15

Advanced Applications in Graph Neural Networks

15.1 Introduction

In PART THREE, we have introduced representative applications of graph neural networks, including natural language proceeding, computer vision, data mining, and biochemistry and healthcare. Graph neural networks have been employed to facilitate more advanced applications as graphs are natural representations of data produced by many real-world applications and systems. Numerous combinational optimization problems on graphs such as minimum vertex cover and the traveling salesman problem are NP-Hard. Graph neural networks have been used to learn the heuristics for these NP-hard problems. Graphs can denote source code in programs from many perspectives, such as data and control flow. Thus, graph neural networks can be naturally leveraged to learn representations for source code to automate various tasks such as variable misuse detection and software vulnerability detection. For dynamical systems in Physics, the objects and their relations can often be denoted as graphs. Graph neural networks have been adopted to infer future states of dynamic systems. This chapter discusses these advanced and sophisticated applications and then introduces how graph neural networks can be applied.

15.2 Combinatorial Optimization on Graphs

Many combinational optimization problems on graphs such as minimum vertex cover (MVC) and travelling salesman problem (TSP) are NP-Hard. In other words, no polynomial-time solutions are available for them (under the condition P≠ NP). These problems are hence usually tackled by approximation algorithms or heuristics. Designing good heuristics is usually a challenging and tedious process, which requires significant problem-specific knowledge

and trial-and-error. Hence, it is desired to learn heuristics automatically. Graph neural networks have been utilized to learn these heuristics from given samples and then try to find solutions for unseen tasks. Next, we first describe some combinatorial optimization problems on graphs and then briefly introduce how graph neural networks can be leveraged to facilitate these tasks.

- **Minimum Vertex Cover (MVC)**. Given a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, a vertex cover $\mathcal{S} \subset \mathcal{V}$ is a subset of vertices that includes at least one endpoint for every edge in $\mathcal{E}$. The problem of minimum vertex cover (MVC) is to find a vertex cover that has the smallest amount of nodes.
- **Maximum Cut (MAXCUT)**. Given a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, a cut $\mathcal{C} = \{\mathcal{S}, \mathcal{V}/\mathcal{S}\}$ is a partition of $\mathcal{V}$ into two disjoint subsets $\mathcal{S}$ and $\mathcal{V}/\mathcal{S}$. Its corresponding cut-set is the subset of edges $\mathcal{E}_c \in \mathcal{E}$ with one endpoint in $\mathcal{S}$ and the other endpoint in $\mathcal{V}/\mathcal{S}$. The problem of maximum cut is to find such a cut $\mathcal{C}$, where the weights of its cut-set $\mathcal{E}_c$ denoted as $\sum_{(u,v)\in\mathcal{E}_c} w(u, v)$ are maximized, where $w(u, v)$ denotes the weight of edge (u, v).
- **Traveling Salesman Problem (TSP)**. Given a collection of cities connected through routes, the traveling salesman problem is to find the shortest route that visits every city once and comes back to the starting city. It can be modeled as a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where nodes are the cities and edges are the routes connecting them. The distance between cities can be modeled as weights on the edges.
- **Maximal Independent Set (MIS)**. Given a graph $\mathcal{G}$, an independent set is a subset of vertices $\mathcal{S} \subset \mathcal{V}$ where no pair of nodes is connected by an edge. The problem of maximal independent set is to find the independent set with the largest number of nodes.

Some of these problems can often be modeled as a node/edge annotation problem, where the goal is to tell whether a node/edge is in the solution or not. For example, the problem of minimum vertex cover (MVC) can be modeled as a node annotation problem (or a node classification problem), where the node in the solution is annotated as 1. In contrast, those not in the solution are annotated as 0. Similarly, the travel salesman problem can be modeled as a problem of node selection or edge annotation. Graph neural networks are suitable to tackle these problems, given rich training samples. However, directly tackling these problems as purely node/edge annotation tasks may lead to invalid solutions. For example, in the task of maximal independent set problem, two connected nodes might be annotated as 1 during the inference. Hence, some search heuristics are usually utilized with graph neural networks to find valid solutions.

In (Khalil et al., 2017), these problems are modeled as a sequential node selection task, which is tackled with reinforcement learning. The graph neural network model is utilized to model the state representations for the deep reinforcement learning framework. A solution is constructed by sequentially adding nodes to a partial solution. These nodes are sequentially selected greedily by maximizing some evaluation functions in the reinforcement learning framework, which is used to measure the quality of the solution (or partial solution). After the nodes are chosen, a helper function is employed to organize them into a valid solution of given tasks. For example, for the MAXCUT task, given the selected set $\mathcal{S}$, its complimentary set $\mathcal{V}/\mathcal{S}$ is found, and the maximum cut-set includes all edges with one endpoint in one set and the other endpoint in the other set.

Instead of sequentially choosing nodes with a reinforcement learning framework, the tasks are modeled as a node annotation task in (Li et al., 2018e). During the training stage, nodes in each training sample are annotated with 1 or 0, where 1 indicates that nodes are in the set of solutions. After training, given a new sample, the graph neural network model can output a probability score for each node, indicating how likely it should be included in the solution. Then a greedy search algorithm is proposed based on these probability scores to build valid solutions recursively. For example, for the MIS task, nodes are first sorted by the probability scores in descending order. Then we iterate all nodes in this order and label each node with 1 and its neighbors with 0. The process stops when we encounter the first node labeled with 0. Next, we remove all labeled nodes (labeled with 1 or 0) and use the remaining nodes to build an induced subgraph. We repeat the process on the induced subgraph. The entire process is terminated until all nodes in the graph are labeled.

In (Joshi et al., 2019), the graph neural network model is trained to annotate edges to solve the travel salesman problem. During training, edges in each training sample are annotated with 1 or 0, indicating whether the edge is in the solution or not. Then, during the inference stage, the model can predict probability scores for edges in the graph. These scores are combined with the beam search to find valid solutions for TSP.

15.3 Learning Program Representations

Machine learning techniques have been adopted to automate various tasks on source code, such as variable misuse detection and software vulnerability detection. A natural way to denote the source code is to treat it as "articles" in a specific language. Then we can transfer the techniques designed for NLP

to deal with source code. However, representing source code as a sequence of tokens usually fails to capture the syntactic and semantic relations in the code. Recently, there are increasing attempts to represent code as graphs and graph neural networks have been employed to learn representations to facilitate down-stream tasks. Next, we first briefly introduce how source code can be denoted as graphs. Then, we describe some downstream tasks and how graph neural networks can be employed to handle these tasks.

There are various ways to construct graphs from source code. Representative ones are listed below:

- **Abstract Syntax Tree (AST)**. One common graph representation for the program is the abstract syntax tree (AST). It encodes the abstract syntactic structure of the source code. Usually, AST is used by code parsers to understand the code structures and find syntactic errors. Nodes in AST consist of syntax nodes (corresponding to non-terminals in the programming language's grammar) and syntax tokens (corresponding to terminals). Directed edges are adopted to represent the child-parent relations.
- **Control Flow Graph (CFG)**. The control flow graph describes all potential paths to be traversed in a program during the execution. CFGs consist of statements and conditions as nodes. The conditional statements such as *if* and *switch* are the key nodes of forming different paths. The edges in CFGs indicate the transfer of the control between statements.
- **Data Flow Graph (DFG)**. A data flow graph describes how variables are used through the program. It has the variables as its nodes, and the edges represent any access or modifications to these variables.
- **Natural Code Sequence (NCS)**. The NCS is a sequence of the source code, where the edges connect neighboring code tokens according to the order in the source code.

These graphs can further be combined to form a more comprehensive graph, which encodes both syntactic and semantic information about the program. Different tasks can be performed with graph neural networks based on the built graphs for a given program. Graph neural networks are usually utilized to learn node representations or graph representations, which are then employed to perform these tasks. There are tasks focusing on nodes in the graphs such as variable misuse detection in a program (Allamanis et al., 2017) and also tasks focusing on the entire program graph such as software vulnerability detection (Zhou et al., 2019).

15.4 Reasoning Interacting Dynamical Systems in Physics

Interacting systems are ubiquitous in nature, and dynamical systems in physics are one of the representatives. Inferring the future states or underlying properties of the dynamical system is challenging due to the complicated interactions between objects in the dynamical system. The objects and their relations in dynamical systems can be typically denoted as a graph, where the objects are the nodes, and the relations between them can be captured as edges. We introduce some dynamical systems in physics and then briefly describe how graph neural networks can be used to infer these dynamical systems.

- **N-body**. In the N-body domain, there are N objects. All N objects in this dynamic system exert gravitational forces to each other, which is dependent on their mass and pair-wise distance. As the relations are pair-wise, there are in total, $N(N-1)$ relations, which can be modeled as a fully-connected graph. Predicting the dynamics of solar systems can be regarded as an N-body problem.
- **Bouncing balls**. In the domain of the bouncing ball, there are two types of objects, i.e., the balls and the walls. The balls are constantly moving, which can collide with other balls and the static walls. Assuming that there are, in total, N objects including both balls and walls, $N(N-1)$ pair-wise relations exist, which again can be modeled as a fully-connected graph.
- **Charged particles**. In the charged particles domain, there are N particles, and each of them carries positive or negative charges. Each pair of particles interact with each other; hence, there are $N(N-1)$ relations and the system can be modeled as a fully connected graph.

The goal of the task is to infer the future status, given the history (or the initial status) of the dynamical system. The status of the dynamical system can be represented by the trajectories of the objects $\mathcal{T} = \{\mathbf{x}_i, \ldots, \mathbf{x}_N\}$, where $\mathbf{x}_i = \{\mathbf{x}_i^{(0)}, \ldots \mathbf{x}_i^{(t)}\}$ with $\mathbf{x}_i^{(t)}$ denoting the status of node i at time t. Usually, the status information about an object includes its positions or velocity.

In (Battaglia et al., 2016), a model named interaction network is proposed to model and predict the future status of the dynamical system. The model can be viewed as a specific type of graph neural networks. It is also based on passing messages through the graphs to update the node representations. Specifically, there are relation-centric and node-centric functions in the interaction network model, where the relation-centric function is adopted to model the effect of the interactions between nodes, while the node-centric function takes the output of the relation-centric function to update the status of the nodes. Hence, compared with the MPNN framework we introduced in Section 5.3.2, the relation-centric

function can be regarded as the message function while the node-centric function can be viewed as the update function. These functions are usually modeled with neural networks. Note that the interaction network can handle different types of objects and relations by designing various types of functions. A more general framework, named as graph networks, is proposed in (Battaglia et al., 2018).

The interaction network assumes that relations between the objects are known, which might not be practical. In (Kipf et al., 2018), a model is proposed to infer the types of relations while predicting the future status of the dynamical system. It takes the form of variational autoencoder, where both the encoder and decoder are modeled by graph neural networks. The encoder, which is applied to the original input graph $\mathcal{G}$, takes the observed trajectories (the history of the dynamical system) as input and predicts the types of relations. The graph with the information of relation types from the encoder is denoted as $\mathcal{G}'$. It is used as the input graph for the decoder. The decoder is also modeled with graph neural networks, and its goal is to predict the future status of the interacting system.

15.5 Conclusion

In this chapter, we discuss some advanced applications of graph neural networks. We introduced their usage to produce heuristics for NP-hard combinational optimizations on graphs such as minimum vertex cover, maximum cut, the traveling salesman problem, and maximal independent set. We illustrated how source code can be denoted as graphs and how graph neural networks can be leveraged to learn program representations to facilitating down-stream tasks. We also presented how to infer the future dynamics of interacting physical systems via graph neural networks.

15.6 Further Reading

Graph neural networks have been proven to be powerful in handling graph-structured data. They are continually being employed for new applications. In (Jeong et al., 2019), musical scores are denoted as graphs, and graph neural networks are applied to these graphs to render expressive piano performance. In (Zhang et al., 2019c), graph neural networks are adopted to speed up the process of distributed circuit design. In (Rusek et al., 2019), graph neural net-

works are utilized to facilitate network modeling and optimization in software defined networks (SDN).