

Lari Valtonen

SYSTEMS USING GRAPH ATTENTION NETWORKS IN SOLVING JOB SHOP SCHEDULING PROBLEMS

Master of Science Thesis
Faculty of Information Technology and Communication Sciences
Examiners: Prof. Tapio Elomaa
Prof. Ioan Tabus
December 2024

ABSTRACT

Lari Valtonen: Systems Using Graph Attention Networks In Solving Job Shop Scheduling Problems
Master of Science Thesis
Tampere University
Master's Programme in Information Technology
December 2024

Several industrial settings are formulable as a well-known computationally intractable combinatorial optimization (CO) problem called the job shop scheduling problem (JSSP). The increasing interest in machine learning (ML) along with the development of more powerful hardware has piqued the curiosity to apply machine learning methods in solving combinatorial optimization problems.

The representability of JSSPs as disjunctive graphs allows the employment of graph neural networks (GNN) to perform representation learning techniques on the problems. Graph attention network (GAT) is a type of a GNN that uses an attention mechanism to calculate the importance of nodes.

This study conducts a semi-systematic literature review on systems using GATs to solve JSSP and its variants. The literature search was conducted on Scopus, IEEE, and arXiv databases. 14 studies were reviewed. The literature was analyzed for GAT utilization, different graph representations, and ML methods.

GAT was received as valuable in extracting rich feature information from disjunctive and heterogeneous graphs. In addition, various graph features for enhancing the graph representation were identified. While the parameters in GAT architecture were a factor in the system performance, other methods used in conjunction with GAT were found to have a more considerable impact. Deep reinforcement learning was applied in all systems to learn feasible solutions along with several algorithms and other techniques. Even with training on small problem instances, the systems were able to solve large scheduling problems. The scheduling problem formulations were simplistic and did not represent a real-world situation.

Keywords: machine learning, neural network, graph attention network, job shop scheduling, combinatorial optimization

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

TIIVISTELMÄ

Lari Valtonen: Attention-mekanismiin Perustuvat Graafineuroverkot Job Shop -skedulointiongelmien Ratkaisemisessa
Diplomityö
Tampereen yliopisto
Tietotekniikan DI-ohjelma
Joulukuu 2024

Lukuisat puitteet teollisuudessa ovat mallinnettavissa tunnettuna ja laskennallisesti haastavana job shop -skedulointiongelmana (JSSP). Kasvava kiinnostus koneoppimista kohtaan ja edistysaskeleet tietoteknisten laitteistojen kehityksessä ovat lisänneet huomiota koneoppimisen hyödyntämisessä kombinatoristen ongelmien ratkaisemiseen.

JSSP:n esittäminen disjunkttiivisena graafina mahdollistaa graafineuroverkkojen käyttämisen graafin suppeamman numeraalisen esityksen muodostamisen oppimisessa. Attention-mekanismiin perustuva graafineuroverkko (GAT) pyrkii laskennallisesti ilmaisemaan kohdesolmuun yhdistettyjen solmujen tärkeyden.

Tämä tutkimus on toteutettu puolisyntetisena kirjallisuuskatsauksena. Katsauksen kohteena ovat JSSP:n ja sen variaatioiden ratkaisemisessa GAT:itä hyödyntävät järjestelmät. Aineistohaku suoritettiin Scopus-, IEEE- ja arXiv-tietokantoihin, joista lopulta valittiin 14 lähdettä kirjallisuuskatsausta varten. Kirjallisuusanalyysillä selvitettiin miten eri järjestelmät käyttävät GAT:tä, erilaisia graafien esitystapoja ja koneoppimismenetelmiä JSSP:n ratkaisemiseen.

GAT havaittiin merkittäväksi graafien olennaisten piirteiden löytämisessä disjunkttiivisista ja heterogeenisistä graafeista. Aineistosta tunnistettiin useita graafeihin sisällytettyjä piirteitä, joilla rikastettiin graafeissa olevaa tietoa. Vaikka GAT:n rakenteeseen liittyvillä parametreillä havaittiin olevan merkitys järjestelmän suorituskyvyssä, GAT:n kanssa käytettävien menetelmien vaikutus oli keskeisempi. Syvää vahvistusoppimista sovellettiin kaikissa järjestelmissä eri algoritmien avulla. Suurin osa järjestelmistä kykeni löytämään ratkaisun mittaviinkin skedulointiongelmiin käyttämällä opettamisessa vain pienempiä ongelmakokoja. Kirjallisuudessa esitetyt ongelmat oli muotoiltu yksinkertaisiksi ja niitä ei voida verrata todellisessa ympäristössä oleviin optimointiongelmiin.

Avainsanat: koneoppiminen, neuroverkko, graafineuroverkko, job shop skedulointi, kombinatorinen optimointi

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -ohjelmalla.

USE OF ARTIFICIAL INTELLIGENCE IN THIS WORK

Artificial intelligence (AI) has been used in generating this work:

- No
- Yes

Acknowledgement of risks

I hereby acknowledge, that as the author of this work, I am fully responsible for the contents presented in this thesis. This includes the parts that were generated by an AI, in part or in their entirety. I therefore also acknowledge my responsibility in the case, where use of AI has resulted in ethical guidelines being breached.

PREFACE

The rollercoaster finally is at ease on an exceptionally colorful November day.

I thank my family for their support during my time in the university. I also thank my supervisor and examiners Tapio Elomaa and Ioan Tabus for the help and patience.

Tampere, 4th December 2024

Lari Valtonen

CONTENTS

1.	Introduction	1
1.1	Thesis Objective.	1
1.2	Thesis Structure.	2
2.	Basics of Graph Theory.	3
2.1	Graph Representation.	3
2.2	Directed Graph	4
2.3	Multi-relational Graph	4
3.	Machine Learning	6
3.1	Artificial Neural Networks	7
3.1.1	Multilayer Perceptron.	8
3.1.2	Activation Function	9
3.2	Graph Neural Networks	11
3.2.1	Graph Representation Learning	12
3.2.2	Message Passing.	12
3.3	Graph Attention Networks.	13
4.	Job Shop Scheduling Problems.	17
4.1	Descriptions	17
4.2	Complexity	19
4.3	Schedule Modeling.	20
4.4	Previous Work	21
5.	Methodology	24
5.1	Literature Review Method.	24
5.2	Semi-systematic Literature Review	24
5.3	Search strategy	24
5.4	Selection Criteria	26
5.5	Literature Analysis.	26
6.	Results	27
6.1	Literature Search Results	27
6.2	Real World Motivations.	28
6.3	General System Architecture.	28
6.4	Graph Utilization	30
6.5	GAT Utilization	34
6.6	Accompanying Techniques.	35

6.7 Experiment Implementation and System Performance	38
6.7.1 Training and Evaluation	38
6.7.2 Performance Comparison	39
6.8 Limitations	40
7. Conclusion	42
References	44

GLOSSARY

AC	Actor-Critic
AI	Artificial Intelligence
ANN	Artificial Neural Network
CO	Combinatorial Optimization
DAG	Directed Acyclic Graph
DJSSP	Dynamic Job Shop Scheduling Problem
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
ELU	Exponential Linear Unit
FJSSP	Flexible Job Shop Scheduling Problem
FSSP	Flow Shop Scheduling Problem
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GIN	Graph Isomorphism Network
GNN	Graph Neural Network
HFSSP	Hybrid Flow Shop Scheduling Problem
JSSP	Job Shop Scheduling Problem
LeakyReLU	Leaky Rectified Linear Unit
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multilayer Perceptron
OSSP	Open Shop Scheduling Problem
PPO	Proximal Policy Optimization
ReLU	Rectified Linear Unit
SAC	Soft Actor-Critic
TSP	Traveling Salesman Problem

1. INTRODUCTION

Solving CO problems is a central part of multitude computational tasks in science and engineering. The ability to solve these problems impacts directly businesses and social life with a wide range of applications from train dispatching to circuit placement planning. Combinatorial optimization has been an essential and well-established field of study in computer science, contributing an extensive collection of algorithmic solvers for various problems. A JSSP is a classic CO problem that has received a remarkable amount of attention. Regardless of the invested efforts, optimal solutions to the problem still remain very challenging to achieve.

Even with growing computer resources and state-of-the-art algorithms, optimization problems are commonly computationally intractable. To tackle this, algorithms that produce approximate solutions have been developed. There has been interest in generating approximate solutions that are possibly nonoptimal but faster to compute.

Advancements in hardware development have allowed the field of ML to expand its popularity, leading to further progression. This is especially true for *artificial neural networks* (ANN), as the last decade has witnessed an upswing of interest in their potential. As a JSSP is representable in a graph form, employing GNNs in producing feasible solutions has gathered increasing interest.

1.1 Thesis Objective

In recent years, GATs have gained popularity in solving graph-based ML problems. Since GAT is a relatively new GNN architecture, a review on the existing literature about solving JSSPs using GATs was not found to have been conducted. This thesis examines the research on the application of GATs for solving multiple variants of JSSPs through a semi-systematic literature review. The research questions are:

RQ1 How are GATs utilized in systems solving JSSPs?

RQ2 How relevant is the GAT architecture to the performance of a system?

RQ3 What other ML techniques are GATs used as an adjunct to?

1.2 Thesis Structure

The rest of the thesis is organized as follows. Chapter 2 explains the necessary graph theory. Chapter 3 provides an explanation and theoretical background of ML, ANNs and their components, and GNNs with a focus on GATs. Chapter 4 provides descriptions of JSSPs and how they are modeled to be in a computationally solvable form. Chapter 5 describes the research method applied in this study. Chapter 6 presents the results of the research. Lastly, chapter 7 is the concluding chapter.

2. BASICS OF GRAPH THEORY

Many applications allow information to be represented in the form of directed or undirected graphs. A graph consists of objects that may have a relationship between each other. Graphs are used to model myriad systems and interactions in numerous fields, including operations research, sociology, chemistry, physics, and software engineering. The objective of this chapter is not to provide an exhaustive overview of graph theory, but rather to illustrate those aspects that are pertinent to this work.

Formally, a graph is a pair $G = (V, E)$, where V is the set of nodes, denoting the previously mentioned objects. E is the set of edges, denoting relationships. Each edge from node $u \in V$ to node $v \in V$ is described as a pair of nodes $(u, v) \in E$. The number of edges a node has is called the node's *degree*.

2.1 Graph Representation

A diagram is the most common way to present a graph [23, p. 1]. This kind of representation is convenient for visual observation, but more favorable approaches exist for computerized processing. There are various ways to represent graphs numerically. One way to do this is to represent a graph in matrix form, as in Figure 2.1.

An adjacency matrix $\mathbf{A} \in \mathbb{R}^{|V| \times |V|}$ is an advantageous way to store the structure of a graph and present it to a computer. The adjacency matrix representation of a graph is represented by ordering the nodes in the graph in a way that every node is an index of a specific row and column in the matrix. This allows to express the edges as elements in the matrix as $\mathbf{A}[u, v] = 1$ if $(u, v) \in E$, otherwise $\mathbf{A}[u, v] = 0$. The matrix \mathbf{A} will be symmetric for graphs containing solely undirected edges, as the edge can be thought as bidirectional. Graphs may include weighted edges that incorporate a feature associated with the edge. In such cases, the elements in the adjacency matrix are denoted by real values instead of $\{0, 1\}$. [36]

Another potential representation of a graph is an adjacency list. In the case of large graphs with few edges, adjacency matrices are sparse. Adjacency list maps each node to a list of neighboring nodes. This provides a contained structure that allows to clearly examine a node's neighbors.

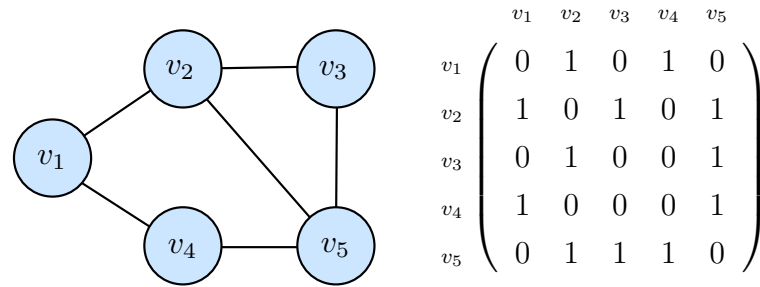


Figure 2.1. Undirected graph with its matrix representation.

2.2 Directed Graph

When no direction is assigned to the edges between nodes, the graph is said to be undirected. An undirected edge denotes a two-way relationship. An example of an undirected graph is a social network of friends where two people are either friends or are not. In such a graph, an edge is considered as an *unordered* pair of nodes (u, v) . However, many practical settings require graphs to be directed, such as flow networks with a specific processing sequence, or city street maps with one-way streets. In a directed graph the set of edges E is mapped onto some *ordered* pair of nodes (u, v) . Directed edges are henceforth referred to as *arcs*.

When graphs are utilized to model flow networks, there are often 2 special nodes called *source* and *sink*. These nodes are so-called 'dummy nodes' with no special features, only denoting the beginning (source) and the completion (sink) of the flow within the network. Arcs may include a feature denoting the flow capacity between nodes.

A cyclic graph consists of edges that form one or more closed paths, allowing graph to be traversed and arrive at the starting node. A cyclic graph can be directed or undirected. An *acyclic graph* does not contain such paths. An acyclic graph, where every relationship between nodes is an arc, is called a *directed acyclic graph*, or DAG, as seen in Figure 2.2.

DAGs are used extensively across diverse fields like computer science, data analysis, and machine learning. In computer science, DAGs provide an essential data structure for tasks such as scheduling and computation optimization [56].

2.3 Multi-relational Graph

In addition to the distinction of undirected edges from directed edges, there can be different types of edges within graphs. The notation can be extended to include an edge type $u, \tau, v \in E$ as an indication of a different kind of relation. The graphs presented thus far have comprised of single types of edges and nodes. This kind of graph with an absence of types is called a *homogeneous graph*. One important subset of multi-

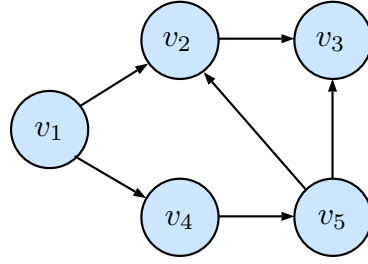


Figure 2.2. Directed acyclic graph.

relational graphs is a *heterogeneous graph*. The nodes in a heterogeneous graph similarly include types, which allows to separate them into disjoint sets $V = V_1 \cup V_2 \cup \dots \cup V_k$, where $V_i \cap V_j = \emptyset, \forall i \neq j$. The edges of a heterogeneous graph ordinarily adhere to some constraint satisfaction criterion regarding node types, usually particular types of nodes are being limited to be connected by specific types of edges $(u, \tau_i, v) \in E \rightarrow u \in V_j, v \in V_k$. [36]

A heterogeneous graph with 3 different node types presented as different colors is illustrated in Figure 2.3.

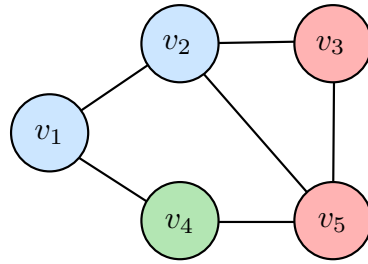


Figure 2.3. Heterogeneous graph with colors indicating node types.

3. MACHINE LEARNING

ML is a branch of AI. The idea of ML is for a computer program to improve the performance of executing a class of tasks through experience. A popular formalization of ML by Mitchell [57] is: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E ".

Various different types of ML exist, depending on what type of tasks the system is built to learn, the performance measure for evaluating the system, and the way the system is trained [60, p. 1]. As an example, ML can be used to learn to recognize or classify certain patterns from input signals, such as image classification or speech recognition.

Two primary objectives can be identified from ML problems of the day, the first being estimation and prediction, and the second being classification and pattern recognition. These objectives can be categorized into four different types of learning: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. The available information has a role in which type of learning may be used.

Supervised learning applies a training set to teach models to provide the desired output. The aim is to learn a function that predicts an output for every input as accurately as possible by providing a set of input-target pairs. The learning is solved by means of optimization problem over a category of functions [8]. The cost function calculates the error between the outputs and the target values. The function of choice depends on the optimization methods and the nature of the task, such as classification or regression. In classification, the problem is to recognize specific entities within the dataset and accurately assign them into mutually exclusive class labels. Regression attempts to understand relationships between variables. Regression is often used to make predictions as the outputs are real values.

Unsupervised learning can be utilized when the training data lacks the target labels. With unsupervised learning, collecting labeled datasets and having to categorize the world into classes becomes unnecessary. While supervised learning includes fitting a conditional model, specifying outputs given inputs, unsupervised learning can be viewed as fitting an unconditional model, that can generate new data [60, p. 14]. With respect to CO problems, unsupervised learning has received limited attention, and its immediate

use seems challenging [8].

Semi-supervised learning fits between supervised learning and unsupervised learning. In semi-supervised learning the learning data contains unlabeled data in addition to labeled data. Often, the set of unlabeled data is considerably larger in comparison to the labeled dataset [63]. The aim is to take advantage of the unlabeled data to produce a model with superior performance than with only labeled data. One of the early semi-supervised learning algorithms first trains with the labeled portion, then iteratively labels a part of the unlabeled data and adds it to the labeled training set.

Reinforcement learning involves an agent that explores and interacts with an environment. The distinction between reinforcement learning and supervised learning is that there is no training data prepared beforehand. Reinforcement learning is also different from unsupervised learning, as it does not attempt to uncover a hidden structure within data. The lack of data leads to long learning times, as the learning happens through an agent exploring the environment. An agent's actions are rewarded with a negative or a positive reward signal, which the agent tries to maximize.

3.1 Artificial Neural Networks

ANN is an attempt to imitate the structure and learning capability of the human brain [37, pp. 31–39]. The human nervous system contains *neurons*, that are linked to a multitude of other neurons through junctions called *synapses*. Neurons operate in parallel, transferring information to other neurons over synapses. The amplitude of the synaptic connection generally changes in reaction to stimuli. Learning in the brain is widely believed to occur in synapses [18].

ANN is a network of computational units linked to one another through weights, computational units being artificial neurons and weights serving as synaptic connections. Each neuron has an activation value, and a weighted connection from one neuron to another. The neurons in an ANN take weighted inputs and propagate the values to the output neurons. Usually, neurons include a constant term called the bias, that often have an important role in practice as they can have a great impact on the accuracy of the outcome of the prediction. A network where input neurons are directly connected to one or more output neurons is called a single-layer network. A multi-layer network has at least a single hidden layer between input and output layers. ANNs often include several layers of neurons. Learning happens by changing the weights that are connected to the neurons. The objective is to use training examples to learn a function that associates one or more inputs with one or more outputs. [2, pp. 3–8]

The simplest ANN only consists of an input layer and an output neuron. This type of network is called a *perceptron*, and is illustrated in Figure 3.1. At the time the perceptron

was first proposed, it was not presented formally as a loss function optimization, as is common for ML methods presented today. The perceptron is limited for classification of linearly separable groups. [2, pp. 5–7]

The simplest computation neuron executes can be written as a line equation $y = kx + b$, where x is the neuron's input, b the bias term, y the neuron output, and k is the slope. In ANNs, k is called weight and is denoted as w . When all the neuron inputs are calculated, the total output is the sum of the products of weights and associated inputs. Now the neuron output can be expressed as

$$z = \left(\sum_{i=1}^m w_i x_i \right) + b, \quad (3.1)$$

where x_i are the inputs, w_i are the corresponding weights, and b is the bias.

Many ML methods focused on optimization are re-creatable with very simple neural network architectures to the point of ANNs being considered as a more potent generalization of these simple models [2, pp. 53–55]. The depth of an ANN is described by the quantity of layers performing the calculations, making the perceptron a single-layer neural network, even as it only consists of an input and an output layer.

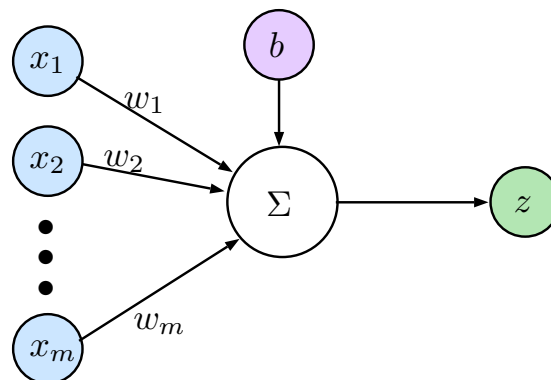


Figure 3.1. Perceptron.

3.1.1 Multilayer Perceptron

A perceptron with the single layer of weights is limited to approximating only linear functions and is not applicable to nonlinear regression or solving problems with a nonlinear discriminant. This limitation can be overcome by introducing hidden layers between the input and the output layers. [4, pp. 279–281]

A *multilayer perceptron* (MLP) is a feedforward neural network model (Figure 3.2) for processing information through groups of interconnected perceptrons. All the input nodes and all the output neurons belong to their respective layers, and the hidden neurons are distributed among 1 or more hidden layers between input and output layers.

The hidden neurons are not part of the network's input or output. The first hidden layer obtains its inputs from the input layer, which consists of sensory units. The second hidden layer is fed the resulting outputs of the first hidden layer, and so on, until the last hidden layer passes its outputs to the output layer. The output layer then produces the output signal of the network. In all layers of an MLP, the nodes express connectivity to a high degree. All the neurons in the network include a differentiable nonlinear activation function. [37, pp. 122–125]

The hidden neurons have an important role in performing as feature detectors of the MLP. During the training process, the neurons gradually learn to find noticeable features that define the training data by performing nonlinear transformations on the input data into a feature space. The feature space allows to more easily separate interesting information than from the original input data space. A common technique for training an MLP is the backpropagation algorithm, that calculates an error signal by comparing the network output with a desired output. The resulting error then propagates backwards through the layers, resulting in fine-tuning of the synaptic weights. [37, pp. 123–126]

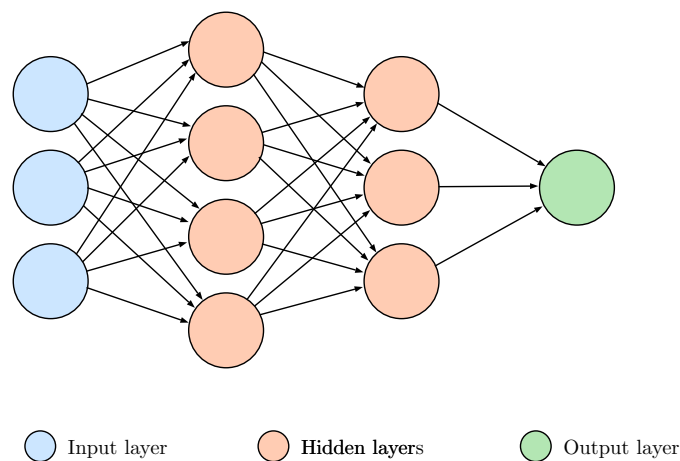


Figure 3.2. Fully connected feed-forward neural network with 2 hidden layers.

3.1.2 Activation Function

An important part of a hidden layer in a neural network is the activation function that restricts the permitted output range of a neuron to some finite value. Any number of hidden layers would be meaningless if the outputs were linear as the linear combination of linear combination returns another linear combination, since the same result is achievable with a simple model without hidden layers. [4, pp. 279–281]

Many real-world relationships that ANNs are used to model are nonlinear. Attempting to model nonlinear relationships with a linear function will produce a very inaccurate and underfit model. ANNs, notably *deep learning networks*, typically require modeling relationships of complex nonlinear mappings. A method of introducing nonlinearity must

be incorporated in order for such mappings to be realized. This is accomplished by including an activation function in a neuron. In essence, any nonlinear activation function will allow an ANN to learn nonlinear mappings. The function is usually the same for every neuron of the same layer. There are several different activation functions applied for different purposes, each having their pros and cons, but only a few are recommended. [44, pp. 77–80]

The activation functions that are recommended for ANNs have changed over time and these changes have notably improved the performance of feedforward networks [32, pp. 220–223]. During the early years of ANNs, threshold functions were used as activation functions. However, the impossibility to compute the gradient at the threshold point, and the backpropagation algorithm requiring a differentiable function, led to the introduction of differentiable activation functions, such as the logistic sigmoid and the hyperbolic tangent [44, p. 127]. The logistic sigmoid activation function is defined as

$$\text{sigmoid}(z) = \frac{1}{1 + \exp(-z)}, \quad (3.2)$$

and the hyperbolic tangent activation function as

$$\tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)} = \frac{1 - \exp(-2z)}{1 + \exp(-2z)}. \quad (3.3)$$

As the number of layers in the ANNs grew, the gradients of the loss functions began to increasingly approach zero, making the network more difficult to train. This problem is called the *vanishing gradient problem*. To improve the error gradient progression through the network, there was another shift into utilizing simpler, segmented linear activation functions.

Currently, the most common activation functions include *rectified linear unit* (ReLU), *leaky rectified linear unit* (LeakyReLU), and *exponential linear unit* (ELU) [45][66]. The advantages of ReLU are in its simplicity, allowing fast computation, and sparsity as it limits the output values between 0 and the input value. Also, this way the derivative stays large, and the gradients also remain consistent when the unit is active. Even though there is a discontinuity when $z = 0$, the software implementations usually return either the left or right derivative rather than state that the derivative is undefined, or causing an error [32, p. 188]. ReLU is defined as

$$\text{ReLU}(z) = \max\{0, z\}. \quad (3.4)$$

There are limitations to ReLU, such as easily causing overfitting. Using the ReLU may also lead to so called "dead neurons", that produce an output of 0 regardless of the input, resulting in a gradient of 0 and causing the weights to not update. Since the

derivative of the function is 0 for all negative inputs, the neuron may stay in this state throughout the training, thwarting its usefulness. [62]

The LeakyReLU was proposed to solve the problem of dead neurons. The solution was to add a negative slope to keep the gradients nonzero during training. The LeakyReLU is given by

$$\text{LeakyReLU}(z) = \begin{cases} z & \text{if } z \geq 0 \\ \alpha z & \text{otherwise,} \end{cases} \quad (3.5)$$

where α is a nonzero negative slope. [62]

ELU uses a nonlinear slope for negative inputs. Unlike ReLU, having negative values brings the mean of the activation outputs towards 0, which enables faster learning [17]. ELU with $\alpha > 0$ is

$$\text{ELU}(z) = \begin{cases} z & \text{if } z > 0 \\ \alpha(\exp(z) - 1) & \text{otherwise,} \end{cases} \quad (3.6)$$

where α is a hyperparameter. When the input z is negative, the output slightly curves towards the α , so the output range is $[\alpha, \infty)$.

The softmax is an activation function that takes K values as input scales these values to sum to 1. This is mostly employed in the final layer of ANNs. The softmax is given as

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^K \exp(z_j)} \quad \text{for } i = 1, 2, \dots, K. \quad (3.7)$$

Activation functions have a major impact on the ANNs' performance, and choosing a suitable activation for a given task is not straightforward as this may require meticulous research and testing. The choice of activation function depends on the task at hand and there are no clear guidelines for choosing the correct activation function. [72]

3.2 Graph Neural Networks

GNNs are general frameworks to implement deep neural networks attempting to learn deep structural properties of a graph using hidden layers. GNNs are currently amidst the most general class of deep learning architectures, and most other deep learning architectures are definable as special cases of the GNN with additional geometric structure [13]. GNNs have been studied since at least as early as 2005 [33][70] but have amassed exceptional interest in recent years. This resurgence of interest is attributed to the success of deep learning and to discovering better solutions for learning graph features [101].

Node-level, edge-level, and graph-level tasks are three common types of learning tasks associated with graphs. Node-level tasks prioritize nodes, such as node classification,

node regression, node clustering, etc. Node classification attempts to assort nodes by labeling them, node regression calculates predictions of feature values of the nodes, and node clustering partitions the nodes by forming separate groups according to some similarity criterion. Edge-level tasks focus on edge classification and link prediction, where edge types are classified or whether an edge exists between nodes. Graph-level tasks are graph classification, graph regression, and graph matching, which are all graph representation learning tasks. [101]

GNNs are neural network architectures that are devised specifically to operate on data with a graph structure. The objective of the GNNs is to first iteratively update the node representations by applying neighborhood propagation to aggregate the representations of the node neighbors. This results in each node obtaining an aggregated representation of the neighboring nodes. [88, pp. 27–37]

3.2.1 Graph Representation Learning

Representation learning serves as a foundational concept in AI and ML that entails transforming input data into points in low-dimensional vector space \mathbb{R}^d that sufficiently but minimally capture information within input data [88, pp. 3–5].

Graph representation learning strives to assign nodes in a graph to low-dimensional representations, *embeddings*, and adequately sustaining the structure of a graph. The motivations for finding effective graph data representations stem from high computational complexity of large graphs, graph data parallelizability, and inapplicability of ML methods. Considerable effort has been committed in undertaking aforementioned challenges to develop innovative graph representation learning techniques, that is, learning compact and continuous low-dimensional vector representations for nodes, while reducing noise and redundancies, and preserving structural information. GNNs can learn embeddings from both graph structure and node and edge features in tandem. [88, pp. 17–19, 31]

3.2.2 Message Passing

A key property of a GNN is the application of a neural message passing technique, where messages get sent from one node to another as vectors, and obtain new information through neural networks [36]. This exchange aims to learn the hidden embeddings that are representation vectors corresponding to each node. Throughout every message-passing iteration l , a hidden embedding $\mathbf{h}_u^{(l)}$ of a node $u \in V$ is updated with information obtained through an aggregation procedure from the target node u 's neighborhood

$\mathcal{N}(u)$:

$$\mathbf{h}_u^{(l+1)} = \text{UPDATE}^{(l)}\left(\mathbf{h}_u^{(l)}, \text{AGGREGATE}^{(l)}(\{\mathbf{h}_v^{(l)}, \forall v \in \mathcal{N}(u)\})\right) \quad (3.8)$$

$$= \text{UPDATE}^{(l)}\left(\mathbf{h}_u^{(l)}, \mathbf{m}_{\mathcal{N}(u)}^{(l)}\right), \quad (3.9)$$

where UPDATE and AGGREGATE are some differentiable functions and $\mathbf{m}_{\mathcal{N}(u)}$ is the aggregated message from u 's neighborhood $\mathcal{N}(u)$, which consists of all nodes that are connected by an edge to u [36]. The superscript l refers to the specific message passing iterations, also referred to as GNN layers.

With repeated iterations, the node u will learn information about the nodes outside of its neighborhood. The aggregation function takes the embeddings of the nodes in u 's neighborhood as the input at each iteration l which results in a message $\mathbf{m}_{\mathcal{N}(u)}^{(l)}$ based on the aggregated information. Then the update function incorporates the message $\mathbf{m}_{\mathcal{N}(u)}^{(l)}$ with the embedding $\mathbf{h}_u^{(l-1)}$ from the previous iteration to update the embedding $\mathbf{h}_u^{(l)}$. The embeddings at $l = 0$ are the input features for every node. The update process runs for a total of L iterations to obtain the final embeddings. [36]

The aggregation is depicted in Figure 3.3. In the figure, a 2-hop neighborhood aggregation is performed on the node A.

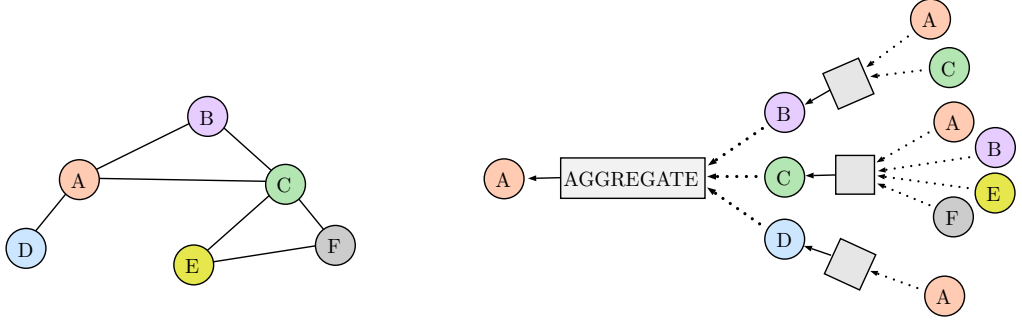


Figure 3.3. Illustration of an input graph (left) and $L = 2$ aggregation employed on a single node (right). Based on [36].

3.3 Graph Attention Networks

In the field of psychology, attention is defined as the ability to actively process a discrete aspect of information in the environment while neglecting impertinent information [91]. As a component in ML, *attention mechanism* mimics this cognitive attention by allowing ML models to focus on certain aspects of the input data when conducting predictions or decisions. Attention facilitates the modeling of global or local dependencies between input and output [78]. This happens by including surrounding elements and their relative importance.

One of the methods to overcome the challenge of obtaining interesting information of a graph is a GAT that attempts to learn how relevant each neighboring node is to a target node by utilizing attention mechanism in a graph attention layer. A GAT is built of one or multiple graph attentional layers that leverage self-attentional layers to cope with the deficiencies in preceding methods, such as graph convolution [79]. Self-attention means that the attention mechanism connects varied positions in a single sequence as a way to compute a new presentation of the sequence [78].

Attentional architecture has a few appealing properties. Attention mechanism allows size-agnostic inputs and to focus on the most interesting input components. Computation of the hidden node representations across node-neighbor pairs is efficient as it is parallelizable. It is applicable to varying degrees by designating arbitrary weights to the neighbors. Lastly, the model is suited for inductive learning problems, such as having to generalize to entirely unobserved graphs. [79]

Graph attentional layer takes a set of node features as an input

$$\mathbf{h} = \{\vec{h}_1, \vec{h}_2, \dots, \vec{h}_N\}, \vec{h}_i \in \mathbb{R}^d,$$

where N is the number of nodes, and d denotes the number of features related to a node. The output from the layer is a new set of node embeddings

$$\mathbf{h}' = \{\vec{h}'_1, \vec{h}'_2, \dots, \vec{h}'_N\}, \vec{h}'_i \in \mathbb{R}^{d'}$$

of possibly some other cardinality d' . The initial step to attain adequate expressive power to reconstruct the input features into higher dimensional vector is to apply a shared linear transformation, parametrized by a weight matrix $\mathbf{W} \in \mathbb{R}^{d' \times d}$, to each node. This is followed by a step that performs self-attention on the nodes. A shared attentional mechanism $a : \mathbb{R}^{d' \times d'} \rightarrow \mathbb{R}$ calculates attention coefficient e_{ij} for all the edges (j, i)

$$e_{ij} = a(\mathbf{W}\vec{h}_i, \mathbf{W}\vec{h}_j), \quad (3.10)$$

indicating how important the features of the neighboring node j are to the node i . Next, a masked attention is performed to only compute e_{ij} for the neighborhood $j \in \mathcal{N}_i$. The most general formulation of the GAT model allows to compute e_{ij} for every node, resulting in a loss of structural information. [79]

The coefficients may not be easily comparable between different nodes. To make this easier, coefficients are normalized using the softmax function

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}. \quad (3.11)$$

The attention mechanism in GAT is a single-layer feedforward ANN parametrized here

by a weight vector $\vec{\mathbf{a}} \in \mathbb{R}^{2d'}$. In the original GAT implementation by Veličković et al. [79], nonlinearity is introduced by applying LeakyReLU activation function

$$e_{ij} = \text{LeakyReLU}(\vec{\mathbf{a}}^T [\mathbf{W}\vec{h}_i \parallel \mathbf{W}\vec{h}_j]). \quad (3.12)$$

The fully expanded form of the computed coefficients by the attention mechanism becomes

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\vec{\mathbf{a}}^T [\mathbf{W}\vec{h}_i \parallel \mathbf{W}\vec{h}_j]))}{\sum_{k \in \mathcal{N}_i} \exp(\text{LeakyReLU}(\vec{\mathbf{a}}^T [\mathbf{W}\vec{h}_i \parallel \mathbf{W}\vec{h}_k]))}, \quad (3.13)$$

where \cdot^T denotes transposition and \parallel represents the concatenation operation.

After obtaining the normalized attention coefficients, these are used to compute linear combination of the corresponding features. After potentially passing them through a nonlinear activation, the final output serves as the new embeddings for each node i

$$\vec{h}'_i = \sigma \left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} \mathbf{W}\vec{h}_j \right), \quad (3.14)$$

where the nonlinear activation is denoted by σ .

The notation so far explains calculation of the attention mechanism using a single-headed attention, meaning the calculations for every node are done once per iteration. The GAT implementation supports the usage of multi-head attention and is found to be advantageous in stabilizing the learning process of self-attention. In multi-head attention, K attention mechanisms are calculated independent of each other, as in Equation 3.14, and then concatenated (Equation 3.15) or averaged (Equation 3.16)

$$\vec{h}'_i = \parallel_{k=1}^K \sigma \left(\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j \right) \quad (3.15)$$

$$\vec{h}'_i = \sigma \left(\frac{1}{K} \sum_{k=1}^K \sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j \right), \quad (3.16)$$

where concatenation is denoted by \parallel . The normalized attention coefficients α_{ij}^k are obtained by the k -th attention head and \mathbf{W}^k is the linear transformation weight matrix of the matching input. Averaging is used in the final layer of the network where concatenation is no longer feasible. The application of nonlinearity σ is also delayed to happen after the averaging. [79]

A modification to GAT was suggested by Brody et al. [12], who commented on its limits, and showed that GAT's expressive power is strictly hindered when constrained to only compute static attention, meaning monotonic attention score for neighbors with respect to per-node scores. This variant is also known as GATv2. The main point of

criticism towards GAT lies in Equation 3.12, particularly the consecutive application of parameters \mathbf{W} and $\vec{\mathbf{a}}$ which is collapsible into one linear layer. The simple adjustment to address this is to move the $\vec{\mathbf{a}}$ after applying nonlinearity with LeakyReLU, and to move the \mathbf{W} after the concatenation

$$e_{ij} = \vec{\mathbf{a}}^T \text{LeakyReLU}(\mathbf{W}[\vec{h}_i \parallel \vec{h}_j]), \quad (3.17)$$

which effectively acts as an MLP.

Some implementations of GAT include a support for edge features. One possible way to enable this is

$$e_{ij} = \vec{\mathbf{a}}^T \text{LeakyReLU}(\mathbf{W}[\vec{h}_i \parallel \vec{h}_j \parallel \vec{e}_{ij}]), \quad (3.18)$$

where \vec{e}_{ij} are multi-dimensional edge features between nodes i and j [65].

4. JOB SHOP SCHEDULING PROBLEMS

Businesses have an imperative to adhere to the transportation dates that have been given to the customers. Neglecting these could result in a significant loss of favorable relations. In addition, they must schedule activities in a manner that optimizes the utilization of available resources. Scheduling theory formalizes scheduling as the allocation of limited resources over time with the attempt of completing an array of assignments. Scheduling is thus a critical decision-making procedure aiming to increase the efficiency of operation. The resources and tasks vary in nature depending on the organization. The resources may be workers, machines, computer processors, airport runways, etc. The tasks, e.g. construction project stages, production process operations, computer program executions, take-offs and landings, are individually competing for the resources. [64, pp. 1–4]

This has led to a virtually endless number of different types of scheduling problem characterizations [14, p. 1]. Some of the problems which have received a great deal of attention are project scheduling [19] [52], computer central processing unit scheduling [42] [21], surgery scheduling [67] [82], airport gate assignments [64, p. 1] [10], and course and exam timetable scheduling in universities [15] [20].

These applications are all testaments to the magnitude and relevance of the subject. The scheduling tasks performed in manufacturing industries are particularly essential, as the continuation of an organization rests on its ability to be profitable.

Job shop scheduling problem (JSSP) is a classic problem in the operations research field and has gained a tremendous amount of attention over the years. JSSP and its variants possibly bear the most similarity to real-world production scenarios [30] [83]. These problems can be found in various practical settings: semiconductor manufacturing [34], glass industry [5], boarding and printing [80], and in mechanical manufacturing and automobile assembly [31].

4.1 Descriptions

A *job shop* is an abstraction of a worksite fitted with a predetermined number of available *machines*. Each machine performs a specific task, an *operation* associated with a *job*,

in the chain of production. There are two common constraints that apply to every shop scheduling problem presented in this section: a job is allowed to be processed in only one machine at once and a machine is allowed only to process a single job at once. In all the job shop problems described in this work, the machines and jobs are assumed to be finite in numbers.

Job shop scheduling problem A classic JSSP can be described as follows. The task is to schedule a size n set of jobs $J = \{J_1, \dots, J_i, \dots, J_n\}$ to a size m set of machines $M = \{M_1, \dots, M_m\}$. Each job $J_i \in J$ contains a series of n_i operations $O_{i1}, O_{i2}, \dots, O_{in_i}$, which are processed in the given order. In other words, this is a precedence constraint of the form $O_{ij} \rightarrow O_{i,j+1}$, where $j = 1, \dots, n_i - 1$. Every operation O_{ij} has a processing time p_{ij} and a machine $\mu_{ij} \in M$ associated with them. Machine μ_{ij} processes each O_{ij} for p_{ij} units of time. The optimization problem is to discover a favourable schedule to minimize an objective function. A common objective function is the *makespan* C_{max} , which is measured as the time that has passed from the start of the first operation to the end of the final operation of the entire process. [14, pp. 178–180]

Flexible job shop scheduling problem FJSSP is a generalization of a JSSP whereby every job is permitted to be processed by any of the eligible machines on each machine stage, which introduces additional complexity compared to the classic JSSP [24]. Following the description of a JSSP, jobs consist of a sequence of consecutive operations O_{in_i} , but instead of being limited to a single machine, each operation O_{ij} can be processed on any machine within a subset $M_{ij} \subseteq M$ of eligible machines. As in a JSSP, the operations have processing times and precedence constraints.

Dynamic job shop scheduling problem A DJSSP is a variation that introduces stochastic elements into the problem. The other problems without random factors are called static, or deterministic, scheduling problems. A DJSSP can be seen as an extension to a JSSP, which considers unpredictable events during processing. A DJSSP follows the description of JSSP while introducing additional dynamic events into the production. Some machines or jobs may not be available at the beginning of production. DJSSPs are more commonly encountered in actual production scenarios than JSSP [51]. The dynamic events in question can be categorized into four classes [58]:

- Job-related events, e.g. dynamic process times and arrival times.
- Machine-related events, e.g. machine breakdowns and limited machine employment.
- Process-related events, e.g. delays, rejected processes, and unstable production.
- Other events, e.g. missing personnel and material defects.

Flow shop scheduling problem An FSSP arises when the problem requires to schedule n jobs on m machines so that every operation of each job follows the same processing order for all machines. An FSSP can be seen as a special case of a JSSP in which for each job J_i , the number of operations equals to the number of machines; $n_i = m$ for $i = 1, \dots, n$ and $\mu_{ij} = \{M_j\}$ for each $i = 1, \dots, n$ and $j = 1, \dots, m$. Each operation O_{ij} has a processing time p_{ij} ($j = 1, \dots, m$) and the operation has to be processed with machine M_j . Each job begins its processing on machine 1, moving on to machine 2, etc., in other words, operations have precedence constraints expressed as $O_{ij} \rightarrow O_{i,j+1}$ ($i = 1, \dots, m - 1$) for each $i = 1, \dots, n$. [14, pp. 174–178]

Hybrid flow shop scheduling problem An HFSSP is a more complex extension to an FSSP and are closer to actual production cases [99]. The problem is to schedule set of jobs $J_i \in J$ for processing in H stages $\{s_1, s_2, \dots, s_h, \dots, s_H\} \in S$. Each stage contains identical parallel machines $NM_{s_h} > 1$, with $NM_{s_h} \geq 2$ at least in one stage. Each J_i is scheduled to any machine in each stage, with the challenge of optimizing some objective function.

Open shop scheduling problem In an OSSP, the challenge is in finding a feasible order for operations related to the same job, and orders of operations to be processed on the same machine [14, pp. 158–160]. Every job i is comprised of m operations O_{ij} ($j = 1, \dots, m$) where O_{ij} is processed by each machine M_j [14, pp. 158–160]. Some of the operations may not have processing time at all [64, p. 15]. An OSSP does not bear any precedence relations regarding the order of operations. Each job is allowed to have a different route through the machines. Furthermore, the flexibility of an OSSP creates a more expansive solution space compared to other shop scheduling problems, and the optimal scheduling schemes are capable of obtaining solutions with makespan closer to the lower bound [50].

4.2 Complexity

JSSPs have earned a reputation for being considered as some of the most computationally intractable combinatorial problems [6]. Computational complexity expresses the computer time required to run an algorithm. Class \mathcal{P} are all the problems which can be solved deterministically in polynomial time. An algorithm can be solved in polynomial time if the total steps needed for the completion for a given input is $O(n^k)$, where k is a nonnegative integer and n denotes the input's complexity. NP-hard problems belong into the group of problems where no polynomial-bounded algorithm is known to exist.

The complexity of makespan minimization for a JSSP depends on the number of jobs, machines, operations, and processing times. A 2-machine problem with jobs that have at most 2 operations with arbitrary processing times belongs to \mathcal{P} [48]. Unsurprisingly,

an efficiently solvable problem quickly becomes NP-hard with only a slight alteration to the problem definition. A makespan minimization problem with $m \geq 2$ and $j \geq 3$ is found to be NP-hard [49].

4.3 Schedule Modeling

Schedules are frequently represented as Gantt charts. Gantt chart is an effective way for understanding and visualizing scheduling. This provides a thorough view of the timeline that serves as a graphical representation of the scheduled tasks. A Gantt chart is illustrated in Figure 4.1.

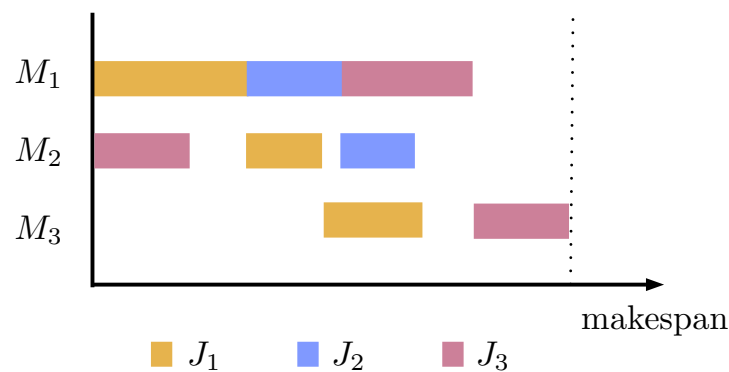


Figure 4.1. Gantt chart of problem with 8 operations, 3 jobs, and 3 machines.

Another informative representation of a scheduling problem is the *disjunctive graph* model, first suggested by Roy and Sussmann [68] in 1964. This has mostly substituted Gantt charts for the solution representation within algorithms. Although, Gantt charts still serve a purpose for graphical representation of a problem solution. [9, pp. 345–352]

Disjunctive graphs provide a succinct representation of structural properties of schedules and have been applied expansively for scheduling problems. A disjunctive graph models nodes as operations and arcs are representations of dependencies created by a job path, or allocations to assign operation sequences on a machine. All feasible operation-machine assignments are modeled as undirected edges. A disjunctive graph of an unsolved problem is illustrated in Figure 4.2, and a solution in Figure 4.3.

In formal terms, the disjunctive graph is a directed graph $G = (V, C, D)$. Operations are indicated as a set of nodes V , the set C consists of conjunctive arcs denoting the precedence constraints linking each 2 sequential operations belonging to the same job. Set D consists of undirected disjunctive edges, which link each of the unordered operations involving the same machine. Arcs may be labelled with a positive value denoting the processing time of the operation where the arc begins. [14, pp. 156–158]

In the disjunctive model, finding a schedule equals to selecting one arc in each disjunction, that is, turning all undirected disjunctive edges into directed conjunctive arcs. The

processing order of each conflicting operation requiring the same machine is obtained by setting the directions of all disjunctive edges, resulting in a complete schedule. [14, pp. 156–158]

The disjunctive graph includes the necessary details required to solve a JSSP, completely or partially. Therefore, the proper representation considerably affects the performance of the problem-solving algorithm. Any valid graph $G(S) = (V, C \cup S)$ that solves a JSSP is acyclic. S denotes the directed conjunctive edges. [14, pp. 156–158]

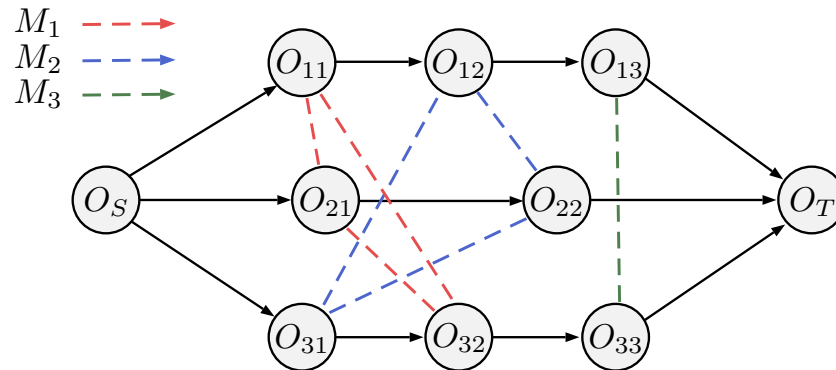


Figure 4.2. Disjunctive graph of an unsolved JSSP.

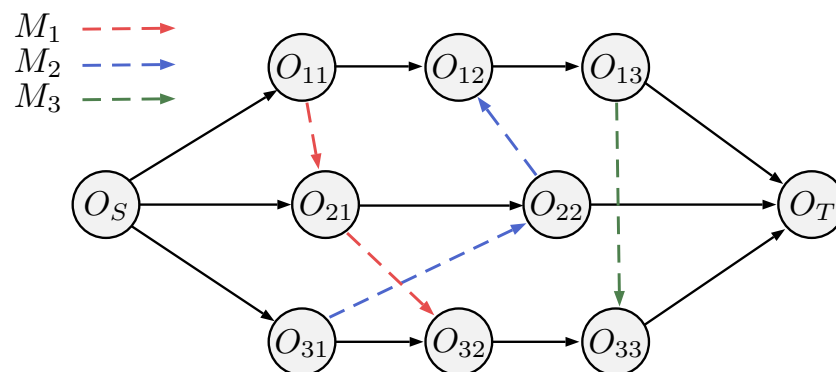


Figure 4.3. A solution to the JSSP in the Figure 4.2.

4.4 Previous Work

This section focuses primarily on presenting past studies where neural networks have been utilized in solving JSSPs. More general and extensive surveys on obtaining scheduling solutions by applying AI techniques and through ML have been conducted in multiple works [69] [43] [3] [16] [25].

Owing to the general limitations of optimal enumeration techniques, methods producing approximal solutions were seen as a feasible alternative [43]. Neural networks are one of the methods that produce approximate solutions. These are fast to provide solutions to scheduling problems but are not guaranteed to discover the optimal solution. The

key concept in neural networks is to recognize patterns and rules ingrained in a quality schedule [34].

A pioneering demonstration of applying ANNs to solve NP-hard optimization problems was performed by Hopfield and Tank [38] in 1985 by solving a *traveling salesman problem* (TSP). Their work presented an analog computational network constructed of neurons modeled as amplifiers in conjunction with feedback circuits. The optimization problem itself was described by an energy function. Hopfield and Tank indicated in their work that if a problem is representable by an energy function, then a corresponding Hopfield network can be utilized to minimize this function. Since then, the Hopfield network and the variants of this architecture have been used extensively in solving various optimization problems [3]. Hopfield was awarded the 2024 Nobel Prize in physics for his work on ANNs [61].

The motivation which sparked from solving a TSP extended to solving JSSPs with neural networks. Foo and Takefuji adopted the Hopfield TSP type network and devised a 2D neuron matrix to map their JSSP [28] [29]. They employed simulated annealing, a stochastic optimization technique, in the system to find the global minimum of the energy function. The energy function acted as the objective function, which was the cost of total completion times of all jobs. This method yielded nearly optimal solutions. Later Foo and Takefuji introduced an extension to the Hopfield network with fewer neurons called a linear programming neural network by describing the problem as an integer linear program [27].

However, the work by Foo and Takefuji met criticism for producing constraint violating solutions, and Van Hulle reformulated the problem as a goal programming problem by replacing the single objective of integer linear programming neural network by several objectives [77]. This goal programming network was also used for JSSPs. Additionally, Hopfield networks were criticized for their poor scaling properties. Zhou et al. [100] identified this scaling issue as being the effect of quadratic energy function causing the quadratic cost of the interconnection network. In their study they proposed a linear energy function, resulting in a scalable network with reduced quantity of neurons and interconnections.

Lo and Bavarian [54] expanded the 2D Hopfield network to 3 dimensions with time being the third dimension. Their work utilized this network to solve JSSPs, as well as a *multiple TSP*. The results of the simulations performed showed that their method provides feasible schedules. However, the number of neurons and large connectivity makes this approach impractical for large problems.

Incorporating an ANN with other AI techniques is not uncommon. Weckman et al. [86] combined an ANN with genetic algorithms to develop an intelligent system for scheduling job shops. In their method, genetic algorithms were employed to produce

both near-optimal and optimal schedules to a known JSSP test instance. The obtained solutions were used to train an ANN to grasp the predictive information with regard to the operation sequences.

Zang et al. [94] proposed a hybrid deep neural network scheduler for solving JSSPs. Their method first divides the problem into several classification-based subproblems to be solved with a deep learning framework comprised of parallel fully connected convolutional layers. The framework applies convolution 2D transformation process to uncover scheduling features by converting irregular scheduling state information into regular information. This allows the convolution deep learning operation to be introduced into solving JSSPs.

GNNs were applied for solving JSSP by Zhang et al. [97], who used a deep reinforcement learning agent to learn priority dispatching rules. The priority dispatch rule-based scheduling was formulated as a Markov decision process. A GNN focused scheme was adopted to create embeddings of the encountered states during solving. The variant of the adopted GNN was graph isomorphism network. The implied rationale for selecting the graph isomorphism network was its proven high discriminative power.

Reinforcement learning was used by Liu et al. [53] as well to solve a traffic management problem formulated as a JSSP. They proposed an end-to-end framework with a single-policy strategy, employing GNNs and deep Q-learning. The training was executed by surveying incoming rewards in addition to following suitable rules. They deemed their model was suited for solving instances of similar sizes.

5. METHODOLOGY

This section explains the research methods used in this study. The objective of this work is to study and review the papers adopting GATs in solving a JSSP or any of its variants. This is accomplished through a literature review. Considering the aim to identify themes and methods, the specificity of the subject, and the time limitations, a semi-systematic review was embraced as the method to conduct the inquiry.

5.1 Literature Review Method

In scientific research, it is essential to build and relate research on existing studies through a literature review. A broad description of a literature review is collecting and synthesizing previous research in a more or less systematic manner. Almost all studies include some form of literature review, but when used as a research method, a literature review should follow the standards and practices of the review type. [73]

5.2 Semi-systematic Literature Review

The semi-systematic literature review positions between systematic and integrative literature reviews. Semi-systematic review combines the methodological rigor of the systematic review with integrative reviews objectives to assess, critique, and synthesize the literature of interest. Whereas systematic reviews require specifically formulated research questions, the semi-systematic review allows to conduct reviews in a more inclusive manner. [73]

A predetermined research strategy and an openness of the process are required in performing a semi-systematic literature review. The researcher is encouraged to develop a research strategy to ensure that the literature covers the points of interest set by the research questions. [73]

5.3 Search strategy

First, the decision was made to use articles, conference papers, journals, and e-prints available on the internet as research material. The following databases were used to

search for literature:

- Scopus
- IEEEExplore Digital Library
- arXiv.

The initial search is performed in Scopus, following IEEE, and lastly arXiv. The reason for using Scopus as the first choice is that it includes abstracts and indexed metadata from a large number of publishers. arXiv is a free e-print archive that is often used to publish pre-prints of papers before they are published. The papers published in arXiv and later published in journals or conferences exhibit the same level of influence [89]. Also, the papers in arXiv are found to have a considerable citation advantage in databases such as Web of Science, Scopus, and Google Scholar [85].

When searching from Scopus, only articles and conference papers are included. The search targets the title, abstract, and keywords. Only papers written in English language are considered. In the case of IEEE, each term in the query targets all metadata. In arXiv, the search is focused on all fields for all search terms. The databases with the respective search queries are presented in Table 5.1.

Database	Search query
Scopus	TITLE-ABS-KEY (("shop problem" OR "shop scheduling") AND ("graph neural" OR "graph attention")) AND (LIMIT-TO (DOCTYPE, "ar") OR LIMIT-TO (DOCTYPE, "cp")) AND (LIMIT-TO (LANGUAGE, "English"))
IEEEExplore	("All Metadata":"shop scheduling" OR "All Metadata":"shop problem") AND ("All Metadata":"graph neural" OR "All Metadata":"graph attention")
arXiv	all="shop problem"; AND all="graph attention"; OR all="shop problem"; AND all="graph neural"; OR all="shop scheduling"; AND all="graph attention"; OR all="shop scheduling"; AND all="graph neural"

Table 5.1. Used databases and search queries.

Firstly, results of the initial query on the first database are collected. Then, the following search on the second database is performed, with a possible removal of all duplicates that appear in the results of the previous searches. The results with duplicates removed are then added to the total results. This process is continued until the searches for all the chosen databases are finished.

5.4 Selection Criteria

Only accessible papers are included in the screening. All the other papers resulting from the queries are screened for their problem formulation and methodologies. The problem the papers focus on must be a JSSP or one of its variants and the methodology must include a GAT architecture in solving the described problem. Studies applying attention-based transformer[78] models are not considered in this work. The papers that fulfill these requirements are examined fully. After a full inspection the decision is made whether to include them in the literature review.

5.5 Literature Analysis

In a semi-systematic literature review, content analysis is often employed to identify, analyze, and report themes found in the literature [73]. The material was read through, and notes were taken during the reading process, especially about the methods and techniques applied in the system architecture.

The content analysis was guided by research questions and the theoretical basis. The different ways and what types of graphs were utilized should be included, along with any graph features. How ML and GATs were applied and what was their role, and any other techniques, both recurring and unique to the study being reviewed are noted. The system performance measurements are viewed to find ways to compare them, but given limited attention, as the goal is not to provide overarching measurement data, and the methods in measuring and reporting vary.

6. RESULTS

The results of the literature review are presented in this chapter, starting from the number and type of different scheduling problems found and chosen for the review. Then any possible motivations are mentioned, followed by how graphs are utilized in the studies. Next are how GAT and additional attention-related techniques are employed. This is followed by mentioning any other methods used in the studies. The system training procedures are stated, and selected studies are presented for comparison. The list of the papers included in the review can be viewed in Table 6.3.

6.1 Literature Search Results

The searches were conducted in April of 2024. The search in Scopus yielded 51 papers. Out of these results, 8 relevant papers were identified. IEEE search resulted in a total of 51 papers. After 26 duplicates were removed from the results and the rest were inspected, 2 additional papers were recognized. The queries in arXiv revealed a total of 11 papers. Following the closer examination, 5 suitable papers were distinguished. 15 papers were chosen for the full examination. Finally, a total of 14 papers were included in the review. The search results are summarized in Table 6.1.

Database	Total sources	Selected sources
Scopus	51	8
IEEEExplore	51	2
arXiv	11	4

Table 6.1. Literature search results.

Most problems of concern were JSSP and FJSSP, with FJSSP being the problem of highest interest. One of the papers focusing in solving a JSSP included an uncertainty factor in the operation durations [41]. One FJSSP formulation included transportation resources and constraints in transportation times [59]. DJSSP, HFSSP, and OSSP were covered in only one paper for each. Paper solving a DJSSP considers machine breakdowns and stochastic job arrival times as the dynamic events. OSSP was formulated as a multi-machine problem with no preemption, that is, operations are processed uninterrupted. The distribution of different problems is shown in Table 6.2.

Problem	#
JSSP	5
FJSSP	6
DJSSP	1
HFSSP	1
OSSP	1

Table 6.2. *Distribution of surveyed papers by problems.*

All the studied papers used makespan as the performance metric for their systems. The proposed solutions were end-to-end systems that take a scheduling problem as an input, train a model with ML by only using the problem, and produce a solution as a final output with no need for any manual feature extraction.

6.2 Real World Motivations

All reviewed papers acknowledge the importance of solving various combinatorial optimization problems and their application to many real-world problems and factors of interest in industry such as profitability, productivity, efficiency of production and delivery, and optimization of resource utilization, but only few suggest any actual scenario.

Only one paper [55] introduced a real-world scenario as a point of focus throughout the paper; a scheduling system for Internet of Vehicles modeled as a FJSSP, utilizing the network aspects with a multi-agent approach. Two papers suggested a possible situation on a superficial, conceptual level where the proposed method would serve as a solution; an Industry 4.0 cloud manufacturing system, which combines distributed manufacturing resources and virtualization into a shared cloud platform [74], and scheduling of automated guided vehicles as a part of a flexible manufacturing system [59].

6.3 General System Architecture

The overall architecture was largely similar across every system. A problem instance is first converted into a graph representation and the feature vectors are constructed. The graph representation is input into the reinforcement learning agent network, which consists of feature extraction and decision-making units. The feature extraction is conducted using GAT layers and the resulting embeddings are forwarded into the decision-making unit. As the action is executed, the reward for the action is generated, the state transitions into the next state, and the graph features are updated. The system tests for some end criterion. If the criterion is met, the system outputs a solution. Otherwise, the new state is input to the next process cycle. Figure 6.1 illustrates the general view

of a system.

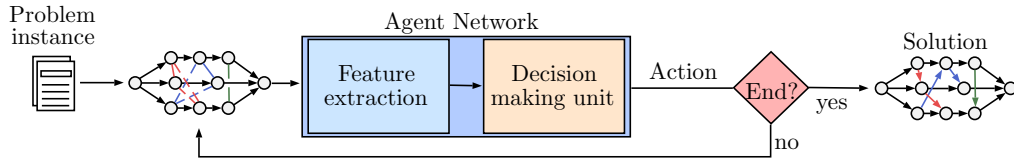


Figure 6.1. Overall architecture.

Ref	Problem	GNN	Graph type	RL algorithm	Other
[93]	JSSP	GAT	disjunctive	AC, PPO	
[39]	JSSP	GATv2	disjunctive	AC, PPO	multi-agent
[95] ¹	JSSP	GAT+GIN	disjunctive	n -step REINFORCE	improvement heuristic
[96]	JSSP	GAT	disjunctive (bidirectional)	entropy regularized REINFORCE	improvement heuristic
[41] ²	JSSP (stochastic)	GATv2	disjunctive (bidirectional)	AC, PPO	rewiring
[74] ³	FJSSP	GAT	heterogeneous	AC, PPO	
[81] ⁴	FJSSP	GAT	disjunctive	AC, PPO	dual attention
[59] ⁵	FJSSP	GAT	heterogeneous (vehicles)	REINFORCE	
[98]	FJSSP	GAT	disjunctive	SAC, D5QN	multi-agent
[24]	FJSSP	GATv2	heterogeneous (job nodes)	AC, PPO	Bayesian optimization
[55]	FJSSP	GAT	disjunctive	AC, PPO	multi-agent
[51]	DJSSP	GAT	disjunctive	AC, PPO	
[99]	HFSSP	GAT	heterogeneous	AC, PPO	semantic attention
[50]	OSSP	GAT	undirected	AC, REINFORCE	discount memory

Table 6.3. List of reviewed papers and the methods used.

¹<https://github.com/zcaicaros/L2S>

²<https://github.com/jolibrain/wheatley/>

³<https://github.com/songwenas12/fjsp-drl>

⁴<https://github.com/wrqccc/FJSP-DRL>

⁵<https://github.com/msh0576/FJSPT-Scheduler>

6.4 Graph Utilization

Graph structures were used to represent scheduling states in every proposed system. The most common way to present the scheduling problem was a disjunctive graph, which was used by 9 of the studies. 4 studies applied heterogeneous graphs to model the relationships between operations and machines. Heterogeneous graphs were used to solve FJSSPs and HFSSP. In addition, vehicle nodes were also used as a third node type to solve a particular problem formulation involving transport constraints [59]. Neither of these were used in solving OSSPs, which employed an undirected graph to present an unscheduled problem that was transformed into a DAG during the solving process to equal as a schedule.

The heterogeneous graph was deemed necessary to include different types of nodes to better express the difference between operations and machines. The heterogeneous representation is a major factor in reducing the number of edges in contrast to the disjunctive graph by reducing the number of possible actions. Presenting an FJSSP as a disjunctive graph is more challenging as operations are processable by more than one machine. This would require much more disjunctive edges in the graph.

In addition to operation and machine nodes, one study [24] included job nodes in the heterogeneous graph. This further increased the state information provided by the graph. Also, this greatly decreased the action space by removing irrelevant nodes and edges and worked as a measure against actions that lead to unsatisfactory schedules.

Multiple Graphs

The Internet of Vehicles service modeled as an FJSSP separated the vehicle location network and service request scheduling state into two different graphs [55]. The vehicle graph has the interconnected vehicles as nodes. The service request scheduling graph was a disjunctive graph with operations as nodes and the vehicles as disjunctive edges.

Method of obtaining information from the disjunctive graph by dividing it into two subgraphs by the node connection types was presented [95]. The subgraphs retain the same nodes but one has the edges that connect the operation nodes of the same job and the other has the conjunctive arcs that connect the operation nodes belonging to the same machine.

A message-passing scheme using dual disjunctive graph was applied to leverage the topological information [96]. In this method, the additional disjunctive graph had backward oriented arcs, in other words, the arcs are pointing from sink towards source. Another important detail was the different operation node features. This was deemed crucial in providing supplementary information of the node neighborhood from both directions.

Graph Rewiring

Another manner of using backward orientation was graph rewiring that was employed as a way to bring information from the future to the present, in a sense [41]. The motivation was to have the agent make choices pertaining to the possible future conflicts in the scheduling. 'Conflict' refers to the jobs requiring the same machine in the JSSP. This method included removing unnecessary edges, then adding reversed arcs that are distinguishable as a different type of connection between nodes, implemented as an edge feature.

Graph Features

Different studies incorporated various different attributes that were realized as feature vectors in the graph implementation. The number of features used varied from 2 to 26. High number of features increases computation and was suspected to have an effect to the time it takes to produce a solution, particularly for large problems. The following lists contain the different features used in the reviewed papers, categorized by problem and part of the problem representation which the features relate to. The number of total features in each system are summarized in Table 6.4.

JSSP		FJSSP		DJSSP		HFSSP		OSSP	
Ref	#	Ref	#	Ref	#	Ref	#	Ref	#
[93]	2	[74]	10	[51]	5	[99]	7	[50]	2
[39]	10	[81]	26						
[95]	3	[59]	15						
[96]	6	[98]	20						
[41]	6	[24]	7						
		[55]	-						

Table 6.4. Number of graph features found in each study.

Features used for JSSP are

- Operation availability, scheduling statuses, processing and completion statuses.
- (Remaining) processing time, remaining job processing time, and remaining time to total job time ratio.
- Earliest possible starting and completion point, and the latest possible starting point.
- Number of remaining operations in the job, processing machine identifier.
- Operation duration as a probability distribution (uncertainty in operation duration).

Features used for operations in FJSSP are

- Operation availability, scheduling statuses, processing and completion statuses.
- (Remaining) processing time, minimum processing time of any machine, average processing time of all machines, total processing time of all machines, and shortest possible processing time among all machines.
- Completion time of the job, starting time of the job, estimated start time, shortest possible completion time estimation, and remaining time of all operations in the job.
- Total number of jobs, number of unscheduled operations in the job, and number of currently completed operations in the job.
- Number of neighboring machines (in heterogeneous graphs), number of machines that operation can be processed on.
- Sum of average processing time of unscheduled operations in the job, the amount of time passed since operation finished processing.
- Ratios such as
 - Completed operations to total number of operations in the job.
 - Average completed operations to total number of operations.
 - Average completed jobs to total number of jobs.

Mostly found with heterogeneous graphs, the FJSSP machine features are

- Number of machines, machine availability, the moment when machine becomes available, and amount of time the machine has been available.
- Remaining processing time, minimum processing time of any operation, and average processing time of operations that the machine can process.
- Number of unscheduled or eligible operations that the machine can process.
- Number of neighboring operations (in heterogeneous graphs).
- Completion time of the last operation or last assigned operation.
- Utilization rate, average utilization rate, and utilization rate as active time to total processing time ratio.

FJSSP operation-machine pair features are

- Processing time, processing time as an edge feature.
- Ratio of processing time to processing time of remaining jobs.
- Various ratios of processing time to the maximum processing times of
 - Operations

- Operation candidates of the machine
- Unscheduled operations
- Unscheduled operation candidates of the machine
- Processing times of compatible pairs.
- Sum of operation and machine waiting times.

Features for FJSSPs used to model problems with vehicles are

- Availability, number of available vehicles, vehicle location, vehicle speed (in Internet of Vehicles).
- Number of neighboring vehicles as an operation feature, number of neighboring operations as a vehicle feature.
- Processing time (operation-vehicle edge feature).
- Amount of time the vehicle has been available.

The HFSSP operation features are

- Operation scheduling status.
- Processing time.
- Number of neighboring nodes.
- Estimated completion time.

And along with the machine features

- Neighboring operation nodes.
- Machine availability.
- Utilization rate as the total sum of processing time.

DJSSP operation features are

- Operation scheduling and completion status.
- Processing time, remaining processing time, estimated completion time.
- Number of unprocessed operations.

OSSP operation features are

- Processing time.
- Machine-job encoding.

6.5 GAT Utilization

In every system, GATs were utilized to automate extraction of useful information and features from the input graph representation of the problem state. The system architectures were designed to be size-agnostic to solve a problem of any size. The original version of GAT was the common choice with only three systems using the more expressive GATv2.

The papers which mentioned the amount of graph attention layers, had 2-4 GAT layers, 2 being the most common. Increasing the number of layers was reported to include information from remote nodes in the message-passing [51]. A test between 2 and 3 layers was conducted with 2 layers achieving better results [51]. 2 layers were considered enough to learn the underlying relationships and features, in addition to having to perform less computation. The number of attention-heads used was 1-8 but was similarly left unmentioned in some of the studies. A higher number of heads was noted to accelerate the learning of finding shorter schedules [95] [96].

GAT was not always used to create embeddings of all the graph feature information. In a study [74] utilizing a heterogeneous graph, machine node features and operation node features were input to two separate neural networks. The machine node embeddings were produced employing GAT, and operation node embeddings with several MLPs with ELU activation function. The motivation for using GAT for machine nodes was to find a way for the system to recognize operations that are expected to be processed sooner as more important than those processed later. In this study, attention mechanism was not deemed useful or convenient for extracting information from different edge types such as arcs to and from other operation nodes and operation-machine edges. This was a somewhat contrasting view with the studies utilizing backward oriented arcs as a source of information. Those studies utilized disjunctive graphs, however.

A dual-attention network [81] with two separate GATs with some dissimilarities and with their own trainable parameters was used. One network was for the operation features and the other for the machine features. GAT for operation features was much like the original GAT implementation while the one for machine features had two different weight matrices, and additional features of the unscheduled operations the machines are competing to process. Attention mechanism was used to express the intensity of competition between machines. Another study [41] had two GATs for disjunctive graph and backward oriented disjunctive graph. However, the GAT structure was the same in both.

Including induced point attention mechanism [47] into the graph attention calculation was proposed to reduce the computational complexity in a multi-agent environment [39]. Induced points are randomly initialized trainable parameters and were part of the GAT layer in this particular system. Induced points also constitute intermediate aggregation.

Computing over the set of unprocessed nodes had a quadratic complexity but with the addition of induced points the computation complexity was lowered to the number of points times the operations.

The system for solving HFSSP [99] employed semantic-level attention accompanied by node-level attention. Semantic-level attention attempts to learn how important different meta-paths are [84]. A meta-path is a sequence of nodes that are joined with different edge types.

In one of the proposed systems [81], the features of the operations and the features of the machines were input into two different graph attention subsystems. This was done to avoid the attention mechanism to relate every operation with each other and to express the competitiveness of the machines in a FJSSP to properly model the machine prioritization. The attention subsystem for operations attempted to find the most important operations based on their features and relate an operation with its preceding and next operations. In the case of machines, the competitiveness of machines over same eligible operations was calculated as attention coefficients. Both subsystems produced new sets of features, and these were then combined and passed to the decision-making unit.

6.6 Accompanying Techniques

This section assembles and shortly describes other ML techniques found in the reviewed systems.

Deep Reinforcement Learning

The learning was conducted with reinforcement learning using deep neural networks, or *deep reinforcement learning* (DRL), in every system. The learning agent's design was similar in many systems. This comprised of feature extraction process using GNN and the decision-making process. The agent takes the graph representation of the problem as an input, selects an action and updates the policy, and is rewarded depending on the action. A policy is the set of actions and their associated probabilities.

In multi-agent reinforcement learning the same environment is shared by multiple agents. Multiple agents were utilized to model machines as competing agents [39], separate job scheduling and vehicle scheduling agents [55], and collaborative agents where the agents exchange information with each other.

Learning Algorithms

Actor-critic (AC) algorithm incorporates policy network as the actor and value network as the critic. In the reviewed systems using AC, the portion of the system responsible

for action selection and updating policy operated as the actor, and value calculation as the critic. *Soft actor-critic* (SAC) [35] is an off-policy variation of AC algorithm, that attempts to learn a policy to maximize reward while choosing actions as randomly as possible. Off-policy learning can exploit information learned by earlier policies.

Other learning algorithms exploited in the systems were *proximal policy algorithm* (PPO), REINFORCE and its modifications n -step REINFORCE and entropy regularized REINFORCE, and Q-learning. REINFORCE [87] is a policy gradient method that estimates the gradient of the return value and optimizes the policy based on this. The n -step REINFORCE [95] algorithm is based on the REINFORCE algorithm but updates the policy periodically every n steps. This method helped with sparse rewards and to avoid memory issues associated with longer training process thus allowing longer training episodes. Entropy regularized REINFORCE [96] expands the n -step REINFORCE algorithm by adding a regularization that stems from the policy's entropy. Entropy in this connection is an indicator of uncertainty, or confidence, in the policy. Regularizing this will discourage the agent from converging to a local optimum.

Most systems utilized PPO [71], which is an algorithm that presents a concept of proximity that limits the scale of updates made to the policy. The algorithm was developed with the motivation to maximize improvement without making too great a change that inadvertently devastates performance. PPO is an on-policy algorithm, meaning that the value function is learned by the present policy.

Deep Q-network (DQN) merges deep learning and reinforcement learning, employing a Q -function to estimate values for each action. D5QN is a combination of DQN, double DQN, noisy DQN, prioritized experience replay DQN, and dueling DQN. These reduce overestimation, add noise to allow more diverse exploration, hasten training, and augment Q -value learning through competition, respectively. This was used in one system along with SAC for two collaborative agents; the agent responsible for operations was trained with SAC and the agent responsible for machines with D5QN [98].

Markov Decision Process

The problem-solving tasks were formulated as a *Markov decision process* (MDP). It is a decision-making framework describing a sequence of events as a stochastic process. The probability of an event is dependent only on the current state. The MDP formulation commonly encompasses state, action, transition, and reward.

State A state was represented as a graph and its features. Differences between states were defined by the graph topology and the feature values. In the improvement heuristic methods, the state during each step was the best solution found so far. In the systems using disjunctive graphs, graph topology is how the disjunctive edges and the conjunc-

tive arcs are connected to nodes. The topological states of heterogeneous graph were differentiated by how operation nodes were connected to other node types.

Action Actions are the set of all possible actions that can be taken in the current state. The probability of different actions was commonly calculated by accumulating the embeddings from the GAT layers and inputting them into softmax function. Since the softmax scales all the inputs to sum to 1, this can be exploited as a probability distribution. These probabilities were used as the probabilities for the actions with respect to the current state. In contrast to the sampling strategy, the greedy strategy chooses the highest value from the distribution. Sampling strategy was found to obtain better results in some of the systems solving FJSSP instances [24] [74] [81]. One study generated several policies, searched for diverse policies using Bayesian optimization, reduced the action space with dispatching rules, and employed a k -nearest neighbor algorithm to choose a feasible policy [24].

Transition Transition in the improvement heuristic was to choose a pair of operations from all the possible candidates and exchange their processing order. In disjunctive graphs, transition meant converting a disjunctive edge to a conjunctive arc and in heterogeneous graphs connection an operation to another node type.

Reward Most methods of determining rewards included difference of makespans between partial solutions, lower bound of the makespan, and sparse reward computed at completion. In improvement heuristic method, the reward was calculated as the difference between makespans of the current and next solution. In the loV problem [55], the reward combined minimizing job completion time, vehicle utilization rate, and job acceptance rate.

DRL-based Improvement Heuristic

Improvement heuristic is a method of exploring iteratively the graph neighborhood for superior solutions. A crucial limitation of this method is the necessity to assess complete solutions. This is computationally taxing, particularly for large problems. Another problem is the rules responsible for choosing the improvement; hand-crafted rules such as always choosing the best improvement might ultimately lead to poor performance. [95]

These limitations were confronted by employing DRL-based improvement heuristic that makes local decisions through a deep policy network to narrow the searched solution space [95] [96]. The motivation behind using improvement heuristic was to tackle the problem of heuristics based on construction being a long way from optimal due to the unsuitability of the graph representation strategy to form partial solutions. Reinforce-

ment learning allows further exploration to make more far-sighted policies and thus improve performance. The improvement heuristic attempts to improve the initial solution generated using simple dispatching rules. The improvement process samples a pair of operations and swaps their order of operation to produce a new solution. Essentially, the nature of the problem turns from scheduling into graph optimization. A problem with this method is the longer solving times as the number of improvement steps is increased.

Graph Isomorphism Network

One study [95] applied another type of GNN alongside with GAT. The network used is called *graph isomorphism network* (GIN) [90] that is said to exhibit a high expressive ability for non-isomorphic graphs. GIN performing well in the graph isomorphism problem, which is a problem of determining if two graphs are topologically identical, was the reason for using GIN to create topological embeddings from the disjunctive graph.

In non-identical solutions there is at least one machine that processes jobs in a different order, meaning the solutions are topologically non-identical. GIN's ability was utilized to aid the policy network to discriminate between states.

Other

Discount memory was used as a way to control how much historical information influences the training process by attempting to focus on important historical information and restricting irrelevant historical information. Discount memory accumulates correlations between embeddings and previous output. The 'discount' is an additional term with a value between 0 and 1, and is multiplied with the previous timestep, causing the accumulation mechanism to forget historical information as time progresses. [50]

6.7 Experiment Implementation and System Performance

This section elucidates the experimentation procedures employed in the studies. The properties and amount of data for training and evaluation are presented. Various studies had different baseline data in their performance presentation, making comparison between systems inconvenient. The performance is described by comparing the system performances with one another as feasible.

6.7.1 Training and Evaluation

The common scheme was to randomly generate synthetic instances for training. Although, some studies did generate training instances following the Taillard rules. Eval-

uation was mostly conducted using public datasets, such as

- Taillard [76]
- ABZ [1]
- FT [26]
- LA [46]
- SWV [75]
- ORB [6]
- YN [92]
- la [11]
- mk [40]
- Behnke [7]
- DMU [22].

Taillard and la were the most popular. The sizes of training instances varied from 5×4 to 30×20 , denoted as "jobs \times machines". The system with transportation constraints is denoted as "jobs \times machines \times vehicles". The evaluation dataset sizes varied from 5×5 to 1000×40 . Extremely large evaluation datasets were used to test the system's ability to generalize. Mentions of the time it took to train a model were scarce; from an hour to a few days depending on the problem size [41]. The different sizes of instances utilized in training and evaluation are listed in Table 6.7.

6.7.2 Performance Comparison

System performance was mostly denoted by both makespan and gap to some baseline. The baseline was either the best solution found from literature or a solution from another solver, such as OR-tools by Google. The different evaluation datasets and the way the papers reported their results varied from one another.

The systems employing improvement heuristic were the best performers in solving JSSPs [95] [96]. In addition, they show that the computational complexity of their graph embedding methods scales approximately linearly for the number of jobs and machines. For FJSSPs, the best average performance was the system utilizing Bayesian optimization and including job nodes in their heterogeneous graph representation [24]. Performance results of selected systems for JSSPs are in Table 6.5, and for FJSSPs in Table 6.6.

GAT was tested against *graph convolutional network* (GCN) and GIN. GAT produced better results than GCN in two systems [51] [98]. GAT generated solutions with lower makespan than GIN in 4 out of 5 Taillard data sets [93].

Size	[95]	[96]	[51]
15 × 15	9.3% (9.3s)	8.0% (12.6s)	17.9%
20 × 15	11.6% (10.1s)	9.9% (14.6s)	26.4%
20 × 20	12.4% (10.9s)	10.0% (17.5s)	24.0%
30 × 15	14.7% (12.7s)	13.3% (17.2s)	26.3%
30 × 20	17.5% (14.0s)	16.4% (19.3s)	35.0%
50 × 15	11.0% (16.2s)	9.6% (23.9s)	29.3%
50 × 20	13.0% (22.8s)	11.9% (24.4s)	23.8%
100 × 20	7.9% (50.2s)	6.4% (42.0s)	15.5%

Table 6.5. Performance comparison of gap to the best solution found in literature for selected JSSP systems for Taillard instances with time taken to produce a solution. The results are from the respective studies. In [51] two different Taillard instances for each size were used to test non-dynamic performance; the results from the two instances are averaged. [95][96] are the performances for 500 improvement steps.

	[74]	[81]	[98]	[24]	UB[7]
Makespan	931.45	925.40	847.13	824.97	809.17
Time	3.54s	4.77s		3.50s	

Table 6.6. Performance comparison of average makespan and average time taken to solve for selected FJSSP systems for la1-30 (vdata) instances. The results are obtained from the respective studies. UB denotes the best known solution in the literature.

6.8 Limitations

The limitations of this study are as follows. The literature search has a possibility of missing studies as this was conducted only in Scopus, IEEE, and arXiv. Other databases such as ACM could contain more literature. In addition, Scopus search results included six inaccessible studies. Another limitation is related to the content analysis of the semi-systematic literature review method. There is typically an element of subjectivity in the analysis of the literature. Particularly the portions that were chosen to be excluded as technical details might have contained noteworthy information.

Ref	Training		Evaluation		
	Min	Max	Min	Max	Dataset
[93]	15×15	20×20	15×15	50×20	Taillard
[39]	15×15	30×20	50×15	100×20	Taillard
[95]	10×10	20×15	6×6	1000×40	synthetic, Taillard,LA, ABZ,FT,YN SWV,ORB
[96]	10×10	20×15	6×6	100×20	Taillard,LA, ABZ,FT,YN SWV,ORB
[41]	6×6	15×15	6×6	100×20	Taillard
[74]	10×5	20×10	10×5	40×10	synthetic, mk, la
[81]	10×5	20×10	10×5	40×10	synthetic, mk, la
[59]	$10 \times 5 \times 5$	$20 \times 10 \times 10$	$30 \times 15 \times 15$	$50 \times 25 \times 25$	synthetic
[98]	10×5	20×5	10×5	30×20	synthetic, mk, la
[24]	5×4	15×13	10×5	100×60	la,Behnke
[55]			200×50	1000×50	
[51]	6×6	10×10	6×6	100×20	Taillard, DMU,synthetic
[99]	10×5	15×10	10×5	60×5	synthetic
[50]	5×5	10×10	5×5	10×10	synthetic

Table 6.7. Sizes of training and evaluation instances.

7. CONCLUSION

JSSP is a classic combinatorial optimization problem encountered in many real-world scenarios. A rich collection of algorithms for solving JSSPs exists, and with the rise in interest in AI, along with the development of increasingly powerful computer hardware, the feasibility of employing ML to undertake combinatorial optimization problems has been a target of interest. As a JSSP can be expressed in graph form, GNNs have been applied in extracting information from the graph representation and leveraging this in solving the scheduling problem.

This study was conducted as a semi-systematic literature review on systems solving various job shop scheduling problems by utilizing GAT architecture. GAT was found to be valuable in solving different job shop problems and useful together with a number of other ML methods. The variations of JSSPs that were covered were its flexible (FJSSP) and dynamic (DJSSP) variants, HFSSP, and OSSP. The most studied problems in the reviewed papers were JSSP and FJSSP. The literature search was conducted on Scopus, IEEE, and arXiv databases in April 2024. A total of 14 papers were chosen to be reviewed.

GAT is a type of GNN that uses attention mechanism to calculate how important a node in a graph is to another node. GAT was deemed effective in extracting important information from graphs and enhancing decision-making ability. Most common reported number of GAT layers was 2. More than 2 layers were found to perform worse in two studies and possibly increase solving time because of increased computation. The number of attention heads was between 1 and 8; more heads were found to improve how fast the learning converges. Three studies employed GATv2 and no comparative testing between GAT and GATv2 was found.

Disjunctive and heterogeneous graph representations were employed to model the problem states, and one study used an undirected graph. Heterogeneous graph was more popular in FJSSPs as it allows to include machines as another node type to effectively reduce complexity by lowering the number of disjunctive edges. Techniques such as graph rewiring, turning the direction of the conjunctive arcs, and dividing graph into subgraphs showed how much information a graph structure can provide, but also how the information is very complex. A method to reduce this complexity was to add a job

node type and remove the unneeded nodes and edges. Several different node and edge features were discovered from the literature. The number and nature of features enriches the available information but possibly increases the time to produce a solution.

DRL was used in every system to explore the graph structure and find feasible policies that minimized makespan. Three studies utilized multiple agents. The scheduling problems were formulated as an MDP. The learning algorithms employed were AC, SAC, PPO, Q-learning, and REINFORCE and its modifications n -step REINFORCE and entropy regulated REINFORCE.

Most systems demonstrated an ability to generalize to solve large problems while only being trained with smaller problem sizes. The best performing system for a FJSSP utilized improvement heuristic method, and showed the computational complexity of their methods is linear for the number of jobs and machines. For a JSSP, the system with a job node type and removal of unneeded nodes and edges outperformed others especially in larger instances. GAT performed better in comparison than GCN and GIN.

As there are no strict standards on how the performance should be measured or reported, or which datasets should be used, comparing the results provided difficult and were presented in limited fashion in this work. The problem formulations were simple and did not accurately represent a real situation. True real-world problems with all the possible features are very complex to model. In addition, the complexity would require a very long training time if one were to use reinforcement learning. Any change in the real-world application would necessitate changing the problem definition, and the model would have to be retrained. The test conducted against GIN was done in limited fashion and not accepting this as a conclusive comparison is encouraged.

To summarize, GAT is a capable architecture for systems solving JSSP variants and the suitable number of layers and attention heads should be tested on case basis. GAT can extract useful information from both disjunctive and heterogeneous graphs and is suited to be used with various ML techniques. The accompanying ML techniques have a more significant impact on the system performance than small changes in the GAT parameters.

The limitations were the subjectivity in the literature analysis, inaccessible studies and possible missing studies. A possible future study could be to see if there is any notable difference in performance between the static attention of GAT and the dynamic attention of GATv2 in extracting information from scheduling problem representations.

REFERENCES

- [1] Adams, J., Balas, E. and Zawack, D. The shifting bottleneck procedure for job shop scheduling. *Management Science* 34.4 (1988), pp. 391–401.
- [2] Aggarwal, C. C. *Neural Networks and Deep Learning: A Textbook*. Springer, 2018.
- [3] Akyol, D. E. and Bayhan, G. M. A review on evolution of production scheduling with neural networks. *Computers and Industrial Engineering* 53.1 (2007), pp. 95–122. DOI: 10.1016/j.cie.2007.04.006.
- [4] Alpaydin, E. *Introduction to Machine Learning*. 3rd edition. MIT Press, 2014.
- [5] Alvarez-Valdes, R., Fuertes, A., Tamarit, J., Giménez, G. and Ramos, R. A heuristic to schedule flexible job-shop in a glass factory. *European Journal of Operational Research* 165.2 (2005), pp. 525–534. DOI: 10.1016/j.ejor.2004.04.020.
- [6] Applegate, D. and Cook, W. Computational study of the job-shop scheduling problem. *ORSA journal on computing* 3.2 (1991), pp. 149–156. DOI: 10.1287/ijoc.3.2.149.
- [7] Behnke, D. and Geiger, M. J. Test instances for the flexible job shop scheduling problem with work centers. *Arbeitspapier / Research Paper / Helmut-Schmidt-Universität, Lehrstuhl für Betriebswirtschaftslehre, insbes. Logistik-Management* (2012). DOI: 10.24405/436.
- [8] Bengio, Y., Lodi, A. and Prouvost, A. Machine learning for combinatorial optimization: A methodological tour d’horizon. *European Journal of Operational Research* 290.2 (2021), pp. 405–421.
- [9] Błażewicz, J., Ecker, K. H., Pesch, E., Schmidt, G. and Weglarz, J. *Handbook on Scheduling: From Theory to Applications*. Springer Berlin Heidelberg, 2007.
- [10] Bouras, A., Ghaleb, M., Suryahatmaja, U. and Salem, A. The airport gate assignment problem: A survey. *Scientific World Journal* 2014 (2014). DOI: 10.1155/2014/923859.
- [11] Brandimarte, P. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research* 41.3 (1993), pp. 157–183. DOI: 10.1007/BF02023073.
- [12] Brody, S., Alon, U. and Yahav, E. *How Attentive are Graph Attention Networks?* 2021. arXiv: 2105.14491 [cs.LG].
- [13] Bronstein, M. M., Bruna, J., Cohen, T. and Veličković, P. *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*. 2021. arXiv: 2104.13478 [cs.LG].
- [14] Brucker, P. *Scheduling Algorithms*. 5th edition. Springer Berlin, 2007.

- [15] Burke, E., McCollum, B., Meisels, A., Petrovic, S. and Qu, R. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research* 176.1 (2007), pp. 177–192. DOI: 10.1016/j.ejor.2005.08.012.
- [16] Çalış, B. and Bulkan, S. A research survey: review of AI solution strategies of job shop scheduling problem. *Journal of Intelligent Manufacturing* 26.5 (2015), pp. 961–973. DOI: 10.1007/s10845-013-0837-8.
- [17] Clevert, D.-A., Unterthiner, T. and Hochreiter, S. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. 2016. arXiv: 1511.07289 [cs.LG].
- [18] Cohen, J. *How neurons learn: The last frontier*. 2017. URL: <https://www.universityofcalifornia.edu/news/how-neurons-learn-last-frontier> (visited on 05/29/2024).
- [19] Creemers, S. Minimizing the expected makespan of a project with stochastic activity durations under resource constraints. *Journal of Scheduling* 18.3 (2015), pp. 263–273. DOI: 10.1007/s10951-015-0421-5.
- [20] Czibula, O., Gu, H., Hwang, F.-J., Kovalyov, M. and Zinder, Y. Bi-criteria sequencing of courses and formation of classes for a bottleneck classroom. *Computers and Operations Research* 65 (2016), pp. 53–63. DOI: 10.1016/j.cor.2015.06.010.
- [21] Davidović, T. and Crainic, T. Parallel Local Search to schedule communicating tasks on identical processors. *Parallel Computing* 48 (2015), pp. 1–14. DOI: 10.1016/j.parco.2015.04.002.
- [22] Demirkol, E., Mehta, S. and Uzsoy, R. Benchmarks for shop scheduling problems. *European Journal of Operational Research* 109.1 (1998), pp. 137–141. DOI: 10.1016/S0377-2217(97)00019-2.
- [23] Deo, N. *Graph Theory with Applications to Engineering & Computer Science*. Dover, 1974.
- [24] Echeverria, I., Murua, M. and Santana, R. *Solving the flexible job-shop scheduling problem through an enhanced deep reinforcement learning approach*. 2024. arXiv: 2310.15706 [cs.AI].
- [25] Fazel Zarandi, M. H., Sadat Asl, A. A., Sotudian, S. and Castillo, O. A state of the art review of intelligent scheduling. *Artificial Intelligence Review* 53.1 (2020), pp. 501–593.
- [26] Fisher, H. Probabilistic learning combinations of local job-shop scheduling rules. *Industrial Scheduling* (1963), pp. 225–251.
- [27] Foo, Y.-P. S. and Takefuji, Y. Integer linear programming neural networks for job-shop scheduling. *IEEE 1988 International Conference on Neural Networks*. Vol. 2. 1988, pp. 341–348.

- [28] Foo, Y.-P. S. and Takefuji, Y. Stochastic neural networks for solving job-shop scheduling. I - Problem representation. *IEEE 1988 International Conference on Neural Networks*. Vol. 2. 1988, pp. 275–282.
- [29] Foo, Y.-P. S. and Takefuji, Y. Stochastic neural networks for solving job-shop scheduling. II - Architecture and simulations. *IEEE 1988 International Conference on Neural Networks*. Vol. 2. 1988, pp. 283–290.
- [30] Gao, J., Sun, L. and Gen, M. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers and Operations Research* 35.9 (2008), pp. 2892–2907. DOI: 10.1016/j.cor.2007.01.001.
- [31] Gao, K., Suganthan, P., Chua, T., Chong, C., Cai, T. and Pan, Q. A two-stage artificial bee colony algorithm scheduling flexible job-shop scheduling problem with new job insertion. *Expert Systems with Applications* 42.21 (2015), pp. 7652–7663. DOI: 10.1016/j.eswa.2015.06.004.
- [32] Goodfellow, I., Bengio, Y. and Courville, A. *Deep Learning*. MIT Press, 2016. URL: <http://www.deeplearningbook.org>.
- [33] Gori, M., Monfardini, G. and Scarselli, F. A new model for learning in graph domains. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005*. Vol. 2. 2005, pp. 729–734.
- [34] Gupta, A. and Sivakumar, A. Job shop scheduling techniques in semiconductor manufacturing. *International Journal of Advanced Manufacturing Technology* 27.11-12 (2006), pp. 1163–1169. DOI: 10.1007/s00170-004-2296-z.
- [35] Haarnoja, T., Zhou, A., Abbeel, P. and Levine, S. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. 2018. arXiv: 1801.01290 [cs.LG]. URL: <https://arxiv.org/abs/1801.01290>.
- [36] Hamilton, W. L. Graph Representation Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 14.3 (), pp. 1–159.
- [37] Haykin, S. S. *Neural Networks and Learning Machines*. 3rd edition. Pearson, 2009.
- [38] Hopfield, J. J. and Tank, D. W. “Neural” computation of decisions in optimization problems. *Biological cybernetics* 52.3 (1985), pp. 141–152.
- [39] Hu, Z., Chen, Y., Li, H. and Deng, X. Research on Job Shop Scheduling Problem Based on Multi-Agent Reinforcement Learning. 2023, pp. 1209–1213. DOI: 10.1145/3652628.3652826.
- [40] Hurink, J., Jurisch, B. and Thole, M. Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spektrum* 15.4 (1994), pp. 205–215. DOI: 10.1007/BF01719451.
- [41] Infantes, G., Roussel, S., Pereira, P., Jacquet, A. and Benazera, E. *Learning to Solve Job Shop Scheduling under Uncertainty*. 2024. arXiv: 2404.01308 [cs.AI].
- [42] Iturriaga, S., Nasmachnow, S., Luna, F. and Alba, E. A parallel local search in CPU/GPU for scheduling independent tasks on large heterogeneous computing

- systems. *Journal of Supercomputing* 71.2 (2015), pp. 648–672. DOI: 10.1007/s11227-014-1315-6.
- [43] Jain, A. S. and Meeran, S. Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research* 113.2 (1999), pp. 390–434. ISSN: 0377-2217.
- [44] Kelleher, J. D. *Deep Learning*. MIT Press, Sept. 2019.
- [45] Kulathunga, N., Ranasinghe, N. R., Vranceanu, D., Kinsman, Z., Huang, L. and Wang, Y. Effects of nonlinearity and network architecture on the performance of supervised neural networks. *Algorithms* 14.2 (2021).
- [46] Lawrence, S. Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (supplement). *Graduate School of Industrial Administration, Carnegie-Mellon University* (1984).
- [47] Lee, J., Lee, Y., Kim, J., Kosiosek, A. R., Choi, S. and Teh, Y. W. *Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks*. 2019. arXiv: 1810.00825 [cs.LG]. URL: <https://arxiv.org/abs/1810.00825>.
- [48] Lenstra, J. K. and Rinnooy Kan, A. H. Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics* 4.C (1979), pp. 121–140.
- [49] Lenstra, J. K., Rinnooy Kan, A. H. and Brucker, P. Complexity of Machine Scheduling Problems. *Annals of Discrete Mathematics* 1 (1977), pp. 343–362.
- [50] Li, J., Dong, X., Zhang, K. and Han, S. Solving Open Shop Scheduling Problem via Graph Attention Neural Network. *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*. Vol. 2020-November. 2020, pp. 277–284. DOI: 10.1109/ICTAI50040.2020.00052.
- [51] Liu, C.-L., Tseng, C.-J. and Weng, P.-H. Dynamic Job-Shop Scheduling via Graph Attention Networks and Deep Reinforcement Learning. *IEEE Transactions on Industrial Informatics* (2024), pp. 1–11. DOI: 10.1109/TII.2024.3371489.
- [52] Liu, Z., Xiao, L. and Tian, J. An activity-list-based nested partitions algorithm for resource-constrained project scheduling. *International Journal of Production Research* 54.16 (2016), pp. 4744–4758. DOI: 10.1080/00207543.2015.1065353.
- [53] Liu, Z., Wang, Y., Liang, X., Ma, Y., Feng, Y., Cheng, G. and Liu, Z. A graph neural networks-based deep Q-learning approach for job shop scheduling problems in traffic management. *Information Sciences* 607 (2022), pp. 1211–1223. DOI: 10.1016/j.ins.2022.06.017.
- [54] Lo, Z.-P. and Bavarian, B. Multiple job scheduling with artificial neural networks. *Computers and Electrical Engineering* 19.2 (1993), pp. 87–101. DOI: 10.1016/0045-7906(93)90039-T.
- [55] Lu, Y., Zhang, P., Duan, Y., Guizani, M., Wang, J. and Li, S. Dynamic Scheduling of IoV Edge Cloud Service Functions Under NFV: A Multi-Agent Reinforcement

- Learning Approach. *IEEE Transactions on Vehicular Technology* 73.4 (2024), pp. 5730–5741. DOI: 10.1109/TVT.2023.3333291.
- [56] Misiakos, P., Wendler, C. and Püschel, M. *Learning DAGs from Data with Few Root Causes*. 2024. arXiv: 2305.15936 [cs.LG].
- [57] Mitchell, T. M. *Machine Learning*. McGraw Hill, 1997.
- [58] Mohan, J., Lanka, K. and Rao, A. N. A review of dynamic job shop scheduling techniques. *Procedia Manufacturing*. Vol. 30. 2019, pp. 34–39.
- [59] Moon, S., Lee, S. and Park, K. Graph-based Reinforcement Learning for Flexible Job Shop Scheduling with Transportation Constraints. *49th Annual Conference of the IEEE Industrial Electronics Society, IECON 2023, Singapore, October 16-19, 2023*. 2023, pp. 1–6. DOI: 10.1109/IECON51785.2023.10312647.
- [60] Murphy, K. P. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022. URL: probml.ai.
- [61] NobelPrize.org. *John Hopfield – Facts – 2024*. URL: <https://www.nobelprize.org/prizes/physics/2024/hopfield/facts/> (visited on 12/02/2024).
- [62] Nwankpa, C., Ijomah, W., Gachagan, A. and Marshall, S. *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*. 2018. arXiv: 1811.03378 [cs.LG].
- [63] Ouali, Y., Hudelot, C. and Tami, M. *An Overview of Deep Semi-Supervised Learning*. 2020. arXiv: 2006.05278 [cs.LG]. URL: <https://arxiv.org/abs/2006.05278>.
- [64] Pinedo, M. L. *Scheduling: Theory, Algorithms, and Systems*. 3rd edition. Addison-Wesley, 2008.
- [65] PyG-Team. *GATv2Conv, PyTorch Geometric Documentation*. 2024. URL: https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.GATv2Conv.html (visited on 05/25/2024).
- [66] Ramadevi, B., Kasi, V. R. and Bingi, K. Fractional ordering of activation functions for neural networks: A case study on Texas wind turbine. *Engineering Applications of Artificial Intelligence* 127 (2024).
- [67] Riise, A., Mannino, C. and Burke, E. Modelling and solving generalised operational surgery scheduling problems. *Computers and Operations Research* 66 (2016), pp. 1–11. DOI: 10.1016/j.cor.2015.07.003.
- [68] Roy, B. and Sussmann, B. Les problèmes d'ordonnancement avec contraintes disjonctives. *Note D. S.* 9 (1964).
- [69] Sabuncuoglu, I. and Gurgun, B. A neural network model for scheduling problems. *European Journal of Operational Research* 93.2 (1996), pp. 288–299. DOI: 10.1016/0377-2217(96)00041-0.
- [70] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M. and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks* 20.1 (2009), pp. 61–80.

- [71] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG]. URL: <https://arxiv.org/abs/1707.06347>.
- [72] Sharma, S., Sharma, S. and Athaiya, A. ACTIVATION FUNCTIONS IN NEURAL NETWORKS. *International Journal of Engineering Applied Sciences and Technology* 04 (May 2020), pp. 310–316.
- [73] Snyder, H. Literature review as a research methodology: An overview and guidelines. *Journal of Business Research* 104 (2019), pp. 333–339. DOI: 10.1016/j.jbusres.2019.07.039.
- [74] Song, W., Chen, X., Li, Q. and Cao, Z. Flexible Job-Shop Scheduling via Graph Neural Network and Deep Reinforcement Learning. *IEEE Transactions on Industrial Informatics* 19.2 (2023), pp. 1600–1610. DOI: 10.1109/TII.2022.3189725.
- [75] Storer, R. H., Wu, S. and Vaccari, R. New search spaces for sequencing problems with application to job shop scheduling. *Management Science* 38.10 (1992), pp. 1495–1509. DOI: 10.1287/mnsc.38.10.1495.
- [76] Taillard, E. Benchmarks for basic scheduling problems. *European Journal of Operational Research* 64.2 (1993). Project Management and Scheduling, pp. 278–285. ISSN: 0377-2217. DOI: [https://doi.org/10.1016/0377-2217\(93\)90182-M](https://doi.org/10.1016/0377-2217(93)90182-M).
- [77] Van Hulle, M. A goal programming network for mixed integer linear programming: A case study for the job shop scheduling problem. *International Journal of Neural Systems* 02.03 (1991), pp. 201–209. DOI: 10.1142/S0129065791000182.
- [78] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. and Polosukhin, I. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].
- [79] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P. and Bengio, Y. *Graph Attention Networks*. 2017. arXiv: 1710.10903 [stat.ML].
- [80] Vilcot, G. and Billaut, J.-C. A tabu search algorithm for solving a multicriteria flexible job shop scheduling problem. *International Journal of Production Research* 49.23 (2011), pp. 6963–6980. DOI: 10.1080/00207543.2010.526016.
- [81] Wang, R., Wang, G., Sun, J., Deng, F. and Chen, J. Flexible Job Shop Scheduling via Dual Attention Network-Based Reinforcement Learning. *IEEE Transactions on Neural Networks and Learning Systems* 35.3 (2024), pp. 3091–3102. DOI: 10.1109/TNNLS.2023.3306421.
- [82] Wang, S., Su, H. and Wan, G. Resource-constrained machine scheduling with machine eligibility restriction and its applications to surgical operations scheduling. *Journal of Combinatorial Optimization* 30.4 (2015), pp. 982–995. DOI: 10.1007/s10878-015-9860-3.
- [83] Wang, X., Gao, L., Zhang, C. and Shao, X. A multi-objective genetic algorithm based on immune and entropy principle for flexible job-shop scheduling

- problem. *International Journal of Advanced Manufacturing Technology* 51.5-8 (2010), pp. 757–767. DOI: 10.1007/s00170-010-2642-2.
- [84] Wang, X., Ji, H., Shi, C., Wang, B., Cui, P., Yu, P. and Ye, Y. *Heterogeneous Graph Attention Network*. 2021. arXiv: 1903.07293 [cs.SI]. URL: <https://arxiv.org/abs/1903.07293>.
- [85] Wang, Z., Glänzel, W. and Chen, Y. The impact of preprints in Library and Information Science: an analysis of citations, usage and social attention indicators. *Scientometrics* 125.2 (2020), pp. 1403–1423. DOI: 10.1007/s11192-020-03612-4.
- [86] Weckman, G., Ganduri, C. and Koonce, D. A neural network job-shop scheduler. *Journal of Intelligent Manufacturing* 19.2 (2008), pp. 191–201. DOI: 10.1007/s10845-008-0073-9.
- [87] Williams, R. J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning* 8.3 (1992), pp. 229–256. DOI: 10.1023/A:1022672621406.
- [88] Wu, L., Cui, P., Pei, J., Zhao, L. and Song, L. *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer, 2022.
- [89] Xie, B., Shen, Z. and Wang, K. *Is preprint the future of science? A thirty year journey of online preprint services*. 2021. arXiv: 2102.09066 [cs.DL].
- [90] Xu, K., Hu, W., Leskovec, J. and Jegelka, S. *How Powerful are Graph Neural Networks?* 2019. arXiv: 1810.00826 [cs.LG].
- [91] Yadav, A. Attention: A cognitive approach in Sports Psychology. *International Journal For Science Technology And Engineering* (2023). DOI: 10.22214/ijraset.2023.52465.
- [92] Yamada, T. and Nakano, R. A genetic algorithm applicable to large-scale job-shop problems. *PPSN* 2 (1992), pp. 281–290.
- [93] Yang, S. Using Attention Mechanism to Solve Job Shop Scheduling Problem. *2022 2nd International Conference on Consumer Electronics and Computer Engineering (ICCECE)*. 2022, pp. 59–62. DOI: 10.1109/ICCECE54139.2022.9712705.
- [94] Zang, Z., Wang, W., Song, Y., Lu, L., Li, W., Wang, Y. and Zhao, Y. Hybrid Deep Neural Network Scheduler for Job-Shop Problem Based on Convolution Two-Dimensional Transformation. *Computational Intelligence and Neuroscience* 2019 (2019). DOI: 10.1155/2019/7172842.
- [95] Zhang, C., Cao, Z., Song, W., Wu, Y. and Zhang, J. *Deep Reinforcement Learning Guided Improvement Heuristic for Job Shop Scheduling*. 2024. arXiv: 2211.10936 [cs.LG].
- [96] Zhang, C., Cao, Z., Wu, Y., Song, W. and Sun, J. *Learning Topological Representations with Bidirectional Graph Attention Network for Solving Job Shop Scheduling Problem*. 2024. arXiv: 2402.17606 [cs.LG].

- [97] Zhang, C., Song, W., Cao, Z., Zhang, J., Tan, P. S. and Xu, C. Learning to dispatch for job shop scheduling via deep reinforcement learning. *Advances in Neural Information Processing Systems*. Vol. 2020-December. 2020.
- [98] Zhang, W., Zhao, F., Li, Y., Du, C., Feng, X. and Mei, X. A novel collaborative agent reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for flexible job shop scheduling problem. *Journal of Manufacturing Systems* 74 (2024), pp. 329–345. DOI: 10.1016/j.jmsy.2024.03.012.
- [99] Zhao, Y., Luo, X. and Zhang, Y. The application of heterogeneous graph neural network and deep reinforcement learning in hybrid flow shop scheduling problem. *Computers and Industrial Engineering* 187.C (Apr. 2024). ISSN: 0360-8352. DOI: 10.1016/j.cie.2023.109802.
- [100] Zhou, D., Cherkassky, V., Baldwin, T. and Hong, D. Scaling neural network for job-shop scheduling. *IJCNN. International Joint Conference on Neural Networks*. 1990, pp. 889–894. DOI: 10.1109/ijcnn.1990.137947.
- [101] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C. and Sun, M. *Graph Neural Networks: A Review of Methods and Applications*. 2021. arXiv: 1812.08434 [cs.LG].