landmark subgraph from 32 to 2048 to analyse the effect on the embedding quality. Figure 5.8 shows that the accuracy tends to increase as we increase the subgraph size. However,

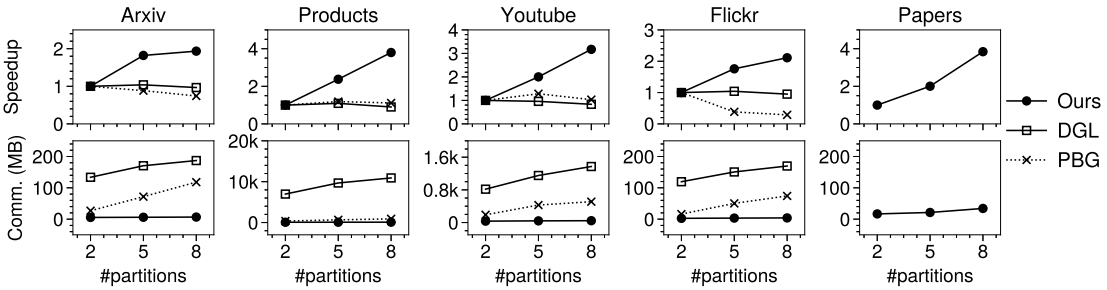


Figure 5.7: Scalability

Table 5.1: Statistics of datasets

	$\|V\|$	$\|E\|$	#features
Flickr	80,513	5,899,882	n/a
Arxiv	169,343	1,166,243	128
Youtube	495,957	1,936,748	n/a
Products	2,449,029	61,859,140	100
Papers	111,059,956	3,231,371,744	128

the rate of increase is small for sizes larger than 128. In general, using more landmarks enables better reconciliation of embedding spaces, which leads to an improvement of the embedding quality. However, increasing the number of landmarks has a diminishing return.

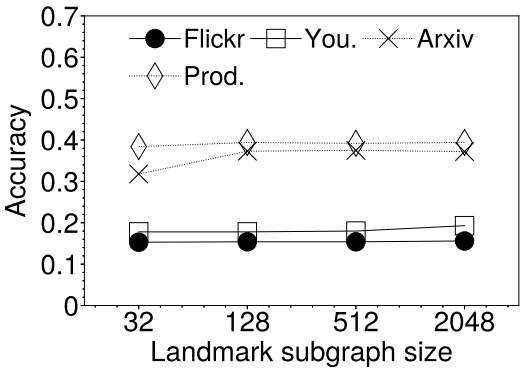


Figure 5.8: Subgraph size

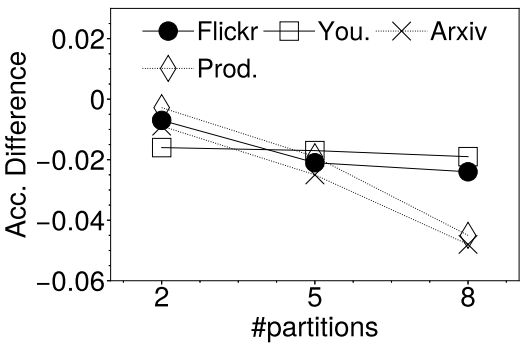


Figure 5.9: Dist. vs. Single.

5.6.4 End-to-end Evaluation

Having evaluated the individual parts of our solution, we turn to its end-to-end performance in comparison to other techniques.

Comparative analysis. We first compare our approach with state-of-the-art distributed graph embedding techniques. Table 5.3 compares the performance of our approach with state-of-the-art techniques. Here, we use GraphSAGE with the aforementioned hyperparameters with m5.4xlarge instances for our approach and DGL. For the Papers dataset, we rely on m5a.12xlarge instances. We do not leverage a GPU to ensure a fair comparison, since PBG cannot exploit it.

Table 5.2: Effectiveness of graph decomposition

	Average degree			Normalised #edge cuts		
	Spinner	DGL	Ours	Spinner	DGL	Ours
Arxiv	674	211	**1214**	3.89	1.08	**0.52**
Products	2323	213	**3331**	35.23	3.77	**1.92**
Youtube	464	11	**7822**	0.46	0.203	**0.09**
Flickr	2383	292	**2487**	1.595	0.95	**0.73**
Papers	906	N/A	**1784**	9.52	N/A	**11.6**

Table 5.3: Comparative analysis

	Time (s)			Accuracy			Communication (GB)		
	PBG	DGL	Ours	PBG	DGL	Ours	PBG	DGL	Ours
Arxiv	76	29	**22**	0.31	0.36	**0.49**	0.04	0.05	**0.006**
Products	649	2081	**361**	0.39	0.55	**0.64**	0.64	4.44	**0.08**
Youtube	312	136	**107**	0.13	**0.21**	0.201	0.6	0.14	**0.04**
Flickr	56	30	**19**	0.15	**0.17**	**0.17**	0.03	0.27	**0.003**
Papers	N/A	3764	**717**	N/A	0.435	**0.478**	N/A	5.324	**0.022**

Our approach leads to better or comparable accuracy of the constructed embeddings, while outperforming the baseline techniques in communication cost and training time. This is expected as the only communication in our framework is from the compute nodes to the DFS during the reduction phase. In contrast, both DGL and PBG require continuous communication between compute nodes during training. This communication overhead also implies higher training times per epoch. On datasets with vertex features such as Papers, our approach is better than DGL in terms of accuracy even when the same graph embedding algorithm is used. As our approach splits the graph and performs graph embedding independently, it can be considered as an ensemble of independent models, which is usually better than a single model. As each model may access only the subgraph, but not the full graph, training relies more on vertex features. Hence, our method performs better on graphs with vertex features. For featureless graphs, such as Flickr or Youtube, our approach achieves comparable accuracy with DGL.

Runtime break-down. We break down the time to learn the vertex embeddings of the Arxiv dataset in Figure 5.12-A. Similar distributions are observed for the other datasets. Here, with more subgraphs, the decomposition takes more time. This can be attributed to harder decomposition constraints, which makes it more difficult to find a good partitioning. On the other hand, the map phase is shorter, as the number of vertices per subgraph, and thus the number of vertex embeddings to be learned, is smaller.

Distributed vs. Single machine. To understand the trade-off between performance and scalability, we measure the difference in accuracy between our distributed version and a single machine setup. We use the node2vec embedding model (10 walks of length 5, batch size 2000, learning rate 0.01). Using this model, only the Flickr, Arxiv, Youtube and Products datasets fit in main memory, with sizes 4.2GB, 3.9GB, 7.1GB, and 24GB, respectively. Figure 5.9 shows that the difference in accuracy is very small, less than 0.05 in absolute terms, across all datasets. While the difference increases with the level of parallelism, even with 8 partitions, the drop in accuracy is only 0.048 in comparison

to a centralized version.

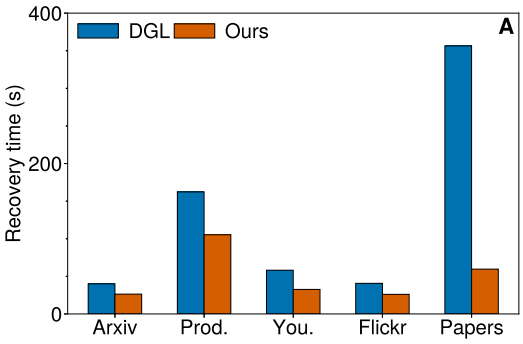


Figure 5.10: Robustness.

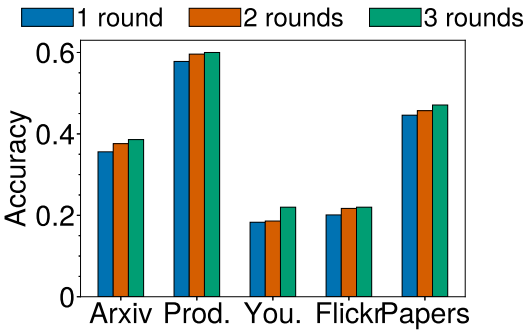


Figure 5.11: Refinement.

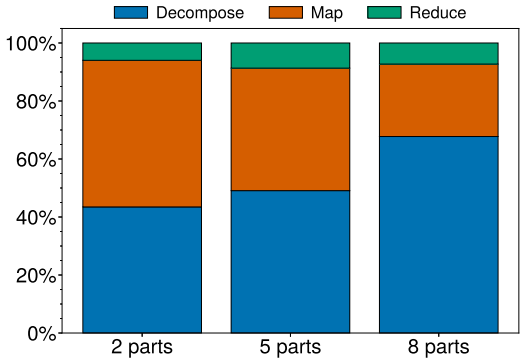


Figure 5.12: Runtime break-down

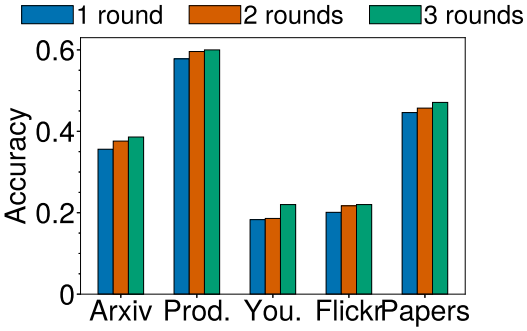


Figure 5.13: Refinement.

Scalability. Next, we analyse the scalability of our approach in terms of the speedup, as we increase the number of partitions (and thus compute nodes) from two to eight. Figure 5.7 shows the speedup as the relative improvement of training time using the setup with two partitions as the reference. The speedup of our approach increases with more partitions and is consistently higher than the one observed for the baseline techniques. For instance, with eight partitions and the Products datase, our speedup is 3.8, whereas DGL and PBG achieve 0.9 and 1.1, respectively. For small datasets, such as Flickr and Arxiv, using more compute nodes actually results in higher training times for PBG, as the increase in I/O and communication overhead dominates the benefits of parallelisation.

Figure 5.7 also shows that our approach maintains stable communication costs as the number of partitions increases. For instance, on the Products dataset, there is an 26% increase in communication, as the number of node increases from two to eight, compared to 56% for DGL and 143% for PBG. For the Papers dataset, we achieve a speedup of 3.9 with 8 partitions and a low communication cost of around 34MB in absolute value. Also, our method transfers only 0.08GB, whereas DGL and PBG require 10.6GB and 0.88GB respectively. We conclude that our approach turns out to show better scalability than the baseline techniques.

Robustness. To assess the robustness to node failure, we measure the recovery cost, i.e., the time required to get back to the same state before the node failure. This time includes the time to load the data to the compute nodes. PBG is excluded in this experiment due to the difficulty in measuring the data loading time, as PBG uses partial

data loading. We simulate node failure by terminating the training process of a node at different training rounds.

Figure 5.12 confirms the robustness of our approach, illustrating that recovery costs are lower than those observed for DGL. The reason being that DGL requires restarting the whole training process, since, even though it supports model checkpointing, all compute nodes need to be restarted before training can continue. Also, note that both PBG and DGL require manual intervention to restart the process, whereas our implementation recovers automatically.

5.6.5 Effects of Iterative Refinement

Finally, we analyse the effects of our optimisation based on iterative refinement on the quality of the embeddings. We increase the number of rounds from one to three for each dataset, expecting an increased embedding quality. This is confirmed in Figure 5.13. However, the improvement is largest initially, reaching a plateau after two rounds. For instance, for the Products dataset, accuracy increases from 0.57 to 0.59, when going from one to two rounds, while another round leads to a minor improvement of 0.01. While we omit the results for training time and communication cost due to space constraint, these measures turned out to increase linearly with the number of rounds.

5.7 Summary

To achieve scalable and robust graph embedding, we proposed a distributed learning process based on the MapReduce model, which can distribute any existing embedding technique. In essence, in the map phase, we learn vertex embeddings for subgraphs, while the reduce phase reconciles the obtained embedding spaces. For the reconciliation to work, we introduced a distributed graph decomposition algorithm based on a vertex-centric computational model. We also presented an implementation of the approach in Spark. Experiments with several real-world datasets confirm the efficiency, scalability, and robustness of our approach.

Chapter 6

Conclusion

6.1 Summary of the Work

With the advance of social media and social networks, data available on the Web have increased not only in quantity in quality. The amount of data is increasing at an unprecedented scale, which makes it extremely difficult for a user to look for a specific information. On the other hand, data are becoming multimodal and heterogenous as users are able to share not only texts but also images, audios, and videos. In addition, data are becoming more connected as relationships between different data elements are also readily available such as in social networks. Traditional information retrieval techniques aim to help users to cope with large amount of data by allowing them to search for documents that satisfy their information need. However, given the heterogeneity of the data, traditional IR systems cannot handle multimodal data well.

Moreover, traditional queries are limited in their capacity to capture users' intention since there are mismatches between supported query terms and available data. Traditional queries are combinations of independent textual query terms while data can be of different modalities or they are connected. It is not possible for a user to describe their information need in other modalities or specify the relationships between query terms even they are available. As a result, there is a need for novel IR system that can support context-rich queries in terms of multimodality and connectivity.

In this thesis, we propose to model data and queries as graphs that can capture data elements/query terms and the relationships between them as well as their modalities. Based thereon, a graph embedding model is applied on the data graph to obtain vertex and subgraph embeddings. This graph embedding model is also applied on the queries to construct the query embeddings. Given the query embeddings and vertex/subgraph embeddings, we follow the vector space retrieval model to answer the queries. In particular, we present graph embedding techniques to support two different settings of the data and queries including *Heterogeneity* and *Connectivity*. We also consider the *Scalability* aspect of embedding techniques as they are required to handle large graphs that are available in practice.

Heterogeneity. In Chapter 3, we propose an approach to construct a multimodal graph/network for any tabular data collection based on Heterogeneous Information Network (HIN). We present methods to construct the vertex embeddings of the HIN that can take into account the modality of the vertices as well as their connections. Our graph embedding model is an adapted instance of the Message-Passing Neural Network (MPNN) for heterogenous graphs. Based on this model, we also propose a method

to create query embeddings for heterogenous queries. Our experiments show that our graph embedding model is more effective than homogeneous ones while our retrieval result based on query embeddings are more relevant than traditional baselines.

Connectivity. In Chapter 4, we propose an approach to construct query embeddings for interconnected queries. We introduce a truncated message passing approach to create embeddings for subgraphs and queries. These embeddings are used to speed up traditional subgraph isomorphism search by reducing matching candidates. We further propose a caching strategy to store past queries and their results to speed up search, which is made possible by the availability of query embeddings. Our evaluation shows that our embedding-based approach can reduce retrieval time significantly. In addition, our cache management strategy enables better cache reuse with higher cache hits.

Scalability. In Chapter 5, we propose a method to scale any graph embedding technique to large graphs. It follows the MapReduce programming paradigm where we divide the graph into small subgraphs, construct graph embedding for each of them before merging them together. We propose a graph decomposition algorithm that can handle large graphs while taking into account hardware availability. We implement our MapReduce algorithm on Spark which makes our algorithm robust to failure and easy to use for end-users. The experiments confirm the scalability of our approach as it outperforms traditional approaches significantly in terms of communication cost, speed up and accuracy.

6.2 Limitations and Future Directions

We realize that the proposed approaches in this dissertation still have several limitations and they can be further improved in a number of ways.

Unstructured Data. In this thesis, we assume that the data is structured as they can be either in tabular format or they can be represented as a graph. To handle unstructured data, we need an information extraction step in which we convert unstructured data to structured format. While information extraction is a well-studied problem, we need an end-to-end approach that can extract data from texts to form a graph directly. Until such approach is available, text-based retrieval is still required to handle textual data which is abundant on the Web.

Dynamic Graphs. The graphs we consider in this thesis are assumed to be static. However, this is not true in practice. For constructed HIN, as data come and go, these changes need to be reflected in the graphs as well. On the other hand, for natural graphs, they are usually dynamic such as social networks where friendships can be made or removed. Graph embedding techniques need to be able to adapt to these changes without retraining the whole graph while handling heterogenous vertices. Our MapReduce-based approach provides a way to design such an approach as vertex embeddings for updated vertices can be learned independently before merging with the whole graph. Other methods [RFC+20, LGP18] are also available but they are limited in scale and do not handle heterogenous graphs.

Featureless Graphs. For graph embedding methods such as graph neural networks to work, a common assumption is that the node have features. In practice, this may not be the case due to difficulty in collection node information such as privacy concerns. While shallow methods such as node2vec [GL16] or DeepWalk [PARS14] can work with featureless graphs, they can not be used on unseen graphs. On the other hand, existing graph neural network methods are ad-hoc as they use structural information such as

the node degree as node features. As such, in this setting, we need a principled and meaningful way to construct the node features.

Search Interface. While our approach can be used to answer context-rich queries, it first needs the users to be able to express their queries in a convenient way. The added expressiveness should not be hindered by the difficulty of specifying a context-rich query. There are two added contextual information in a context-rich query. First, as our queries are multimodal, users should be able to specify different modalities such as images, audios and videos. Second, users can specify the connections between query terms if they are available. The search interface needs to allow users to specify different modalities as well as connections in an efficient way. A complex query specifying process would discourage users in trying context-rich queries even they are better than traditional text-based queries.

Explainability. While graph embedding enables applying vector space retrieval model in graphs, they can not explain why they come up with the retrieval result. In critical applications where explanations are required, further research into explainability of graph embedding models is needed. There are several research directions including what-if analysis, counterfactual analysis. In these methods, we aim to understand the underlying model by changing the input parameters and observing their outcomes. For instance, by adding or removing a node or an edge on a graph, we would understand its importance in the model.

Vector Indexing. To speed up retrieval, an index on the embedding space is required. While there are several methods on indexing vectors [Gut84, DBCVKO08], they do not satisfy all the requirements in our setting. We need an indexing mechanism that can handle high-dimensional vectors, dynamic admission or removal of embeddings, distributed computation of similarity. These requirements in combination require a managed system to go beyond traditional indexing techniques such as a vector database system.

Bibliography

[AAB+19] Aaron Archer, Kevin Aydin, Mohammad Hossein Bateni, Vahab Mirrokni, Aaron Schild, Ray Yang, and Richard Zhuang. Cache-aware load balancing of data center applications. 12(6):709–723, 2019. 52

[Abb07] Noureddine Abbadeni. An approach based on multiple representations and multiple queries for invariant image retrieval. In *VISUAL*, pages 570–579, 2007. 32

[Abb09] Noureddine Abbadeni. Information retrieval from visual databases using multiple representations and multiple queries. In *SAC*, pages 1523–1527, 2009. 30, 32

[AKA17] Tahir Azim, Manos Karpathiotakis, and Anastasia Ailamaki. Recache: Reactive caching for fast analytics over heterogeneous data. *VLDB*, 11(3):324–337, 2017. 66

[ALA16] Mikel Artetxe, Gorka Labaka, and Eneko Agirre. Learning principled bilingual mappings of word embeddings while preserving monolingual invariance. In *EMNLP*, pages 2289–2294, 2016. 83, 84

[APTP03] Khalil Amiri, Sanghyun Park, Renu Tewari, and Sriram Padmanabhan. Dbproxy: A dynamic data cache for web applications. In *ICDE*, pages 821–831. IEEE, 2003. 52

[BBZ17] Petra Budikova, Michal Batko, and Pavel Zezula. Fusion strategies for large-scale multi-modal image retrieval. In *TLDKS*, pages 146–184. 2017. 22, 28

[BCL+16] Fei Bi, Lijun Chang, Xuemin Lin, Lu Qin, and Wenjie Zhang. Efficient subgraph matching by postponing cartesian products. In *SIGMOD*, pages 1199–1214, 2016. 23, 24

[BFG+10] Vincenzo Bonnici, Alfredo Ferro, Rosalba Giugno, Alfredo Pulvirenti, and Dennis Shasha. Enhancing graph database indexing by suffix tree structure. In *IAPR-PRIB*, pages 195–203. Springer, 2010. 24, 55, 66

[BFW+21] Cristian Bodnar, Fabrizio Frasca, Yu Guang Wang, Nina Otter, Guido Montúfar, Pietro Lio, and Michael Bronstein. Weisfeiler and lehman go topological: Message passing simplicial networks. *arXiv preprint arXiv:2103.03212*, 2021. 16

[Bil00] Dania Bilal. Children's use of the yahooligans! web search engine: I. cognitive, physical, and affective behaviors on fact-based search tasks. *Journal of the American Society for information Science*, 51(7):646–665, 2000. 1

[BYRN11] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. Modern information retrieval: The concepts and technology behind search, 2011. 1, 3, 20, 21, 22

[CFSV04] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub) graph isomorphism algorithm for matching large graphs. *TPAMI*, 26(10):1367–1372, 2004. 23, 51, 52, 59, 60, 66

[CKP14] Bokai Cao, Xiangnan Kong, and S Yu Philip. Collective prediction of multiple types of links in heterogeneous information networks. In *ICDM*, pages 50–59, 2014. 35

[CLG18] Eduar Castrillo, Elizabeth León, and Jonatan Gómez. Dynamic structural similarity on graphs. *arXiv preprint arXiv:1805.01419*, 2018. 58

[CLR+18] Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word translation without parallel data. *ICLR*, 2018. 83, 84

[CLS+19] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *KDD*, pages 257–266. ACM, 2019. 17

[CLW+16] Yue Cao, Mingsheng Long, Jianmin Wang, Qiang Yang, and Philip S Yu. Deep visual-semantic hashing for cross-modal retrieval. In *KDD*, pages 1445–1454, 2016. 22

[CLWL17] Yue Cao, Mingsheng Long, Jianmin Wang, and Shichen Liu. Collective deep quantization for efficient cross-modal retrieval. In *AAAI*, pages 3974–3980, 2017. 22, 28

[CLX15] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *CIKM*, pages 891–900, 2015. 13

[DBCVKO08] Mark De Berg, Otfried Cheong, Marc Van Kreveld, and Mark Overmars. Orthogonal range searching: Querying a database. *Computational Geometry*, pages 95–120, 2008. 52, 59, 101

[DBVB17] Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson. Fma: A dataset for music analysis. In *ISMIR*, pages 1–8, 2017. 42

[DCS17] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*, pages 135–144, 2017. 41

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. 81, 82

[DHD+19] Chi Thang Duong, Thanh Dat Hoang, Ha The Hien Dang, Quoc Viet Hung Nguyen, and Karl Aberer. On node features for graph neural networks. *arXiv preprint arXiv:1911.08795*, 2019. 51

[DSV+18] Matthias Dorfer, Jan Schlüter, Andreu Vall, Filip Korzeniowski, and Gerhard Widmer. End-to-end cross-modality retrieval with cca projections and pairwise ranking loss. *IJMIR*, 7(2):117–128, 2018. 22, 28, 46

[DYH+20] Chi Thang Duong, Hongzhi Yin, Dung Hoang, Minn Hung Nguyen, Matthias Weidlich, Quoc Viet Hung Nguyen, and Karl Aberer. Graph embeddings for one-pass processing of heterogeneous queries. In *ICDE*, pages 1994–1997, 2020. 14, 51

[EHSM08] Hugo Jair Escalante, Carlos A Hérnadez, Luis Enrique Sucar, and Manuel Montes. Late fusion of heterogeneous methods for multimedia image retrieval. In *ICMR*, pages 172–179, 2008. 22, 28

[FF15] Andreas Emil Feldmann and Luca Foschini. Balanced partitions of trees and applications. *Algorithmica*, 71(2):354–376, 2015. 88

[FL19] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv e-prints*, pages arXiv–1903, 2019. 91

[FZMK20] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of The Web Conference 2020*, pages 2331–2341, 2020. 16

[GL16] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pages 855–864, 2016. 13, 100

[GMA+08] Charles Garrod, Amit Manjhi, Anastasia Ailamaki, Bruce Maggs, Todd Mowry, Christopher Olston, and Anthony Tomasic. Scalable query result caching for web applications. *VLDB*, 1(1):550–561, 2008. 52

[GSR+17] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, pages 1263–1272, 2017. 13, 14

[Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984. 52, 59, 101

[GY13] Shahram Ghandeharizadeh and Jason Yap. Cache augmented database management systems. In *DBSocial*, pages 31–36, 2013. 52

[HBZ+18] Will Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. Embedding logical queries on knowledge graphs. In *NIPS*, pages 2026–2037, 2018. 22, 39

[HDQ+19] Thanh Trung Huynh, Chi Thang Duong, Thang Huynh Quyet, Quoc Viet Hung Nguyen, Abdul Sattar, et al. Network alignment by representation learning on structure and attribute. In *PRICAI*, pages 698–711, 2019. 51

[HDWS20] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, pages 2704–2710, 2020. 16

[HFZ+20] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. In *NeurIPS*, 2020. 93

[HLJ06] Xiuzhen Huang, Jing Lai, and Steven F Jennings. Maximum common subgraph: some upper bound and lower bound results. *BMC bioinformatics*, 7(4):S6, 2006. 61

[HLL13] Wook-Shin Han, Jinsoo Lee, and Jeong-Hoon Lee. Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *SIGMOD*, pages 337–348, 2013. 23, 24, 51, 53, 66

[HYL17a] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1024–1034, 2017. 15, 42, 44, 52, 79, 83

[HYL17b] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *DEBU*, 2017. 12, 79, 81, 83

[JFLW17] Li Jin, Ling Feng, Gangli Liu, and Chaokun Wang. Personal web revisitation by context and content keywords with relevance feedback. *TKDE*, 29(7):1508–1521, 2017. 30

[JK02] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *TOIS*, 20(4):422–446, 2002. 32

[JSN+17] Peiguang Jing, Yuting Su, Liqiang Nie, Xu Bai, Jing Liu, and Meng Wang. Low-rank multi-view embedding learning for micro-video popularity prediction. *TKDE*, 30(8):1519–1532, 2017. 30, 32

[JSP05] Bernard J Jansen, Amanda Spink, and Jan Pedersen. A temporal comparison of altavista web searching. *Journal of the American Society for Information Science and Technology*, 56(6):559–570, 2005. 1

[KK95] George Karypis and Vipin Kumar. Metis–unstructured graph partitioning and sparse matrix ordering system, version 2.0. 1995. 85

[KK98] George Karypis and Vipin Kumar. A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. *University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN*, 1998. 87

[KKM11] Karsten Klein, Nils Kriege, and Petra Mutzel. Ct-index: Fingerprint-based graph indexing combining cycles and trees. In *ICDE*, pages 1115–1126. IEEE, 2011. 24, 55, 66

[KNT15] Foteini Katsarou, Nikos Ntarmos, and Peter Triantafillou. Performance and scalability of indexed subgraph query processing methods. *VLDB*, 8(12):1566–1577, 2015. 24, 52

[KS97] Norio Katayama and Shin'ichi Satoh. The sr-tree: An index structure for high-dimensional nearest neighbor queries. *ACM Sigmod Record*, 26(2):369–380, 1997. 59

[KSH+18] Kyoungmin Kim, In Seo, Wook-Shin Han, Jeong-Hoon Lee, Sungpack Hong, Hassan Chafi, Hyungyu Shin, and Geonhwa Jeong. Turboflux: A fast continuous subgraph matching system for streaming graph data. In *SIGMOD*, pages 411–426, 2018. 23, 25

[KW17] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017. 13, 66, 82

[LGP18] Jiongqian Liang, Saket Gurukar, and Srinivasan Parthasarathy. Mile: A multi-level framework for scalable graph embedding. *arXiv preprint arXiv:1802.09612*, 2018. 17, 18, 100

[LGZ04] P-A Larson, Jonathan Goldstein, and Jingren Zhou. Mtcache: Transparent mid-tier database caching in sql server. In *ICDE*, pages 177–188. IEEE, 2004. 52

[LGZ+18] Lailong Luo, Deke Guo, Xiang Zhao, Jie Wu, Ori Rottenstreich, and Xueshan Luo. Near-accurate multiset reconciliation. *TKDE*, 31(5):952–964, 2018. 30

[LHKL12] Jinsoo Lee, Wook-Shin Han, Romans Kasperovics, and Jeong-Hoon Lee. An in-depth comparison of subgraph isomorphism algorithms in graph databases. *VLDB*, 6(2):133–144, 2012. 66, 67

[LKDL12] Wangchao Le, Anastasios Kementsietsidis, Songyun Duan, and Feifei Li. Scalable multi-query optimization for sparql. In *ICDE*, pages 666–677. IEEE, 2012. 24

[LKF07] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2–es, 2007. 93

[LNC+18] Di Lu, Leonardo Neves, Vitor Carvalho, Ning Zhang, and Heng Ji. Visual attention model for name tagging in multimodal social media. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1990–1999, 2018. 2

[LPW14] Chen Luo, Wei Pang, and Zhe Wang. Semi-supervised clustering on heterogeneous information networks. In *PAKDD*, pages 548–559, 2014. 35

[LRU14] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2014. 46

[LWS+19] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. Pytorch-biggraph: A large-scale graph embedding system. *arXiv preprint arXiv:1903.12287*, 2019. xiii, 18, 79, 80, 93

[LWSP14] Mingsheng Long, Jianmin Wang, Jiaguang Sun, and S Yu Philip. Domain invariant transfer kernel learning. *TKDE*, 27(6):1519–1532, 2014. 32

[LZ19] Yongjiang Liang and Peixiang Zhao. Workload-aware subgraph query caching and processing in large graphs. In *ICDE*, pages 1754–1757. IEEE, 2019. 23, 24, 61

[MAB+10] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD*, pages 135–146. ACM, 2010. 81, 86, 87, 93

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. 13

[ML09] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340):2, 2009. 59

[MLLS17] Claudio Martella, Dionysios Logothetis, Andreas Loukas, and Georgos Siganos. Spinner: Scalable graph partitioning in the cloud. In *ICDE*, pages 1083–1094. Ieee, 2017. 82, 86, 87, 89, 93

[MlP14] Dániel Marx and Micha l Pilipczuk. Everything you always wanted to know about the parameterized complexity of subgraph isomorphism. 2014. 52

[MMM15] André Mourão, Flávio Martins, and João Magalhães. Multimodal medical information retrieval with unsupervised rank fusion. *CMIG*, 39:35–45, 2015. 22, 28, 47

[MRF+19] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *AAAI*, volume 33, pages 4602–4609, 2019. 16, 52, 55, 56

[MSL+16] Tong Man, Huawei Shen, Shenghua Liu, Xiaolong Jin, and Xueqi Cheng. Predict anchor links across social networks via an embedding approach. In *IJCAI*, pages 1823–1829, 2016. 83

[MTX13] Lei Meng, Ah-Hwee Tan, and Dong Xu. Semi-supervised heterogeneous fusion for multimedia data co-clustering. *TKDE*, 26(9):2293–2306, 2013. 30

[NCC+16] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *arXiv preprint arXiv:1606.08928*, 2016. 52

[OAU11] Rifat Ozcan, Ismail Sengor Altingovde, and Özgür Ulusoy. Cost-aware strategies for query result caching in web search engines. *TWEB*, 5(2):1–25, 2011. 52

[OCP+16] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. In *KDD*, pages 1105–1114, 2016. 13

[PARS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*, pages 701–710, 2014. 13, 52, 79, 100

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999. 2

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, pages 8024–8035, 2019. 82

[PH97] Annabel Pollock and Andrew Hockley. What's wrong with internet searching. *D-lib magazine*, 3(3):1–5, 1997. 1

[PIC19] Georgios Paschos, George Iosifidis, and Giuseppe Caire. Cache optimization models and algorithms. *arXiv preprint arXiv:1912.12339*, 2019. 63

[RFC+20] Emanuele Rossi, Fabrizio Frasca, Ben Chamberlain, Davide Eynard, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020. 17, 18, 100

[RRK+90] Dennis W Ruck, Steven K Rogers, Matthew Kabrisky, Mark E Oxley, and Bruce W Suter. The multilayer perceptron as an approximation to a bayes optimal discriminant function. *IEEE Transactions on Neural Networks*, 1(4):296–298, 1990. 83

[RW15] Xuguang Ren and Junhu Wang. Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs. *VLDB*, 8(5):617–628, 2015. 23, 24

[RW16] Xuguang Ren and Junhu Wang. Multi-query optimization for subgraph isomorphism search. *VLDB*, 10(3):121–132, 2016. 23, 24, 51, 53, 57, 66, 67

[SAH08] Chanop Silpa-Anan and Richard Hartley. Optimised kd-trees for fast image descriptor matching. In *CVPR*, pages 1–8. IEEE, 2008. 59

[SH13] Yizhou Sun and Jiawei Han. Mining heterogeneous information networks: a structural analysis approach. *KDD Expl. News.*, 14(2):20–28, 2013. 29, 35

[SHY+11] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S Yu, and Tianyi Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. In *VLDB*, pages 992–1003, 2011. 16, 33, 35, 41

[SKB+18] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018. 16

[SKP15] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, pages 815–823, 2015. 41, 42

[SLZ+16] Chuan Shi, Yitong Li, Jiawei Zhang, Yizhou Sun, and S Yu Philip. A survey of heterogeneous information network analysis. *TKDE*, 29(1):17–37, 2016. 29, 33, 35

[SMR08] Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. *Introduction to information retrieval*, volume 39. Cambridge University Press Cambridge, 2008. 1, 3, 20, 21

[soc] 2

[SRNC+00] Ilmerio Silva, Berthier Ribeiro-Neto, Pavel Calado, Edleno Moura, and Nivio Ziviani. Link-based and content-based evidential information in a belief network model. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 96–103, 2000. 2

[SSH14] Bahjat Safadi, Mathilde Sahuguet, and Benoit Huet. When textual and visual information join forces for multimedia retrieval. In *ICMR*, pages 265–272, 2014. 22, 28

[SY17] Saeid Sattari and Adnan Yazici. Multimedia information retrieval using fuzzy cluster-based model learning. In *FUZZ-IEEE*, pages 1–6, 2017. 22, 28

[SY18] Saeid Sattari and Adnan Yazici. Multimodal query-level fusion for efficient multimedia information retrieval. *IJIS*, 33(10):2019–2037, 2018. 28, 32

[SYH09] Yizhou Sun, Yintao Yu, and Jiawei Han. Ranking-based clustering of heterogeneous information networks with star network schema. In *KDD*, pages 797–806, 2009. 35

[SZK+12] Chuan Shi, Chong Zhou, Xiangnan Kong, Philip S Yu, Gang Liu, and Bai Wang. Heterecom: a semantic-based recommendation system in heterogeneous networks. In *KDD*, pages 1552–1555, 2012. 35

[SZLY08] Haichuan Shang, Ying Zhang, Xuemin Lin, and Jeffrey Xu Yu. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. *VLDB*, 1(1):364–375, 2008. 59, 60

[TL09] Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *KDD*, pages 817–826, 2009. 93

[TTVV+20] Huynh Thanh Trung, Nguyen Thanh Toan, Tong Van Vinh, Hoang Thanh Dat, Duong Chi Thang, Nguyen Quoc Viet Hung, and Abdul Sattar. A comparative study on network alignment techniques. *ESWA*, 140:112883, 2020. 51

[TVVT+20] Huynh Thanh Trung, Tong Van Vinh, Nguyen Thanh Tam, Hongzhi Yin, Matthias Weidlich, and Nguyen Quoc Viet Hung. Adaptive network

alignment with unsupervised and multi-order convolutional networks. In *ICDE*, pages 85–96, 2020. 51

[Ull76] Julian R Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM (JACM)*, 23(1):31–42, 1976. 51, 59

[VAFK17] Damir Vandic, Steven Aanen, Flavius Frasincar, and Uzay Kaymak. Dynamic facet ordering for faceted product search engines. *TKDE*, 29(5):1004–1016, 2017. 28

[VCC+17] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 16

[VFH+18] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *arXiv preprint arXiv:1809.10341*, 2018. 16, 66

[WJS+19] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, pages 2022–2032, 2019. 16

[WJZ+19] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. Simplifying graph convolutional networks. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 6861–6871. PMLR, 2019. 17, 18

[WLLZ18] Zhichun Wang, Qingsong Lv, Xiaohan Lan, and Yu Zhang. Cross-lingual knowledge graph alignment via graph convolutional networks. In *EMNLP*, pages 349–357, 2018. 32

[WOY+14] Wei Wang, Beng Chin Ooi, Xiaoyan Yang, Dongxiang Zhang, and Yueting Zhuang. Effective multi-modal retrieval based on stacked auto-encoders. In *VLDB*, pages 649–660, 2014. 22, 28, 46

[WYO+16] Wei Wang, Xiaoyan Yang, Beng Chin Ooi, Dongxiang Zhang, and Yueting Zhuang. Effective deep learning-based multi-modal retrieval. *VLDBJ*, 25(1):79–101, 2016. 22, 28

[XHLJ18] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018. 17, 37

[XHS+18] Huijuan Xu, Kun He, Leonid Sigal, Stan Sclaroff, and Kate Saenko. Text-to-clip video retrieval with early fusion and re-captioning. *arXiv:1804.05113*, 2018. 22, 28, 47

[YKY+18] Adnan Yazici, Murat Koyuncu, Turgay Yilmaz, Saeid Sattari, Mustafa Sert, and Elvan Gulen. An intelligent multimedia information system for multimodal content extraction and querying. *MTA*, 77(2):2225–2260, 2018. 22

[You08] Neal E Young. Online paging and caching. 2008. 53, 62, 63, 66

[YYM+18] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *arXiv preprint arXiv:1806.08804*, 2018. 17

[ZCH+20] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020. 12, 13, 16

[ZLY09] Shijie Zhang, Shirong Li, and Jiong Yang. Gaddi: distance index based subgraph matching in biological networks. In *EDBT*, pages 192–203, 2009. 51, 59, 60

[ZMW+20] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. Distdgl: Distributed graph neural network training for billion-scale graphs. *arXiv preprint arXiv:2010.05337*, 2020. 18, 80, 93

[ZQYC19] Yuanyuan Zhu, Lu Qin, Jeffrey Xu Yu, and Hong Cheng. Answering top-k graph similarity queries in graph databases. *TKDE*, 2019. 23, 25

[ZSH+19] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. Heterogeneous graph neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 793–803, 2019. 16

[ZXW+16] Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache spark: a unified engine for big data processing. *Commun. ACM*, 59(11):56–65, 2016. 80, 81, 90

[ZYP07] Peixiang Zhao, Jeffrey Xu Yu, and S Yu Philip. Graph indexing: Tree+ delta¿= graph. In *VLDB*, volume 7, pages 938–949, 2007. 23, 25

Chapter 7

Curriculum vitae

THANG DUONG

Chemin du Mottey 35, 1020 Renens, Switzerland
+41 78 923 45 24 ⋄ duongchithang@gmail.com

EDUCATION

PhD of Computer Science September 2016 - now
École polytechnique fédérale de Lausanne (EPFL), Lausanne, Switzerland
Supervisor: Prof. Karl Aberer

- Thesis: Graph Embeddings for Retrieval: Techniques and Applications

Master of Computer Science September 2012 - February 2015
École polytechnique fédérale de Lausanne (EPFL), Lausanne, Switzerland

- Thesis: Argument Mining via Crowdsourcing (mark 5.5/6)
- GPA: 5.34/6

Bachelor of Computer Science September 2007 - April 2012
Ho Chi Minh University of Technology (HCMUT), Vietnam

- Thesis: Probabilisitc Schema Covering (mark 9.7/10)
- Rank: 2/28 (in class), 4/268 (in faculty)
- GPA: 8.79/10.00
- Graduated with **Honor**

AWARDS

EPFL-EDIC PhD Fellowship, EPFL 2016 - 2017
EPFL Excellence Scholarship, EPFL 2012 - 2015
University Academic Merit Scholarship, HCMUT 2007-2012
Kitagawa Scholarship, Awarded for excellent academic results 2010
University Merit Award, Awarded for high-ranking students Fall 2009
University Entrance Examincation - top 2 out of 1.8 million students[1]

- **First runner-up**, HCMUT - Mark: 29.5/30 2007
- **First runner-up**, Medical University - Mark: 29.5/30 2007

StackOverflow reputation: **top 3%** of users

PUBLICATION

Duong, C. T., Dung Hoang, Hongzhi Yin, Matthias Weidlich, Quoc Viet Hung Nguyen, Karl Aberer, "Scalable Robust Graph Embedding with Spark." In submission **VLDB** (2022)

Duong, C. T., Dung Hoang, Hongzhi Yin, Matthias Weidlich, Quoc Viet Hung Nguyen, Karl Aberer, "Efficient Streaming Subgraph Isomorphism with Graph Neural Networks." In: **VLDB** (2021) pp. 730-742

Duong, C. T., Tam Thanh Nguyen, Hongzhi Yin, Matthias Weidlich, Son Mai, Karl Aberer, Quoc Viet Hung Nguyen, "Efficient and Effective Multi-Modal Queries through Heterogeneous Network Embedding." In: **TKDE** (2021)

Duong, C. T., Hongzhi Yin, Dung Hoang, Minh Hung Nguyen, Matthias Weidlich, Quoc Viet Hung Nguyen, Karl Aberer, "Graph Embeddings for One-pass Processing of Heterogeneous Queries." In: **ICDE** (2020), pp.1994-1997

Nguyen, Q. V. H., **Duong, C. T.**, Nguyen, T. T., Weidlich, M., Aberer, K., Yin, H., Zhou, X. "Argument discovery via crowdsourcing." In: **VLDBJ** (2017): 1-25.

[1] https://vnexpress.net/nua-trieu-thi-sinh-da-truot-dai-hoc-2088332.html

Hung, N. Q. V., **Duong. C. T.**, Tam, N. T., Weidlich, M., Aberer, K., Yin, H., Zhou, X. "Answer validation for generic crowdsourcing tasks with minimal efforts." In: **VLDBJ** (2017): 1-26.

Duong, C. T., Nguyen, Q. V. H., Wang, S., Stantic, B. . "Provenance-Based Rumor Detection". In: **Australasian Database Conference** (2017): 125-137

Nguyen, T. T., **Duong, C. T.**, Weidlich, M., Yin, H., Nguyen, Q. V. H. "Retaining data from streams of social platforms with minimal regret." In: **IJCAI** 2017.

N. Q. V. Hung, **Duong C. T.**, M. Weidlich, and K. Aberer. "Minimizing Efforts in Validating Crowd Answers". In: **SIGMOD** ACM. 2015, pp. 999-1014.

N. Quoc Viet Hung, **Duong C. T.**, M. Weidlich, and K. Aberer. "ERICA: Expert Guidance in Validating Crowd Answers". In: **SIGIR** ACM. 2015, pp. 1037-1038.

Duong C. T., N. T. Tam, N. Q. V. Hung, and K. Aberer. "An Evaluation of Diversification Techniques". In: **Database and Expert Systems Applications. (DEXA)** Springer. 2015, pp. 215-231.

N. Q. V. Hung, S. Sathe, **Duong C. T.**, and K. Aberer. "Towards enabling probabilistic databases for participatory sensing". In: **CollaborateCom** 2014, pp. 114-123. *Best runner-up paper*

WORK EXPERIENCE

Graph Management Researcher Sep 2017 - Now
LSIR, EPFL, Switzerland

- Research on efficient retrieval of nodes, subgraphs and patterns on graphs based on graph embeddings.
- Tackle scalablity problem of graph embedding methods.

Technology Monitoring and Management Researcher Mar 2019 - Now
Cyber Defense Campus, Armasuisse, Switzerland

- Developed an end-to-end framework for technology extraction, classification and retrieval.
- Collected data from all Swiss companies to identify their potential technology offerings.

Mobile Application Intern Mar 2014 - Aug 2014
Swissquote Ltd., Gland, Switzerland

- Developed an Android application FXBook for foreign exchange trading

Founder - Startup Apr 2012 - Sep 2012
Crowd.vn, Ho Chi Minh City, Vietnam

- Developed a microtask/crowdsourcing platform using Ruby on Rails

Data Integration & Visualization Intern Aug 2011 - Dec 2011
LSIR lab, EPFL - NisB project - EU 7th Framework

- Focused on probabilistic schema covering and the network of schemas.
- Implemented a visualization tool for schema covering

Security Monitoring Researcher 2009 - 2010
ASIS lab, HCMUT

- Developed techniques for security visualization on web servers.
- Implemented an application to monitor/protect web servers from malicious attacks.

Microsoft Student Partner 2009 - 2010
Microsoft - HCMUT, Vietnam

- Imparted Microsoft technology to HCMUT students
- Organize Microsoft activities (talks, events,...) in HCMUT