Deep Learning and Deep Reinforcement Learning for Graph Based Applications

Ramin Hasibi

Thesis for the degree of Philosophiae Doctor (PhD)
University of Bergen, Norway
2024

UNIVERSITY OF BERGEN

Deep Learning and Deep Reinforcement Learning for Graph Based Applications

Ramin Hasibi

Thesis for the degree of Philosophiae Doctor (PhD)
at the University of Bergen

Date of defense: 26.01.2024

Year: 2024

Title: Deep Learning and Deep Reinforcement Learning for Graph Based Applications

Name: Ramin Hasibi

Print: Skipnes Kommunikasjon / University of Bergen

Scientific environment

The work presented in this thesis was carried at the Computational Biology Unit (CBU), Department of Informatics, Faculty of Mathematics and Natural Sciences, University of Bergen (UiB).

During my studies my main supervisor was Prof. Dr. Tom Michoel, and I was co-supervised by Associate Prof. Dr. Pekka Parviainen. Addiotionally, I have been a member of the National Research School in Bioinformatics, Biostatistics, and Systems Biology (NORBIS).

Acknowledgements

First and foremost, I would like to thank my supervisor Dr. Tom Michoel whose amazing guidance is the main reason that this thesis was made possible. From day one, he helped me with ideas and ways to tackle hurdles and inspired me to keep going even when results seemed unachievable. His efforts have kept me motivated during the four-year Ph.D. period and he has taught me what it means to be a perfect project leader.

Next, I would like to thank my wife Elham for her immense kindness and support. Without her, none of the blessings that I have in my life would be possible. I would also like to thank my family back home who I miss very dearly. Their constant words of encouragement have helped me remain strong throughout this journey. I would also like to thank my friends in Bergen who have created unforgettable memories for me that have made the living experience in Norway extra enjoyable.

Finally, I would like to thank the Department of Informatics and University of Bergen for allowing me to pursue my education and favorite research topic at this institute and for providing me with the necessary resources to do so.

Ramin Hasibi
Bergen, 2023

Abstract

Deep learning has provided state-of-the-art performance in many applications such as computer vision, text analysis, biology, etc. The success of deep learning has also helped with the emergence of deep reinforcement learning for optimal decision-making and has shown great promise, especially in optimization problems. Additionally, graphs as a mathematical representation for structured complex systems have proven to be a powerful tool for analysis and problem-solving that offer a fresh perspective on the formulation of the problem. Introducing graphs as an input modality for machine learning problems enables deep learning models to either utilize the structure of the graph in their representation learning scheme or optimize the graph structure for a downstream evaluation task. Doing so will also lead to model methods and pipelines that leverage the structural information provided by graphs to improve performance compared to traditional machine learning models. In this thesis, we introduce five different use-case applications, in the format of five research papers, that can be modeled as graphs and aim to provide novel models that address problems using deep graph representation learning and deep reinforcement learning models. Our main three application domains are bioinformatics, computer vision, and logistics.

First, we aim to address two problems in the domain of bioinformatics. In **Paper I**, we address the issue of integration of continuous omics datasets with biological networks. We introduce an auto-encoder scheme focused on representation learning of node features in biological networks and showcase the application of the designed framework in a real-world example through the imputation of missing values in an example omics dataset. **Paper II** looks at utilizing graph representation learning for processing metabolic networks. In the proposed approach, we introduce a machine learning pipeline (from feature extraction to model architecture) based on graph neural networks and evaluate the pipeline on the task of gene essentiality prediction which is a well-known application of metabolic pathway networks.

The second domain of applications is the computer vision domain specifically the problem of human gesture recognition. In **Paper III** and the follow-up **Paper IV**, we introduce a gesture recognition system that is both faster and more accurate compared to the state-of-the-art prediction of human subject gestures from mmWave Radar generated point clouds. We achieve this by modeling the input point cloud as a spatio-temporal graph and processing the created graph using the proposed graph representation learning technique. We further evaluate the system in different experimental conditions in terms of the angle of the subject with respect to sensing and propose an ensemble approach for mitigating the effect of changing the sensing angle on the performance of the model.

The last application that we address is the use of deep reinforcement learning to optimize the structure of the graphs in combinatorial optimization problems in logistics. **Paper V** introduces a general problem-independent hyperheuristic that utilizes the decision-making capability of deep reinforcement learning using a problem-independent state feature information. The proposed framework is trained on a general reward function to achieve state-of-the-art performance among popular solvers in the field of combinatorial optimization. We evaluate the performance of the proposed approach on three example routing problems as well as a scheduling problem to showcase the effectiveness

of our method in different problems.

Sammendrag

Dyplæring har gitt state-of-the-art ytelse i mange applikasjoner som datasyn, tekstanalyse, biologi, osv. Suksessen med dyp læring har også hjulpet fremveksten av dyp forsterkende læring for optimal beslutningstaking og har vist stort potensiale, spesielt i optimaliseringsproblemer. I tillegg har grafer som matematisk representasjon for strukturerte komplekse systemer vist seg å være et kraftig verktøy for analyse og problemløsning, og gitt et nytt perspektiv på formuleringen av problemet. Ved å introdusere grafer som en inputmodalitet for maskinlæringsproblemer kan dyplæringsmodeller enten bruke strukturen til grafen i sine representasjonslæringsskjema, eller optimalisere grafstrukturen i en nedstrøms evalueringsoppgave. Dette vil også føre til modellmetoder og pipelines som utnytter den strukturelle informasjonen gitt av grafer til forbedret ytelse, sammenlignet med tradisjonelle maskinlæringsmodellers kapasitet. I denne oppgaven introduserer vi fem forskjellige use-case-applikasjoner, gjennom fem forskningsartikler, som kan modelleres som grafer og tar sikte på å skape nye modeller som adresserer problemer ved bruk av dyp grafrepresentasjonslæring og dype forsterkningslæringsmodeller. Våre tre viktigste applikasjonsdomener er bioinformatikk, datasyn og logistikk.

Først tar vi sikte på å adressere to problemer innen bioinformatikk. I **Paper I** tar vi opp spørsmålet om integrering av kontinuerlige omics-datasett med biologiske nettverk. Vi introduserer et auto-koderskjema fokusert på representasjonslæring av nodefunksjoner i biologiske nettverk, og viser anvendelsen av det utformede rammeverket i et virkelighetseksempel gjennom imputering av manglende verdier i et eksempeldatasett for omics. **Paper II** ser på bruk av grafrepresentasjonslæring for å behandle metabolske nettverk. I den foreslåtte tilnærmingen introduserer vi en maskinlæringspipeline (fra funksjonsekstraksjon til modellarkitektur) basert på grafiske nevrale nettverk og evaluerer pipelinen basert på prediksjon av genessensalitet, som er en velkjent bruk av metabolske banenettverk.

Det andre domenet av applikasjoner er datasynsdomenet, spesifikt problemet med gjenkjennelse av menneskelige gester. I **Paper III**, og oppfølgingen **Paper IV**, introduserer vi et gestgjenkjenningssystem som er både raskere og mer nøyaktig enn den avanserte prediksjonen av menneskelige motivbevegelser fra mmWave Radar genererte punktskyer. Vi oppnår dette ved å modellere inngangspunktskyen som en spatiotemporal graf og å bearbeide den opprettede grafen ved bruk av den foreslåtte læringsteknikken for grafrepresentasjon. Videre evaluerer vi systemet under forskjellige eksperimentelle forhold ut ifra vinkelen til emnet med hensyn til sansing, og foreslår en ensembletilnærming for å dempe effekten av å endre sansevinkelen på ytelsen til modellen.

Den siste applikasjonen vi tar for oss er bruken av dyp forsterkningslæring for å optimalisere strukturen til grafene i kombinatoriske optimaliseringsproblemer i logistikk. **Paper V** introduserer en generell problemuavhengig hyperheuristikk som utnytter beslutningsevnen til dyp forsterkende læring, ved å bruke en problemuavhengig tilstandsfunksjonsinformasjon. Det foreslåtte rammeverket er trent på en generell belønningsfunksjon for å oppnå høykvalitets ytelse blant populære løsere innen kombinatorisk optimalisering. Vi evaluerer ytelsen til den foreslåtte tilnærmingen med tre eksempler på ruting problemer samt et planleggingsproblem, for å vise effektiviteten til metoden vår i forskjellige typer problemstillinger.

Outline

This thesis consists of an introduction and five scientific papers. Chapter 1 gives the background information needed to understand the papers. The main objectives for this thesis are provided in chapter 2. A brief summary of the papers is given in chapter 3. The papers included in this thesis (chapter 5) are:

1. **Ramin Hasibi** and Tom Michoel, (2021) *A Graph Feature Auto-Encoder for the Prediction of Unobserved Node Features on Biological Networks*, BMC Bioinformatics **22/1**,
   https://doi.org/10.1186/s12859-021-04447-3

2. **Ramin Hasibi**, Tom Michoel, and Diego A. Oyarzun, (2023) *Integration of genome-scale metabolic models and deep graph neural networks for gene essentiality prediction*, biorxive preprint available
   https://doi.org/10.1101/2023.08.25.554757

3. Dariush Salami[1], **Ramin Hasibi**[1], Sameera Palipana, Petar Popovski, Tom Michoel, and Stephan Sigg, (2022) *Tesla-Rapture: A Lightweight Gesture Recognition System From mmWave Radar Sparse Point Clouds*, IEEE Transactions on Mobile Computing **22/08**,
   https://doi.org/10.1109/TMC.2022.3153717

4. Dariush Salami, **Ramin Hasibi**, Stefano Savazzi, Tom Michoel, and Stephan Sigg, (2022) *Integrating Angle-Agnostic Sensing into Cellular Networks using NR Sidelink*, (Submitted to ACM Transactions on Internet of Things)
   https://doi.org/10.48550/arXiv.2109.07253

5. Jakkob Kallestad, **Ramin Hasibi**, Ahmad Hemmati, and Keneth Sörensen (2023) *A general deep reinforcement learning hyperheuristic framework for solving combinatorial optimization problems*, European Journal of Operational Research **309/1**,
   https://doi.org/10.1016/j.ejor.2023.01.017

[1] Equal contribution

Contents

1 Background

1.1 Graphs as mathematical structures

A graph can be described as a mathematical notion to express the structure of a complex system. Simply put, graphs are derived from sets of elements (nodes) that are connected through a set of relations (edges). The nodes and edges of a graph can be of the same or different types which is one of the reasons that graphs are expressive in describing a wide range of systems. (*Barabási and Pósfai*, 2016).

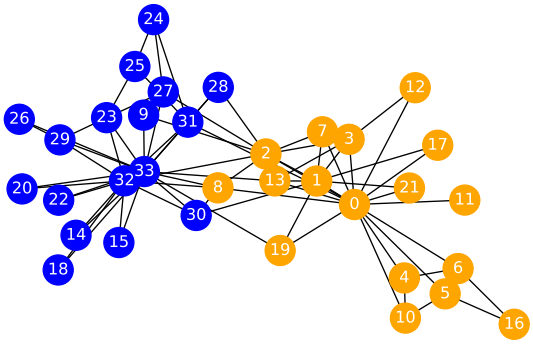


Figure 1.1: ***Example graph of Zachary Karate Club describing a social network of students attending a club.*** *Nodes and edges correspond to each member and whether they interacted outside the club. Colors show the two groups that they eventually split into after a conflict between members. Zachary (1977)*

Many real-world complex systems can be described as graphs. A commonly used example is the social network which is produced from the friends and acquaintances of individuals with nodes representing the persons and edges between the people who are friends or know each other from their social life. One example of such a graph is presented in Fig 1.1. Another important aspect of graphs is their generalizability to different domains just by representing nodes and edges as elements of said domain. For example, the same graph notation in social networks can also be applied in chemistry to describe molecules with nodes and edges representing the atoms and bonds (*Gilmer et al.*, 2017). This flexibility has introduced graphs in many application domains for analyzing complex systems:

- In biology, a graph can describe the interaction between genes, proteins, metabo-

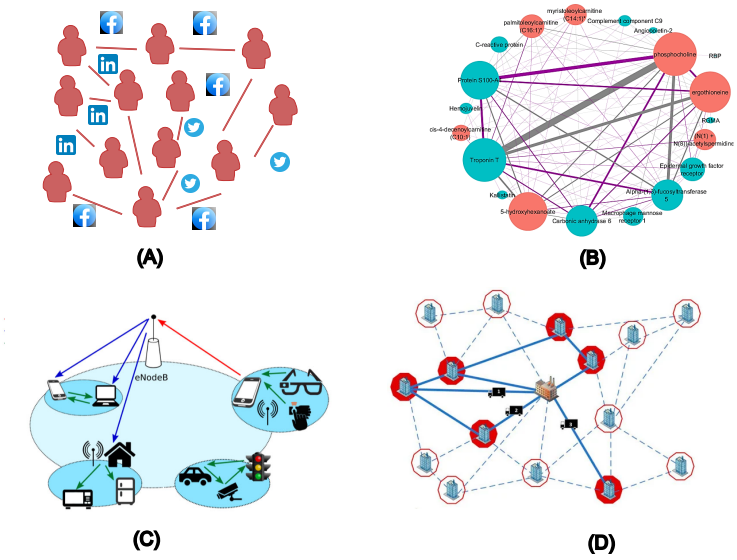


Figure 1.2: ***Some example applications of graphs for representing real-world systems*** *(A) a graph showing social networks and people connected by different websites; (B) A correlation network between proteins and metabolites Mastej et al. (2020); (C) A graph depicting the connection of devices and the direction of dataflow between different nodes in a communication network Tilwari et al. (2021); (D) An instance of a classic vehicle routing problem in logistics is represented as a graph Kucharska (2019).*

lites, and reactions inside a cell (*Emmert-Streib et al.*, 2014).

- In communications networks, the connection between devices, either wired or wireless, can be mapped to a graph structure and analyzed through graph perspective (*Jiang*, 2022).
- Power grids can be modeled as graphs with the power stations as nodes and power transmission lines as edges (*Hadaj et al.*, 2022).
- In transportation and logistics, nodes and edges can be the locations and roads that connect them and many analyses such as shortest path calculation can be done on the graph structure (*Kool et al.*, 2019).

Analyzing complex systems through the eyes of graph structure can result in finding laws and similarities between different systems that were not previously known. Understanding and discovering such laws and the relationship between the structure and the nature of the systems is the main motivation behind the emergence of *"Network Science"*. Network science is a study that offers a set of analysis tools that help uncover such empirical laws from different domain datasets and unfold the complexity of structured systems. (*Easley and Kleinberg*, 2010)

With the ever-growing field of network science and graph theory, the amount of datasets containing such structural information has also increased massively. These high-quality datasets are generated through many industries that work with these complex systems such as social networking websites, the pharmaceutical industry (for developing

drugs), logistics, etc. Other than network science, machine learning can also be used to process graph-based datasets. In line with discovering similarities and rules about the structure of graphs, machine learning can take advantage of training certain algorithms that are tailored to graph-based input datasets and extract a generalized rule for obtaining a preferred result from the input graph. Such trained models can then be applied in other scenarios to predict for unseen datasets or generate new graph structures with desired properties to improve the quality of a certain product (*Hamilton*, 2020). For instance, a generative machine learning model can learn to predict new and stable drugs by analyzing the graph structure and features of already available drug datasets (*You et al.*, 2018). In the following sections, we further illustrate the way machine learning models can take advantage of graph structures.

1.2 Graph formulation

Before going into details about machine learning techniques on graphs, one needs to understand how to formulate a graph object. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by a set of nodes $\mathcal{V}$ with $N = |\mathcal{V}|$ which represents the main domain objects in the system and a set of edges $\mathcal{E}$ which are used to describe the underlying connections between the nodes. In an undirected graph, a connection from node i to node j can be shown using the unordered set of $\{i, j\} \in \mathcal{E}$. However, in a directed graph, a connection from node i to node j does not guarantee a connection in the opposite direction. Thus, the edge from i to j is represented using ordered pair (i, j). A more straightforward way to represent all the above information about $\mathcal{G}$ is to use the adjacency matrix. The adjacency matrix $\mathbf{A} \in \{0, 1\}^{N \times N}$ is typically a square matrix the rows and columns of which correspond to each node in the graph. The elements of the matrix represent the existence of each edge in the graph. For every two adjacent nodes i and j (nodes that share an edge), the (i, j)th entry of the matrix will be equal to 1. All other elements of the matrix will be zero which corresponds to no connection between the two corresponding nodes of the matrix entry. The edges in a graph can also be weighted in which case a numerical value is assigned to each edge in $\mathcal{E}$. This can be reflected in the adjacency entries by replacing the value 1 with the edge weight for the corresponding entry (*Rosen*, 2006). Finally, another aspect of the graph that is used to characterize the underlying system is the attributions or features that are assigned to each node or edge in the graph. This aspect is especially important for machine learning methods as they usually work with features as input to find the appropriate function approximation. In the case of node features, the values are stored in a Matrix $\mathbf{X} \in \mathbb{R}^{N \times F}$ in which, each row corresponds to the features of the node in the same row of the adjacency $\mathbf{A}$. An example of such features can be personal data or image for each person in a social network or spatial information of each atom in a molecule. The edge set attributes may also be available for some graph datasets. For instance, the type of relationship between two entities in an author-paper network (e.g., written, cited, etc.) or the energy levels of a bound between two atoms can be considered example attributes for a sample edge.

1.3 Machine learning on graphs

Machine learning is the study of building automated systems through training an algorithm based on available knowledge to solve a specific prediction problem. When it comes to graphs, the algorithms are applied to datasets that can be modeled as graphs. For a typical problem in machine learning, the available knowledge (also known as *training data*) is used to train or tune a learning algorithm to predict a desired outcome. The performance of the algorithm is then evaluated on a separate unseen (during training) dataset known as *test data*. Traditionally, machine learning problems are categorized into three main families of *supervised*, *unsupervised*, and *RL* based on the feedback that the learning model receives from data. In a supervised setting, the main goal is to predict a ground truth label for each sample in the dataset. In such a setting, the model leverages the ground truth labels as feedback during training to try and tune the prediction model. On the other hand, Unsupervised learning lacks (or ignores) label information and tries to uncover hidden patterns from the characteristics (features) of the dataset itself rather than their relationship to ground truth labels. In RL, an environment replaces the dataset and the aim is to maximize the feedback signal that is received from the environment over a certain amount of steps. RL can be modeled as a high-level decision-making process where the machine learning model aims to choose the best set of consecutive actions to solve an episodic problem. Introducing the graph structure into the problem formulation results in a different type of problem categorization when it comes to machine learning methods. Tasks on graphs relate to the prediction of a specific target for each element or the whole structure of the graph or the generation of a graph with desired properties. Based on this fact, *Hamilton* (2020) suggested five categories for machine learning problems on graphs:

- Node level prediction,
- Edge level prediction,
- Graph level prediction,
- Sub-graph (community) level prediction,
- Graph structure optimization.

Both supervised and unsupervised machine learning techniques can be used for the first four tasks while reinforcement learning is mostly used as an approach to Graph structure optimization. Out of these 5 categories, in this thesis, we mostly focus on node and graph-level applications through supervised methods as well as structure optimization with RL.

1.3.1 Node level prediction

Many machine learning problems on graphs are modeled as node-level predictions. In this task, the aim is to come up with a learning function that accurately predicts value y_i associated with node i in the graph while utilizing attributes and the structure of the graph. The target value of a node can be the type (classification), a singular continuous value (regression), or multiple values associated with a node (multi-class or

multi-regression). An example application of this task is to try and predict the spam reviews in a graph of reviewer products (*Rayana and Akoglu*, 2015) or to find the research subject of a scientific paper based on a citation network *Yang et al.* (2016).

Although this problem resembles the classic task of supervised learning in machine learning, there is a fundamental difference in node-level prediction owed to the connection of nodes through the structure of the graph. In normal supervised learning problems, the instances are completely independent of each other and are looked at separately by the learning model. On the other hand, in graphs, a node contains neighborhood information that is a result of it being part of the graph. Utilizing this extra information can help the algorithm to better predict the labels given the input features. This hypothesis stems from the fact that in a network, similar nodes tend to connect to one another more often, a phenomenon that is also known as *"homophily"*. Incorporating this fact into the machine learning model has been shown to improve the performance specifically in node level predictions (*Hamilton*, 2020). Additionally, the notion of *semi-supervised* learning has also been used for node-level tasks. In the semi-supervised approach, the model has access to the features of the nodes in the test set however is "unaware" of the labels of the test nodes. By doing so, the model uses the features of all the nodes in the graph to better capture the neighborhood information surrounding each node and use this extra information for its prediction of the label of each node. During training the model is tuned based on the labels of the training nodes and for evaluation of the model, the labels of the test nodes are predicted by the tuned model (*Kipf and Welling*, 2016).

1.3.2 Graph level prediction

The task of predicting a value based on the whole graph structure is more similar to the classic supervised approach compared to node-level tasks. In graph-level prediction, each sample in the dataset is drawn from a distribution independently from other samples. The difference between this type of prediction and node-level prediction is the fact that a single target y is predicted for the entire set of nodes in the graph and the training and test sets are comprised of multiple graphs. The machine learning model has to take advantage of the features of all the nodes as well as features extracted from the overall structure of the graph to make its prediction. Example applications of this category include trying to predict energy properties of a given molecular structure (*Wu et al.*, 2017), and classifying fake news using news propagation graphs (*Dou et al.*, 2021).

1.3.3 Graph structure optimization

The main objective of structure optimization is to generate graph structures that fulfill certain criteria in terms of the properties of the structure. For instance, in some applications, the aim is to create graphs that are similar to the ones that are available in the training dataset. One of the major applications of this task is the generation of realistic molecules which is used in the drug design process (*You et al.*, 2018). Another area that is concerned with the optimization of graph structure is the task of optimizing a cost according to the graph structure. For example, in logistics, a graph represents the route that a ship has to take to deliver goods to the destination (*Korte and Vygen*, 2012). Therefore, creating graphs that describe a route that achieves minimum cost is the goal of this problem. In this thesis, we mainly focus on the second type of graph structure

learning which is to optimize a solution for a given graph-based optimization problem.

1.4 Traditional approach to node and graph level tasks

Before the introduction of deep learning for structured datasets, prediction tasks on node and graph levels were typically solved using network science algorithmic frameworks. One example is the family of network propagation algorithms. This framework is based on the homophily principle of graph-structured datasets and typically follows an iterative process in which, at each step, information is propagated through the edges to nearby nodes until a certain convergence criterion is met *Cowen et al.* (2017a). A classic example of this method is the *"label propagation"* algorithm in which (for the simplest case) classification of the unlabeled nodes is done by assigning the most frequent label of the neighbors as the label of the node in each step until each node's label is the most frequent in its neighboring nodes (*Zhu and Ghahramani*, 2002). This family of algorithms has also been explored in the format of diffusion processes in statistical physics in which heat (information) is propagated through the structure of the graph from nodes to its neighbors (*Thanou et al.*, 2017). Going beyond one-hop neighbors in the graph, another family of methods, namely *"Graph Embedding"*, looks to encode high-dimensional discrete graphs into low-dimensional, dense, and continuous embedding vector spaces while preserving the structural properties of the graph inside the embedding (*Cai et al.*, 2018; *Belkin and Niyogi*, 2001). However, traditional approaches are limited due to the fact that they require careful, hand-engineered features. Such hand-crafted features are limited in generalization capability as they are not optimized for a specific task and do not go through a learning process. Furthermore, designing these features can be a time-consuming and expensive process. Therefore, the introduction of learning representations to the domain of graphs has made graph-based machine learning more powerful and expressive due to the ability to find the optimized set of representations for each task as well as training powerful non-linear function approximators through the use of deep learning methods (*Hamilton*, 2020). Details about such methods and the reason behind their superior performance are discussed in the following section.

1.5 Supervised deep learning on node and graph level tasks

Deep learning has become the flagship methodology in many of the machine learning problems over the last decade. The main success of deep learning is owed to its superior performance on high-dimensional datasets. Learning in high dimensions is a difficult task that gets exponentially more complicated with increasing the number of dimensions, a phenomenon that is known as the *"Curse of Dimensionality"*. However, in deep learning, the architecture of the models tries to soften this effect by taking advantage of the domain that the data is representing (e.g., vision, text, graphs, etc.). This is done through inducing bias from the rules of the data domain. While many deep learning architectures such as Convulotional Neural Networks (CNNs) (*LeCun et al.*, 2010) and Recurrent Neural Networks (RNNs) (*Rumelhart et al.*, 1986) have taken advantage of domain structures (i.e., locality in images and time warping in text or sound), graphs often do not share the same underlying structural constraints as images or time-series datasets. In fact, as shown in Figure 1.3, contrary to images, graph-based datasets do

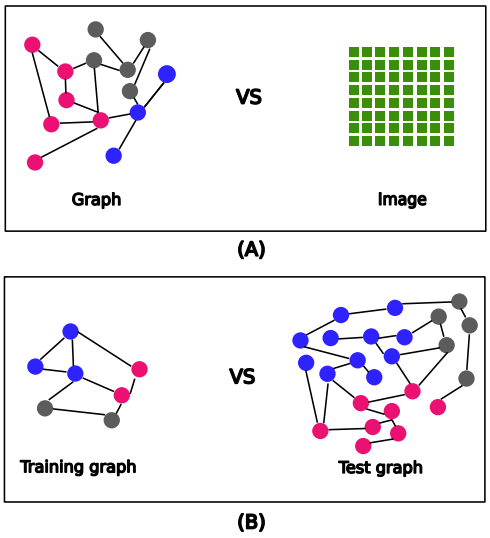


Figure 1.3: ***A graph does not follow a specific structure constraint (A)*** *In a graph, each node can have a different number of neighbors whereas an image is always in the form of a grid with nearby pixels forming a structured neighborhood;* ***(B)*** *The structure and size of the training graphs (available data) and testing graphs (real-world data) might be different. As a result, the deep learning model must be able to process both graphs through the same approach to be applicable to real-world applications.*

not have any constraints in terms of the size of each graph instance and the number of neighbors for each node. For instance, the graphs in the training dataset might be of smaller size compared to the ones that are produced in real-world environments. Therefore, the deep learning architecture that processes them should not only be able to process any graph (considering their structural irregularity) but also take advantage of this rule in the processing pipeline to improve the performance of the model. In this section, first, we discuss the difficulty of learning in high dimensional tasks and then delve into the methods for inducing bias for deep representation learning for the domain of graph-structured datasets.

1.5.1 Sources of error in supervised learning

In order to understand how deep learning techniques can benefit from inductive bias one needs to identify the sources of error when it comes to supervised learning tasks. In traditional supervised learning problem, a number of observed data points $\mathcal{D} \in \mathbb{R}^{M \times d}$ are available for which a label vector $Y \in \mathbb{R}^{M}$ is assigned. The aim of the learning algorithm is to find a function mapping $f \in \mathcal{F}$ which maps the data points to the labels i.e., $f(M) = Y$. In machine learning, this is done through estimating the true function f using some family of parameterized functions $\mathcal{F} = \{f_{\theta \in \Theta}\}$. This estimation is done by minimizing an empirical loss function (e.g., squared loss $\frac{1}{2}|Y - f_\theta(M)|^2$ through some optimization technique (e.g., stochastic gradient descent) which results in the estimated function of $\tilde{f}_\theta$ (*Bishop*, 2007). Given this formulation, three sources of error can be identified which prevent us from estimating the true function f:

1. *Approximation error*: This error arises from the fact that the family of functions $\mathcal{F}$ lacks the expressive power to estimate the true function f. An example of this is to try and estimate a non-linear function using linear estimation which never results in perfect estimation.

2. *Statistical error*: In the learning task, a finite number of data points are provided for training. However, most functions require a large amount of data (which increases exponentially with the number of dimensions) to be able to estimate the true function f. The low number of available data samples limits the approximation capability of true function f especially in noisy applications where the true data distribution is not known.

3. *Optimization error*: This error reflects the quality of optimization techniques to find the global minimum of the error loss. A lot of the loss functions especially in deep learning architectures are non-convex functions and optimization algorithms on these loss functions are prone to finding local minima. Therefore, this error is defined as the difference in the loss calculated for the final parameter set θ and the global minima of the chosen loss function.

While neural networks are known to be universal approximators (*Hornik et al.*, 1989) and therefore using them would result in a near zero approximation error, the statistical error would actually be higher (due to overfitting) compared to a simpler family of functions (e.g., linear). Thus, these two errors act in opposite directions of each other. However, it can be shown that inducing bias into the architecture of the learning function f_θ using the underlying laws of the domain structure can decrease the approximation error while keeping the statistical error and the complexity of the model (in terms of number of parameters) the same (*Bronstein et al.*, 2021). For example CNNs take advantage of the fact that images are pixels on a grid space and neighboring pixels contain useful information when investigated together. This local information is processed using the local kernels in the CNN architecture and that is one of the main contributing factors to the success of such models on the image-based datasets (*Krizhevsky et al.*, 2012).

1.5.2 Set representation learning

Inductive bias in the graph domain datasets arises from the definition of graphs. Graphs are made from a *set* of nodes and in sets, there are not any constraints on the ordering of the elements. This characteristic, which is referred to as permutation invariance, is the main focus of deep learning for inducing bias and decreasing the approximation error of the processing module of graph structures. The deep learning architectures that take graphs as input also process them in such a way that changing the order of the nodes does not change the output of the learning algorithm. To further illustrate this, first, we assume a graph that does not have any edges ($\mathcal{E} = \emptyset$). A function f that takes the input set of nodes $\mathcal{V}$ achieves permutation invariance by applying an individual processing step on each of the nodes separately and aggregating the results for all the nodes to generate the final output (*Zaheer et al.*, 2017).

$$S(\mathbf{X}, \mathcal{V}) = \sum_{i \in \mathcal{V}} (\psi(x_i)). \tag{1.1}$$

In this setting, ψ is usually a differentiable parameterized function and the resulting vector of the aggregation is the representation vector of the entire node set $\mathcal{V}$. Adding edges back into the mix builds on top of the *set representation learning* method in formula (1.1) by considering the local neighborhood of a node as a set and applying the same formula on the neighborhood rather than the entire graph. Therefore, formula (1.1) for each node is calculated as:

$$S(\mathbf{X}_{\mathcal{N}_i}, \mathcal{N}_i) = \sum_{j \in \mathcal{N}_i} (\psi(x_j)), \tag{1.2}$$

in which, $\mathcal{N}_i = \{j : (j, i) \in \mathcal{E}\}$ is the neighbourhood set of node i. The output of the formula (1.2) is the representation vector for the neighborhood of each node and can be combined with the features of each node and used as input for predicting the labels in a downstream task. For graph-level prediction, a permutation invariant pooling operator is used to aggregate all the nodes' representation vectors.

1.5.3 Message passing neural networks

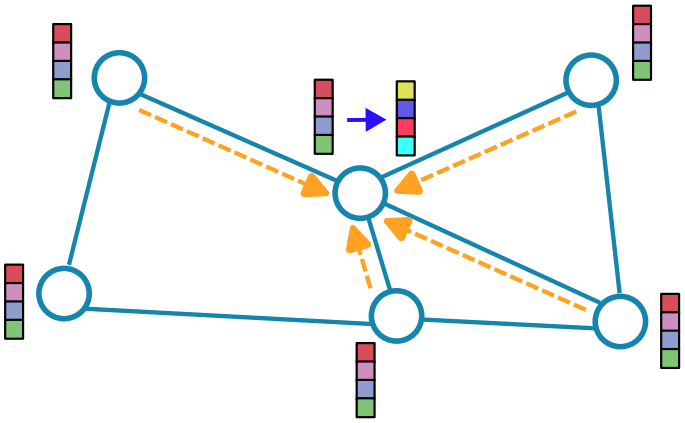


Figure 1.4: ***The message passing scheme for graph processing****; In each layer, the representation of each node is updated by aggregating the messages from neighboring nodes.*

The graph representation learning framework can also be modeled as a *Message Passing* algorithm in which each node in the graph gathers the neighborhood information and after combining it with its own representation sends a message over the edge to the nodes that are connected to it. By doing so, the message function for each edge can be computed as:

$$m_{ji} = \text{MSG}(x_j, x_i, x_{\mathcal{E}ji}), \forall (j, i) \in \mathcal{E} \tag{1.3}$$

in which, $x_{\mathcal{E}ji}$ are the optional edge features between nodes i and j and MSG is a parameterized differentiable function. Finally, the output representation of each node will be the aggregation of all the messages that it received from neighboring nodes and the features of the node itself as follows:

$$h_i = \text{AGG}\left(\{m_{ji}, j \in N(i)\}, x_i\right), \tag{1.4}$$

in which the AGG function performs a permutation invariant pooling to keep the inductive bias assumption of set representation learning. In order to combine this framework with the neural network as an example of an expressive family of functions, the MSG function is usually a Multi-Layer Perceptron (MLP) with non-linear activation functions. Additionally, in order to increase the power of the model, multiple layers of message passing are used where the output of each layer is fed into the next layer, and the resulting model is referred to as *Deep* Message Passing Neural Networks (MPNNs) or *Deep* Graph Neural Network (GNN). The formula for MPNNs is defined as

$$h_i^{(k)} = \gamma^{(k)} \left(h_i^{(k-1)}, \mathrm{AGG}_{j \in \mathcal{N}_i} \left(\mathrm{MSG}(h_i^{(k-1)}, h_j^{(k-1)}, h_{\mathcal{E}(j,i)}^{(k-1)}) \right) \right), \tag{1.5}$$

in which, $0 < k <= K$ refers to the number of the message passing layer and $h^{(k)}$ is the representation vector of each node at layer k. It should also be noted that the input to the first layer of MPNN are the original node and edge feature set (i.e., $h_i^0 = x_i$ and $h_{\mathcal{E}(i,j)}^0 = x_{\mathcal{E}ji}$)

Based on the downstream task and the level of prediction (node or graph level), there typically is an addition of the Read-out function which given the input representation of all the nodes $\mathbf{H} = \{h_i | i \in \mathcal{V}\}$ calculates the final prediction values for the target $\mathbf{Y}$.

$$\tilde{\mathbf{Y}} = R(\mathbf{H}). \tag{1.6}$$

Different choices for γ, MSG, AGG, and R results in different MPNN frameworks tailored to specific tasks or scenarios. For instance, on node-level tasks, the read-out function R is usually an MLP function that is calculated on each node representation vector $h_i^{(K)}$ separately and predicts the target value for that node. But on graph-level prediction, it can be a pooling operation such as sum-pool or average-pool combined with and MLP to calculate the target prediction for the entire graph.

1.6 Deep reinforcement learning for graph structure optimization

RL is a subfield of machine learning that is concerned with maximizing the expected sum of a numerical reward signal over a sequence of actions through multiple time steps. In a typical RL setting, an autonomous agent is tasked with choosing between a set of actions provided by a dynamic entity called *environment* over an episode of multiple sequential decision-making steps. At each step, the chosen action is applied to the environment, and in return, the environment provides the agent with a set of observable features also known as *states* to indicate the change that the action has caused and a numerical *reward* value which informs the agent how desirable the chosen action was at that particular step (*Sutton and Barto*, 2018).

Deep Reinforcement Learning (Deep RL) is an extension of RL methods that take advantage of the superior performance of deep learning techniques to make the decision-making process of RL more powerful. In Deep RL the agent uses the same inductive bias that the deep learning techniques are known for based on the domain of the state information (e.g., image, graph, etc.,) to be able to learn in high dimensional environments. While traditional RL uses tabular functions to map the state information to the action space, in the continuous high dimensional state features, the tabular methods can not

model the environment behavior properly. In such environments, an artificial neural network is mainly used as a function approximator to be able to capture the representation of the environment feature set and form a Deep RL agent.

Tasks related to graph structure optimization using Deep RL can be classified into two subgroups based on the objective of structure generation. The first class is concerned with the generation of realistic graphs that resemble the graphs from a certain dataset. In this particular task, the generation process that uses machine learning involves training on a set of available training set and during the generation phase, a graph structure is sampled from the learned probability distribution of the graph structures. A popular application of this domain is the generation of molecular structures for discovering new drug molecules that need to be stable (maintain the correct structural constraints) and also novel to be able to discover potential new drugs. An example work that utilizes Deep RL for molecular generation is the work of *You et al.* (2018), in which an agent is tasked with the generation of molecules from the ground up by adding a molecule and a bond at each step of the generation process. The second category of graph structure optimization acts solely based on a reward function and tries to optimize a cost value defined over graphs. This family of problems mostly belongs to combinatorial optimization problems. In this thesis, we address the second type of problem and introduce a novel way of using Deep RL to solve most applications in this family of problems. In the rest of this section, first, we introduce combinatorial optimization, then mention the typical methods to solve them, and how they relate to graphs.

1.6.1 Combinatorial optimization on graphs

Combinatorial optimization is defined as finding a selection of elements (solution) between a set of elements $\mathcal{P} = \{1, 2, ..., n\}$. Each selection is assigned an objective value calculated using the objective function $\mathcal{O} : 2^{\mathcal{P}} \mapsto \mathbb{R}$, that assigns a value to each possible selection $s \in \mathcal{S}$. Not all possible solutions are acceptable (feasible) for most problems. In fact, each feasible solution has to abide by certain rules (also known as problem constraints) to be accepted as a possible solution for the combinatorial optimization problem. The space of feasible solutions $\mathcal{S} \subseteq 2^{\mathcal{P}}$ is a finite set, however finding feasible solutions can be a challenging task in some problem spaces due to the set of constraints that are defined for the solution space $\mathcal{S}$. The main aim of this optimization task is to find the optimal solution s^* which achieves the lowest (or highest) amount of objective function value (*Papadimitriou and Steiglitz*, 1982).

Almost all combinatorial optimization problems can be modeled as a graph structure optimization problem (*Papadimitriou and Steiglitz*, 1982). In this setting, the set of choice elements is the edge set of the potential edges of the graph $\mathcal{E}$ and the resulting graph $\mathcal{G}'$ from the chosen edge set $\mathcal{E}'$ represents a potential solution s'. To further demonstrate this, we can look at the classic example of Travelling Salesman Problem (TSP) (*Dantzig et al.*, 1954) where the aim is to find the shortest path among the possible paths between multiple cities and return to the original starting point. As shown in Fig. 1.5, This problem can be easily modeled as a graph the nodes of which are positioned on the 2-dimensional Euclidean space. The weighted edge set is the potential routes among each pair of cities and a solution is constructed by choosing a subset of edges for constructing the solution path.

While small instances of certain combinatorial optimization problems could be solved

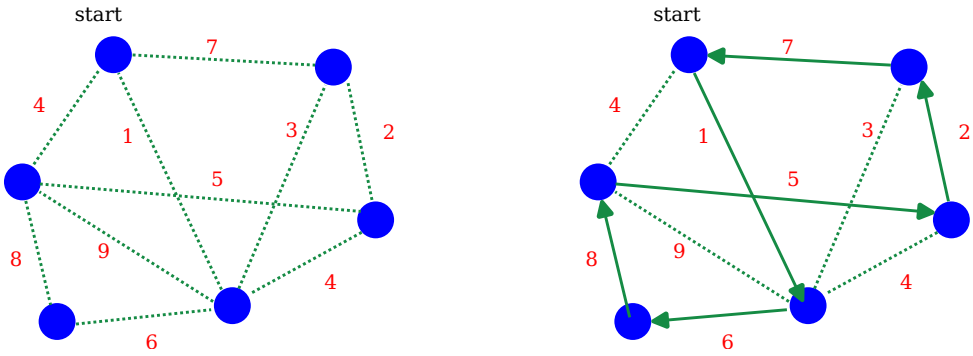


Figure 1.5: Example TSP instance modeled as a graph. The nodes represent cities and the edge weights indicate the cost of traveling between cities. An example solution for this problem is depicted on the right where the objective value is $O(s') = 24$.

using exhaustive search, the number of possible solutions grows exponentially with larger instance sizes. In some cases, even finding feasible solutions can be a cumbersome task. Many of cases of this family of problems can not be solved in polynomial time and are of computational complexity of NP-complete. This means that there is no exact algorithm that obtains the global best solution in polynomial time for these problems (*Festa*, 2014).

To solve combinatorial optimization problems, there exist multiple approaches. The first family of methods is known as *Exact Solvers*. In this approach, the problem is usually solved using a variation or combination of dynamic programming (*Bellman*, 1957) or branch and bound (*Lawler and Wood*, 1966) where only a subset of solution space is explored where the global best is guaranteed to appear. Although these methods often guarantee the global best solution, the time and computational complexity of such solvers can be too resource-consuming. This has led to another family of methods referred to as ***Heuristics***. In this method, the optimality guarantee no longer stands but the solutions provided by heuristics are computed in comparatively fast time with acceptable quality in terms of objective value. This makes the heuristics suitable for larger instances where using exact solvers is intractable in terms of computation time. The heuristics are algorithmic strategies designed for each problem specifically to achieve high-quality solutions. Two main ways to design heuristics are:

1. **Constructive**: In this type of heuristics, we start from an empty solution set and elements of the solution are chosen one after the other in each step. Although they are not as powerful as exact methods or the second family of heuristics, they are quite fast and provide a good initial solution or a baseline for the second family of heuristic methods.

2. **Perturbative**: In such heuristics, we start from an existing solution, and at each step, a neighboring solution is generated by applying changes to the current solution.

The amount of change in the solution in perturbative heuristics determines how much of the neighborhood is explored. While it is important to diversify, we also want to maintain the quality of the current solution and not start from scratch (intensification). Therefore, a balance between two strategies is required to obtain a good solution.

1.6.2 Metaheuristics

While heuristics are typically problem dependent, a subgroup of heuristics titled ***Metaheuristics*** aim to be more generalizable by providing a high-level set of guidelines and strategies to develop heuristics for optimization problems (*Sörensen and Glover*, 2013). generalizability of metaheuristics is often achieved by having some underlying assumptions about the optimization problems that are to be solved. Metaheuristics are usually closely related to perturbative heuristics. Some classic examples of these algorithms are Genetic Algorithm (*Holland*, 1992), Particle SwarmOptimization (*Kennedy and Eberhart*, 1995), Ant Colony (*Merkle and Middendorf*, 2006), Tabu search (*Glover and Laguna*, 1997), and Simulated Annealing (*Kirkpatrick et al.*, 1983a).

1.6.3 Hyperheuristics

Hyperheuristic is a heuristic that is used to guide the selection or generation process of low-level heuristics to achieve a better solution quality. The hyperheuristic framework, first introduced by *Cowling et al.* (2001), can be classified into two groups of *selection hyperheuristics* and *generation hyperheuristics*. In this thesis, the main focus is on the former group. The selection hyperheuristic is an iterative process that selects and applies a low-level heuristic of a solution vector at each step of the search. The selection process is usually done using a learning algorithm that applies machine learning techniques to learn from experience either during the search (online) or from training on separate datasets (offline). The main aim in hyperheuristic research is to build a model that can be applied on many optimization problems with minimal information about the nature of the problem. For instance, the only information that a hyperheuristic is provided can be the number of low-level heuristics, the direction of the optimization (minimization or maximization), and the objective function value. One of the contributions of this thesis is to apply Deep RL as a learning selection hyperheuristic approach for solving combinatorial optimization on graphs.

1.6.4 Deep RL for combinatorial optimization

A Deep RL agent can be seen as a heuristic method that can be trained to choose from a set of elements. In this application, each set element represents an action that changes the solution space (state information) and also the objective value which can be the source for the reward function. Using the Deep RL agent, just like other heuristics can be in two manners of "constructive" or "perturbative". In the constructive Deep RL, an agent starts from an empty solution and at each step selects an element to add to the solution. For instance, in solving TSP, the agent at each step chooses the next node of the graph to visit from the last visited node. Typically, in such a setting, the deep learning architecture is used to process the solution representation, which is a graph representing the current solution. It can also have information about the global setting of the problem instance. An instance of this approach is the work of *Kool et al.* (2019) for solving routing problems. In this work, an attention-based GNN (*Vaswani et al.*, 2017) is used to process the input instance as a graph, and at each step this representation vector along with the current solution is used to choose the next node to visit in the tour. Although, much research has examined the role of the Deep RL agent in a constructive manner, using Deep RL in a perturbative manner has produced higher quality solutions

due to covering a large portion of solution space during the search. For example, in *Lu et al.* (2020), the authors propose to use a Deep RL agent with a deep attention graph encoder, to select between multiple low-level operators for a certain number of iterations. However, most frameworks in this field have employed Deep RL agent as a heuristic, have used it in a problem-specific manner and none has tried to use it as a hyperheuristic method which makes the framework problem independent.

1.7 Applications of graph processing on different domains

As previously discussed, graphs are used in many real-world applications and in many industries. In this thesis, we address the applications of deep learning and deep reinforcement learning on graphs in specific applications ranging from biology to computer vision and logistics. In this section, we introduce each application and the graphs that are used in each domain.

1.7.1 Bioinformatics applications

Complex biological systems operate based on the interaction and coordination between many small events and their corresponding units. A straightforward way to represent such interactions and study them is through using graphs. The amount of generated data in recent years has helped with the generation of such graph-based datasets and has provided extra motivation for graph-based analysis methods due to the large amount of easily available data. This is owed to the advent of high throughput technologies that allow the identification of components such as genes, or proteins, and their activation levels as well as the interaction between them in different biological conditions and organisms in omics datasets (*Zhu et al.*, 2007). The term *omic* next to a biological molecular term indicates the global assessment of a set of molecules. One such assessment is genomics where the entire genome space of an organism is studied to extract useful information regarding a disease or a phenotype. Genomes are built from deoxyribonucleic acid (DNA) molecules and specific areas in the genome that are transcribed into Ribonucleic acid (RNA) are referred to as genes. Some genes are classified as protein-encoding genes meaning that they will be transcribed into messenger RNA (mRNA) which is later encoded into proteins for specific functions in the organism. One of the most important tools in genomics is expression profiling where transcription levels are measured for each gene in multiple experimental conditions, tissues, or individuals (*Hasin et al.*, 2017). In humans, such measurements can be used to detect disease-gene activity association to develop methods for the detection of diseases or monitor their progression. Cellular components such as genes, metabolites, and proteins are typically modeled as the nodes in biological graphs and the interactions that are between these elements often form the edges in a specific network type. The four types of networks that we consider in this thesis are: **Transcprition Factor (TF)** networks, **Protein-Protein Interaction (PPI)**, **Genetic interaction** networks, and **Metabolite networks**. In the following, we introduce each network and mention how they are built and some classic methodologies that are used to process them.

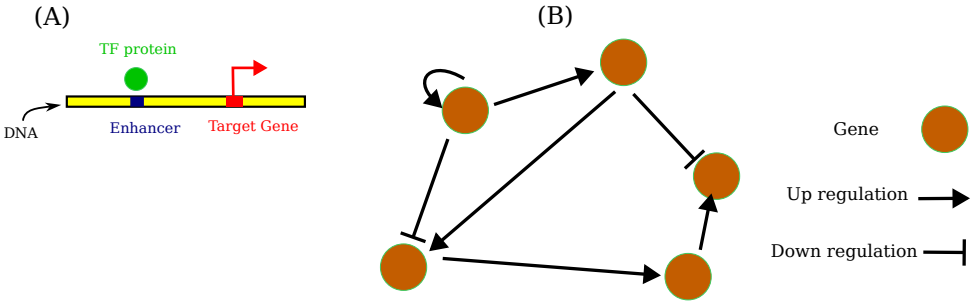


Figure 1.6: ***Transcription factor network as a type of gene regulatory graph.*** ***(A)*** *a protein is binding to the enhancer region of a gene regulating its production rate;* ***(B)*** *an example of 5 gene TF network is shown with up, down, and auto-regulation examples of connections.*

TF Network

TFs are DNA-binding proteins that regulate the mRNA production rate of the genes that they target by binding to regions called promoters and enhancers near the gene location (*MacNeil and Walhout*, 2011). The TF network is a directed network that maps the TF protein to the genes that the protein regulates. There is also a variation of such networks (referred to as the gene regulatory network) where the TF nodes are replaced with the genes that encode them. There are two types of effects that a binding can have on the production rate of a mRNA. The binding can either cause up-regulation resulting in an increased rate of mRNA production or down-regulation which has the opposite effect. There is also autoregulation in some cases where a TF produced by a gene, binds to the binding site of the same gene and affects its own production rate (*MacNeil and Walhout*, 2011). Fig. 1.6 depicts an example of such a network with 5 genes and their interactions.

PPI network

Proteins play a major role in vital cellular functions. Most of these functions are done through protein interactions examples of which are assembly of structural components, transcription, translation, and many more (*Muzio et al.*, 2020). In this thesis, to integrate the gene expression profiles with PPI networks, proteins are identified by the genes that encode them and the expression information for each gene is mapped to the nodes in the graph.

Genetic interaction network

It is often possible that a mutation in a single gene does not have any major effect on the growth or fitness of a biological organism. However, when mutations happen in two different genes, they could lead to a much stronger phenotype. Such observations can be modeled as a graph in which for a certain phenotype an edge between two genes (nodes in the graph) represents the fact that the effect of deleting both genes simultaneously on the phenotype is significantly greater than the sum of the effects of deleting one gene at a time (*Zhu et al.*, 2007).

Metabolite network

Metabolism at genome scale includes the set of multiple interconnected reactions that produce energy and convert nutrients into biomolecules. Representing metabolic models as a graph can be as simple as a network with metabolites as nodes and the edges are representative of the reactions that share those metabolites. Another way to build the graph is to consider the reactions as nodes and the edges are between the reactions where one consumes the molecules produced by the other (directed graph). Building such graphs and assigning the appropriate properties to them in terms of how the nodes are connected, edge weights, etc., can affect the performance of the downstream task that utilizes machine learning to process the graph structure (*Dusad et al.*, 2021).

Integration of omics datasets with biological graphs

It has been demonstrated that the structure of biological networks is informative of biological functions at multiple scales. For instance, degree distributions are a sign of the relative importance of genes or proteins in a cell; 3–4 node network motifs have well-defined information-processing roles; and network clusters or communities contain genes or proteins involved in similar biological processes (*Alon*, 2020; *Barabási and Oltvai*, 2004). At the same time, *omics* datasets can measure the variation or activation of certain cellular components across different individuals or experimental conditions. These datasets are however of a continuous nature and their integration with graph-based datasets requires methods that can represent the discrete graph structures in a meaningful wayAn edge between genes means that the effect on the phenotype of deleting both genes simultaneously is significantly greater than the sum of effects of deleting one gene at a time, in order to benefit the learning model when processing both data sources. There is a rich history of integrating the complementary viewpoints of biological networks and omics data. For instance, "active subnetwork" identification methods treat omics data as features of network nodes in order to identify well-connected subnetworks that are perturbed under different conditions (*Nguyen et al.*, 2019). Network propagation or smoothing methods on the other hand use biological networks to extend partial information on some nodes (e.g., disease association labels, partially observed data) to other nodes (e.g., to discover new disease-associated genes or impute missing data) (*Cowen et al.*, 2017b; *Ronen and Akalin*, 2018). However, such methods treat biological networks as discrete structures, which are intrinsically difficult to integrate with continuous node features or activity measures.

Gene essentiality prediction with metabolic graphs

One popular example of use cases of biological networks is the prediction of essential genes using metabolic networks. With essential applications in biomedicine and biotechnology (for example, for identifying therapeutic targets in complex diseases (*Cacheiro, P et al*, 2020)), identification of essential genes has been done through screening assays where multiple mutants are built and phenotyped with a suitable fitness selection strategy. The high cost and complexity of knock-out assays have resulted in a growing interest in computational approaches for the prediction of knockout genes. These computational approaches often employ machine learning techniques combined with information from protein sequence, gene homologies, gene-function ontologies, and protein

interaction networks (*Campos et al.*, 2019; *Li et al.*, 2020; *Zhang et al.*, 2020; *Mobegi et al.*, 2017; *Aromolaran et al.*, 2021). When it comes to metabolic genes that code for catalytic enzymes in metabolic pathways, Flux Balance Analysis (FBA) is a widely employed method for predicting essentiality (*Orth et al.*, 2010). While FBA has shown great performance over the well-studied simple organism of *E. coli* (*Monk, J. et al*, 2017), the predictive power of FBA decreases massively in case of higher order organisms (e.g., eukaryotes). One could associate this with the objective function that methods such as FBA assume for deletion strains. While FBA assumes that the objective function (typically chosen as growth rate) stays identical, it is plausible that gene deletions alter cell physiology to meet other objectives for survival.

There are numerous variants of FBA and its related algorithms (*Lewis et al.*, 2012), but at its core FBA computes genome-scale flux distributions that optimize a cellular fitness objective. Such objectives are typically taken to be the cellular growth rate modeled as a linear combination of synthesis rates of amino acids, lipids and other biomass components. By imposing constraints on each metabolic flux, FBA problems can be solved with efficient linear programming algorithms, which allows to rapidly simulate the impact of gene deletions on the predicted growth rate and draw predictions on the essentiality of metabolic genes. In a steady state, a metabolic network can be described by

$$\mathbf{S}v = 0, \tag{1.7}$$

in which, v is a n-dimensional vector of reaction fluxes, and $\mathbf{S}$ is a $n \times m$ stoichiometric matrix with m metabolites and n enzymatic reactions. The aim of FBA is to obtain the solution vector v^* that satisfies the above condition and at the same time solves the following optimization problem:

$$\begin{aligned} v^* &= \arg\max_{v} \quad c'v \\ &\text{subject to} \quad \begin{cases} \mathbf{S}v = 0, \\ v_{\text{lb}} < v < v_{\text{ub}}, \end{cases} \end{aligned} \tag{1.8}$$

in which, c is a vector of flux weights, and (v_{lb}, v_{ub}) are lower and upper bounds on reaction fluxes, respectively.

1.7.2 Computer vision applications

Graphs in computer vision have many applications such as mediator to process images and videos by extracting human pose or scene structure as a graph (*Jain et al.*, 2015), label processing through knowledge graphs *Chen et al.* (2019), or k-shot learning methods (*Garcia and Bruna*, 2018). In this thesis, we mostly focus on the application of graphs in processing *point clouds* as the input media for a 3-D scenery. Specifically, we mention mmWave radar-generated point clouds for capturing human movements. First, we introduce point clouds and their application and then we provide a brief overview of radar-generated point cloud and their properties.

Point Clouds

Figure 1.7: ***Point cloud samples from the ShapeNet dataset (Chang et al., 2015)*** *Three different objects of airplane, chair, and table are visualized; colors represent the labels for part segmentation problem.*

While visual scenes are often represented by images in 2-dimensions, 3-D scenery can be represented through many different formats, e.g., RGB-depth images, voxel images, meshes, etc. One of the most popular ways to represent 3-D space is by using *point clouds*. There are several major benefits of using point clouds among which we can point to the preservation of 3-D geometric information without the need for discretization or low cost and complexity of processing the point cloud datasets compared to voxel-based images (*Guo et al.*, 2021). A point cloud can be defined as a set of points $P = \{p_1, ..., p_n\} \subseteq \mathbb{R}^3$ in the 3-D space where each point can also be assigned a feature set of $x_{p_i} = \{f_i^1, ..., f_i^F\}$. This feature set can be any values including the 3-dimensional coordinates, depth, intensity, etc., so as to add more input information for the processing model. A more obvious way to process the point cloud using a machine learning method is to look at its definition as a set and apply the set representation learning method (see section 1.5.2) on the input points (*Zaheer et al.*, 2017; *Qi et al.*, 2016). However, a more informative way of looking at such inputs is to create a graph on the set of points and process the input graph using machine learning methods. Doing so would help the model capture local dependencies and hidden structures in the point cloud through graph edges as well as the global representation of the overall scene (*Wang et al.*, 2018).

mmWave radars and moving point cloud generations

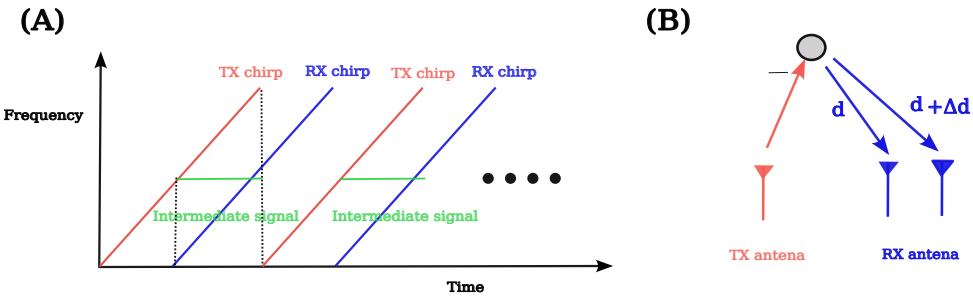


Figure 1.8: ***Object reflection using mmWave sensors:*** *(A) Chirps are sent and received by the antennas of the sensor and the range and velocity of the object are calculated using the frequency and shift of the intermediate signal; (B) The angle of the object is calculated by using multiple receiving antennas and their distance.*

mmWave radars are a class of radar-based technology that utilizes short-wavelength electromagnetic waves to detect objects in their vicinity. Such radars are able to send waves and based on the received reflection of the objects, the range, velocity, and angle of the moving object in their viewpoint is calculated. Two major parts of such radars are the transmitter (TX) and receiver (RX) components used to send and receive the radio frequency waves at objects (*Iovescu and Rao*, 2017).

To detect the properties of an object, the radar first sends an electromagnetic signal the frequency of which rises over a short period of time (Figure 1.8A). This wave is also known as a *chirp*. In return, a chirp is reflected from the object and is received by the RX antenna after a short period of time. the formula for the two transmitter (TX) and RX waves are as follows:

$$a_{tx} = sin(\omega_{tx}t + \Phi_{tx}) \tag{1.9}$$

$$a_{rx} = sin(\omega_{rx}t + \Phi_{rx}). \tag{1.10}$$

Once the return signal is received, an intermediate signal is calculated from the difference of the two TX and RX waves as:

$$a_{int} = sin((\omega_{tx} - \omega_{rx})t + (\Phi_{tx} - \Phi_{rx})). \tag{1.11}$$

The time delay of the RX chirp τ is calculated by

$$\tau = \frac{2d}{c}, \tag{1.12}$$

in which, d and c are the distance of the object and speed of light, respectively. Following this, one can derive the constant frequency and the initial phase of the wave by:

$$f_{ini0} = S\tau, \tag{1.13}$$

$$\Phi_{int0} = \frac{4\pi d}{\lambda}, \tag{1.14}$$

in which, λ is the wavelength and S is the slope of change in the frequency of the initial TX chirp. Thus, by calculating the frequency of the intermediate wave, one can obtain the distance of an object.

In order to extract the velocity of the moving object, the radar sends a second TX chirp with a time delay of T_c. The two chirps received by RX will have the same frequency as the intermediate signal however, their phase will have a shift. The shift of phase and consequently the velocity of the object is calculated by:

$$\Delta\Phi = \frac{4\pi v T_c}{\lambda}, \tag{1.15}$$

$$v = \frac{\lambda\Delta\Phi}{4\pi T_c}. \tag{1.16}$$

To calculate the angle of the object from the radar sensor, two separate RX antennas are used in the sensing device. The TX chirp is received by each of the antennas separately resulting in a phase shift between the two received signals due to the distance between the antennas. The phase shift between the two received signals can be calculated as:

$$\Delta\Phi = \frac{2\pi\Delta d}{\lambda}, \tag{1.17}$$

in which Δd is the difference of distance between the object and the two antennas (Figure 1.8B). The Δd can be calculated as

$$\Delta d = l sin(\theta), \tag{1.18}$$

in which, l is the distance between two antennas and θ is the angle of the object with respect to the sensor. As a result, the angle θ can be calculated by

$$\theta = sin^{-1}\left(\frac{\lambda\Delta\Phi}{2\pi l}\right). \tag{1.19}$$

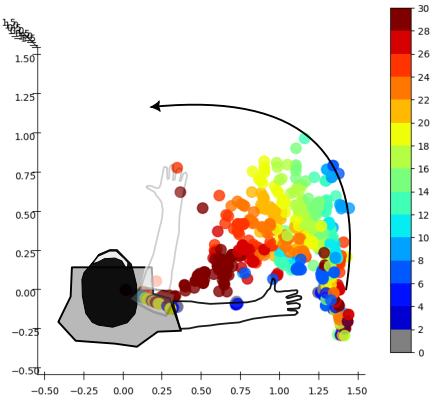


Figure 1.9: ***Point cloud samples from the swipe left gesture for a human subject.*** *Points and gestures are taken from Pantomime dataset (Palipana et al., 2021). The colors indicate the time stamp of the gesture which is performed over 30 frames.*

When a human subject performs gestures in front of a mmWave radar sensor, a set of points is calculated according to the above formulae based on the number of reflections and the resolution of the radar. Through these calculations, the range, azimuth angle, and elevation of each point are obtained by using multiple antennas, and a set of points with time stamps are gathered to represent the performed gestures as shown in Figure. 1.9. By replacing the nodes of the graph with the set of points and generating neighborhood graphs over the set of nodes, we can use GNNs to process the point clouds which is the approach taken in this thesis.

Gesture recognition with point clouds

Gesture recognition has been one of the most important tasks in computer vision. There have been many machine learning, and in particular, deep learning models for performing gesture recognition. One aspect that is important in these models, is the fact that the input representation plays an important role in both the accuracy and time-complexity of deep learning-based systems, especially in 3D scenery processing. Some example input media are RGB-depth images *Yang et al.* (2018); *Molchanov et al.* (2016); *Abavisani et al.* (2019), spectrograms of Doppler signals (*Kim and Toomajian*, 2016), and point clouds (*Qi et al.*, 2017; *Min et al.*, 2020)). Point clouds often lead to lower complexity systems compared to other input formats of 3D scenery processing. Additionally, models that process other input formats often have to create the format from a direct point cloud output of the sensor and as a result, might lose some geometric properties through the transformation process. The point cloud that is generated from mmWave radar sensors is of type **motion point cloud** which assigns a time stamp or frame number to each point to build a sequential data structure. In order to process such data structures, the deep learning models typically follow an RNN based pipeline which extracts the spatial features iteratively, over multiple time steps to emulate the evolution of gesture through time. However, in the case of the mmWave radar point clouds which have the innate property of being sparse in each frame (in average 5-10 points per frame), extraction of frame-wise spatial features does not contribute to the latent representation of gestures. Moreover, such recursive representation learning, where each frame is processed separately over multiple time steps, results in computation overhead. As a result a more resource-consuming prediction model. Therefore, a different representation learning scheme can address such shortcomings by incorporating graph structure into the prediction pipeline for input motion point clouds.

Another issue with point cloud processing models is that they are not typically rotation invariant (*Li et al.*, 2021). This means that rotating the input point cloud in 3D space confuses the model into thinking that a different gesture has been performed while the nature of the gesture is still the same. This can be translated in terms of capturing the gesture into positioning the subject at a different angle in front of the sensor compared to the angle that is provided in the test set. Moreover, when it comes to radar-generated point clouds, the observed patterns that originate from reflections off the same moving object differ conditioned on their distance, angle, and translation relative to the sensing due to shadowing effects. Such changes are not a direct result of geometric transformations on the base gesture point cloud. Therefore, the rotation invariant or equivariant methods for point cloud processing (*Thomas et al.*, 2018) will not be effective in this scenario.

1.7.3 Logistics and scheduling applications

The final graph application domain that we address is the domain of logistics and more specifically combinatorial optimization in logistics. Many of the problems in the domain of logistics are modeled as combinatorial optimization problems. While the routing and traveling problems heavily dominate this industry, there are also scheduling and inventory management problems that are modeled as a graph through combinatorial optimization. In this thesis, we introduce three variants of routing problems and an instance of a scheduling problem to represent the variation and difficulty of different problems that exist for operations research. Later, we use instances of these problems for the evaluation of proposed methods in a research article.

Capacitated Vehicle Routing Problem (CVRP)

One of the most studied problems in routing, CVRP is simply defined as a set of orders that need to be delivered by a set of vehicles. Each vehicle has a maximum capacity of goods that can carry and each order contains a certain amount of goods. Each vehicle starts from a depot and in a *"tour"* visits a sequence of destinations to deliver orders. The main objective of the problem is to deliver all orders using a set of tours and minimize the cost of traveling across all the combined tours.

Pickup and Delivery Problem (PDP)

A variation of the CVRP, in PDP an order needs to be picked up from a certain pickup point and delivered at another point. The vehicles in this problem are also of maximum capacity and can carry a certain amount of goods at a time. The objective is also similar to CVRP, in which all orders must be picked up and delivered at the lowest amount of travel cost possible while not exceeding the vehicle capacity in each route.

Pickup and Delivery Problem with Time Windows (PDPTW)

PDPTW is a variation of the PDP introduced in *Hemmati et al.* (2014). In this problem, vehicles are heterogeneous, in the sense that each vehicle has a different maximum capacity of goods and starts from a different starting point from others. Moreover, each call has a time window, within which it has to be picked up and delivered to its destination and certain calls can only be addressed by certain vehicles. There is also the possibility of outsourcing the orders at a higher price compared to delivering them using the predefined set of vehicles.

Parallel Job Scheduling Problem (PJSP)

This problem can be modeled as a graph with two types of nodes also known as a bipartite graph. In this problem, a set of jobs needs to be assigned to a set of machines that solve those jobs. Each machine has a certain processing speed and each job needs a certain amount of operations to be done in order to be solved. Each job must be solved before its preset deadline and the delay in finishing the job after the deadline increases the cost of the problem. The objective of the problem is to solve all jobs with the minimum amount of delay possible.

State of the art metaheuristic in solving combinatorial optimization problems

Among available metaheuristics, Adaptive Large Neighbourhood Search (ALNS) framework of *Ropke and Pisinger* (2006) has shown good results and decent generalizability to most combinatorial optimization problems in the domain. This framework which is an adaptation of Large Neighbourhood Search (LNS) (*Shaw*, 1998), uses an adaptive agent to balance between intensification and diversification by choosing between a pool of low-level heuristics. The algorithm for ALNS is presented at Algorithm 1.

Algorithm 1: ALNS

Function `ALNS`
- Generate an initial solution s_{init} with objective function of $\mathcal{O}(s)$
- $s_{best} = s, \mathcal{O}(s_{best}) = \mathcal{O}(s)$
- **Repeat**
  - $s^{'} = s$
  - choose h based on adaptive agent
  - Apply heuristic h to $s^{'}$
  - **if** $f(s^{'}) < f(s_{best})$ **then**
    - $s_{best} = s$
  - **end**
  - **if** $accept(s^{'}, s)$ **then**
    - $s = s^{'}$
  - **end**
- **Until** *Stop-criterion met*
- **return** s_{best}

The adaptive agent assigns probabilities of being chosen at each iteration of the search to each low-level heuristic and is updated every few iterations based on the performance of heuristics that were chosen during the previous iterations. Based on this strategy, the probability of each heuristic $h \in H$ can be calculated as:

$$P_h = \frac{w_h}{\sum_{m \in H} w_m}, \tag{1.20}$$

in which w_h is the weight of each heuristic that is calculated based on the performance of each chosen heuristic at each point during the search. Additionally, the acceptance criteria for ALNS usually follows a Simulated Annealing framework (*Kirkpatrick et al.*, 1983b). The ALNS framework has several advantages. For most optimization problems, a number of well-performing heuristics are already known which can be used as the operators in the ALNS framework. Due to the large size and diversity of the neighborhoods, the ALNS algorithm will explore huge chunks of the solution space in a structured way. As a result, ALNS is very robust as it can adapt to different characteristics of the individual instances, and is able to avoid being trapped in local optima (*Pisinger and Ropke*, 2019). According to *Turkeš et al.* (2021), the adaptive layer of ALNS has only minor impact on the objective function value of the solutions in the studies that have employed this framework. Moreover, the information that the adaptive layer uses for selecting heuristics is limited to the past performance of each heuristic. This limited data can make the adaptive layer naïve in terms of decision-making capability because it is not able to capture other (problem-independent) information about the current state of the search process.

Another area where ALNS struggles is when faced with a large number of heuristics to choose from. In order to find the best set of available heuristics for ALNS for a specific setting, initial experiments are often required to identify and remove inefficient heuristics. This extra step can be both time-consuming and computationally expensive (*Hemmati and Hvattum*, 2017). Furthermore, some heuristics are known to perform very well for specific problem variations or specific conditions during the search, but they may have a poor average performance. In this case, it might be beneficial to remove these from the pool of heuristics available to ALNS in order to increase the average performance of ALNS, but this results in a less powerful pool of heuristics that is unable to perform as well during these specific problem variations and conditions.

2 Aim of this study

In this thesis, we aim to address some applications in three domains of ***bioinformatics***, ***computer vision***, and ***logistics*** which can be modeled as graphs. While for some applications, methods such as network science or mathematical optimization can be used, we leverage the machine learning techniques of deep learning on graphs and Deep RL to improve the available methods as well as introduce new ways of dealing with the tasks at hand. In the following, we will discuss each domain separately and mention the need for using machine learning or improvements to the state-of-the-art methods for each task.

2.1 Bioinformatics

in **Paper 1**, we propose the use of GNNs to bridge the gap between the discrete biological graph structures and continuous domains omics datasets. To this end, we aim to come up with a systematic measurement of the relationship between the structure of a network and the node feature (e.g., gene expression profile) values. Additionally, we look at ways to train a GNN auto-encoder with a focus on node feature reconstruction rather than graph structure encoding. This approach will help the model leverage the structure as an additional modality in preserving the representation of node features and will have real-world applications such as imputing missing node values in graph-based datasets.

After establishing the benefit of using GNNs for biological networks, we aim to address the problem of prediction of gene essentiality by inventing a novel pipeline using deep learning on graphs and solution vector to the FBA wild-type conditions. To this end, in **Paper 2**, we propose FlowGAT, a gene essentiality prediction framework that tries to leverage the graph representation learning with dynamic graph construction in space of reaction nodes from metabolite pathways (*Beguerisse-Díaz et al.*, 2018) to predict the essentiality label of the reaction nodes in the graph.

2.2 Computer vision

In the task of gesture recognition with radar-generated point clouds, our goal is to create a model that is both fast and accurate in predicting the class of performed gestures from human subjects. The proposed model needs to be lightweight and fast as this model is aimed to be deployed on embedded devices which typically have memory and computing constraints. Additionally, the model should take advantage of the unique structure of sparse motion point clouds modeled as graphs. To tackle this issue, in **Paper 3**, we propose a direct point cloud processing method, **Tesla-Rapture** (**TE**mporal graph **S**e**L**f **A**ttention convolution), an MPNN graph convolution based architecture tailored to sparse point clouds generated by mmWave radars which offer high accuracy and low computational complexity for inference of input gesture.

After introducing the Tesla-Rapture model, we aimed to increase the robustness of the model to changing the subject angle with respect to the radar sensor. In **Paper 4**, we sought to address this problem by providing a dataset that captures the gesture

from 8 different angles from the same gesture and a prediction pipeline that utilizes the representation learning scheme of Tesla-Rapture along with the voting mechanism of an ensemble of models to predict the gestures more robustly against changes in sensing angle. Additionally, a mechanism for sharing the output of each model is proposed which can be sent over the New Radio sidelink for 5th-generation networks and make distributed learning a possibility for the proposed framework.

2.3 Logistics

To address the shortcomings of ALNS as a general metaheuristic framework, in **Paper 5**, we propose **Deep Reinforcement Learning Hyperheuristic (DRLH)**, a general approach to selection hyperheuristic framework for solving combinatorial optimization problems. In DRLH, we replace the adaptive layer of ALNS with a Deep RL agent responsible for selecting heuristics at each iteration of the search to achieve superior performance compared to the baseline of ALNS and a random selection agent. To make DRLH more intelligent compared to ALNS, we sought to come up with state features consisting of a problem-independent feature set from the search process and train the agent with a problem-independent reward function that encourages better solutions. The main goal of this framework is to be generalizable and easily adapted to most of the combinatorial optimization problems in the literature.

3 Summary of the papers

Paper I: A Graph Feature Auto-Encoder for the Prediction of Unobserved Node Features on Biological Networks

Ramin Hasibi *and Tom Michoel*

BMC Bioinformatics, **22/1** (2021), doi:10.1186/s12859-021-04447-3

Background: Molecular interaction networks summarize complex biological processes as graphs, whose structure is informative of biological function at multiple scales. Simultaneously, omics technologies measure the variation or activity of genes, proteins, or metabolites across individuals or experimental conditions. Integrating the complementary viewpoints of biological networks and omics data is an important task in bioinformatics, but existing methods treat networks as discrete structures, which are intrinsically difficult to integrate with continuous node features or activity measurements. Graph neural networks map graph nodes into a low-dimensional vector space representation, and can be trained to preserve both the local graph structure and the similarity between node features.

Results: We studied the representation of transcriptional, protein-protein, and genetic interaction networks in E. coli and mouse using graph neural networks. We found that such representations explain a large proportion of variation in gene expression data, and that using gene expression data as node features improves the reconstruction of the graph from the embedding. We further proposed a new end-to-end Graph Feature Auto-Encoder framework for the prediction of node features utilizing the structure of the gene networks, which is trained on the feature prediction task, and showed that it performs better at predicting unobserved node features than regular MultiLayer perceptrons. When applied to the problem of imputing missing data in single-cell RNAseq data, the Graph Feature Auto-Encoder utilizing our new graph convolution layer called FeatGraphConv outperformed a state-of-the-art imputation method that does not use protein interaction information, showing the benefit of integrating biological networks and omics data with our proposed approach.

Conclusion: Our proposed Graph Feature Auto-Encoder framework is a powerful approach for integrating and exploiting the close relation between molecular interaction networks and functional genomics data.

Paper II: Integration of genome-scale metabolic models and deep graph neural networks for gene essentiality prediction

***Ramin Hasibi**, Tom Michoel, and Diego A. Oyarzun*

Background: Genome-scale metabolic models are powerful tools for understanding cellular physiology. Flux balance analysis (FBA), in particular, is an optimization-based approach widely employed for predicting metabolic phenotypes. In model microbes such as *Escherichia coli*, FBA has been successful at predicting essential genes, i.e. those genes that impair survival when deleted. A central assumption in this approach, however, is that both wild-type and deletion strains optimize the same fitness objective. The optimality assumption may hold for the wild-type metabolic network, but deletion strains are not subject to the same evolutionary pressures, and knock-out mutants may steer their metabolism to meet other objectives for survival.

Resutls: In this paper, we present FlowGAT, a hybrid FBA-machine learning strategy for predicting essentiality directly from wild-type metabolic phenotypes. The approach is based on a graph-structured representation of metabolic fluxes predicted by FBA, where nodes correspond to enzymatic reactions and edges quantify the propagation of metabolite mass flow between a reaction and its neighbors. We integrate this information into a graph neural network that can be trained on knock-out fitness assay data. Comparisons across different model architectures reveal that FlowGAT predictions for *E. coli* are close to those of FBA for several growth conditions. Additionally, FlowGAT displays encouraging generalization power across growth conditions, even in cases where the underlying graphs and node features differ substantially. This observation suggests that the proposed architecture and feature extraction method can learn internal representations that are useful predictors of gene essentiality.

Conclusion: Our approach demonstrates the benefits of combining the mechanistic insights afforded by genome-scale models with the ability of deep learning models to extract patterns from complex data. Our results suggest that gene essentiality can be accurately predicted by exploiting the network structure of metabolism, without additional assumptions beyond optimality of the wild-type.

Paper III: **Tesla-Rapture: A Lightweight Gesture Recognition System From mmWave Radar Sparse Point Clouds**

Dariush Salami and Ramin Hasibi*, Sameera Palipana, Petar Popovski, Tom Michoel, and Stephan Sigg*

IEEE Transactions on Mobile Computing, **22/08** (2022), doi:10.1109/TMC.2022.3153717

Background: State-of-the-art gesture recognition models are either too resource-consuming or not sufficiently accurate for integration into real-life scenarios using wearable or constrained equipment such as IoT devices (e.g. Raspberry PI), XR hardware (e.g. HoloLens), or smartphones. The input representation plays an important role in both accuracy and time-complexity of deep learning based systems. In particular, converting the raw Analog to Digital Con- version (ADC) data from the antenna arrays to point clouds (i.e., unordered sets of points in space), massively reduces the data size by several magnitudes (e.g. GBytes to MBytes), which results in faster data transfer, pre-processing, and inference time. Unlike spectrograms of Doppler signals, point clouds are easily interpretable since the motions occur in a 3D space. Furthermore, strong point-cloud processing models exist, since this format is the standard output of a wide range of sensors.

Results: We present Tesla-Rapture, a gesture recognition system for sparse point clouds generated by mmWave Radars. State-of-the-art gesture recognition models are either too resource-consuming or not sufficiently accurate for integration into real-life scenarios using wearable or constrained equipment such as IoT devices (e.g. Raspberry PI), XR hardware (e.g. HoloLens), or smartphones. To tackle this issue, we have developed Tesla, an MPNN graph convolution approach for mmWave radar point clouds. The model outperforms the state of the art on three datasets in terms of accuracy while reducing the computational complexity and, hence, the execution time. In particular, the approach is able to predict a gesture almost 8 times faster than the most accurate competitor. Our performance evaluation in different scenarios (environments, angles, distances) shows that Tesla generalizes well and improves the accuracy up to 20% in challenging scenarios, such as a through-wall setting and sensing at extreme angles. Utilizing Tesla, we develop Tesla-Rapture, a real-time implementation using a mmWave Radar on a Raspberry PI 4, and evaluate its accuracy and time complexity. We also publish the source code, the trained models, and the implementation of the model for embedded devices.

Conclusion: Introducing Tesla-Rapture system, as a fast and accurate gesture recognition interface is a step forward in human-computer interaction scenarios for integration with many off-the-shelf devices. Given the robustness of the system in different environments, angels, and distances as well as real-time performance, Tesla-Rapture system can be incorporated into a wide range of applications e.g., smart homes, vehicular settings, and human-robot interaction. Furthermore, the model can be trained on a customized set of gestures and deployed on Tesla-Rapture for a specific real-time application.

Paper IV: Integrating Angle-Agnostic Sensing into Cellular Networks using NR Sidelink

*Dariush Salami, **Ramin Hasibi**, Stefano Savazzi, Tom Michoel, and Stephan Sigg*

Background: The angle of the human subject with respect to the sensor plays an important role in Radar based gesture recognition system. It is a known fact that changing the angle from 0deg results in a different point cloud shape. This new shape is not exactly reproducible by rotation of the gestures from 0deg due to shadowing effect. Therefore, a rotation invariant-equivariant framework does not apply in this scenario.

Results: Two different approaches of angle invariant and gesture orientation tracking are considered for a gesture recognition system using point clouds. In each approach, a gesture from a subject is captured from 8 different angles from training and each angle point cloud is processed using a TeslaConv layer from Paper III. In the first approach, the weights of the TeslaConv layer is shared among all angles and the prediction of the gesture is done using an angle invariant pooling mechanism. In the second approach, each angle's gesture is predicted separately through a different set of weights and the label is predicted separately. The final prediction for the gesture is computed through voting ensemble score. gesture sequences to capture spatio-temporal relation. Angle-agnosticity is achieved through variations in pooling as well as through orientation tracking. Our approach outperforms the state-of-the-art algorithms for RF-sensing achieving 100 % accuracy. Moreover, our approach is significantly more resilient to missing angles compared to state-of-the-art models outperforming them with a margin of 70% when 7 angles out of 8 angles are not available. We make openly available our dataset comprising 15 subjects, performing 21 gestures which are recorded from 8 angles. Additionally, in order to implement this framework in the 5th generation mobile network (5g), we recommend the use of New Radio (NR) sidelink for transmitting the sensor output in order to enable the remote sensing of the subject through the cellular network. In our proposed framework, a sensing user equipment can participate with multiple users and share their sensed gesture for integrated classification. Moreover, a federated learning scheme is proposed where a central agent is trained based on the information gathered from multiple sensing users.

Conclustion: Our data aggregation and processing tool-chain outperforms the state-of-the-art point cloud based gesture recognition approaches for angle-diverse gesture recordings. Moreover, our proposed pipeline provides a feasible mechanism for integrating RF-sensing into cellular communication systems.

Paper V: **A general deep reinforcement learning hyperheuristic framework for solving combi- natorial optimization problems**

Jakkob Kallestad, ***Ramin Hasibi****, Ahmad Hemmati, and Keneth Sörensen*

European Journal of Operational Research, **309/1** (2023), doi:10.1016/j.ejor.2023.01.017

Background: Many problem-specific heuristic frameworks have been developed to solve combinatorial optimization problems, but these frameworks do not generalize well to other problem domains. Metaheuristic frameworks aim to be more generalizable compared to traditional heuristics, however their performances suffer from poor selection of low-level heuristics (operators) during the search process. An example of heuristic selection in a metaheuristic framework is the adaptive layer of the popular framework of Adaptive Large Neighborhood Search (ALNS).

Resutls: We propose a selection hyperheuristic framework that uses Deep Reinforcement Learning (Deep RL) as an alternative to the adaptive layer of ALNS. Unlike the adaptive layer which only considers heuristics' past performance for future selection, a Deep RL agent is able to take into account additional information from the search process, e.g., the difference in objective value between iterations, to make better decisions. This is due to the representation power of Deep Learning methods and the decision making capability of the Deep RL agent which can learn to adapt to different problems and instance characteristics. In this paper, by integrating the Deep RL agent into the ALNS framework, we introduce Deep Reinforcement Learning Hyperheuristic (DRLH), a general framework for solving a wide variety of combinatorial optimization problems and show that our framework is better at selecting low-level heuristics at each step of the search process compared to ALNS and a Uniform Random Selection (URS). Our experiments also show that while ALNS can not properly handle a large pool of heuristics, DRLH is not negatively affected by increasing the number of heuristics.

Conclusion: For quite some time now, it has increasingly become evident that the fields of machine learning and (heuristic) optimization can mutually benefit from an integration. On the one hand, recent advances in optimization can support the development of advanced machine learning methods, since these methods generally solve an optimization problem (e.g., what is the optimal subset of features from a data set that predict a certain outcome). This paper addressed the mirror issue: how can optimization approaches benefit from an integration of machine learning methods. We believe that approaches like the one presented in this paper have the potential to make the development of a powerful heuristic less dependent on the knowledge of an experienced developer with a deep insight into the structure of the specific problem being solved, and may therefore be instrumental in the integration of metaheuristics ideas into general purpose software packages

4 Discussion and future aspects

Graphs are considered a powerful tool for modeling a wide variety of systems and often provide a novel view for analyzing systems as well as a different approach to problem-solving. Machine learning on graphs offers a gateway to applying the powerful models of machine learning to process datasets with an underlying structure. Among the machine learning models, deep learning and deep reinforcement learning offer the most promising performance owing to the representation power of deep learning architectures. However, the domain of graphs needs its own deep learning architectures as their structural nature is different from those of images and time series datasets. Adapting the domain of graphs to deep learning models helps develop new applications of machine learning as well as providing novel approaches to previously known problems that were solved using traditional machine learning. To achieve this, the graph can be leveraged as a roadmap on how to efficiently process the input samples (what GNNs refers to as message passing) to improve the performance or the structure of the graph can be optimized using deep reinforcement learning. In this thesis, we focused on applications that can be modeled as graphs and provide appropriate deep learning and deep reinforcement learning methodologies to achieve state-of-the-art performance.

First, we provided two applications of graph processing solved using GNNs in bioinformatics

In **Paper I**, we studied whether GNN, which learn embeddings of nodes of a graph in a low-dimensional space, can be used to integrate discrete structures such as biological interaction networks with information on the activity of genes or proteins in certain experimental conditions. Traditionally, this is achieved by for instance network propagation methods, but these methods do not extract quantitative information from a graph that could be used for downstream modeling or prediction tasks. GNN on the other hand can include node features (gene or protein expression levels) in the learning process, and thus in theory can learn a representation that better respects the information contained in both data types. So far the integration of node features in graph representation learning has mainly been pursued for the task of link prediction. Here instead we focused on the task of predicting unobserved or missing node feature values. A potential drawback of our method is that it assumes that the interaction graph is known and of high quality. Future work in this direction could investigate whether it is feasible to learn graph representations that can do link prediction and node feature prediction simultaneously, or whether network inference followed by graph representation learning for one type of omics data can aid the prediction of another type of omics data.

In **Paper II**, we presented FlowGAT, a GNN that can be trained on knock-out fitness data to predict the essentiality of metabolic genes. The architecture exploits the inherent graph structure of metabolic fluxes predicted by Flux Balance Analysis through a combination of mass flow graphs and node features that describe local connectivity. Using data from *E. coli* and its latest genome-scale metabolic model, we show that FlowGAT can identify most of the genes that are correctly called as essential by Flux Balance Analysis, and even correct some of its misclassified essential genes. Our approach is based solely on the wild-type phenotype predicted by FBA; since it does not require the assumption of optimality in the deletion strains, FlowGAT may provide benefits when applied

to organisms where the growth optimality assumption is not warranted. Additionally, FlowGAT displays encouraging generalization power across growth conditions, even in cases where the underlying graphs and node features differ substantially. This observation suggests that the proposed architecture and feature extraction method can learn internal representations that are useful predictors of gene essentiality. We also found, however, that FlowGAT struggled to predict non-essential genes and can be substantially outperformed by traditional FBA. This phenomenon could arise from various sources, such as the data imbalance that in our case favors essential labels, or because predicting non-essential genes is intrinsically more challenging than essential ones (*Bernstein et al.*, 2023). Our approach illustrates the potential of exploring new ways of combining traditional tools such as Flux Balance Analysis with modern data-driven approaches and adds to the growing body of literature at the interface of genome-scale metabolic modeling with machine learning.

Next, we look at the application of gesture recognition on point clouds modeled as spatio-temporal graphs.

Paper III introduced Tesla-Rapture system, a fast and accurate gesture recognition interface that is a step forward in human-computer interaction scenarios for integration with many off-the-shelf devices. Given the robustness of the system in different environments, angles, and distances as well as real-time performance, Tesla-Rapture system can be incorporated into a wide range of applications e.g., smart-homes, vehicular settings, and human-robot interaction. Furthermore, the model can be trained on a customized set of gestures and deployed on Tesla-Rapture for a specific real-time application. Due to the computational efficiency and robustness to different environments and angles, Tesla-Rapture system can be extended to scenarios where egocentric gestures should be recognized on constrained devices. Tesla prediction model can be modified to adapt to new applications for wearable devices, e.g., Microsoft HoloLens[1]. Currently, HoloLens 2 captures hand gestures using RGB-D sensors. Given the benefits of radars over RGB-D cameras, integration of Tesla-Ra[true with HoloLens improves the performance of hand gesture recognition which is one of the main interaction mechanisms of this device. One potential drawback of Tesla is that it does not reach state-of-the-art performance when it comes to a set of dense gestures collected using a depth camera as evidenced by our experiments in the paper. However, our approach outperforms PointLSTM (state-of-the-art model on dense point cloud datasets), with a margin of up to 12.4%, 11.1%, and 4.6% accuracy on mmWave radar generated point cloud datasets of Pantomime, mHomeGes, and RadHAR, respectively. While this approach introduces a custom K Nearest Neighbors (KNN) algorithm to reflect the temporal dependency in graph generation, the graph is still being created statically using the KNN algorithm. Introducing an RL agent, imitating the cognitive reward-based learning process, can lead to improvement in the graph according to the accuracy of the classification model. Therefore, dynamic graph generation using RL is one possible direction for improving the temporal graph.

Paper IV is a follow-up on the work of the previous article in which we investigate the effect of human subjects with respect to the sensing radar. In this work, first, We have proposed a mechanism for radar sensing to be integrated into cellular communication systems. In particular, we suggested integrating radar sensing with NR sidelink in Device-To-Device (D2D) communication. We further investigated a common issue

[1] https://www.microsoft.com/en-us/hololens

related to Radio sensing, which is its angle and rotation dependence. In particular, we discussed transformations of mmWave point-cloud data which achieve rotational invariance, as well as distributed processing based on such rotational invariant inputs at distributed, angle, and distance diverse devices. Furthermore, and to process the data, we employed the graph-based encoder of Tesla to capture spatio-temporal features of the data as well as four approaches for multi-angle learning. The approaches are compared on a newly recorded and openly available dataset comprising 15 subjects, performing 21 gestures which are recorded from 8 angles. We showed that our data aggregation and processing toolchain outperforms the state-of-the-art point cloud-based gesture recognition approaches for angle-diverse gesture recordings.

In our last article, we look at the use of Deep Reinforcement Learning (DRL) in optimizing the structure of graph-based problems in combinatorial optimization.

To demonstrate this, in **Paper V**, we proposed DRLH for solving combinatorial optimization problems, which utilizes a trained DRL agent to select low-level heuristics to be applied on the solution in each iteration of the search based on a search state consisting of features from the search process. In our experiments, we solved four combinatorial optimization problems (CVRP, PJSP, PDP, and PDPTW) using our proposed approach and compared its performance with the two separate baselines. Our results show that DRLH is able to select heuristics in a way that achieves better results in fewer iterations for almost all of the problem variations compared to ALNS and URS. Furthermore, the performance gap between DRLH and the baselines is shown to increase for larger problem sizes, making DRLH a suitable option for large real-world problem instances. Additional experiments on an extended set of heuristics show that DRLH is not negatively affected when selecting from a large set of available heuristics, while the performance of ALNS is much worse in this situation. Enriching or refining the state representation with additional information is possible with very little effort. We have experimented with adding problem-dependent information into the state representation and seen that this gives even better results than sticking with the simple chosen state representation. Yet once we start to introduce problem-dependent structure and constraint information into the state representation we lose some of the generality that we strive for with DRLH as we would have to separately engineer a different state representation for each new problem. For this reason, we deem this outside of the scope of this paper and leave this area open for future work. Future research in this area should provide more empirical evidence for the superiority of DRLH over ALNS by applying this novel hyperheuristic to different problems. A potential direction for improving the model in the future is designing a reward function that is both stable and takes into account the difference in objective value at each iteration of the search. Initial experiments on alternative reward functions have shown promising results, but are time-consuming to train and not very stable compared to the proposed reward function that we have used in this paper.

Overall, our main purpose with this thesis is to showcase the benefit of integration of deep learning and deep reinforcement learning methodologies with problems that have an underlying structure. For all studied applications, we leverage the structural information, often modeled as graphs, along with other provided information about the problem set to improve state-of-the-art performance. We achieve this by leveraging deep learning architectures into the problem-solving pipelines.

Bibliography

Abavisani, M., H. R. V. Joze, and V. M. Patel (2019), Improving the performance of unimodal dynamic hand-gesture recognition with multimodal training, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 21

Alon, U. (2020), *An introduction to systems biology: design principles of biological circuits*, 2nd ed., CRC Press, London. 16

Aromolaran, O., D. Aromolaran, I. Isewon, and J. Oyelade (2021), Machine learning approach to gene essentiality prediction: a review, *Briefings in Bioinformatics*, *22*(5). 17

Barabási, A.-L., and Z. N. Oltvai (2004), Network biology: understanding the cell's functional organization, *Nat Rev Genet*, *5*, 101–113. 16

Barabási, A.-L., and M. Pósfai (2016), *Network science*, Cambridge University Press, Cambridge. 1

Beguerisse-Díaz, M., G. Bosque, D. Oyarzún, J. Picó, and M. Barahona (2018), Flux-dependent graphs for metabolic networks, *npj Systems Biology and Applications*, *4*(1). 25

Belkin, M., and P. Niyogi (2001), Laplacian eigenmaps and spectral techniques for embedding and clustering, in *Advances in Neural Information Processing Systems*, vol. 14, edited by T. Dietterich, S. Becker, and Z. Ghahramani, MIT Press. 6

Bellman, R. (1957), *Dynamic Programming*, 1 ed., Princeton University Press, Princeton, NJ, USA. 12

Bernstein, D. B., B. Akkas, M. N. Price, and A. P. Arkin (2023), Critical assessment of E. coli genome-scale metabolic model with high-throughput mutant fitness data, doi:10.1101/2023.01.05.522875, pages: 2023.01.05.522875 Section: New Results. 34

Bishop, C. M. (2007), *Pattern Recognition and Machine Learning (Information Science and Statistics)*, 1 ed., Springer. 7

Bronstein, M. M., J. Bruna, T. Cohen, and P. Velickovic (2021), Geometric deep learning: Grids, groups, graphs, geodesics, and gauges, *CoRR*, *abs/2104.13478*. 8

Cacheiro, P et al (2020), Human and mouse essentiality screens as a resource for disease gene discovery, *Nature Communications 2020*, *11*(1), 1–16. 16

Cai, H., V. W. Zheng, and K. C.-C. Chang (2018), A comprehensive survey of graph embedding: Problems, techniques, and applications, *IEEE Transactions on Knowledge and Data Engineering*, *30*(9), 1616–1637, doi:10.1109/tkde.2018.2807452. 6

Campos, T. L., P. K. Korhonen, R. B. Gasser, and N. D. Young (2019), An Evaluation of Machine Learning Approaches for the Prediction of Essential Genes in Eukaryotes Using Protein Sequence-Derived Features, *Computational and Structural Biotechnology Journal*, *17*, 785–796. 17

Chang, A. X., T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu (2015), Shapenet: An information-rich 3d model repository, cite arxiv:1512.03012. 18

Chen, Z., X. Wei, P. Wang, and Y. Guo (2019), Multi-label image recognition with graph convolutional networks, *CoRR*, *abs/1904.03582*. 17

Cowen, L., T. Ideker, B. J. Raphael, and R. Sharan (2017a), Network propagation: a universal amplifier of genetic associations, *Nature Reviews Genetics*, *18*(9), 551–562, doi:10.1038/nrg.2017.38. 6

Cowen, L., T. Ideker, B. J. Raphael, and R. Sharan (2017b), Network propagation: a universal amplifier of genetic associations, *Nature Reviews Genetics*, *18*(9), 551. 16

Cowling, P., G. Kendall, and E. Soubeiga (2001), A hyperheuristic approach to scheduling a sales summit, in *Practice and Theory of Automated Timetabling III*, edited by E. Burke and W. Erben, pp. 176–190, Springer Berlin Heidelberg, Berlin, Heidelberg. 13

Dantzig, G. B., D. R. Fulkerson, and S. M. Johnson (1954), Solution of a large-scale traveling-salesman problem, *Operations Research*, *3*, 393–410. 11

Dou, Y., K. Shu, C. Xia, P. S. Yu, and L. Sun (2021), User preference-aware fake news detection, in *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 5

Dusad, V., D. Thiel, M. Barahona, H. C. Keun, and D. A. Oyarzún (2021), Opportunities at the interface of network science and metabolic modeling, *Frontiers in Bioengineering and Biotechnology*, *8*, doi:10.3389/fbioe.2020.591049. 16

Easley, D. A., and J. M. Kleinberg (2010), *Networks, Crowds, and Markets - Reasoning About a Highly Connected World.*, I-XV, 1-727 pp., Cambridge University Press. 2

Emmert-Streib, F., M. Dehmer, and B. Haibe-Kains (2014), Gene regulatory networks and their applications: understanding biological and medical problems in terms of networks, *Frontiers in Cell and Developmental Biology*, *2*, doi:10.3389/fcell.2014.00038. 2

Festa, P. (2014), A brief introduction to exact, approximation, and heuristic algorithms for solving hard combinatorial optimization problems, pp. 1–20, doi:10.1109/ICTON.2014.6876285. 12

Garcia, V., and J. Bruna (2018), Few-shot learning with graph neural networks. 17

Gilmer, J., S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl (2017), Neural message passing for quantum chemistry, *CoRR*, *abs/1704.01212*. 1

Glover, F., and M. Laguna (1997), *Tabu Search*, Kluwer Academic Publishers, Norwell, MA, USA. 13

Guo, Y., H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun (2021), Deep learning for 3d point clouds: A survey, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *43*(12), 4338–4364, doi:10.1109/TPAMI.2020.3005434. 18

Hadaj, P., D. Strzałka, M. Nowak, M. Łatka, and P. Dymora (2022), The use of PLANS and NetworkX in modeling power grid system failures, *Scientific Reports*, *12*(1), doi:10.1038/s41598-022-22268-z. 2

Hamilton, W. L. (2020), *Graph Representation Learning*, Springer International Publishing, doi:10.1007/978-3-031-01588-5. 3, 4, 5, 6

Hasin, Y., M. Seldin, and A. Lusis (2017), Multi-omics approaches to disease, *Genome Biology*, *18*(1), doi:10.1186/s13059-017-1215-1. 14

Hemmati, A., and L. M. Hvattum (2017), Evaluating the importance of randomization in adaptive large neighborhood search, *International Transactions in Operational Research*, *24*(5), 929–942, doi:https://doi.org/10.1111/itor.12273. 24

Hemmati, A., L. M. Hvattum, K. Fagerholt, and I. Norstad (2014), Benchmark suite for industrial and tramp ship routing and scheduling problems, *INFOR: Information Systems and Operational Research*, *52*(1), 28–38, doi:10.3138/infor.52.1.28. 22

Holland, J. H. (1992), Genetic algorithms, *Scientific American*. 13

Hornik, K., M. Stinchcombe, and H. White (1989), Multilayer feedforward networks are universal approximators, *Neural Networks*, *2*(5), 359–366, doi:https://doi.org/10.1016/0893-6080(89)90020-8. 8

Iovescu, C., and S. Rao (2017), The fundamentals of millimeter wave sensors. 19

Jain, A., A. R. Zamir, S. Savarese, and A. Saxena (2015), Structural-rnn: Deep learning on spatio-temporal graphs, *CoRR*, *abs/1511.05298*. 17

Jiang, W. (2022), Graph-based deep learning for communication networks: A survey, *Computer Communications*, *185*, 40–54, doi:https://doi.org/10.1016/j.comcom.2021.12.015. 2

Kennedy, J., and R. C. Eberhart (1995), Particle swarm optimization, in *Proceedings of the IEEE International Conference on Neural Networks*, pp. 1942–1948. 13

Kim, Y., and B. Toomajian (2016), Hand gesture recognition using micro-doppler signatures with convolutional neural network, *IEEE Access*, *4*, 7125–7130. 21

Kipf, T. N., and M. Welling (2016), Semi-supervised classification with graph convolutional networks, *CoRR*, *abs/1609.02907*. 5

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983a), Optimization by simulated annealing, *Science*, *220*(4598), 671–680, doi:10.1126/science.220.4598.671. 13

Kirkpatrick, S., C. D. Gelatt, and M. P. Vecchi (1983b), Optimization by simulated annealing, *Science*, *220*(4598), 671–680, doi:10.1126/science.220.4598.671. 23

Kool, W., H. van Hoof, and M. Welling (2019), Attention, learn to solve routing problems! 2, 13

Korte, B., and J. Vygen (2012), *Combinatorial Optimization*, Springer Berlin Heidelberg, doi:10.1007/978-3-642-24488-9. 5

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012), Imagenet classification with deep convolutional neural networks, in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, p. 1097–1105, Curran Associates Inc., Red Hook, NY, USA. 8

Kucharska, E. (2019), Dynamic vehicle routing problem—predictive and unexpected customer availability, *Symmetry*, *11*(4), doi:10.3390/sym11040546. 2

Lawler, E. L., and D. E. Wood (1966), Branch-and-bound methods: A survey, *Operations Research*, *14*(4), 699–719, doi:10.1287/opre.14.4.699. 12

LeCun, Y., K. Kavukcuoglu, and C. Farabet (2010), Convolutional networks and applications in vision, in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pp. 253–256, doi:10.1109/ISCAS.2010.5537907. 6

Lewis, N. E., H. Nagarajan, and B. O. Palsson (2012), Constraining the metabolic genotype-phenotype relationship using a phylogeny of in silico methods, *Nature Reviews Microbiology*, *10*(4), 291–305, doi:10.1038/nrmicro2737. 17

Li, F., K. Fujiwara, F. Okura, and Y. Matsushita (2021), A closer look at rotation-invariant deep point cloud analysis, in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 16,198–16,207, doi:10.1109/ICCV48922.2021.01591. 21

Li, X., W. Li, M. Zeng, R. Zheng, and M. Li (2020), Network-based methods for predicting essential genes or proteins: a survey, *Briefings in Bioinformatics*, *21*(2), 566–583, doi:10.1093/bib/bbz017. 17

Lu, H., X. Zhang, and S. Yang (2020), A learning-based iterative method for solving vehicle routing problems, in *International Conference on Learning Representations*. 14

MacNeil, L. T., and A. J. Walhout (2011), Gene regulatory networks and the role of robustness and stochasticity in the control of gene expression, *Genome Research*, *21*(5), 645–657, doi:10.1101/gr.097378.109. 15

Mastej, E., L. Gillenwater, Y. Zhuang, K. A. Pratte, R. P. Bowler, and K. Kechris (2020), Identifying protein–metabolite networks associated with copd phenotypes, *Metabolites*, *10*(4), doi:10.3390/metabo10040124. 2

Merkle, D., and M. Middendorf (2006), Marco dorigo and thomas stützle, ant colony optimization, mit press (2004) isbn 0-262-04219-3., *Eur. J. Oper. Res.*, *168*(1), 269–271. 13

Min, Y., Y. Zhang, X. Chai, and X. Chen (2020), An efficient pointlstm for point clouds based gesture recognition, in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5760–5769, doi:10.1109/CVPR42600.2020.00580. 21

Mobegi, F. M., A. Zomer, M. I. de Jonge, and S. A. F. T. van Hijum (2017), Advances and perspectives in computational prediction of microbial gene essentiality, *Briefings in Functional Genomics*, *16*(2), 70–79, doi:10.1093/bfgp/elv063. 17

Molchanov, P., X. Yang, S. Gupta, K. Kim, S. Tyree, and J. Kautz (2016), Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural network, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4207–4215. 21

Monk, J. et al (2017), iML1515, a knowledgebase that computes Escherichia coli traits, *Nature Biotechnology*, *35*(10), 904–908. 17

Muzio, G., L. O'Bray, and K. Borgwardt (2020), Biological network analysis with deep learning, *Briefings in Bioinformatics*, *22*(2), 1515–1530, doi:10.1093/bib/bbaa257. 15

Nguyen, H., S. Shrestha, D. Tran, A. Shafi, S. Draghici, and T. Nguyen (2019), A comprehensive survey of tools and software for active subnetwork identification, *Frontiers in genetics*, *10*, 155. 16

Orth, J. D., I. Thiele, and B. Ø. Palsson (2010), What is flux balance analysis?, *Nature biotechnology*, *28*(3), 245–8. 17

Palipana, S., D. Salami, L. A. Leiva, and S. Sigg (2021), Pantomime: Mid-air gesture recognition with sparse millimeter-wave radar point clouds, *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, *5*(1), doi:10.1145/3448110. 20

Papadimitriou, C. H., and K. Steiglitz (1982), *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Inc., USA. 11

Pisinger, D., and S. Ropke (2019), Large neighborhood search, in *Handbook of metaheuristics*, pp. 99–127, Springer. 23

Qi, C. R., H. Su, K. Mo, and L. J. Guibas (2016), Pointnet: Deep learning on point sets for 3d classification and segmentation, *CoRR*, *abs/1612.00593*. 18

Qi, C. R., L. Yi, H. Su, and L. J. Guibas (2017), Pointnet++: Deep hierarchical feature learning on point sets in a metric space, in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, p. 5105–5114, Curran Associates Inc., Red Hook, NY, USA. 21

Rayana, S., and L. Akoglu (2015), Collective opinion spam detection: Bridging review networks and metadata, in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, p. 985–994, Association for Computing Machinery, New York, NY, USA, doi:10.1145/2783258.2783370. 5

Ronen, J., and A. Akalin (2018), netSmooth: Network-smoothing based imputation for single cell RNA-seq, *F1000Research*, *7*. 16

Ropke, S., and D. Pisinger (2006), An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows, *Transportation Science*, *40*(4), 455–472, doi:10.1287/trsc.1050.0135. 23

Rosen, K. H. (2006), *Discrete Mathematics and Its Applications: And Its Applications*, McGraw-Hill Higher Education. 3

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986), Learning internal representations by error propagation, in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, edited by D. E. Rumelhart and J. L. Mcclelland, pp. 318–362, MIT Press, Cambridge, MA. 6

Shaw, P. (1998), Using constraint programming and local search methods to solve vehicle routing problems, in *Principles and Practice of Constraint Programming — CP98*, edited by M. Maher and J.-F. Puget, pp. 417–431, Springer Berlin Heidelberg, Berlin, Heidelberg. 23

Sutton, R. S., and A. G. Barto (2018), *Reinforcement Learning: An Introduction*, second ed., The MIT Press. 10

Sörensen, K., and F. Glover (2013), *Metaheuristics*, pp. 960–970, doi:10.1007/978-1-4419-1153-7_1167. 13

Thanou, D., X. Dong, D. Kressner, and P. Frossard (2017), Learning heat diffusion graphs, *IEEE Transactions on Signal and Information Processing over Networks*, *3*(3), 484–499, doi:10.1109/TSIPN.2017.2731164. 6

Thomas, N., T. E. Smidt, S. Kearnes, L. Yang, L. Li, K. Kohlhoff, and P. Riley (2018), Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds, *CoRR*, *abs/1802.08219*. 21

Tilwari, V., M. N. Hindia, K. Dimyati, D. N. K. Jayakody, S. Solanki, R. S. Sinha, and E. Hanafi (2021), Mbmqa: A multicriteria-aware routing approach for the iot 5g network based on d2d communication, *Electronics*, *10*(23), doi:10.3390/electronics10232937. 2

Turkeš, R., K. Sörensen, and L. M. Hvattum (2021), Meta-analysis of metaheuristics: Quantifying the effect of adaptiveness in adaptive large neighborhood search, *European Journal of Operational Research*, *292*(2), 423–442, doi:https://doi.org/10.1016/j.ejor.2020.10.045. 23

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017), Attention is all you need, *CoRR*, *abs/1706.03762*. 13

Wang, Y., Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon (2018), Dynamic graph CNN for learning on point clouds, *CoRR*, *abs/1801.07829*. 18

Wu, Z., B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. S. Pande (2017), Moleculenet: A benchmark for molecular machine learning, *CoRR*, *abs/1703.00564*. 5

Yang, X., P. Molchanov, and J. Kautz (2018), Making convolutional networks recurrent for visual sequence learning, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6469–6478. 21

Yang, Z., W. W. Cohen, and R. Salakhutdinov (2016), Revisiting semi-supervised learning with graph embeddings, *CoRR*, *abs/1603.08861*. 5

You, J., B. Liu, R. Ying, V. S. Pande, and J. Leskovec (2018), Graph convolutional policy network for goal-directed molecular graph generation, *CoRR*, *abs/1806.02473*. 3, 5, 11

Zachary, W. W. (1977), An information flow model for conflict and fission in small groups, *Journal of Anthropological Research*, *33*(4), 452–473, doi:10.1086/jar.33.4.3629752. 1

Zaheer, M., S. Kottur, S. Ravanbakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola (2017), Deep sets, *CoRR*, *abs/1703.06114*. 8, 18

Zhang, X., W. Xiao, and W. Xiao (2020), DeepHE: Accurately predicting human essential genes based on deep learning, *PLOS Computational Biology*, *16*(9), e1008,229, doi:10.1371/journal.pcbi.1008229. 17

Zhu, X., and Z. Ghahramani (2002), Learning from labeled and unlabeled data with label propagation. 6

Zhu, X., M. Gerstein, and M. Snyder (2007), Getting connected: analysis and principles of biological networks, *Genes & Development*, *21*(9), 1010–1024, doi:10.1101/gad.1528707. 14, 15

5 Scientific results

Paper I

A Graph Feature Auto-Encoder for the Prediction of Unobserved Node Features on Biological Networks

Ramin Hasibi and Tom Michoel
BMC Bioinformatics, **22**/**1** (2021)

Hasibi and Michoel *BMC Bioinformatics* (2021) 22:525
https://doi.org/10.1186/s12859-021-04447-3

BMC Bioinformatics

RESEARCH Open Access

A Graph Feature Auto-Encoder for the prediction of unobserved node features on biological networks

Ramin Hasibi* and Tom Michoel

*Correspondence:
Ramin.Hasibi@uib.no
Computational Biology Unit, Department of Informatics, University of Bergen, Bergen, Norway

Abstract

Background: Molecular interaction networks summarize complex biological processes as graphs, whose structure is informative of biological function at multiple scales. Simultaneously, omics technologies measure the variation or activity of genes, proteins, or metabolites across individuals or experimental conditions. Integrating the complementary viewpoints of biological networks and omics data is an important task in bioinformatics, but existing methods treat networks as discrete structures, which are intrinsically difficult to integrate with continuous node features or activity measures. Graph neural networks map graph nodes into a low-dimensional vector space representation, and can be trained to preserve both the local graph structure and the similarity between node features.

Results: We studied the representation of transcriptional, protein–protein and genetic interaction networks in *E. coli* and mouse using graph neural networks. We found that such representations explain a large proportion of variation in gene expression data, and that using gene expression data as node features improves the reconstruction of the graph from the embedding. We further proposed a new end-to-end Graph Feature Auto-Encoder framework for the prediction of node features utilizing the structure of the gene networks, which is trained on the feature prediction task, and showed that it performs better at predicting unobserved node features than regular MultiLayer Perceptrons. When applied to the problem of imputing missing data in single-cell RNAseq data, the Graph Feature Auto-Encoder utilizing our new graph convolution layer called FeatGraphConv outperformed a state-of-the-art imputation method that does not use protein interaction information, showing the benefit of integrating biological networks and omics data with our proposed approach.

Conclusion: Our proposed Graph Feature Auto-Encoder framework is a powerful approach for integrating and exploiting the close relation between molecular interaction networks and functional genomics data.

Keywords: Gene regulatory networks, Gene expression, Graph neural networks, Graph representation learning, Molecular networks, Omics, Feature prediction, Feature auto-encoder

Introduction

Biological networks of genetic, transcriptional, protein–protein, or metabolic interactions summarize complex biological processes as graphs, whose structure or topology is informative of biological function at multiple scales. For instance, degree distributions reflect the relative importance of genes or proteins in a cell; 3–4 node network motifs have well-defined information-processing roles; and network clusters or communities contain genes or proteins involved in similar biological processes [1–3]. Simultaneously, genomics, transcriptomics, proteomics, and metabolomics technologies measure the variation or activity of genes, proteins, or metabolites across individuals or experimental conditions [4, 5]. There is a rich history of integrating the complementary viewpoints of biological networks and omics data. For instance, "active subnetwork" identification methods treat omics data as features of network nodes in order to identify well-connected subnetworks that are perturbed under different conditions [6]. Network propagation or smoothing methods on the other hand use biological networks to extend partial information on some nodes (e.g., disease association labels, partially observed data) to other nodes (e.g., to discover new disease-associated genes or impute missing data) [7, 8]. However, existing methods treat biological networks as discrete structures, which are intrinsically difficult to integrate with continuous node features or activity measures.

Recently, with the advent of deep learning, the idea of representation learning on graphs has been introduced. In this concept, nodes, subgraphs, or the entire graph are mapped into points in a low-dimensional vector space [9]. These frameworks are known as graph neural networks (GNNs), and use deep auto-encoders to preserve the local structure of the graph around each node in the embedding, without having to specify in advance what "local" means. However, not much attention has been paid so far to the representation of the node features in these embeddings [10, 11].

In this paper, we propose a new framework using graph representation learning on biological networks which results in embeddings that are compatible with or informative for molecular profile data, concentrating for simplicity on gene expression data. The three main contributions of this study are:

1. We introduce a method to systematically measure the relationship between the structure of a network and the node feature (gene expression) values. This is done using the Graph Auto-Encoder (GAE) approach of [12] and measuring (i) the performance of reconstructing the network from the embedding, with and without expression data, and (ii) measuring the variance in expression values explained by the embedding matrix.
2. We propose the framework of *Graph Feature Auto-Encoder (GFAE)* for the prediction of expression values utilizing gene network structures, and introduce a new convolution layer named FeatGraphConv using a message passing neural networks (MPNNs) framework, tailored to reconstructing the representation of the node features rather than the graph structure.
3. We show that our new approach to gene expression prediction has practical applications in tasks such as imputation of missing values in single cell RNA-seq data and similar scenarios.

Related work on GNN

Assume that an undirected, unweighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $N = |\mathcal{V}|$ number of nodes has an adjacency matrix $\mathbf{A}$, where $A_{ij} = 1$ if there is an edge between nodes i and j and zero otherwise, and degree matrix $\mathbf{D}$, a diagonal matrix with the degrees (number of neighbours) of each node on the diagonal. Matrix $\mathbf{X} \in \mathbb{R}^{N \times Q}$, called the feature matrix, denotes node features. One of the first attempts at learning neural networks over graph structures was the convolution operation on graphs. For an input graph signal $x \in \mathbb{R}^N$, the spectral convolution is defined as

$$\mathbf{g}_\theta * x = \mathbf{U}\mathbf{g}_\theta \mathbf{U}^T x, \tag{1}$$

in which $\mathbf{U}$ is the matrix of eigenvectors of the symmetric Laplacian $\mathbf{L} = \mathbf{D} - \mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$. $\mathbf{U}^T x$ is called the Fourier transform of signal x and $\mathbf{g}_\theta$ is a matrix function of $\mathbf{\Lambda}$, the diagonal matrix of eigenvalues of $\mathbf{L}$.

Due to the high cost of calculating the eigenvalues in the case of large matrices, Hammond et al. [13] proposed to use a Chebyshev series expansion truncated after the K^{th} term to approximate the graph convolution operation with a K^{th}-order polynomial:

$$\mathbf{g}_\theta * x \approx \mathbf{U} \sum_{k=0}^{K} \theta'_k T_k(\tilde{\mathbf{\Lambda}}) \mathbf{U}^T x = \sum_{k=0}^{K} \theta'_k T_k(\tilde{\mathbf{\Lambda}}) x, \tag{2}$$

in which $T_k(.)$ and θ'_k are the k^{th}-order Chebyshev polynomials and expansion coefficients, respectively, $\tilde{\mathbf{\Lambda}} = \frac{2}{\lambda_{max}}\mathbf{\Lambda} - \mathrm{I}_N$ with λ_{max} the largest eigenvalue of $\mathbf{\Lambda}$ and I_N an identity matrix with size $N \times N$, and finally $\tilde{\mathbf{L}} = \mathbf{U}\tilde{\mathbf{\Lambda}}\mathbf{U}^T = \frac{2}{\lambda_{max}}\mathbf{L} - \mathrm{I}_N$.

In the graph convolutional network (GCN) [14], further approximations were done by setting $K = 1$, $\lambda_{max} \approx 2$, and $\theta = \theta'_0 = -\theta'_1$. As a result, formula (2) was transformed into

$$\mathbf{g}_\theta * x \approx (\mathrm{I}_N + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})x. \tag{3}$$

Repeated application of $\mathbf{g}_\theta$ resulting in high powers of $\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ can cause numerical instabilities. Kipf and Welling [14] suggested to set the diagonal elements of $\mathbf{A}$ to 1 (add self-loops) and to recompute $\mathbf{D}$ according to the updated adjacency matrix. Therefore, they used the symmetrically normalized adjacency matrix $\tilde{\mathbf{A}}$ in their convolution layer, with

$$\tilde{\mathbf{A}} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}. \tag{4}$$

Thus, the forward operation in a GCN for $\mathbf{X}$ is computed as

$$\mathrm{GCN}(\mathbf{X}, \tilde{\mathbf{A}}) = \sigma(\tilde{\mathbf{A}}\,\mathrm{ReLU}(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}_0)\mathbf{W}_1) \tag{5}$$

with weight matrices $\mathbf{W}_i$ containing the trainable weights for each input feature, and σ a non-linear task specific function such as softmax for a node classification problem [10].

Additional studies on GNNs have shown that a GNN can be viewed as a message-passing approach based on graph structure, where every node's message for its neighbours is the aggregation of the neighbourhood information, in which the aggregation is done through a trainable neural network [15]. This framework is also known as a

MPNN. The forward pass in such a network consists of a message passing phase and a readout phase. In the message passing phase, the hidden representation of each node is updated through an update function, which aggregates the previous node representation and the messages of its neighbours according to:

$$h_i^k = \gamma^k(h_i^{k-1}, \text{Pool}_{j\in N(i)}(M(h_i^{k-1}, h_j^{k-1}, e_{ij}))), \tag{6}$$

in which h_i^k is the hidden representation of node i in layer k, with h_i^0 being the node i's input features and e_{ij} is the edge attribute between nodes i and j. Additionally, γ and M are both differentiable functions called the update and message functions, respectively and Pool is a permutation invariant pooling function. Furthermore, $N(i)$ denotes the set of neighbouring nodes of node i.

In the readout phase, the feature vector of each node in the graph is computed using some learnable, differentiable readout function R according to

$$\mathbf{Y} = \mathbf{R}(\{h_i | i \in \mathcal{V}\}), \tag{7}$$

in which, $\mathbf{Y}$ is the predicted labels. Different settings of γ, M, Pool, and $\mathbf{R}$ can lead to different MPNN convolution layers specific to different tasks and scenarios.

Methods

In this section, we present two different GFAE frameworks of predicting expression values, leveraging the gene regulatory network structure, whose pipelines are depicted in Fig. 1. In this scenario, $\mathbf{X} \in \mathbb{R}^{N\times Q}$ is a matrix of expression values in Q different experiments for N number of genes (molecular profiles) and $\mathbf{A}$ denotes the adjacency matrix of the gene network.

Structural embedding for indirect prediction of expression values

In this approach, we used the Non-probabilistic GAE model of [12] to represent the structure of a gene network as depicted in Fig. 1A. First the GCN operation, shown in formula (5), is modified by setting σ in formula (5) to the identity function:

$$\text{GCN}(\mathbf{X}, \mathbf{A}) = \tilde{\mathbf{A}}\text{ReLU}(\tilde{\mathbf{A}}\mathbf{X}\mathbf{W}_0)\mathbf{W}_1, \tag{8}$$

in which W_0 and $\mathbf{W}_1$ are trainable weights. Therefore, the embedding matrix of the graph adjacency $\mathbf{Z}$ is calculated by

$$\mathbf{Z} = GCN(\mathbf{X}, \mathbf{A}). \tag{9}$$

Furthermore, the weight matrices $\mathbf{W}_0$ and $\mathbf{W}_1$ in formula (8) are trained by measuring how well the embedding reconstructs the graph adjacency matrix, where the reconstructed adjacancy matrix $\hat{\mathbf{A}}$ is defined as

$$\hat{\mathbf{A}} = \text{Sigmoid}(\mathbf{Z}\mathbf{Z}^T). \tag{10}$$

The cross-entropy error over all the edges in the matrix is used as a loss function,

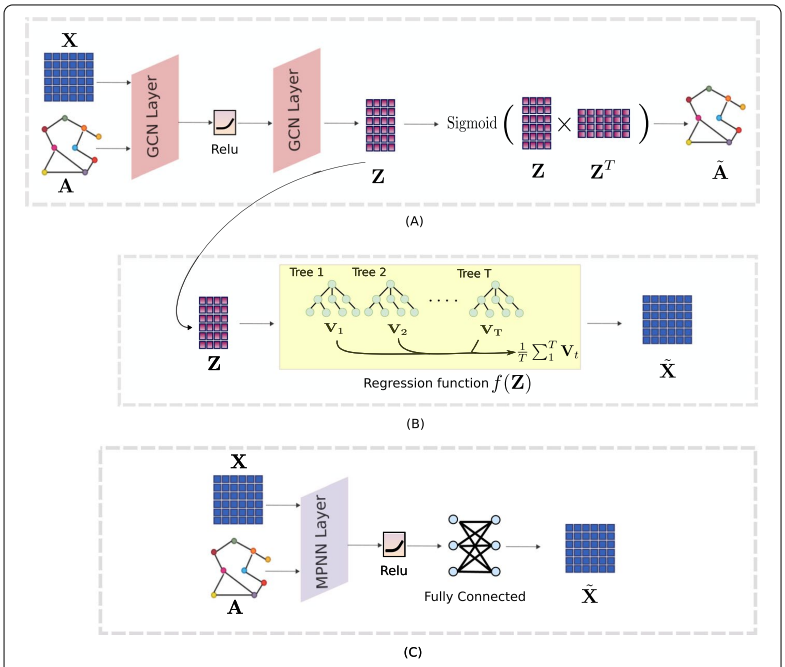


Fig. 1 **A** Depicts the GAE scheme of [12] tailored to the reconstruction of the adjacency matrix of the graph. In **B** we take the embedding matrix of a GAE and train an indirect regression task for the prediction of the expression values. **C** Illustrates our proposed GFAE approach for end-to-end learning of graph node features

$$\mathcal{L} = -\sum_{n=1}^{N} \mathbf{A}_n \ln \hat{\mathbf{A}}_n, \tag{11}$$

where $\mathbf{A}_n$ and $\hat{\mathbf{A}}_n$ are the adjacency rows of the nth node in $\mathbf{A}$ and $\tilde{\mathbf{A}}$, respectively. The training of the neural network is done by gradient descent and stochasticity added by dropout rate. We use the metrics of average precision and area under the ROC curve related to the reconstruction of $\mathbf{A}$, and the Variance Explained of $\mathbf{X}$ by $\mathbf{Z}$ to quantify the relationship between the node features and the graph structure.

As shown in Fig. 1B, the expression values of the genes are obtained following

$$\tilde{\mathbf{X}} = f(\mathbf{Z}) \tag{12}$$

where $\tilde{\mathbf{X}}$ denotes the predicted expression values. Moreover, f is a regression function for which we consider linear regression (LR) and random forest (RF) regression as examples.

Message passing neural network for end-to-end prediction of expressions

In the second method, we apply the message passing formula of (6) on the input expression values in which the messages (hidden representations) are propagated to each gene from neighbours in the gene network. As shown in Fig. 1C, in this approach, the model is trained in an end-to-end framework to predict the expression

values directly from the input, without the need for training a separate regression model. To establish the performance of this framework, we used three popular message passing schemes for finding the hidden representation of the genes, as well as introducing our own, for the task of predicting gene expression values. These three methods are inductive GCN, GraphSAGE [16], and the GNN operator from [17] (from here on referred to as GraphConv). According to [15], a single GCN layer can also be viewed as a message passing scheme between the nodes in the graph in the format of formula (6):

$$h_i^k = \sum_{j \in N(i) \cup i} \frac{1}{\sqrt{deg(i)} * \sqrt{deg(j)}} . (\mathrm{W} h_j^{k-1}), \tag{13}$$

in which $deg(i)$ is the number of neighbours of node i, W is a trainable weight matrix, and $\sum$ is the sum pooling operator. This scheme is equivalent to running a single GCN layer in formula (8). Another MPNN layer is GraphSAGE, whose formula is given by

$$h_i^k = \mathrm{W}_1(h_i^{k-1}) + \mathrm{W}_2 \mathrm{Mean}_{j \in N(i) \cup i}(h_j^{k-1}), \tag{14}$$

in which Mean is the mean pooling operator, and $\mathbf{W}_1$ and $\mathbf{W}_2$ are trainable weight matrices. GraphSAGE MPNN assumes that the representation of each node is the summation of the output from the previous layer and the average of the representation of the adjacent nodes. The final MPNN layer, named GraphConv, is calculated through

$$h_i^k = \mathrm{W}_1 h_i^{k-1} + \sum_{j \in N(i)} \mathrm{W}_2 . h_j^{k-1}, \tag{15}$$

in which $\mathbf{W}_1$ and $\mathbf{W}_2$ are trainable weight matrices, and $\sum$ is the sum pooling operator. In this layer, the representation of each node is the sum product of the previous layer representation and the summation of the incoming messages from adjacent nodes.

In our proposed version of MPNN, FeatGraphConv, we first obtain a representation of every node's features by running them through a linear layer in the message function M. This step helps the layer to find optimized message representations for the propagation phase. Then we aggregate the incoming neighbours' messages by a mean pooling operator, based on the hypothesis that in a gene network, a gene's expression value is intermediate between its neighbours expression values. For the update function γ, we concatenate the node's embedding with its aggregated messages, and run them through a shared weight network, which determines how important each of these values are in predicting the features of the node. Hence, the formulae for our FeatGraphConv operator are as follows

$$\begin{aligned} g_i^k &= \mathrm{W}_1 * h_i^{k-1} \\ h_i^k &= \mathrm{W}_2(g_i^k || \mathrm{Mean}_{j \in N(i) \cup i}(g_j^k)), \end{aligned} \tag{16}$$

in which $(.||.)$ is the concatenation function and $\mathbf{W}_1$ and $\mathbf{W}_2$ represent trainable weight matrices. In all of the four mentioned layers, the readout function $\mathbf{R}$ is defined as a fully connected linear layer. Thus, $\tilde{\mathbf{X}}_i$ the predicted expression values for node i are obtained through

$$\tilde{\mathbf{X}}_i = W * h_i + b, \tag{17}$$

in which W and b represent trainable weights and bias matrices respectively. For finding the optimal weights in this framework, mean squared error (MSE) on predicted expression values is used as the loss function. This method is considered to be a semi-supervised training framework due to utilizing the complete structure of the graph in training the model.

Prediction of expressions from expressions

For comparison of the results obtained through "Structural embedding for indirect prediction of expression values" and "Message passing neural network for end-to-end prediction of expressions" sections, we also consider prediction of $\tilde{\mathbf{X}}$ directly from $\mathbf{X}$ through simple machine learning algorithms. These algorithms include:

- *Multi layer perceptron (MLP)* A simple form of a neural network which maps the input features into output features through multiple layers of neurons (computing units).
- *Linear regression* A linear model for mapping the input to the output.
- *Random forest* A set of decision tree models that determine the output value through the aggregation of the output of decision trees that each are trained on a subset of $\mathbf{X}$
- *Markov affinity-based graph imputation of cells (MAGIC)* Uses signal-processing principles similar to those used to clarify blurry and grainy images to recover missing values from already existing ones in a matrix [18]

Experimental setup

In this section, we describe three experiments that are done to measure the relationship between gene network structure and expression values as well as thoroughly evaluate the performance of the proposed GFAE. The hyper-parameters of all the experiments were determined after some initial experiments on a separate validation set and were kept the same for all the models, to measure the predictive performance of different approaches under the same set of initial circumstances. These hyper-parameters are listed in Table 1. In order to make a sound and thorough performance evaluation, two masking methods are used to divide the data for training and evaluation, the details of which are explained below.

Table 1 Hyper-parameters of the graph neural network

Hyper-parameter	Node embedding	MPNN
Epochs	500	20,000
Initial learning rate	0.001	0.001
First hidden layer size	64	64
Second hidden layer size	32	32

Masking mechanism for the separation of train and test expression values

For evaluation purposes, we separate the expression values of $\mathbf{X}$ and $\tilde{\mathbf{X}}$ into two sets of training and testing. For this goal, two different masking techniques are used, the schemes of which are illustrated in Fig. 2. First, The input expression values $\mathbf{X}$ and the expression values that are to be predicted $\tilde{\mathbf{X}}$ are split into train and test through

$$\begin{aligned} \mathbf{X}_{train} &= M_{train} \circ \mathbf{X}, \qquad \tilde{\mathbf{X}}_{train} = \tilde{M}_{train} \circ \tilde{\mathbf{X}}, \\ \mathbf{X}_{test} &= M_{test} \circ \mathbf{X}, \qquad \tilde{\mathbf{X}}_{test} = \tilde{M}_{test} \circ \tilde{\mathbf{X}}, \end{aligned} \tag{18}$$

where $\circ$ is the Hadamard product and $M_{train}, M_{test}, \tilde{M}_{train}, \tilde{M}_{test} \in \{0,1\}^{N \times Q}$ are binary matrices which have the value 1 in train and test indices, respectively. The goal is to train the models to predict the values of $\tilde{\mathbf{X}}_{train}$ using the values in $\mathbf{X}_{train}$ as input and evaluate the models when predicting $\tilde{\mathbf{X}}_{test}$ with $\mathbf{X}_{test}$ as input features. In the first masking method as depicted in Fig. 2A, the masking is done in such a way that both experiments (columns) and genes (rows) in expression profile matrix are split into separate train and test sets. Furthermore, in this approach, a model is trained to predict each column of the $\tilde{\mathbf{X}}$ independently (experiments based) to make the evaluation possible for regression functions, since they are only capable of predicting one value for each gene.

In the second masking mechanism (Fig. 2B), also refered to as the imputation masking, following the imputation mechanism in auto-encoders, we set $\tilde{\mathbf{X}} = \mathbf{X}$ to measure the reconstruction ability of each model in an auto-encoder framework resulting in only two splits of $\mathbf{X}_{train}$ and $\mathbf{X}_{test}$. Thus, M_{train} is set to 1 for some elements of $\mathbf{X}$ at random and M_{test} is calculated as

$$M_{test} = \neg M_{train}, \tag{19}$$

where $\neg$ indicates the logical not operator. Additionally, K-fold cross-validation is used in both masking techniques to ensure the soundness of all obtained results with K set to 10 or 3 depending on the time complexity of the specific experiment.

Experiment on gene network structure embedding

In this experiment, we obtain the embedding matrix of the graph structure $\mathbf{Z}$ and measure the performance of the graph auto-encoder in calculating $\tilde{\mathbf{A}}$ (see "Structural embedding for indirect prediction of expression values" section). We used the PytorchGeometry implementation of the graph auto-encoder provided by [19]. For our approach, the normal auto-encoder provided in the package was used, and the variational auto-encoder was omitted.

Five different sets of input graphs and features to the model were tested:

1. *Random graph* In this approach $\mathbf{Z}$ is calculated by

$$\mathbf{Z} = \mathrm{GCN}(\mathrm{I}_N, \mathbf{A}_{rand}), \tag{20}$$

in which, I_N and $\mathbf{A}_{rand}$ represent an identity matrix of size $N \times N$ and the adjacency matrix of a random graph, respectively. For generating random graphs, we used the random graph generator of the Python3 package NetworkX, using the Erdős–Rényi model [20] (Additional file 1).

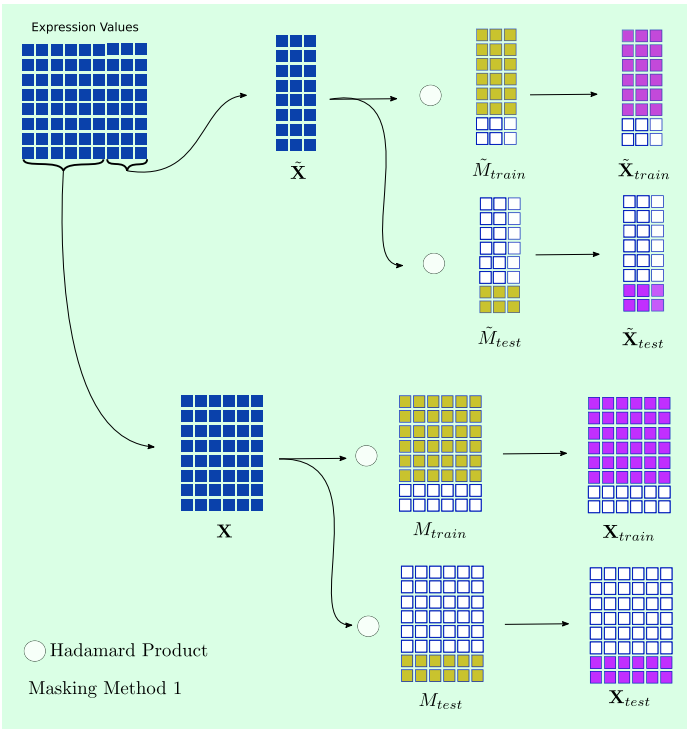


Fig. 2 The two different approaches of masking of expression values. In **A**, each dimension of $\mathbf{X}_{test}$ is predicted using a separate model trained on $\mathbf{X}_{train}$. In **B**, a single model is used to predict all the values of the feature matrix

2. *Expression + random graph* In this approach, the identity matrix is replaced with the actual expression values of genes as input features:

$$\mathbf{Z} = \mathrm{GCN}(\mathbf{X}, \mathbf{A}_{rand}). \tag{21}$$

3. *Real graph* Following the approach used by [12], the embedding matrix $\mathbf{Z}$ in this case is calculated by

$$\mathbf{Z} = \mathrm{GCN}(\mathrm{I}_N, \mathbf{A}), \tag{22}$$

where $\mathbf{A}$ is the adjacency matrix of the input graph.

4. *Expression + real graph* The embedding $\mathbf{Z}$ in this case is calculated through formula (9),

$$\mathbf{Z} = \mathrm{GCN}(\mathbf{X}, \mathbf{A}).$$

5. *Expression* The network in this model, is inferred from the (absolute) correlation between the expression values of different genes. In this approach, the correlation directly outputs the probability of the edge between two nodes.

By choosing the identity matrix as input features in input setting 1 and 3, each of the nodes has a distinct set of features, which do not give any indication about the functionality of the node. This way the model will only pay attention to the graph structure when producing the embedding matrix. The edge set of the graph ($\mathcal{E}$) is split into train and test sets and performance metrics of average precision (AP), area under ROC curve (AUC) on the test edge set, as well as the variance of $\mathbf{X}$ explained by $\mathbf{Z}$ are calculated for evaluation. The benefit of this experiment is that it allows to measure the relationship between the expression values and the structure of different gene networks through different metrics obtained from five different input settings as mentioned above.

Experiment on the prediction of expression values using the proposed GFAE

In this experiment, the first masking mechanism (Fig. 2A and "Masking mechanism for the separation of train and test expression values" section) is used to evaluate different models for predicting expression values utilizing the structure of the network. Three sets of models are compared: indirect (Fig. 1B and "Structural embedding for indirect prediction of expression values" section), end-to-end framework (Fig. 1C and "Message passing neural network for end-to-end prediction of expressions" section), and baseline regression models ("Prediction of expressions from expressions" section). Moreover, two settings of inputs, $(\mathbf{X}, A)$ and $(\mathrm{I}_N, \mathbf{A})$, are used in the indirect and end-to-end models to compare their performance with and without input expression values. The purpose of this experiment is that it allows for a simple performance comparison between graph-based prediction methods in a sample regression task. Furthermore, prediction of expression values solely based on graph structure is possible in this setting. The average MSE for the prediction of each column of $\tilde{\mathbf{X}}_{test}$ is reported as the performance metric.

Experiment on the imputation performance of the proposed GFAE

For this experiment, the second masking approach ("Masking mechanism for the separation of train and test expression values" section) or imputation masking is applied

on the input $\mathbf{X}$ in end-to-end models (Fig. 1C and "Message passing neural network for end-to-end prediction of expressions" section). The goal of this experiment is to evaluate the proposed GFAE in imputation tasks such as the imputation of missing values in Single Cell RNA-seq datasets, as well as to compare the reconstruction ability of the proposed framework against traditional auto-encoders. The MSE of $\mathbf{X}_{test}$ is the metric used in this experiment to compare different auto-encoders.

Datasets

We evaluated the performance of our method on data for the organisms *Escherichia Coli* (*E. coli*) and *Mus Musculus*.

- *Network datasets* For *E. coli*, we used transcriptional, protein–protein, and genetic interaction networks. The transcription network was obtained from RegulonDB [21]. All the positive and negative regulatory effects from the TF-gene interactions dataset file were included regardless of their degree of evidence (strong or weak) to construct the adjacency matrix. A PPI and genetic interaction network were obtained from BioGRID [22, 23]. We extracted the interactions from the file "BIOGRID-ORGANISM-Escherichia_coli_K12_W3110-3.5.180", and considered the "physical" and "genetic" values of 'Experimental System Type' column for constructing the PPI and genetic networks, respectively. For *Mus Musculus*, we used a protein–protein interaction network extracted in the same way from the file "BIOGRID-ORGANISM-Mus_musculus-3.5.182".
- *Expression level dataset* For *E. coli*, we used the Many Microbes Microarray Database (M3DB) [24, 25]. All the experiments from the file "avg_E_coli_v4_Build_6_exps466probes4297" were used to construct the feature matrix. For *Mus Musculus*, the single cell RNA-Seq data from [26] were obtained from the Gene Expression Omnibus.

For each of the networks, the common genes between the network and the expression data were extracted, and an adjacency matrix and a matrix of features were constructed from the network and expression level datasets, respectively. A detailed description of each of the networks is available in Table 2.

Computational resources and source code

All the experiments were done on a Tesla V100 with Python 3. The source code of the experiments is availabe at https://github.com/RaminHasibi/GraphFeatureAutoencoder.

Table 2 Summary description of benchmark datasets

		TF_net		PPI		Genetic	
Input	Expressions	Nodes	Edges	Nodes	Edges	Nodes	Edges
E. coli	466	1559	3184	1929	11,592	3688	147,475
Mus Musculus	1468	–	–	9951	75,587	–	–

Results

Graph structural embeddings reconstruct gene networks and explain variation in gene expression

We obtained low-dimensional embeddings of transcriptional regulatory (TF_net), protein–protein interaction (PPI) and genetic interaction networks in *E. coli* and a PPI network in mouse (MPPI), with and without using expression data as node features, and trained the GAE to optimize the reconstruction of the original graph from the node embedding (see "Methods" section for details). Figure 3 shows the results for the graph reconstruction task for various embeddings. As seen in Fig. 3A, B, embeddings learned from the structure of the real graph alone ("Graph") performed considerably better than embeddings learned from random graphs ("Random Graph"), as expected, in terms of both AUC and AP. The same was true for a standard Pearson correlation coexpression network inferred from the expression data alone ("Expression"), showing that graph embeddings and gene expression data independently predict graph structure.

When gene expression data were used as node feature inputs to the GAE (see "Structural embedding for indirect prediction of expression values" and "Experiment on gene network structure embedding" sections), graph reconstruction performance further increased ("Expression + Graph" row), but this was not the case when expression

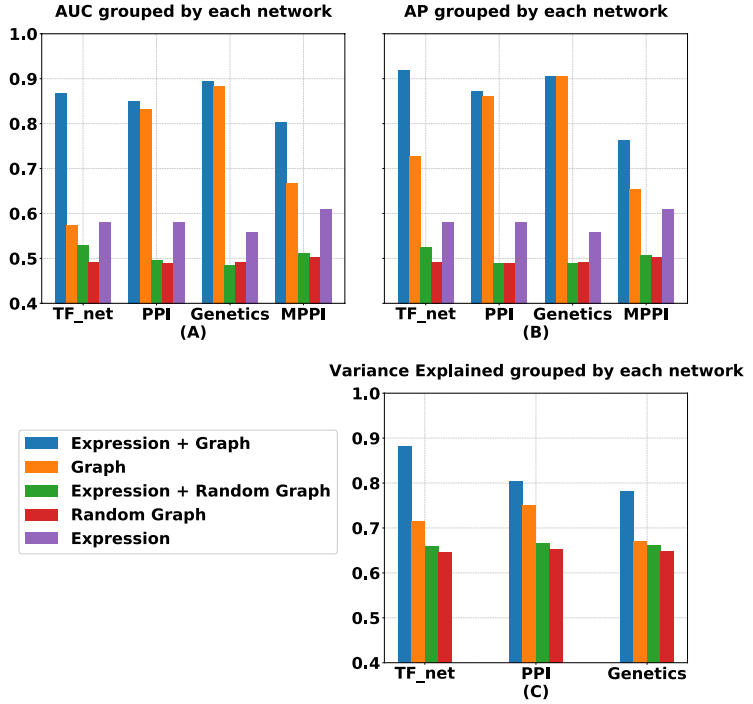


Fig. 3 **A**, **B** Show AUC and AP of test set edge prediction on different gene networks given different input settings. **C** Shows the variance explained of the expression data matrix **X** by the embedding matrix **Z** (for exact values see the Additional file 1)

data was combined with random graphs ("Expression + Random Graph" row). In other words, graph embeddings where the distance between nodes respects both their graph topological distance and their expression similarity result in better graph reconstruction than embeddings that are based on topological information alone. This shows that expression profiles are informative of graph structure in a way that is consistent with, but different from, the traditional view where networks are inferred directly from expression data using expression similarity measures.

Next we computed the variance of the expression data explained by the different embeddings (see Fig. 3C and details in the Additional file 1). Despite being trained on the graph reconstruction task, graph embeddings learned with and without expression data as node features explained a high percentage of variation in the expression data, but not when random graphs were used.

In summary, graph representation learning results in low-dimensional node embeddings that faithfully reconstruct the original graph as well as explain a high percentage of variation in the expression data, suggesting that graph representation learning may aid the prediction of unobserved expression data.

Indirect and end-to-end GFAE predict unobserved expression values

As mentioned in "Experiment on the prediction of expression values using the proposed GFAE" section, we considered three categories of prediction methods: (i) standard baseline methods that don't use graph information (LR, RF, MLP, see "Prediction of expressions from expressions" section), (ii) standard regression methods trained on graph embeddings instead of directly on the training data (LR-embedding and RF-embedding, see "Structural embedding for indirect prediction of expression values" section), and (iii) graph MPNN methods for end-to-end learning of features (GCN, GraphSage, GraphConv, and FeatGraphConv, see "Message passing neural network for end-to-end prediction of expressions" section and Fig. 1. Table 3 shows the performance (average mean squared error) of all methods on the *E. coli* data. For this experiment the mouse single-cell RNA-seq data was omitted due to sparsity of the expression values. The "Features" and "Graph" columns indicate the input settings of $(\mathbf{X}, A)$ and $(\mathrm{I}_N, \mathbf{A})$, respectively.

The newly proposed graph convolution layer of FeatGraphConv is able to predict the unobserved expression values better than the other graph convolutions, due to the fact

Table 3 The average MSE of predicting test expression values $\tilde{\mathbf{X}}_{test}$ using different models

Method	TF_Net		PPI		Genetics	
	Features	Graph	Features	Graph	Features	Graph
GCN	7.791 ± 3.550	15.127 ± 2.280	6.208 ± 0.607	11.106 ± 0.52198	5.988 ± 0.696	4.560 ± 0.351
GraphSAGE	0.332 ± 0.160	8.078 ± 2.592	0.265 ± 0.135	**2.844 ± 0.349**	0.233 ± 0.127	4.466 ± 1.605
GraphConv	0.318 ± 0.154	13.812 ± 4.534	0.308 ± 0.139	3.094 ± 0.431	0.234 ± 0.116	5.226 ± 1.054
FeatGraphConv	**0.285 ± 0.135**	**7.525 ± 2.941**	**0.244 ± 0.130**	5.207 ± 1.476	**0.201 ± 0.112**	**3.414 ± 0.691**
LR-embedding	**1.583 ± 0.200**	2.279 ± 0.403	1.091 ± 0.166	1.453 ± 0.264	**1.863 ± 0.332**	**1.653 ± 0.271**
RF-embedding	1.945 ± 0.318	**2.150 ± 0.363**	**1.472 ± 0.267**	**1.452 ± 0.267**	1.883 ± 0.343	1.897 ± 0.351
MLP	0.424 ± 0.170	–	0.354 ± 0.153	–	0.332 ± 0.134	–
LR	**0.215 ± 0.126**	–	**0.147 ± 0.105**	–	**0.108 ± 0.084**	–
RF	0.507 ± 0.143	–	0.194 ± 0.103	–	1.882 ± 0.343	–

(Bold indicates lowest error mean per category of experiments for each group of methods (end-to-end, indirect, or baseline)

that this layer is tailored to the prediction of features rather than the reconstruction of the graph. As expected, all end-to-end methods perform considerably better when training data is included as node features. The end-to-end methods, with the exception of GCN, also perform better than the indirect methods where regression models are trained on graph embeddings. We also observe that the lowest MSE overall is in fact obtained by baseline LR on the training data alone. However, experiments on FeatGraphConv with 20,000 iterations (as opposed to the default of 500 used for all end-to-end methods in Table 3) showed that this model can decrease the MSE to 0.204 ± 0.12, 0.133 ± 0.089, and 0.107 ± 0.083 for each of the TF_net, PPI, and Genetic networks, respectively, which is better than LR. However, due to the high number of experiments and the need to train a different model for each of the experiments of $\tilde{\mathbf{X}}_{test}$, it is not computationally efficient to train the more complex GNN models with a higher number of iterations by default for this prediction task.

On the other hand, when the graph structure alone is used ("Graph"), the indirect embedding-based methods achieve lower error. This could be due to the fact that these models better capture the structure of the graph, since their loss function is defined on the reconstruction of the adjacency matrix. Hence when the graph structure is the only information provided to the model, they are able to better capture this information and therefore obtain an embedding that better predicts expression data (on the basis of the results in "Graph structural embeddings reconstruct gene networks and explain variation in gene expression" section), compared to end-to-end models which try to predict the expression directly and are operating blindly when expression values are provided as input.

Graph feature auto-encoding improves the imputation of randomly missing values in single-cell RNA-seq data

Based on the results in the previous section, we next considered the more challenging prediction task where unobserved node features are randomly distributed over the nodes and differ between experiments, that is, the task of imputing randomly missing data (Fig. 2B). Since there are no fixed sets of training and test nodes, neither the baseline regression methods of LR and RF, nor the indirect frameworks are applicable in this case ("Structural embedding for indirect prediction of expression values" section). In contrast, the end-to-end GFAE methods allow to train a single model for the prediction of all the $\mathbf{X}_{test}$ values, which may be placed in any possible order inside the feature matrix. We used these models for the prediction of expression values in *E. coli* and of non-zero values in the single-cell RNAseq data in mouse, and benchmarked them against two methods that don't use graph information, namely a normal MLP used in an auto-encoder scheme, and MAGIC, a method designed specifically to impute missing data in single-cell RNA-seq data [18] (see "Prediction of expressions from expressions" section).

As shown in Table 4, our FeatGraphConv convolution layer is able to predict missing features more accurately than all other methods. It is interesting to note that graph convolution layers, with the exception of GCN, outperform MAGIC on the single-cell RNAseq imputation task, although the MLP, which does not use graph information, also

Table 4 The average MSE of predicting randomly distributed test values using different auto-encoder models

Model	*E. coli*			*Mus Musclus*
	TF_Net	**PPI**	**Genetics**	**PPI**
GCN	0.043 ± 0.00175	0.065 ± 0.004	0.114 ± 0.004	0.011 ± 0.001
GraphSAGE	0.027 ± 0.0007	0.023 ± 0.0004	0.026 ± 0.0003	0.004 ± 0.0006
GraphConv	0.041 ± 0.003	0.068 ± 0.05	0.182 ± 0.046	2.06 ± 2.73
FeatGraphConv (our)	$\mathbf{0.025 \pm 0.0008}$	$\mathbf{0.023 \pm 0.0006}$	$\mathbf{0.025 \pm 0.0004}$	$\mathbf{0.003 \pm 0.0002}$
MLP Auto-encoder	0.031 ± 0.0007	0.028 ± 0.0003	0.027 ± 0.0004	0.004 ± 0.0005
MAGIC	3.505 ± 0.006	3.661 ± 0.017	3.215 ± 0.003	0.050 ± 0.0002

(Bold indicates lowest error mean per network)

performs well in this case. The under-performance of GCN in these experiments can be explained by the fact that a GCN is primarily concerned with the structure representation of each node through multiplication of the degrees of neighbouring nodes (formula (13)). This captures the graph structure well, but has a negative effect on the prediction of node features. This is evident from the fact that other convolution layers that did not take node degrees into consideration performed better in the tasks given.

Discussion

In this paper we studied whether GNN, which learn embeddings of nodes of a graph in a low-dimensional space, can be used to integrate discrete structures such as biological interaction networks with information on the activity of genes or proteins in certain experimental conditions. Traditionally, this is achieved by for instance network propagation methods, but these methods do not extract quantitative information from a graph that could be used for downstream modelling or prediction tasks. GNN on the other hand can include node features (gene or protein expression levels) in the learning process, and thus in theory can learn a representation that better respects the information contained in both data types. Thus far the integration of node features in graph representation learning has mainly been pursued for the task of link prediction. Here instead we focused on the task of predicting unobserved or missing node feature values.

We showed that representations learned from a graph and a set of expression profiles simultaneously result in better reconstruction of the original graph and higher expression variance explained than using either data type alone, even when the representations are trained on the graph reconstruction task. We further proposed a new end-to-end GFAE which is trained on the feature reconstruction task, and showed that it performs better at predicting unobserved node features than auto-encoders that are trained on the graph reconstruction task before learning to predict node features.

Predicting or imputing unobserved node features is a common task in bioinformatics. In this paper we demonstrated the value of our proposed GFAE on the problem of imputing missing data in single-cell RNAseq data, where it performs better than a state-of-the-art method that does not include protein interaction data. Other potential application areas are the prediction of new disease-associated genes from a seed list of known disease genes on the basis of network proximity [7], or the prediction of non-measured

transcripts or proteins from new low-cost, high-throughput transcriptomics and proteomics technologies that only measure a select panel of genes or proteins [27, 28] which we intend to look into in our future works.

A potential drawback of our method is that it assumes that the interaction graph is known and of high-quality. Future work could investigate whether it is feasible to learn graph representations that can do link prediction and node feature prediction simultaneously, or whether network inference followed by graph representation learning for one type of omics data (e.g. bulk RNAseq data) can aid the prediction of another type of omics data (e.g. single-cell RNAseq).

In summary, our GFAE framework is a stepping stone in a new direction of applying graph representation learning to the problem of integrating and exploiting the close relation between molecular interaction networks and functional genomics data, not only for network link prediction, but also for the prediction of unobserved functional data.

Abbreviations

GNN: Graph neural networks; LR: Linear regression; RF: Random forest; GCN: Graph convolution networks; MPNN: Message passing neural network; MLP: Multi layered perceptron; MAGIC: Markov affinity-based graph imputation of cells; MSE: Mean square error; TF_net: Transcriptional regulatory network; PPI: Protein–protein interaction; AUC: Area under the ROC curve; AP: Average precision.

Supplementary Information

The online version contains supplementary material available at https://doi.org/10.1186/s12859-021-04447-3.

Additional file 1. Appendix of the paper containing extra information regarding the experiments.

Acknowledgements

We would like to express our deepest gratitude and appreciation for all reviewers and editors.

Authors' contributions

RH designed and implemented all the baseline methods, the improved auto-encoder layer, and the training scheme and implementation, performed the experiments, and drafted the manuscript. TM contributed to the design of experiments and the auto-encoder layer, and to improving the writing of the manuscript. Both authors read and approved the final manuscript.

Funding

TM acknowledges funding from the Research Council of Norway (#312045).

Availability of data and materials

The source code of this project is written in python and is available on https://github.com/RaminHasibi/GraphFeatureAutoencoder.

Declarations

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Received: 2 February 2021 Accepted: 13 October 2021
Published online: 27 October 2021

References

1. Barabási A-L, Oltvai ZN. Network biology: understanding the cell's functional organization. Nat Rev Genet. 2004;5:101–13.

2. Zhu X, Gerstein M, Snyder M. Getting connected: analysis and principles of biological networks. Genes Dev. 2007;21(9):1010–24. https://doi.org/10.1101/gad.1528707.
3. Alon U. An introduction to systems biology: design principles of biological circuits. 2nd ed. London: CRC Press; 2020.
4. Ritchie MD, Holzinger ER, Li R, Pendergrass SA, Kim D. Methods of integrating data to uncover genotype–phenotype interactions. Nat Rev Genet. 2015;16(2):85–97.
5. Hasin Y, Seldin M, Lusis A. Multi-omics approaches to disease. Genome Biol. 2017;18(1):1–15.
6. Nguyen H, Shrestha S, Tran D, Shafi A, Draghici S, Nguyen T. A comprehensive survey of tools and software for active subnetwork identification. Front Genet. 2019;10:155.
7. Cowen L, Ideker T, Raphael BJ, Sharan R. Network propagation: a universal amplifier of genetic associations. Nat Rev Genet. 2017;18(9):551.
8. Ronen J, Akalin A. netSmooth: network-smoothing based imputation for single cell RNA-seq. F1000Research. 2018;7:8.
9. Hamilton WL, Ying R, Leskovec J. Representation learning on graphs: methods and applications. CoRR arXiv:1709.05584 (2017).
10. Wu Z, Pan S, Chen F, Long G, Zhang C, Yu PS. A comprehensive survey on graph neural networks. CoRR arXiv:1901.00596 (2019).
11. Chami I, Abu-El-Haija S, Perozzi B, Ré C, Murphy K. Machine learning on graphs: a model and comprehensive taxonomy. arXiv preprint arXiv:2005.03675 (2020).
12. Kipf TN, Welling M. Variational graph auto-encoders. arXiv:1611.07308 (2016).
13. Hammond DK, Vandergheynst P, Gribonval R. Wavelets on graphs via spectral graph theory. Appl Comput Harmon Anal. 2011;30(2):129–50. https://doi.org/10.1016/j.acha.2010.04.005.
14. Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. CoRR arXiv:1609.02907 (2016).
15. Gilmer J, Schoenholz SS, Riley PF, Vinyals O, Dahl GE. Neural message passing for quantum chemistry. CoRR arXiv:1704.01212 (2017).
16. Hamilton WL, Ying R, Leskovec J. Inductive representation learning on large graphs. CoRR arXiv:1706.02216 (2017).
17. Morris C, Ritzert M, Fey M, Hamilton WL, Lenssen JE, Rattan G, Grohe M. Weisfeiler and leman go neural: higher-order graph neural networks. CoRR arXiv:1810.02244 (2018).
18. van Dijk D, Sharma R, Nainys J, Yim K, Kathail P, Carr AJ, Burdziak C, Moon KR, Chaffer CL, Pattabiraman D, Bierie B, Mazutis L, Wolf G, Krishnaswamy S, Pe'er D. Recovering gene interactions from single-cell data using data diffusion. Cell. 2018;174(3):716–72927. https://doi.org/10.1016/j.cell.2018.05.061.
19. Fey M, Lenssen JE. Fast graph representation learning with PyTorch Geometric. In: ICLR workshop on representation learning on graphs and manifolds; 2019.
20. Erdös P, Rényi A. On random graphs I. Publ Math Debr. 1959;6:290.
21. RegulonDB: RegulonDB network interactions–gene interaction. Data retrieved from RegulonDB Downloadable Experimental Datasets. http://regulondb.ccg.unam.mx/menu/download/datasets/index.jsp; 2019.
22. BioGRID: BioGRID PPI interaction network Ecoli. Data retrieved from BioGRID download page available at https://downloads.thebiogrid.org/BioGRID/Release-Archive/BIOGRID-3.5.180/; 2019.
23. Oughtred R, Stark C, Breitkreutz B-J, Rust J, Boucher L, Chang C, Kolas N, O'Donnell L, Leung G, McAdam R, Zhang F, Dolma S, Willems A, Coulombe-Huntington J, Chatr-Aryamontri A, Dolinski K, Tyers M. The BioGRID interaction database: 2019 update. Nucleic Acids Res. 2018;47(D1):529–41. https://doi.org/10.1093/nar/gky1079.
24. Faith JJ, Driscoll ME, Fusaro VA, Cosgrove EJ, Hayete B, Juhn FS, Schneider SJ, Gardner TS. Many microbe microarrays database; 2007. Data retrieved from M3DB download page available at http://m3d.mssm.edu/norm/.
25. Faith JJ, Driscoll ME, Fusaro VA, Cosgrove EJ, Hayete B, Juhn FS, Schneider SJ, Gardner TS. Many microbe microarrays database: uniformly normalized affymetrix compendia with structured experimental metadata. Nucleic Acids Res. 2008;36(Database-Issue):866–70.
26. Cheng S, Pei Y, He L, Peng G, Reinius B, Tam PPL, Jing N, Deng Q. Single-cell RNA-seq reveals cellular heterogeneity of pluripotency transition and x chromosome dynamics during early mouse development. Cell Rep. 2019;26(10):2593–26073. https://doi.org/10.1016/j.celrep.2019.02.031.
27. Subramanian A, Narayan R, Corsello SM, Peck DD, Natoli TE, Lu X, Gould J, Davis JF, Tubelli AA, Asiedu JK, et al. A next generation connectivity map: L1000 platform and the first 1,000,000 profiles. Cell. 2017;171(6):1437–52.
28. Suhre K, McCarthy MI, Schwenk JM. Genetics meets proteomics: perspectives for large population-based studies. Nat Rev Genet. 2020;22:1–19.

Publisher's Note

Appendix

A.1 Variance explained on features

For an embedding matrix $\mathbf{Z} \in \mathbb{R}^{N \times H}$ and feature matrix $\mathbf{X} \in \mathbb{R}^{N \times M}$, we calculated the amount of variance in $\mathbf{X}$ explained by $\mathbf{Z}$ as

$$\mathbb{V}_{\mathbf{Z}} = \frac{\mathrm{tr}\big(\mathbf{P}_{\mathbf{Z}}\mathbf{X}^T\mathbf{X}\big)}{\mathrm{tr}(\mathbf{X}^T\mathbf{X})} \tag{A1}$$

where $\mathbf{P}_{\mathbf{Z}}$ is the projection matrix onto the subspace of $\mathbb{R}^N$ spanned by the columns of $\mathbf{Z}$,

$$\mathbf{P}_{\mathbf{Z}} = \mathbf{Z}(\mathbf{Z}^T\mathbf{Z})^{-1}\mathbf{Z}^T.$$

Note that if we write the eigendecomposition of $\mathbf{X}^T\mathbf{X}$ as $\mathbf{X}^T\mathbf{X} = \mathbf{V}^T\mathbf{\Delta}\mathbf{V}$, then the columns of $\mathbf{V}$ corresponding to the nonzero eigenvalues in $\mathbf{\Delta}$ are the principal components of $\mathbf{X}$. If $\mathbf{Z}$ consist of the i^{th} principal component, then eq. (A1) reduces to the familiar variance explained by this component, $\Delta_i/(\sum_j \Delta_j)$. If $\mathbf{Z}$ consists of a single vector $z \in \mathbb{R}^N$ with unit length, $\|z\| = 1$, then eq. (A1) reduces to

$$\mathbb{V}_z = \sum_{i=1}^{N} \frac{\Delta_i}{\sum_j \Delta_j}(z^T u_i)^2,$$

a weighted sum of the variances explained by each principal component, with weights determined by the extent of overlap between z and each principal component. Eq. (A1) generalizes this to summing the variances explained by multiple vectors simultaneously that need not be mutually orthogonal.

A.2 Random Graph Generation using Erdő s–Rényi model

According to Erdő s–Rényi, a random graph $G(n, p)$ has $\binom{n}{2}p$ edges placed at random. The degree distribution of each node is calculated through:

$$P(\deg(v) = k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}, \tag{A2}$$

which is the binomial distribution. We adjusted the value of p so that the random graphs would approximately have the same number of edges as the real networks.

A.3 Results of gene regulatory networks reconstruction and gene expression variance explained

Input	AUC				AP			
		Ecoli		Mus Musculus		Ecoli		Mus Musculus
	TF_net	PPI	Genetics	PPI	TF_net	PPI	Genetics	PPI
Expression + Graph	0.868±0.017	0.8502±0.002	0.894±0.01	0.803±0.000	0.918±0.007	0.872±0.003	0.904±0.006	0.762±0.000
Graph	0.574±0.020	0.8316±0.018	0.882±0.014	0.6674±0.0000	0.727±0.012	0.860±0.010	0.904±0.006	0.653±0.000
Expression + Random Graph	0.529±0.01	0.4958±0.002	0.485±0.005	0.5108±0.0000	0.525±0.011	0.488±0.001	0.489±0.004	0.506±0.000
Random Graph	0.492±0.006	0.4892±0.006	0.490±0.007	0.5025±0.0000	0.496±0.005	0.486±0.005	0.5002±0.003	0.505±0.000
Expression	0.579±0.02	0.580±0.018	0.557±0.02	0.610±0.000	0.611±0.002	0.624±0.018	0.590±0.02	0.627±0.002

Table A1: Area under the ROC curve (AUC) and average precision (AP) for reconstructing biological network in E. coli and mouse from five graph structure embedding approaches

Table A2: The average Variance Explained on Gene Expression from the Regulatory Networks graph embedding

Input	Ecoli		
	TF_net	PPI	Genetics
Expression + Graph	0.883±0.005	0.805±0.003	0.7824±0.006
Graph	0.716±0.010	0.752±0.008	0.6697±0.010
Expression + Random Graph	0.660±0.010	0.667±0.013	0.6618±0.014
Random Graph	0.647±0.017	0.652±0.018	0.6483±0.010

Paper II

Integration of graph neural networks and genome-scale metabolic models for predicting gene essentiality

Ramin Hasibi, Tom Michoel, and Diego A. Oyarzun,
biorxive, (2023)

II

Integration of graph neural networks and genome-scale metabolic models for predicting gene essentiality

Ramin Hasibi and Tom Michoel

Computational Biology Unit, Department of Informatics, University of Bergen, Norway

Diego A. Oyarzún

School of Biological Sciences, University of Edinburgh, UK

School of Informatics, University of Edinburgh, UK

The Alan Turing Institute, London, UK and

Corresponding author: d.oyarzun@ed.ac.uk

Abstract

Genome-scale metabolic models are powerful tools for understanding cellular physiology. Flux balance analysis (FBA), in particular, is an optimization-based approach widely employed for predicting metabolic phenotypes. In model microbes such as *Escherichia coli*, FBA has been successful at predicting essential genes, i.e. those genes that impair survival when deleted. A central assumption in this approach, however, is that both wild type and deletion strains optimize the same fitness objective. The optimality assumption may hold for the wild type metabolic network, but deletion strains are not subject to the same evolutionary pressures and knock-out mutants may steer their metabolism to meet other objectives for survival. Here, we present FlowGAT, a hybrid FBA-machine learning strategy for predicting essentiality directly from wild type metabolic phenotypes. The approach is based on graph-structured representation of metabolic fluxes predicted by FBA, where nodes correspond to enzymatic reactions and edges quantify the propagation of metabolite mass flow between a reaction and its neighbours. We integrate this information into a graph neural network that can be trained on knock-out fitness assay data. Comparisons across different model architectures reveal that FlowGAT predictions for *E. coli* are close to those of FBA for several growth conditions. This suggests that gene essentiality can be accurately predicted by exploiting the network structure of metabolism, without additional assumptions beyond optimality of the wild type. Our approach demonstrates the benefits of combining the mechanistic insights afforded by genome-scale models with the ability of deep learning models to extract patterns from complex data.

I. INTRODUCTION

The identification of essential genes is crucial for understanding the minimal functional modules required for survival of organisms [48]. Gene essentiality has key applications in biomedicine and biotechnology, for example, to identify therapeutic targets in complex diseases [6], find strategies to combat pathogens [16], or optimize chemical production in genetically engineered microbes [10]. Identification of essential genes requires screening assays where multiple knock-out mutants are phenotyped with a suitable fitness selection strategy. Such screens have been performed on many organisms, including model microbes such as *Escherichia coli* [2, 31, 37], *Saccharomyces cerevisiae* [45] and *Bacillus subtilis* [26], as well as pathogens such as *Candida albicans* [38] and *Aspergillus fumigatus* [23]. In human cells, recent work has produced high-resolution deletion assays [6], leveraging progress in high-throughput technologies such as RNA interference and CRISPR-based screens [48] to produce detailed maps of gene essentiality in different conditions.

As a result of the cost and complexity of knock-out fitness assays, there is a growing interest in computational methods that can complement the experimental work with *in silico* prediction of fitness effects. These computational approaches often employ machine learning techniques combined with information from protein sequence, gene homologies, gene-function ontologies, and protein interaction networks [1, 8, 29, 30, 49]. In the case of metabolic genes, i.e. those that code for catalytic enzymes in metabolic pathways, Flux Balance Analysis (FBA) is a widely employed method for predicting essentiality [34]. There are numerous variants of FBA and its related algorithms [28], but at its core FBA computes genome-scale flux distributions that optimize a cellular fitness objective. Such objective is typically taken to be the cellular growth rate modeled as a linear combination of synthesis rates of amino acids, lipids and other biomass components. By imposing constraints on each metabolic flux, FBA problems can be solved with efficient linear programming algorithms, which allows to rapidly simulate the impact of gene deletions on the predicted growth rate and draw predictions on the essentiality of metabolic genes.

Flux Balance Analysis has shown good prediction accuracy for gene essentiality in the *E. coli* bacterium [31] and other model microbes, but predictions for eukaryotes and higher-order organisms have produced mixed results [18, 22]. The quality of FBA predictions have also been shown to vary strongly across published models as well as the performance

metrics employed to quantify prediction accuracy [4]. An often overlooked limitation of the FBA approach is the tacit assumption that the metabolism of deletion strains optimizes the same objective as the wild type. In many cases, deletion strains display suboptimal growth phenotypes [44] and they are not subject to the same long-term evolutionary pressures as the wild type. It has also been postulated that deletions of metabolic genes can alter cell physiology to meet other objectives for survival; for example, an early work hypothesized that knockout strains may minimize their phenotypic deviation from the wild type [44], while various works have explored the impact of alternative objective functions [17, 42] and multiobjective optimization principles [43] in the classic FBA formulation.

Here, we sought to determine if gene essentiality can be predicted directly from wild type metabolic phenotypes. We developed a hybrid algorithm to predict gene essentiality using a combination of FBA and deep graph neural networks trained on knock-out fitness data. This approach does not require the assumption of optimality of deletion strains and takes maximal advantage of the inherent graph structure of cellular metabolism. Early attempts to augment the predictive power of FBA with machine learning explored the use of flux features for improved prediction of gene essentiality [33, 36], and other works have attempted to predict essentiality from the metabolic graph topology [15]. Most recently, several authors have developed integrated pipelines aimed at improving FBA predictions for biomedical [27, 35] and biotechnology tasks [13, 41].

In our approach, starting from wild type FBA solutions we first represent genome-scale flux distributions as a weighted digraph in a space of reaction nodes, and employ a flow-based representation for each node based on the redistribution of chemical mass flows between various paths in the graph. To integrate the graph structure and node features into a single predictive model, we employ a Graph Neural Network (GNN) with an attention mechanism [46] termed FlowGAT. We show that FlowGAT can be trained on a small amount of labelled data from knock-out screens. We demonstrate the effectiveness of this approach using the latest metabolic model of *E. coli*, and show that prediction accuracy near the FBA gold standard. Moreover, model predictions appear to generalize well across various growth conditions without the need for further training data. The results highlight the advantages of integrating FBA pipelines with state-of-the-art machine learning algorithms for improved phenotypic predictions.

II. RESULTS

A. Model architecture and training

In this paper, we propose FlowGAT, a GNN based model to predict gene essentiality from graphs generated from FBA solutions. As shown in Figure 1A, each node in the graph corresponds to a metabolic reaction, and we pair each node with a set of flow-based features and binary essentiality labels obtained from knock-out fitness assays. The graph structure and node features are integrated into a GNN for binary classification, so as to use a message passing scheme to propagate node features through the structure of the graph; this allows learning a rich embedding of the input that contains information from the k-hop neighbourhood of each node [19]. We next detail the different components of the model and our training strategy.

a. Graph construction We consider metabolic networks with m metabolites and n enzymatic reactions described by the following differential equation model

$$\frac{\mathrm{d}X}{\mathrm{d}t} = \mathbf{S}v, \tag{1}$$

where X is an m-dimensional vector of metabolite concentrations, v is a n-dimensional vector of reaction fluxes, and $\mathbf{S}$ is a $n \times m$ stoichiometric matrix. In steady state, the relation $\mathbf{S}v = 0$ describes all flux vectors that can sustain a specific metabolic state. A common strategy to estimate v at the genome-scale is to employ FBA to compute a flux vector $v^\star$ that optimizes a meaningful biological objective; details on FBA can be found in the Methods section. To convert such FBA solution vectors $v^\star$ into a graph, we used the Mass Flow Graph (MFG) construction proposed by Beguerisse-Diaz *et al* [3] and illustrated in Figure 1B. Starting from the stoichometric matrix $\mathbf{S}$, we first build a directed graph with reactions as nodes, where two nodes are connected if and only if the source reaction produces a metabolite that is consumed by the target reaction. Each edge in the graph has a weight $w_{i,j}$ that represents the normalized mass flow from node i to node j. We first compute the flow of metabolite X_k from reaction i to j according to:

$$\mathrm{Flow}_{i\to j}(X_k) = \mathrm{Flow}^+_{R_i}(X_k) \times \frac{\mathrm{Flow}^-_{R_j}(X_k)}{\sum_{\ell \in C_k} \mathrm{Flow}^-_{R_\ell}(X_k)}, \tag{2}$$

where $\mathrm{Flow}^+_{R_i}(X_k)$ and $\mathrm{Flow}^-_{R_j}(X_k)$ are the production and consumption flows of metabolite X_k by reaction R_i, respectively. The set C_k contains the indices of all reactions that consume

metabolite X_k. The edge weight $w_{i,j}$ is thus defined as the total mass flow between two nodes, aggregated over all metabolites X_k that are produced by node i and consumed by node j:

$$w_{i,j} = \sum_{k=1}^{p} \mathrm{Flow}_{i \to j}(X_k). \tag{3}$$

Mass flow graphs allow converting FBA solutions into a directed graph, and thus can be used to represent the network structure of metabolism in different growth conditions or genetic perturbations. In Figure 1C, we show MFGs built from genome-scale metabolic models for three model microbes available in the BiGG model database [24] (*Escherichia coli*, *Saccharomyces cerevisiae* and *Bacillus subtilis*). Further details on the construction of the mass flow graphs can be found in the Methods section.

b. Design of node features Besides the graph topology, we ascribe a feature vector to each reaction node that can be exploited for improved performance by the representation learning approach. This approach is analogous to the structural and positional encoding schemes employed in graph Transformer architectures to feed models with extra information about the local connectivity of nodes [32]. Since the edge weights in (3) relate to the mass flow between reactions, we opted to employ flow-based features that can aggregate information on incoming and outgoing mass flows from each node. To this end, we employ the Flow Profile Encoding (FPE) first defined by Cooper and Barahona for general directed graphs [9]. Given a directed MFG with weighted adjacency matrix $\mathbf{A}$, for each node i we define the inflow profile of length k as

$$\mathrm{inflow}_i^k = [\mathbf{A}^k \mathbf{1}^{n\times 1}]_i, \tag{4}$$

where $\mathbf{A}^k$ is the matrix k-th power, $\mathbf{1}^{n\times 1}$ is an m-dimensional vector of ones, and $[\,\cdot\,]_i$ is the i-th element of a vector. The inflow of node i is thus defined as the weight sum across all incoming paths of length k. We similarly define the outflow of node i as:

$$\mathrm{outflow}_i^k = [(\mathbf{A}')^k \mathbf{1}^{n\times 1}]_i, \tag{5}$$

where $\mathbf{A}'$ is the matrix transpose. We concatenate inflows and outflows up to maximal length k_m for each node:

$$\mathrm{FPE}_i = \left[\beta^1 \times \mathrm{inflow}_i^1, \cdots, \beta^{k_\mathrm{m}} \times \mathrm{inflow}_i^{k_\mathrm{m}}, \beta^1 \times \mathrm{outflow}_i^1, \cdots, \beta^{k_\mathrm{m}} \times \mathrm{outflow}_i^{k_\mathrm{m}}\right], \tag{6}$$

where k_{m} is a hyperparameter that defines the maximum path length, $\beta = \alpha/\lambda_1$ is a scaling factor, λ_1 is the largest eigenvalue of the adjacency matrix $\mathbf{A}$, and α is a hyperparameter that controls for the variable weights of the short and long paths; normalization by the largest eigenvalue λ_1 ensures convergence for large k, in the sense that $\lim_{k\to\infty} ||\mathbf{A}^{k+1}||/||\mathbf{A}^{k}|| = \lambda_1$. The definition in (6) allows computing a feature vector of length $2k_{\mathrm{m}}$ for each node in the graph.

c. Representation learning Graph representation learning is concerned with mapping the nodes into a low dimensional vector which is optimized for downstream tasks such as classification or regression [21]. GNNs are a family of deep learning methods on graphs which obtain the embedding vector by incorporating the features of the node and its local neighbourhood according to a customized message passing scheme called Message Passing Neural Network (MPNN) [19]. Doing so helps the model capture local and global structural information about the graph and results in a more expressive embedding space. For more information about MPNNs refer to the Methods section. In this study, we employ a MPNN architecture named Graph Attention (GAT) to compute the neighbourhood information importance in finding the representation of the node [46]. Each layer l of GAT updates the representation of node i according to:

$$h_i^l = \sum_{j\in\mathcal{N}(i)\cup i} a_{ji}\Theta^l h_j^{l-1}, \tag{7}$$

where h_i^l is the representation vector, $\mathcal{N}(i)$ is the set of neighbouring nodes for node i, Θ is a set of differentiable weights, and a_{ij} is an attention coefficient that is dynamically calculated for each node $j \in \mathcal{N}(i) \cup i$ as

$$a_{ji} = \frac{\exp(\phi(h_i^{l-1}, h_j^{l-1}))}{\sum_{v\in\mathcal{N}(i)\cup i}\exp(\phi(h_i^{l-1}, h_v^{l-1}))}, \tag{8}$$

where ϕ is a differentiable function optimized through gradient descent optimization algorithms [39]. Details on the message passing and attention schemes can be found in the Methods section.

d. Data pre-processing FlowGAT can be trained on knock-out growth assay data, where each gene is labelled as non-essential (0) or essential (1) depending on whether a fitness score is above or below prescribed threshold. For model training, the binary gene labels must be converted into their corresponding reaction node labels in the MFG. To

this end, we use the Gene-Protein-Reaction (GPR) map included in genome-scale metabolic models. The GPR is a Boolean function that specifies which gene codes for which proteins, and conversely how each protein affects a metabolic reaction. The GPR can account for reactions that are catalyzed by multiple enzymes or by enzymatic complexes encoded in multiple genes. For those genes that map one-to-one into a single reaction, we transferred the gene label directly into a reaction label. For those genes that map into multiple reactions (many-to-one), we transferred the gene label to all reactions deactivated by the gene deletion. When multiple genes map to multiple reactions (many-to-many), the structure of the GPR does not allow to infer reaction labels from gene labels, and therefore we considered such reactions as unlabelled. Note that the data also contains nodes that lack essentiality labels because their corresponding genes have not been measured in the growth assay. The unlabeled nodes are made available for model training to make sure that the graph representation learning can take advantage of the full graph structure without limiting the representation power of the GAT; the classification loss for training and evaluation of the model is only calculated on the labeled nodes. We also note that the reaction labels are typically imbalanced because the MFG is enriched for essential reaction nodes. By definition in (3), those reactions with zero flux in the wild type FBA solution will have nil edge weights and thus are disconnected from the graph. During training, the model has access to the features of all nodes (labeled and unlabeled) through the message passing, but the training loss is calculated on the labels of the training nodes in semi-supervised fashion (Figure 1D). Details on model training can be found in the Methods section.

B. Performance evaluation of FlowGAT

To evaluate the performance of FlowGAT, we employed the growth knock-out data for the *Escherichia coli* bacterium reported by Monk and colleagues [31], and the iML1515 genome-scale model reported in the same work. We chose the *E. coli* model because it is the most complete and best curated metabolic reconstruction in the literature, and thus allows us to mitigate the impact of misclassification errors caused by poor model quality and focus on the predictive power of FlowGAT itself. The dataset contains growth rate data for 3,892 *E. coli* genes grown in various carbon sources.

We built the MFG for *E. coli* using the wild type FBA solution using glucose as the sole

carbon source and the default objective function included in the iML1515 model (growth rate). The resulting MFG has 444 nodes and after converting the gene labels to reaction labels with the GPR map we obtained 255 labeled nodes (191 essential, 64 nonessential). We first compared FlowGAT trained on binary cross-entropy loss with classical binary classifiers including Support Vector Classifier (SVC), Multi Layer Preceptron (MLP) classifier, and random forests (RF) classifier using the flow profile embeddings in (6) as feature vectors; details on model training and hyperparameter selection can be found in Methods. The results in Figure 2A show precision-recall curves, averaged across $N = 50$ rounds of training and testing (5 test folds with 20% of nodes resampled 10 times for model retraining); details on our strategy for model evaluation can be found in the Methods. Among the considered classifiers, FlowGAT achieves the best Area Under the Precision-Recall Curve (PRAUC) across all test folds and performs above the no-skill classifier while the classic models significantly underperform; we note that due to the class imbalance the baseline precision of the no-skill classifier is 74.9%.

We also compared FlowGAT trained on two other popular node embedding techniques (Local Degree Profile (LDP) [7] and Random Walk Embedding (RWE) [11]) that have shown good performance in a number of tasks on molecular graphs, as well as two other message passing schemes (Graph Convolution Network [25] and Graph SAGE, see Supplementary Figure S1). Details on these additional node embeddings and message passing strategies can be found in the Methods section. The results (Figure 2A) show that graph attention delivers the best performance, and models trained on LDP and RWE node features are outperformed by the flow profile encodings, possibly because the former do not take account for directionality and weight of the edges of the MFG.

We further investigated the sensitivity of FlowGAT to the random seed employed for weight initialization; the distributions in Figure 2B show the PRAUC scores for all models across the 50 runs. The results suggest that FlowGAT performance is relatively robust; only the RF classifier delivers tighter predictions, but at the cost of an average performance below the no-skill baseline.

We finally sought to explore an alternative training scheme using a regression approach. Since the gene essentiality labels are based on a binarization of continuous measurements of growth rate, we reasoned that recasting the prediction problem as a regression task could improve performance. To this end, we employed the non-binarized fitness measurements of

growth rate in Monk et al [31] and re-trained FlowGAT as a regressor using Mean Squared Error (MSE) loss on predicting the non-binary growth rate values; all model hyperparameters were left unchanged. Following the same evaluation scheme as the above, we used the FlowGAT regressor to predict growth rates for the reaction nodes in each test fold. We then used the predicted growth rate as classification scores and computed the precision-recall curve on the test fold. Upon comparison with the classification approach in Figure 2A–B, the regression results in Figure 2C led to a performance increase in terms of average PRAUC, as well as tighter predictions that are less sensitive to weight initialization.

After finding the best setting for FlowGAT in terms of architectural design choices, we fixed the cut-off threshold for the output prediction of FlowGAT trained as a regressor in Figure 2C to produce binary essentiality predictions for all 50 evaluations by classifying the nodes that score above the threshold as essential and others as non-essential. We measure the performance of FlowGAT in terms of three metrics of Precision, Recall, and F1 for the binary predictions (Figure 2D). The predictions of the FlowGAT regressor manage to keep both precision and recall above 75% and 90%, respectively.

To better understand the performance of our model, we compared the output of FlowGAT trained as a regressor (Figure 2C) with those from FBA applied to the genes that appear in the MFG. First, we mapped each reaction node back to its corresponding into gene labels using the GPR map. In total, 240 labeled genes appear in the constructed MFG (180 essentials and 60 non-essential. This number is lower than the number of reaction nodes (255) because some reactions correspond to the same gene based on many-to-one mapping that was used to assign labels to nodes earlier; for such genes, we aggregated reactions by maximum prediction value of corresponding reactions. We collected predictions across all genes and compared these results with the essentiality prediction of FBA for each gene in Figure 2E. The results suggest that both FlowGAT and FBA find most of the essential genes, but FlowGAT finds on average 19 essential genes that are misclassified by FBA. In the case of non-essential genes, however, we found that FlowGAT underperforms and misses more genes than FBA, likely as a result of non-essential genes being the minority class.

C. Essentiality prediction in different growth conditions

The essentiality of metabolic genes can be highly dependent on environmental conditions. Difference carbon sources can produce important differences on the metabolic phenotype and, as a result, some genes that are essential in one condition source may be non-essential in another one.

To test the predictive power of FlowGAT in other growth conditions beyond glucose, we trained the model using *E. coli* knock-out fitness data in ten other carbon sources that cover different entry points into central carbon metabolism [31]. We built the corresponding mass flow graphs from wild type FBA solutions of the iML1515 model instanced to each carbon source. To build condition-dependent graphs, we constrained the flux of each nutrient exchange reaction to a fixed value (Supplementary Table S2). This resulted in 10 different MFGs that differ on their nodes and their edge weights. Inspection of the reaction nodes per graph (Figure 3A) reveals differences across graphs for reactions that become active for specific carbon sources, as well as a large number of reactions that are shared across conditions. We then evaluated the performance of FlowGAT trained in each of the ten mass flow graphs, using the growth knock-out fitness data and the regression strategy of Figure 2C; model hyperparameters were left unchanged. In each graph, the number of essential and non-essential nodes varies and thus, the no-skill baseline varies depending on the class imbalance in that graph. As seen in Figure 3B, we found that while the PRAUC scores vary across growth conditions, in all cases FlowGAT outperformed the no-skill classifier by at least 6%. These encouraging results can likely still be improved by introducing condition-specific hyperparameters for the FlowGAT architecture.

We finally aimed to determine the ability of FlowGAT to generalize predictions across growth conditions. We conducted a cross-training evaluation, where the model was trained on a mass flow graph and fitness data from a single carbon source, and tested on the reaction nodes in a different growth condition. To this end, we also included the FlowGAT model for glucose discussed in the previous section. As shown in Figure 3C, all 90 cross-tests show an improvement in PRAUC with respect to each MFG no-skill classifier. Although each FlowGAT model was trained on a different graph and fitness data, these results suggest that the model captures a well-performing representation of the data. To test if this is a result of the similarity between the nodes present in each graph (Figure 3A), we quantified the

graph-to-graph similarity using the distance between the distribution of node features. We estimated the probability density function of the flow profile encodings for each graph using kernel density estimation and computed the Jensen Shannon divergence between all pairs of distributions. The results (Figure 3C) do not show a correlation between graph similarity and the PRAUC scores. For example, the maltose graph embedding is nearly equidistant from both the acetate and galactose graphs, but FlowGAT trained on maltose has a performance of approximately 5% better when tested on galactose than in acetate. Likewise, FlowGAT trained on mannitol performs better when tested in galactose than glycerol, despite the galactose graph being more dissimilar to the mannitol graph. These observations suggest that the generalization performance of FlowGAT results from its representation power rather than the similarity between the input graphs.

III. DISCUSSION

Gene essentiality refers to the concept that some genes are indispensable for the survival of an organism. These genes often encode proteins that play critical roles in fundamental cellular processes needed for growth. Since the quantification of gene essentiality requires knock-out fitness assays across a large number of genes and growth conditions, there is substantial interest in computational methods that can aid the identification of genes from a reduced number of measurements. In this paper, we presented FlowGAT, a graph neural network that can be trained on knock-out fitness data to predict the essentiality of metabolic genes. The architecture exploits the inherent graph structure of metabolic fluxes predicted by Flux Balance Analysis through a combination of mass flow graphs and node features that describe local connectivity.

Using data from *E. coli* and its latest genome-scale metabolic model, we show that FlowGAT can identify most of the genes that are correctly called as essential by Flux Balance Analysis, and even correct some of its misclassified essential genes. Our approach is based solely on the wild-type phenotype predicted by FBA; since it does not require the assumption of optimality in the deletion strains, FlowGAT may provide benefits when applied to organisms where the growth optimality assumption is not warranted. Additionally, FlowGAT displays encouraging generalization power across growth conditions, even in cases where the underlying graphs and node features differ substantially. This observation suggests that the

proposed architecture and feature extraction method can learn internal representations that are useful predictors of gene essentiality. We also found, however, that FlowGAT struggled to predict non-essential genes and can be substantially outperformed by traditional FBA. This phenomenon could arise from various sources, such as the data imbalance that in our case favors essential labels, or because predicting non-essential genes is intrinsically more challenging than essential ones [4]. Our approach illustrates the potential of exploring new ways of combining traditional tools such as Flux Balance Analysis with modern data-driven approaches, and adds to the growing body of literature at the interface of genome-scale metabolic modeling with machine learning [40, 47].

IV. METHODS

A. Flux Balance Analysis

Flux Balance Analysis (FBA) is one of the popular methods for the analysis of cellular metabolism. In a steady state, a metabolic network can be described by

$$\mathbf{S}v = 0. \tag{9}$$

The aim of FBA is to obtain the solution vector v^* that satisfies the above condition and at the same time solves the following optimization problem:

$$\begin{aligned} v^* = \arg\max_{v} \quad & c'v \\ \text{subject to} \quad & \begin{cases} \mathbf{S}v = 0, \\ v_{\text{lb}} < v < v_{\text{ub}}, \end{cases} \end{aligned} \tag{10}$$

in which, c is a vector of flux weights, and $(v_{\text{lb}}, v_{\text{ub}})$ are lower and upper bounds on reaction fluxes, respectively.

B. Mass Flow Graphs

Originally introduced in [3], MFGs are designed to reflect the directional flow of metabolites produced or consumed through enzymatic reactions. In these graphs, reactions are considered as vertices, and two reactions are connected through a directed edge if they share

a metabolite (either as reactants or products). The construction pipeline of these graphs can incorporate different experimental conditions through varying flux distributions. For instance, one key advantage of such a pipeline is the automatic pruning of pool metabolites through mapping onto weak connections between graph nodes, therefore reducing their impact on the overall network structure.

To construct an MFG from a metabolite network consisting of m reactions and n metabolites, first, we obtain the solution vector v^* from FBA. Then, we unfold the v^* into two-fold forward and reverse reaction fluxes through

$$v^*_{2m} = \frac{1}{2} \begin{bmatrix} \text{abs}(v^*) + v^* \\ \text{abs}(v^*) - v^* \end{bmatrix}. \tag{11}$$

Next, the corresponding stoichiometric matrix of v^*_{2m} is defined as

$$\mathbf{S}_{2m} = \begin{bmatrix} \mathbf{S} & -\mathbf{S} \end{bmatrix} \begin{bmatrix} \mathbf{I}_m & 0 \\ 0 & \text{diag}(r) \end{bmatrix}, \tag{12}$$

in which, S in the $n \times m$ stoichiometric matrix corresponds to n reactions and m metabolites of the original network, and r is an m dimensional Boolean vector indicating whether a reaction is reversible or not. Finally, the adjacency matrix of the MFG can be calculated as

$$A(v^*) = (\mathbf{S}^+_{2m}\mathbf{S}^*)' \mathbf{J}^\dagger_v (\mathbf{S}^-_{2m}\mathbf{V}^*), \tag{13}$$

where $\dagger$ is the matrix pseudoinverse operator, and $\mathbf{V}^* = \text{diag}(v^*_{wm})$, $\mathbf{J} = \text{diag}(\mathbf{S}^+_{2m} v^*_{2m})$ with

$$\mathbf{S}^+_{2m} = \frac{1}{2}(\text{abs}(\mathbf{S}_{2m}) + \mathbf{S}_{2m}), \tag{14}$$

$$\mathbf{S}^-_{2m} = \frac{1}{2}(\text{abs}(\mathbf{S}_{2m}) - \mathbf{S}_{2m}). \tag{15}$$

C. Node feature generation

Mass Flow Graphs are none-attributed, i.e., no specific node features are provided for reactions (nodes) in the graph. As a result, it is necessary to design a feature generation pipeline that considers the structure of the graph as well as the edge weights that appear in the adjacency of the graph. For this task, we propose a node encoding algorithm analogous to positional encoding of Transformer architectures.

Apart from the proposed FPE features in Eq. (6), a second approach to node encoding is to gather local neighborhood structural statistics based on the degree of each node and its neighboring nodes [7]. In this approach, the local degree profile of each node is defined as

$$\mathrm{LDP}_i = [\deg(i), \min(\mathrm{DN}(i)), \max(\mathrm{DN}(i)), \mathrm{avg}(\mathrm{DN}(i)), \mathrm{std}(\mathrm{DN}(i))], \tag{16}$$

in which $\mathrm{DN}(i)$ is the set of out-degree values for all the neighboring nodes of node i. For this set the minimum, maximum, average, and standard deviation are calculated and used as node features of node i. Additionally, a third encoding method relies on random walks from each node. In this method, a random walk encoding for each node i is calculated as:

$$\mathrm{RWE}_i = [RW_{ii}^2, RW_{ii}^3, \ldots, RW_{ii}^{K_{\max}}], \tag{17}$$

where $K_{\max}$ is a hyperparameter for maximum length of the random walks, and $RW = \mathbf{A}\mathbf{D}^{-1}$ is the random walk operator and only the random walks that end in node i are considered for encoding (RW_{ii} is this the i-th element of the diagonal).

D. Message-passing neural networks (MPNN)

For representation learning of the graph features we employed GAT architecture [46] which is an instance of a MPNN scheme. In a typical graph representation learning task, the representation of each node is updated through a message passing scheme in which the information from neighboring nodes is gathered using message formula and aggregated with the features of the node itself. Thus, a message passing formula for each message from node j to node i can be written as

$$m_{ji}^{(l)} = \mathrm{MSG}^{(l)}\left(h_{j\in\{\mathcal{N}(i)\cup i\}}^{(l-1)}, e_{j,i}\right) \tag{18}$$

, where $h_i^{(l)}$ is the representation vector of node i in layer (l) of the MPNN, $e_{j,i}$ are the features of the edge between node i and node j, and $\mathcal{N}(i)$ is the set of neighbouring nodes to node i. The operator MSG^l is the custom message function which is different in each layer design. One typical example of such a function is an MLP applied on the input values. Moreover, the messages for each node are aggregated to obtain the representation of node i in layer l using

$$h_i^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{m_{ji}^{(l)}, u \in \mathcal{N}(i)\right\}, h_v^{(l-1)}\right), \tag{19}$$

in which, AGG is a custom permutation invariant operator with regards to messages for each node.

GAT formulates the message equation in (18) as the multiplication of the attention as the learnable importance factor of each message by the representation of neighbors. Thus, the formula in (18) becomes:

$$m_{ji}^{(l)} = a_{ji}\Theta^l h_j^{l-1}, \tag{20}$$

in which, a_{ji} is the attention coefficient and is usually calculated through feeding the features of both neighboring nodes i and j through a learnable function and calculating the importance through the softmax function. Other popular examples of MPNN framework are Graph Convolution Network (GCN) [25] and GraphSAGE [20] which change the message function and use different aggregation functions. In GCN, the message function in (18) is calculated as:

$$m_{ji}^{l} = \frac{\Theta^l h_j^{l-1}}{\sqrt{\deg(i)}\sqrt{\deg(j)}}, \tag{21}$$

with the sum pooling operator as the aggregator function. In GraphSAGE, the message function MSG is the identity function and the aggregation function AGG is calculated as:

$$h_i^l = \Theta^l h_i^{l-1} + \Theta'^l \mathrm{MEAN}_{j\in\mathcal{N}(i)} h_j^{l-1}. \tag{22}$$

where MEAN is the mean pooling operator. The comparison between the performance of different MPNN schemes is presented in the Supplementary Figure S1.

E. Graph construction

To build the MFGs, we employed the iML1515 model of *E. coli* MG1655 introduced by Monk *et al* [31]. To label the reaction nodes in the graph, we employed the growth assay data from the same work on strain BW25113. Since BW25113 lacks several genes from MG1655, we produced FBA solutions by setting their reaction bounds to zero and assuming aerobic growth. The reaction bounds can be found in Supplementary Table S1. To simulate *E. coli* growth in specific carbon sources, we set the corresponding exchange flux to a fixed value and deactivated all other carbon exchange fluxes. The list of all carbon sources and their corresponding exchange reactions can be found in Supplementary Table S2. All calculations were done with the COBRApy toolbox v0.26.3 using the `glpk` solver and the default objective function included in the iML1515 model.

F. Performance evaluation of binary classifiers

a. Training and evaluation in a single carbon source We start our evaluations of Flow-GAT from MFG resulting from glucose as the sole carbon source in Figure 2A. After mapping reactions to genes based on GPR rule set, growth rate values were converted to essentiality labels based on the threshold of 0.5 and were assigned to corresponding nodes in the MFG. As mentioned in Section II A, the labeled nodes in the MFG graph are imbalanced with a higher number of essentials compared to non-essential nodes. Therefore, for model training, we employed stratified sampling into 5 folds using built-in scikit-learn [5] functions with 1 fold for testing and 4 folds for training with the labeled nodes and 25% of the training set is set chosen as validation set (Figure 1D). For the initial tuning of hyperparameters, we employed grid search for each model and chose the best model settings based on the performance on the validation set; hyperparameters were kept constant for all other evaluations in the paper. All GNN based models were implemented and trained using the `GraphGym` in `PyG` package [14]; classic models (SVC, MLP, RF) were implemented using `scikit-learn`. The list of chosen hyperparameters for each model is available in Supplementary Tables S3 and S4.

Due to the small number of available labeled data in our dataset (255 in case of glucose MFG), to compare the performance of different mmodels (Figure 2) we trained the GNN models on the training folds and evaluated the performance on the test fold 5 times, each time changing the train and test fold to ensure that the results are not caused by split bias. In the case of GNN based models, for each evaluation step, 25% of the training fold was considered as the early stopping set. We kept track of the best model on the early stopping set, in terms of the loss value after each training epoch, until the maximum number of epochs was reached. The weights of the best model at the end of training were then saved and employed to predict for the nodes of the test fold. Additionally, each training and evaluation step on a test fold was repeated 10 times with the model retrained with a different initial random seed to make sure the predictions were not a result of random seed selection for weight initialization. In total, 50 evaluation steps (5 folds and 10 times for each fold) were gathered for each model. For the classic models (SVC, RF, MLP), we employed the same process except for the use early stopping set. We followed the same procedure for model evaluation in other carbon sources (Figure 3B).

b. Training and evaluations across carbon sources To produce the evaluations in Figure 3C, for each MFG the training and early stopping folds were chosen with a 4:1 ratio; in all cases we tested each model on all nodes of the other MFGs. Following the same scheme as in the previous section, the best performing model on the early stopping set was chosen for the evaluation of the test set; the training set was resampled 5 times and each model was retrained 10 times with different initial weights.

In Figures 3B–C, we quantified the performance improvement of FlowGAT over the no-skill classifier $100 \times (\mathrm{PRAUC}_{\mathrm{FlowGAT}} - \mathrm{PRAUC}_{\text{no-skill}}) / \mathrm{PRAUC}_{\text{no-skill}}$.

ACKNOWLEDGMENTS

TM was supported by the Research Council of Norway (grant number 312045). DAO was supported by the United Kingdom Research and Innovation (grant EP/S02431X/1, UKRI Centre for Doctoral Training in Biomedical AI. Computations were performed on resources provided by Sigma2 - the National Infrastructure for High Performance Computing and Data Storage in Norway (project NS9715).

REFERENCES

[1] Olufemi Aromolaran, Damilare Aromolaran, Itunuoluwa Isewon, and Jelili Oyelade. Machine learning approach to gene essentiality prediction: a review. *Briefings in Bioinformatics*, 22 (5), sep 2021. ISSN 14774054.

[2] Tomoya Baba, Takeshi Ara, Miki Hasegawa, Yuki Takai, Yoshiko Okumura, Miki Baba, Kirill A. Datsenko, Masaru Tomita, Barry L. Wanner, and Hirotada Mori. Construction of Escherichia coli K-12 in-frame, single-gene knockout mutants: the Keio collection. *Molecular Systems Biology*, 2:2006.0008, 2006. ISSN 1744-4292. doi:10.1038/msb4100050.

[3] Mariano Beguerisse-Díaz, Gabriel Bosque, Diego Oyarzún, Jesús Picó, and Mauricio Barahona. Flux-dependent graphs for metabolic networks. *npj Systems Biology and Applications*, 4(1), August 2018. doi:10.1038/s41540-018-0067-y.

[4] David B. Bernstein, Batu Akkas, Morgan N. Price, and Adam P. Arkin. Critical assessment

of E. coli genome-scale metabolic model with high-throughput mutant fitness data, January 2023. Pages: 2023.01.05.522875 Section: New Results.

[5] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[6] Cacheiro, P et al. Human and mouse essentiality screens as a resource for disease gene discovery. *Nature Communications 2020*, 11(1):1–16, jan 2020. ISSN 2041-1723.

[7] Chen Cai and Yusu Wang. A simple yet effective baseline for non-attribute graph classification. *CoRR*, abs/1811.03508, 2018.

[8] Tulio L. Campos, Pasi K. Korhonen, Robin B. Gasser, and Neil D. Young. An Evaluation of Machine Learning Approaches for the Prediction of Essential Genes in Eukaryotes Using Protein Sequence-Derived Features. *Computational and Structural Biotechnology Journal*, 17: 785–796, jan 2019. ISSN 2001-0370.

[9] Kathryn Cooper and Mauricio Barahona. Role-based similarity in directed networks. December 2010. doi:10.48550/arXiv.1012.2726. arXiv:1012.2726 [physics, q-bio].

[10] Varshit Dusad, Denise Thiel, Mauricio Barahona, Hector C. Keun, and Diego A. Oyarzún. Opportunities at the Interface of Network Science and Metabolic Modeling. *Frontiers in Bioengineering and Biotechnology*, 8, 1 2021. ISSN 2296-4185.

[11] Vijay Prakash Dwivedi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Graph Neural Networks with Learnable Structural and Positional Representations. February 2022. doi:10.48550/arXiv.2110.07875. arXiv:2110.07875 [cs].

[12] Ali Ebrahim, Moritz E. Beber, Synchon Mandal, Matthias König, Henning Redestig, Christian Diener, Dr Scientist, Peter St John, akaviaLab, Hemant_Yadav, Zak King, Nikolaus Sonnenschein, Maximilian Greil, JuBra, Maureen Carey, Ali Kaafarani, Benjamín Sánchez, Erik Cederstrand, Greg Medlock, Dr Daniel Weilandt, Afif Elghraoui, Vivek Rai, Svetlana Kutuzova (Galkina), Justin Taylor, Marvin van Aalst, SigitaR, Wanderrful, Achilles Rasquinha, Christian Lieven, and Anthon van der Neut. CobraPy. *Zenodo*, 2023. doi: 10.5281/zenodo.7823793.

[13] Léon Faure, Bastien Mollet, Wolfram Liebermeister, and Jean-Loup Faulon. A neural-

mechanistic hybrid approach improving the predictive power of genome-scale metabolic models. *Nature Communications*, 14(1):4669, August 2023. ISSN 2041-1723. doi:10.1038/s41467-023-40380-0. Number: 1 Publisher: Nature Publishing Group.

[14] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[15] Lilli J. Freischem, Mauricio Barahona, and Diego A. Oyarzún. Prediction of gene essentiality using machine learning and genome-scale metabolic models. *IFAC-PapersOnLine*, 55(23): 13–18, 2022. doi:10.1016/j.ifacol.2023.01.006.

[16] Ci Fu, Xiang Zhang, Amanda O. Veri, Kali R. Iyer, Emma Lash, Alice Xue, Huijuan Yan, Nicole M. Revie, Cassandra Wong, Zhen-Yuan Lin, Elizabeth J. Polvi, Sean D. Liston, Benjamin VanderSluis, Jing Hou, Yoko Yashiroda, Anne-Claude Gingras, Charles Boone, Teresa R. O'Meara, Matthew J. O'Meara, Suzanne Noble, Nicole Robbins, Chad L. Myers, and Leah E. Cowen. Leveraging machine learning essentiality predictions and chemogenomic interactions to identify antifungal targets. *Nature Communications*, 12(1):6497, November 2021. ISSN 2041-1723. doi:10.1038/s41467-021-26850-3.

[17] Carlos Eduardo García Sánchez and Rodrigo Gonzalo Torres Sáez. Comparison and analysis of objective functions in flux balance analysis. *Biotechnology Progress*, 30(5):985–991, 2014. ISSN 1520-6033. doi:10.1002/btpr.1949.

[18] Francesco Gatto, Heike Miess, Almut Schulze, and Jens Nielsen. Flux balance analysis predicts essential genes in clear cell renal cell carcinoma metabolism. *Scientific Reports*, 5(1):1–18, jun 2015. ISSN 2045-2322.

[19] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for Quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1263–1272. JMLR.org, August 2017.

[20] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 1025–1035, December 2017. ISBN 978-1-5108-6096-4.

[21] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation Learning on Graphs: Methods and Applications. 2018. doi:10.48550/arXiv.1709.05584. arXiv:1709.05584 [cs].

[22] Benjamin D. Heavner and Nathan D. Price. Comparative Analysis of Yeast Metabolic Net-

work Models Highlights Progress, Opportunities for Metabolic Reconstruction. *PLOS Computational Biology*, 11(11):e1004530, nov 2015. ISSN 1553-7358.

[23] Wenqi Hu, Susan Sillaots, Sebastien Lemieux, John Davison, Sarah Kauffman, Anouk Breton, Annie Linteau, Chunlin Xin, Joel Bowman, Jeff Becker, Bo Jiang, and Terry Roemer. Essential gene identification and drug target prioritization in Aspergillus fumigatus. *PLoS pathogens*, 3 (3):e24, March 2007. ISSN 1553-7374. doi:10.1371/journal.ppat.0030024.

[24] Zachary A. King, Justin Lu, Andreas Dräger, Philip Miller, Stephen Federowicz, Joshua A. Lerman, Ali Ebrahim, Bernhard O. Palsson, and Nathan E. Lewis. BiGG Models: A platform for integrating, standardizing and sharing genome-scale models. *Nucleic Acids Research*, 44 (D1):D515–D522, January 2016. ISSN 0305-1048. doi:10.1093/nar/gkv1049.

[25] Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. February 2017. doi:10.48550/arXiv.1609.02907. arXiv:1609.02907 [cs, stat].

[26] K. Kobayashi, S. D. Ehrlich, A. Albertini, G. Amati, K. K. Andersen, M. Arnaud, K. Asai, S. Ashikaga, S. Aymerich, P. Bessieres, F. Boland, S. C. Brignell, S. Bron, K. Bunai, J. Chapuis, L. C. Christiansen, A. Danchin, M. Débarbouillé, E. Dervyn, E. Deuerling, K. Devine, S. K. Devine, O. Dreesen, J. Errington, S. Fillinger, S. J. Foster, Y. Fujita, A. Galizzi, R. Gardan, C. Eschevins, T. Fukushima, K. Haga, C. R. Harwood, M. Hecker, D. Hosoya, M. F. Hullo, H. Kakeshita, D. Karamata, Y. Kasahara, F. Kawamura, K. Koga, P. Koski, R. Kuwana, D. Imamura, M. Ishimaru, S. Ishikawa, I. Ishio, D. Le Coq, A. Masson, C. Mauël, R. Meima, R. P. Mellado, A. Moir, S. Moriya, E. Nagakawa, H. Nanamiya, S. Nakai, P. Nygaard, M. Ogura, T. Ohanan, M. O'Reilly, M. O'Rourke, Z. Pragai, H. M. Pooley, G. Rapoport, J. P. Rawlins, L. A. Rivas, C. Rivolta, A. Sadaie, Y. Sadaie, M. Sarvas, T. Sato, H. H. Saxild, E. Scanlan, W. Schumann, J. F. M. L. Seegers, J. Sekiguchi, A. Sekowska, S. J. Séror, M. Simon, P. Stragier, R. Studer, H. Takamatsu, T. Tanaka, M. Takeuchi, H. B. Thomaides, V. Vagner, J. M. van Dijl, K. Watabe, A. Wipat, H. Yamamoto, M. Yamamoto, Y. Yamamoto, K. Yamane, K. Yata, K. Yoshida, H. Yoshikawa, U. Zuber, and N. Ogasawara. Essential Bacillus subtilis genes. *Proceedings of the National Academy of Sciences*, 100(8):4678–4683, April 2003. doi:10.1073/pnas.0730515100. Publisher: Proceedings of the National Academy of Sciences.

[27] Joshua E. Lewis and Melissa L. Kemp. Integration of machine learning and genome-scale metabolic modeling identifies multi-omics biomarkers for radiation resistance. *Nature Com-*

munications, 12(1):2700, May 2021. ISSN 2041-1723. doi:10.1038/s41467-021-22989-1.

[28] Nathan E. Lewis, Harish Nagarajan, and Bernhard O. Palsson. Constraining the metabolic genotype-phenotype relationship using a phylogeny of in silico methods. *Nature Reviews Microbiology*, 10(4):291–305, 2012. ISSN 17401534. doi:10.1038/nrmicro2737.

[29] Xingyi Li, Wenkai Li, Min Zeng, Ruiqing Zheng, and Min Li. Network-based methods for predicting essential genes or proteins: a survey. *Briefings in Bioinformatics*, 21(2):566–583, March 2020. ISSN 1477-4054. doi:10.1093/bib/bbz017.

[30] Fredrick M. Mobegi, Aldert Zomer, Marien I. de Jonge, and Sacha A. F. T. van Hijum. Advances and perspectives in computational prediction of microbial gene essentiality. *Briefings in Functional Genomics*, 16(2):70–79, March 2017. ISSN 2041-2649. doi:10.1093/bfgp/elv063.

[31] Jonathan M Monk, Colton J Lloyd, Elizabeth Brunk, Nathan Mih, Anand Sastry, Zachary King, Rikiya Takeuchi, Wataru Nomura, Zhen Zhang, Hirotada Mori, Adam M Feist, and Bernhard O Palsson. iML1515, a knowledgebase that computes escherichia coli traits. *Nature Biotechnology*, 35(10):904–908, October 2017. doi:10.1038/nbt.3956.

[32] Luis Müller, Mikhail Galkin, Christopher Morris, and Ladislav Rampášek. Attending to Graph Transformers. *arXiv*, 2023. doi:10.48550/arXiv.2302.04181.

[33] Sutanu Nandi, Abhishek Subramanian, and Ram Rup Sarkar. An integrative machine learning strategy for improved prediction of essential genes in Escherichia coli metabolism using flux-coupled features. *Molecular BioSystems*, 13(8):1584–1596, 2017. doi:10.1039/C7MB00234C. Publisher: Royal Society of Chemistry.

[34] Jeffrey D Orth, Ines Thiele, and Bernhard Ø Palsson. What is flux balance analysis? *Nature biotechnology*, 28(3):245–8, mar 2010. ISSN 1546-1696.

[35] Gianvito Pio, Paolo Mignone, Giuseppe Magazzù, Guido Zampieri, Michelangelo Ceci, and Claudio Angione. Integrating genome-scale metabolic modelling and transfer learning for human gene regulatory network reconstruction. *Bioinformatics*, 38(2):487–493, 2022. ISSN 1367-4803. doi:10.1093/bioinformatics/btab647.

[36] Kitiporn Plaimas, Roland Eils, and Rainer König. Identifying essential genes in bacterial metabolic networks with machine learning methods. *BMC systems biology*, 4, may 2010. ISSN 1752-0509.

[37] Morgan N. Price, Kelly M. Wetmore, R. Jordan Waters, Mark Callaghan, Jayashree Ray, Hualan Liu, Jennifer V. Kuehl, Ryan A. Melnyk, Jacob S. Lamson, Yumi Suh, Hans K.

Carlson, Zuelma Esquivel, Harini Sadeeshkumar, Romy Chakraborty, Grant M. Zane, Benjamin E. Rubin, Judy D. Wall, Axel Visel, James Bristow, Matthew J. Blow, Adam P. Arkin, and Adam M. Deutschbauer. Mutant phenotypes for thousands of bacterial genes of unknown function. *Nature*, 557(7706):503–509, May 2018. ISSN 1476-4687. doi:10.1038/s41586-018-0124-0. Number: 7706 Publisher: Nature Publishing Group.

[38] Terry Roemer, Bo Jiang, John Davison, Troy Ketela, Karynn Veillette, Anouk Breton, Fatou Tandia, Annie Linteau, Susan Sillaots, Catarina Marta, Nick Martel, Steeve Veronneau, Sebastien Lemieux, Sarah Kauffman, Jeff Becker, Reginald Storms, Charles Boone, and Howard Bussey. Large-scale essential gene identification in Candida albicans and applications to antifungal drug discovery. *Molecular Microbiology*, 50(1):167–181, 2003. ISSN 1365-2958. doi:10.1046/j.1365-2958.2003.03697.x.

[39] Sebastian Ruder. An overview of gradient descent optimization algorithms. 2017. doi:10.48550/arXiv.1609.04747. arXiv:1609.04747 [cs].

[40] Ankur Sahu, Mary-Ann Blätke, Jedrzej Jakub Szymański, and Nadine Töpfer. Advances in flux balance analysis by integrating machine learning and mechanism-based models. *Computational and Structural Biotechnology Journal*, 19:4626–4640, January 2021. ISSN 2001-0370. doi:10.1016/j.csbj.2021.08.004.

[41] Song-Min Schinn, Carly Morrison, Wei Wei, Lin Zhang, and Nathan E. Lewis. A genome-scale metabolic network model and machine learning predict amino acid concentrations in Chinese Hamster Ovary cell cultures. *Biotechnology and Bioengineering*, 118(5):2118–2123, 2021. ISSN 1097-0290. doi:10.1002/bit.27714.

[42] Robert Schuetz, Lars Kuepfer, and Uwe Sauer. Systematic evaluation of objective functions for predicting intracellular fluxes in Escherichia coli. *Molecular Systems Biology*, 3(1):119, January 2007. ISSN 1744-4292. doi:10.1038/msb4100162. Publisher: John Wiley & Sons, Ltd.

[43] Robert Schuetz, Nicola Zamboni, Mattia Zampieri, Matthias Heinemann, and Uwe Sauer. Multidimensional optimality of microbial metabolism. *Science (New York, N.Y.)*, 336(6081): 601–4, May 2012. ISSN 1095-9203. doi:10.1126/science.1216882.

[44] Daniel Segrè, Dennis Vitkup, and George M. Church. Analysis of optimality in natural and perturbed metabolic networks. *Proceedings of the National Academy of Sciences*, 99(23): 15112–15117, 2002. doi:10.1073/pnas.232349399.

[45] Evan S. Snitkin, Aimée M. Dudley, Daniel M. Janse, Kaisheen Wong, George M. Church, and Daniel Segrè. Model-driven analysis of experimentally determined growth phenotypes for 465 yeast gene deletion mutants under 16 different conditions. *Genome Biology*, 9(9):R140, September 2008. ISSN 1474-760X. doi:10.1186/gb-2008-9-9-r140.

[46] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

[47] Guido Zampieri, Supreeta Vijayakumar, Elisabeth Yaneske, and Claudio Angione. Machine and deep learning meet genome-scale metabolic modeling. *PLOS Computational Biology*, 15 (7):e1007084, jul 2019. ISSN 1553-7358.

[48] Tianzuo Zhan and Michael Boutros. Towards a compendium of essential genes-from model organisms to synthetic lethality in cancer cells. *Critical Reviews in Biochemistry and Molecular Biology*, 51:74–85, 3 2016. ISSN 15497798.

[49] Xue Zhang, Wangxin Xiao, and Weijia Xiao. DeepHE: Accurately predicting human essential genes based on deep learning. *PLOS Computational Biology*, 16(9):e1008229, September 2020. ISSN 1553-7358. doi:10.1371/journal.pcbi.1008229.

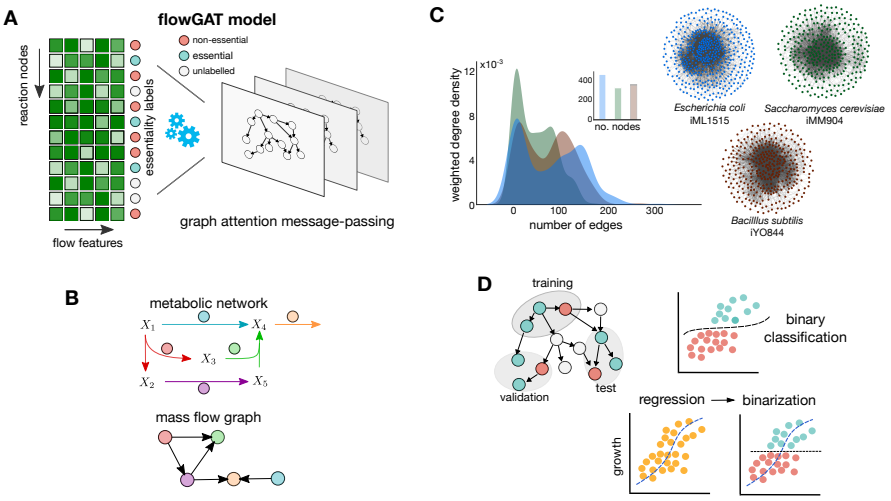


FIG. 1. **Elements of the FlowGAT model for gene essentiality prediction.** (**A**) Schematic of the FlowGAT architecture proposed in this paper. The model integrates a digraph representation of FBA solutions (Mass Flow Graphs, MFG), where nodes are reactions and edges encode the re-distribution of metabolite mass flows between reactions. We featurize each node with flow-based scores and label them as essential or non-essential using data from gene knock-out assays. Using a graph neural network with an attention layer, FlowGAT predicts essentiality for unlabelled reactions. (**B**) Construction of mass flow graphs from FBA solutions. The top network is an exemplar metabolic network, and the bottom digraph is the corresponding MFG constructed; nodes are reactions and two nodes are connected if they share metabolites as reactants or products. The edge weights are computed from the metabolite mass flows as described in (3); more details on the MFG construction can be found in [3]. (**C**) Exemplar MFGs for several microbes computed from their genome-scale models using standard FBA [12] with the default growth condition in each case; density plots show the distribution of edge weights in each case. (**D**) For model training and validation, labeled nodes in the MFG are separated into training, validation and test sets. The validation set is used for early stopping and performance metrics are computed on the test set. We explored two training frameworks for FlowGAT; as a binary classifier and as a regressor of growth rate that can be binarized to produce essentiality predictions.

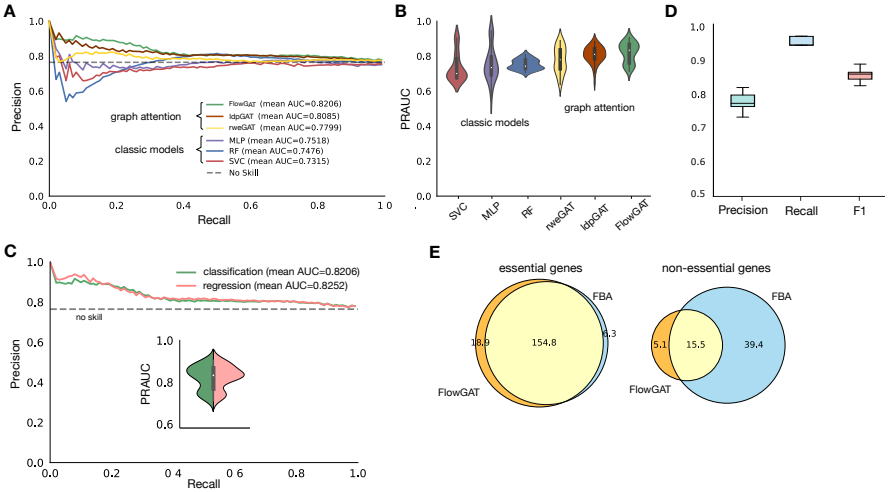


FIG. 2. **Performance of FlowGAT as predictor of metabolic gene essentiality in *Escherichia coli*.** (**A**) Precision-Recall (PR) curves for classic binary classifiers and graph neural networks trained on essentiality measurements for *E. coli* growing in aerobic conditions with glucose as the sole carbon course [31]. Classic models were trained using the Flow Profile Embeddings (FPE) defined in Eq. (6) computed from wild type FBA solutions. The graph neural networks were trained using the mass flow graph and the FPE as node feature vectors, as well as two other popular node embedding techniques (Local Degree Profile, LDP [7] and Random Walk Embedding, RWE [11]). Results show PR curves averaged across 50 model evaluations consisting of 5 rounds of testing on 20% test folds and and 10 rounds of model re-training for different random seeds; the dashed line represented the precision (74.9%) of the no-skill classifier given the class imbalance of the data. (**B**) Distribution of PRAUC scores across the 50 evaluations. The graph attention models outperform classic binary classifier; among the three considered node embeddings, FPE provides the best performance. (**C**) Retraining FlowGAT as a regressor provides slight gains in performance; inset shows the distribution of PRAUC scores across 50 evaluations; the regressor was trained on growth rate data [31] and predictions are subsequently binarized to produce essentiality labels. (**D**) Exemplar classification results by the best performing model (FlowGAT trained as regressor, as in panel C); results show precision, recall and F1-score for the 50 evaluations and fixed classification threshold. (**E**) Comparison between FlowGAT and FBA predictions over the entire gene set in glucose MFG; Venn diagrams show the number of genes called correctly for each model, averaged across 10 rounds of re-training FlowGAT for different random seeds (for details of training refer to Methods)

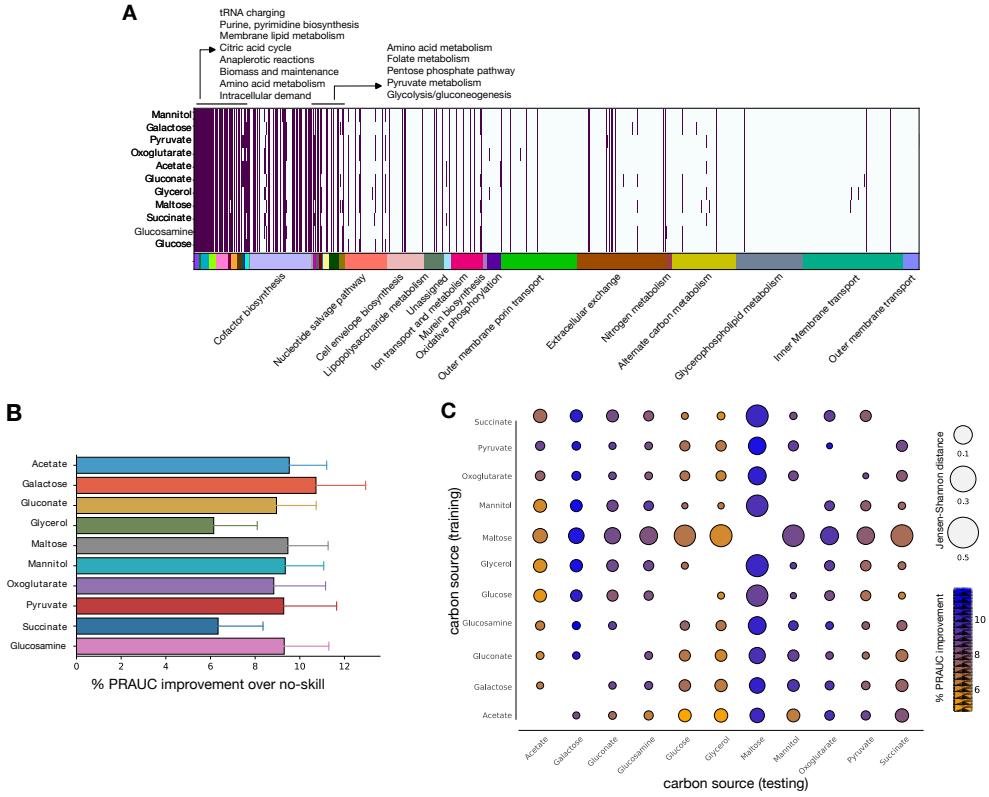
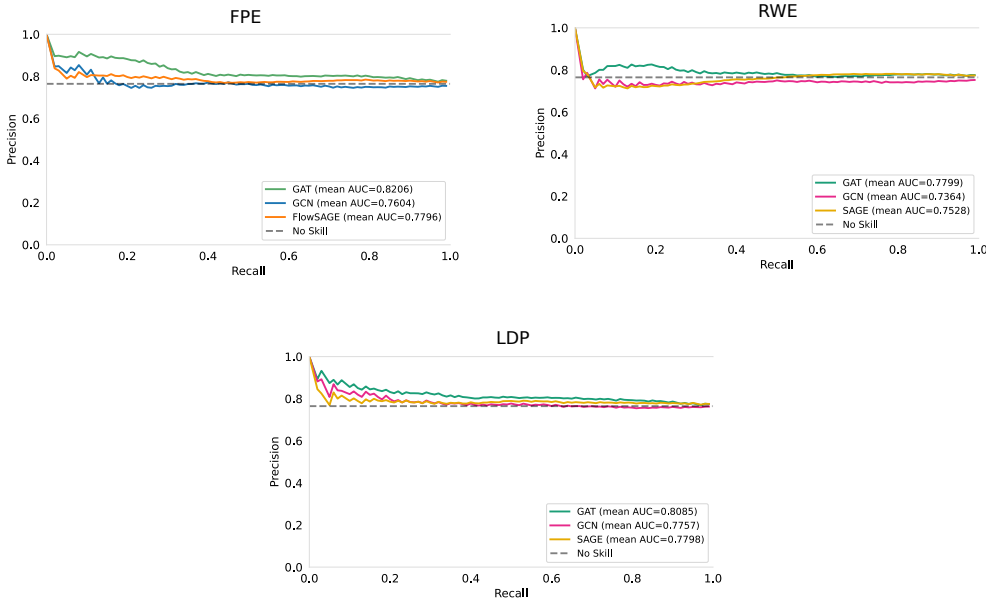
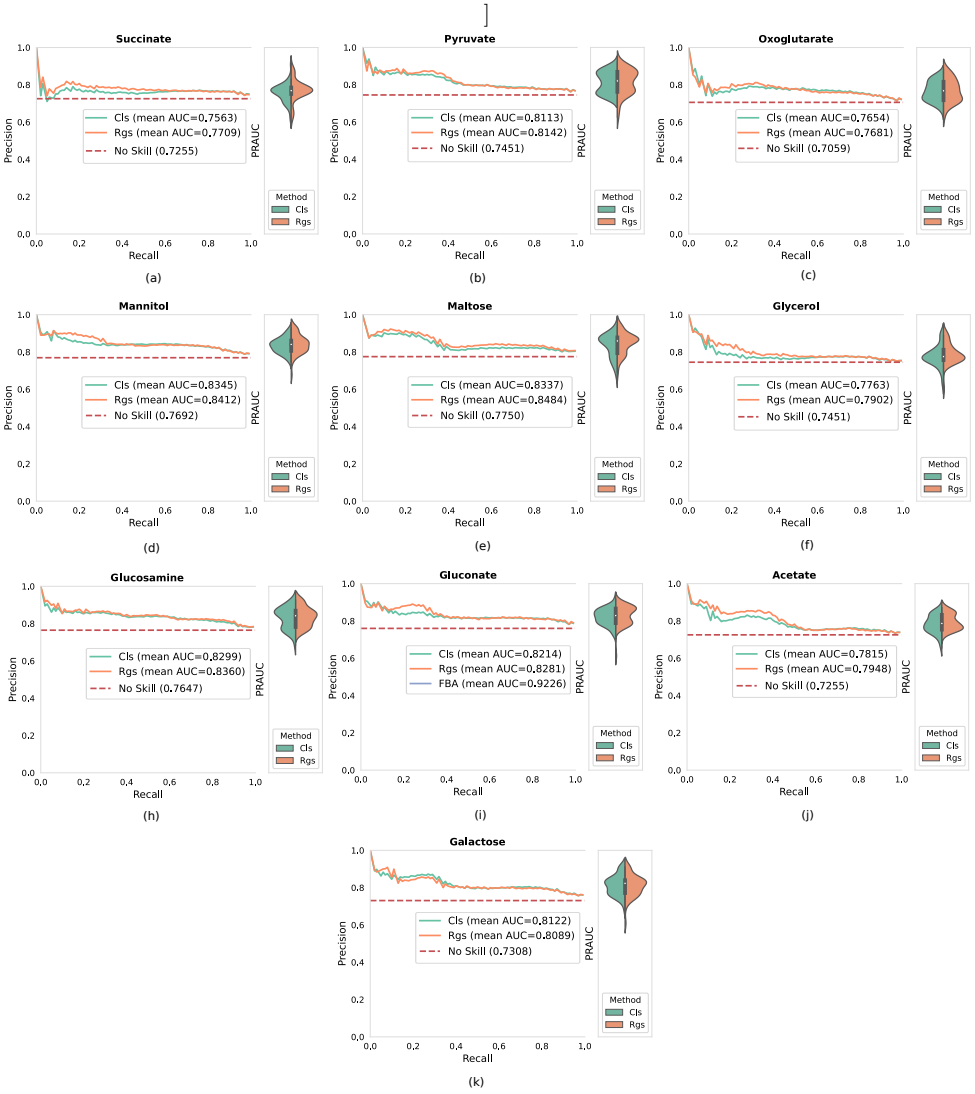


FIG. 3. **Essentiality predictions of FlowGAT for *Escherichia coli* growing in different carbon sources.** **(A)** Heatmap of reactions present in mass flow graphs (MFG) computed from wild type FBA solutions computed for ten carbon sources. Each MFG is obtained through changing the carbon source; the color bar denotes the different metabolic subsystems as annotated in the latest genome-scale metabolic model iML1515 [31]. **(B)** Prediction performance of FlowGAT trained and tested on the different condition-dependent MFGs. Bars show the average improvement of PRAUC scores across the 50 evaluations with respect to the no-skill classifier; error bars denote one standard deviation of the PRAUC. Full precision-recall curves for each case can be found in Supplementary Figure S2. **(C)** Performance of FlowGAT in cross-testing across different carbon sources; in each case, the model was trained on one graph and tested on the nodes of all other graphs, totalling 90 cross-test evaluations. The color bar indicates the improvement in PRAUC over the no-skill classifier; the bubble radius denotes the graph-to-graph distance computed as the Jansen-Shannon divergence between the distribution of node features.

Supplementary Figures and Tables



Supplementary Figure S1. **Performance comparison between different message passing schemes and node embeddings.** Model training and evaluation followed the same procedure as Figure 2A in the main text. Results show three node embeddings (Flow Profile Embeddings defined in Eq. (6), Local Degree Profile, and Random Walk Embeddings as explained in the Methods) and three popular message passing schemes (Graph Attention as in Figure 2A, Graph Convolution Networks, and Graph SAGE).



Supplementary Figure S2. **Performance comparison of classification and regression training for FlowGAT trained on labels and mass flow graphs for various carbon sources.** For each of the ten graphs, FlowGAT was trained on the nodes of the graph using both classification and regression training scheme, as in Figure 2C in the main text. Precision-recall curves were averaged over 5 cross-validation folds and 10 rounds of re-training for different random seeds.

Reaction	Bounds
L-arabinose isomerase (ARAI)	
L-ribulokinase (RBK_L1)	
Rhamnulose-1-phosphate aldolase (RMPA)	
Lyxose isomerase (LYXI)	$v_{lb} = v_{ub} = 0$
L-rhamnose isomerase (RMI),	
Rhamnulokinase (RMK)	
B-galactosidase (LACZ)	
Oxygen exchange (EX_o2_e)	$v_{lb} = -20$

Supplementary Table S1. **Bounds for exchange reactions for the *E. coli* iML1515 model growing in aerobic conditions with glucose as the only carbon source.**

Carbon source	Reaction	Bounds
Succinate	EX_succ_e	
Pyruvate	EX_pyr_e	
Oxoglutarate	EX_akg_e	
Mannitol	EX_mnl_e	
Maltose	EX_malt_e	
Glycerol	EX_glyc_e	$v_{lb} = -10$
Glucosamine	EX_acgam_e	
Gluconate	EX_glcn_e	
Acetate	EX_ac_e	
Galactose	EX_gal_e	

Supplementary Table S2. **Exchange reactions for different carbon sources in the *E. coli* iML1515 genome-scale metabolic model.** For each carbon source, the corresponding reaction bounds were adjusted and all other reaction bounds for other sources were set to 0.

	Hyperparameter	Value
Architecture	Number of message passing layers	4
	Number of post message passing layers	1
	Dimension of hidden layers	16
	Activation function	ReLU
	Dropout rate	0.1
	k_{m}	8
Training	Base learning rate	0.01
	Learning rate decay	0.1
	Momentum	0.9
	Optimization algorithm	ADAM
	Scheduler	Cosine Annealing
	Max epochs	200
	Min epochs	20
	Classification loss	Cross entropy
	Regression loss	MSE

Supplementary Table S3. **Hyperparameters list for GNN based models.** For each GNN based model the list of above hyperparameters was determined after initial tuning on a chosen validation set through grid search. Afterward, the above set was used for all evaluations reported in the paper. For a fair comparison amongst different GNNs, the architecture and training hyperparameters are kept the same and the message passing formula is the only change in the architecture (e.g., GAT, GCN, SAGE)

Model	Hyperparameter	Value
RF	Criterion	entropy
	Max depth	50
	Max features	sqrt
	Num estimators	300
SVC	Kernel	rbf
	Gamma	scale
MLP	Activation function	ReLU
	Optimizer	ADAM
	Hidden layer size	100
	Number of layers	2

Supplementary Table S4. **Hyperparameters list for classic ML models.** For each model the list of above hyperparameters was determined after initial tuning on a chosen validation set through grid search.

Paper III

Tesla-Rapture: A Lightweight Gesture Recognition System From mmWave Radar Sparse Point Clouds

Dariush Salami[1], Ramin Hasibi[1], Sameera Palipana, Petar Popovski, Tom Michoel, and Stephan Sigg,
IEEE Transactions on Mobile Computing, **22/08** (2022)

III

[1]Equal contribution

Paper IV

IV

Integrating Angle-Agnostic Sensing into Cellular Networks using NR Sidelink

Dariush Salami, Ramin Hasibi, Stefano Savazzi, Tom Michoel, and Stephan Sigg
(2022)

Paper V

A general deep reinforcement learning hyperheuristic framework for solving combinatorial optimization problems

Jakkob Kallestad, Ramin Hasibi, Ahmad Hemmati, and Keneth Sörensen
European Journal of Operational Research, **309**/**1** (2023)

V

European Journal of Operational Research 309 (2023) 446–468

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Interfaces with Other Disciplines

A general deep reinforcement learning hyperheuristic framework for solving combinatorial optimization problems

Jakob Kallestad [a], Ramin Hasibi [a,*], Ahmad Hemmati [a], Kenneth Sörensen [b]

[a] Department of Informatics, University of Bergen, Norway
[b] Faculty of Business and Economics, ANT/OR - University of Antwerp Operations Research Group, Belgium

ARTICLE INFO

Article history:
Received 11 October 2021
Accepted 11 January 2023
Available online 16 January 2023

Keywords:
Heuristics
Hyperheuristic
Adaptive metaheuristic
Deep reinforcement learning
Combinatorial optimization

ABSTRACT

Many problem-specific heuristic frameworks have been developed to solve combinatorial optimization problems, but these frameworks do not generalize well to other problem domains. Metaheuristic frameworks aim to be more generalizable compared to traditional heuristics, however their performances suffer from poor selection of low-level heuristics (operators) during the search process. An example of heuristic selection in a metaheuristic framework is the adaptive layer of the popular framework of Adaptive Large Neighborhood Search (ALNS). Here, we propose a selection hyperheuristic framework that uses Deep Reinforcement Learning (Deep RL) as an alternative to the adaptive layer of ALNS. Unlike the adaptive layer which only considers heuristics' past performance for future selection, a Deep RL agent is able to take into account additional information from the search process, e.g., the difference in objective value between iterations, to make better decisions. This is due to the representation power of Deep Learning methods and the decision making capability of the Deep RL agent which can learn to adapt to different problems and instance characteristics. In this paper, by integrating the Deep RL agent into the ALNS framework, we introduce Deep Reinforcement Learning Hyperheuristic (DRLH), a general framework for solving a wide variety of combinatorial optimization problems and show that our framework is better at selecting low-level heuristics at each step of the search process compared to ALNS and a Uniform Random Selection (URS). Our experiments also show that while ALNS can not properly handle a large pool of heuristics, DRLH is not negatively affected by increasing the number of heuristics.

1. Introduction

A metaheuristic is an algorithmic framework that offers a coherent set of guidelines for the design of heuristic optimization methods. Classical frameworks such as Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Simulated Annealing (SA) are examples of such frameworks (Dokeroglu, Sevinc, Kucukyilmaz, & Cosar, 2019). Moreover, there is a large body of literature that addresses solving combinatorial optimization problems using metaheuristics. Among these, Adaptive Large Neighbourhood Search (ALNS) (Ropke & Pisinger, 2006) is one of the most widely used metaheuristics. It is a general framework based on the principle of Large Neighbourhood Search (LNS) of Shaw (1998), where the objective value is iteratively improved by applying a set of "removal" and "insertion" operators on the solution. In ALNS, each of the removal and insertion operators have weights associated with them that determine the probabilities of selecting these during the search. These weights are continuously updated after a certain number of iterations (called a segment) based on their recent effect on improving the quality of the solution during the segment. The ALNS framework was early on an approach specific to routing problems. However, in recent years, there has been a growing number of studies that employ this approach to other problem types, e.g., scheduling problems (Laborie & Godard, 2007). Its high quality of performance at finding solutions has made it a go-to approach in many recent studies in combinatorial optimization problems (Aksen, Kaya, Sibel Salman, & Özge Tüncel, 2014; Chen, Demir, & Huang, 2021; Demir, Bektaş, & Laporte, 2012; Friedrich & Elbert, 2022; Grangier, Gendreau, Lehuédé, & Rousseau, 2016; Gullhav, Cordeau, Hvattum, & Nygreen, 2017; Li, Chen, & Prins, 2016). The ALNS framework has several advantages. For most optimization problems, a number of

* Corresponding author.
E-mail addresses: jakobkallestad@gmail.com (J. Kallestad), Ramin.Hasibi@uib.no (R. Hasibi), Ahmad.Hemmati@uib.no (A. Hemmati), kenneth.sorensen@uantwerpen.be (K. Sörensen).

https://doi.org/10.1016/j.ejor.2023.01.017

well-performing heuristics are already known which can be used as the operators in the ALNS framework. Due to the large size and diversity of the neighborhoods, the ALNS algorithm will explore huge chunks of the solution space in a structured way. As a result, ALNS is very robust as it can adapt to different characteristics of the individual instances, and is able to avoid being trapped in local optima (Pisinger & Ropke, 2019). According to Turkeš, Sörensen, & Hvattum (2021), the adaptive layer of ALNS has only minor impact on the objective function value of the solutions in the studies that have employed this framework. Moreover, the information that the adaptive layer uses for selecting heuristics is limited to the past performance of each heuristic. This limited data can make the adaptive layer naïve in terms of decision making capability because it is not able to capture other (problem-independent) information about the current state of the search process, e.g., the difference in cost between past solutions, whether the current solution has been encountered before during the search, or the number of iterations since the solution was last changed, etc. We refer to the decision making capability of ALNS as performing on a "macro-level" in terms of adaptability, i.e., the weights of each heuristic is only updated at the end of each segment. This means that the heuristics selected within a segment are sampled according to the fixed probabilities of the segment. This limitation makes it impossible for ALNS to take advantage of any short-term dependencies that occur within a segment that could help aid the heuristic selection process.

Another area where ALNS struggles is when faced with a large number of heuristics to choose from. In order to find the best set of available heuristics for ALNS for a specific setting, initial experiments are often required to identify and remove inefficient heuristics, and this can be both time consuming and computationally expensive (Hemmati & Hvattum, 2017). Furthermore, some heuristics are known to perform very well for specific problem variations or specific conditions during the search, but they may have a poor average performance. In this case, it might be beneficial to remove these from the pool of heuristics available to ALNS in order to increase the average performance of ALNS, but this results in a less powerful pool of heuristics that is unable to perform as well during these specific problem variations and conditions.

To address the issues in ALNS, one can use Reinforcement Learning (RL). RL is a subset of machine learning concerned with "learning how to make decisions"—how to map situations to actions—so as to maximize a numerical reward signal. One of the main tasks in machine learning is to generalize a predictive model based on available training data to new unseen situations. An RL agent learns how to generalize a good policy through interaction with an environment which returns the reward in exchange for receiving an action from the agent. Therefore, through a trial-and-error search process, the agent is trained to achieve the maximum expected future reward at each step of decision making conditioned on the current situation (state). Thus, training an RL agent (to achieve the best possible results in similar situations), makes the agent aware of the dynamics of the environment as well as adaptable to similar environments with slightly different settings. One of the more recent approaches in RL is Deep RL which benefits from the powerful function approximation property of deep learning tools. In this approach, different functions that are used to train and make decisions in an RL agent are implemented using Artificial Neural Networks (ANNs). Different Deep RL algorithms dictate the training mechanism and interaction of the ANNs in the decision making process of the agent (Sutton & Barto, 2018). Therefore, integration of the Deep RL into the adaptive layer of the ALNS can make the resulting framework much smarter at making decisions at each iteration and improve the overall performance of the framework.

In this paper, we propose **Deep Reinforcement Learning Hyperheuristic (DRLH)**, a general approach to selection hyperheuristic framework (definition in Section 2) for solving combinatorial optimization problems. In DRLH, we replace the adaptive layer of ALNS with a Deep RL agent responsible for selecting heuristics at each iteration of the search. Our Deep RL agent is trained using Proximal Policy Optimization (PPO) method of Schulman, Wolski, Dhariwal, Radford, & Klimov (2017) which is a standard approach for stable training of the Deep RL agent in different environments. The proposed DRLH utilizes a search state consisting of a problem-independent feature set from the search process and is trained with a problem-independent reward function that encourages better solutions. This approach makes the framework easily applicable to many combinatorial optimization problems without any change in the method and given the proper training step for each problem separately. The training process of DRLH makes it adaptable to different problem conditions and settings, and ensures that DRLH is able to learn good strategies of heuristic selection prior to testing, while also being effective when encountering new search states. In contrast to the macro-level decision making of ALNS, the proposed DRLH makes decisions on a "micro-level", meaning that only the current search state information affects the probabilities of choosing heuristics. This allows for the probabilities of selecting heuristics to change quickly from one iteration to the next, helping DRLH adapt to new information of the search as soon as it becomes available. The Deep RL agent in DRLH is able to effectively leverage this search state information at each step of the search process in order to make better decisions for selecting heuristics compared to ALNS.

To evaluate the performance and generalizability of DRLH, we choose four different combinatorial optimization problems to benchmark against different baselines in terms of best objective found and the speed of convergence as well as the time it takes to solve each problem. These problems include the Capacitated Vehicle Routing Problem (CVRP), the Parallel Job Scheduling Problem (PJSP), the Pickup and Delivery Problem (PDP), and the Pickup and Delivery Problem with Time Windows (PDPTW). These problems are commonly used for evaluation in the literature and are diverse in terms of difficulty to find good and feasible solutions. They additionally correspond to a broad scope of real world applications. For each problem, we create separate training and test datasets. In our experiments, we compare the performance of DRLH on different problem sizes and over an increasing number of iterations of the search and demonstrate how the heuristic selection strategy of DRLH differs from other baselines throughout the search process.

Our experiments show the superiority of DRLH compared to the popular method of ALNS in terms of performance quality. For each of the problem sets, DRLH is able to consistently outperform other baselines when it comes to best objective value specifically in larger instances sizes. Additionally, DRLH does not add any overhead to the instance solve time and the performance gain is a result of the decision making capability of the Deep RL agent used. Further experiments also validate that unlike other algorithms, the performance of DRLH is not negatively affected by increasing the number of available heuristics to choose from. In contrast to this, ALNS struggles when handling a large number of heuristics to choose from. This advantage of our framework makes the development process for DRLH very simple as DRLH seems to be able to automatically discover the effectiveness of different heuristics during the training phase without the need for initial experiments in order to manually reduce the set of heuristics.

The remainder of this paper is organized as follows: In Section 2, related previous work in hyperheuristics and Deep RL is presented. In Section 3, we propose the overall algorithm of DRLH as well as the choice of heuristics and parameters. The

descriptions of the four combinatorial optimization problems used for benchmarking purposes are illustrated in Section 4. The experimental setup and the results of our evaluation are presented in Sections 5 and 6, respectively.

2. Related work

In this section, we first define the term "Hyperheuristic" and review some of the traditional work that fall into this category and point out their limitations. We also mention some of the methods that employ Deep RL for solving combinatorial problems and their shortcomings. In the end, we explain how we combine the best of two domains (Hyperheuristic and Deep RL) to take advantage of both their methodologies.

The term *hyperheuristic* was first used in the context of combinatorial optimization by Cowling, Kendall, & Soubeiga (2001) and described as *heuristics to choose heuristics*. Burke et al. (2010) later extended the definition of hyperheuristic to "a search method or learning mechanism for selecting or generating heuristics to solve computational search problems". The most common classification of hyperheuristics makes the distinction between *selection hyperheuristics* and *generation hyperheuristic*. Selection hyperheuristics are concerned with creating a selection mechanism for heuristics at each step of the search, while generation hyperheuristics are concerned with generating new heuristics using basic components from already existing heuristic methods. This paper will focus on selection hyperheuristics methods.

Although it is possible to create highly effective problem-specific and heuristic-specific methods for heuristic selection, these methods do not always generalize well to other problem domains and different sets of heuristics. A primary motivation of hyperheuristic research is therefore the development of general-purpose, problem-independent methods that can deliver good quality solutions for many combinatorial optimization problems without having to make significant modifications to the methods. Thus, advancements done in hyperheuristic research aims to be easily applicable by experts and non-experts alike, to various problems and heuristics sets without requiring extra effort such as domain knowledge about the specific problem to be solved.

A classic example of using RL in hyperhueristics is the work of Özcan, Misir, Ochoa, & Burke (2010) in which they propose a framework that uses a traditional RL method for solving examination timetabling. Performance is compared against a simple random hyperheuristic and some previous work, and results show that using RL obtains better results than simply selecting heuristics at random. The RL used here learns during the search process by adjusting the probabilities of choosing heuristics based on their recent performance during the search. This type of RL framework shares many similarities with the ALNS framework, and therefore suffers from the same limitations as those mentioned for ALNS.

Apart from RL, supervised learning, which is another machine learning technique, has also been utilized in hyperheuristic frameworks to improve the performance. A hyperheuristic method for the Vehicle Routing Problem named *Apprentice Learning-based Hyper-heuristic (ALHH)* was proposed by Asta & Özcan (2014) in which an apprentice agent seeks to imitate the behavior of an expert agent through supervised learning. The training of the ALHH works by running the expert on a number of training instances and recording the selected actions of the expert together with a *search state* that consists of the previous action used and the change in objective function value for the past n steps. These recordings of search state and action pairs build up a training dataset in which a decision tree classifier is used in order to predict the action choice of the expert. This makes up a supervised classification problem in which the final accuracy of the model is reported to be around 65%. In the end ALHH's performance is compared against the expert and is reported to perform very similarly to the expert, and even slightly outperforming the expert for some instances.

Tyasnurita, Özcan, Shahriar, & John (2015) further improved upon the apprentice learning approach by replacing the decision tree classifier with a multilayer perceptron (MLP) neural network, and named their approach MLP-ALHH. This change increased the representational power of the search state and resulted in a better performance that is reported to even outperform the expert. A limitation of ALHH and MLP-ALHH is their use of the supervised learning framework which makes performance of these approaches bounded by the expert algorithm's performance. A consequence of this is that the feedback used to train the predictive models of ALHH and MLP-ALHH is binary, i.e. it either matches that of the expert or not, leaving no room for alternative strategies that might perform even better than the expert. In contrast, DRLH uses a Deep RL framework that neither requires, nor is bounded by an expert agent and therefore has more potential to outperform existing methods by coming up with new ways of selecting heuristics. The feedback used to train DRLH depends on the effect of the action on the solutions, and the amount received varies depending on several factors. Additionally, DRLH takes future iterations of the search into account, while ALHH and MLP-ALHH only consider the immediate effect of the action on the solution. Because of this, diversifying behavior is encouraged in DRLH when it gets stuck, as it will help improve the solution in future iterations. Another difference of DRLH compared to ALHH and MLP-ALHH is that the features of the search state used by DRLH contain more information compared to the search state of the other two methods which ultimately makes the agent more aware of the search state and thus capable of making effective decisions.

In addition to hyperheuristic approaches there have also recently been many attempts at solving popular routing problems using Deep RL by the machine learning community. A big limitation of these works is that they all rely on problem-dependent information, and are usually designed to solve a single problem or a small selection of related problems, often requiring significant changes to the approach in order to make them work for several problems. In first versions of these studies, Deep RL is used as a constructive heuristic approach for solving the vehicle routing problem in which the agent, representing the vehicle, selects the next node to visit at each time step (Kool, van Hoof, & Welling, 2019; Nazari, Oroojlooy, Snyder, & Takac, 2018). Although this is very effective when compared to simple construction heuristics for solving routing problems, it lacks the quality of solutions provided by iterative metaheuristic approaches as well as being unable to find feasible solutions in the case of more difficult routing problems that involve more advanced constraints such as pickup and delivery problem with time windows.

Another approach that leverages Deep RL for solving combinatorial optimizations is to take advantage of the decision making ability of the agent in generating or selecting low-level heuristics to be applied on the solution. Hottung & Tierney (2019) have used a Deep RL agent to generate a heuristic for rebuilding partially destroyed routes in the CVRP using a large neighbourhood search framework. This method is an example of heuristic generation and is specifically designed to solve the CVRP. Thus, it can not easily be generalized to other problem domains. In Chen & Tian (2019), a framework is presented for using two Deep RL agents for finding a node in the solution and the best heuristic to apply on that node at each step. Although the authors claim that this method is generalizable to three different combinatorial optimization problems, the details in representation of the problem and type of ANNs used for the agents from one problem to another change a lot depending on the nature of the problem. Additionally, one would have to come up with new inputs and representation when applying this method to other optimization problems that are not discussed in the study

which reduces the generalizability of the framework. Lu, Zhang, & Yang (2020) suggested the use of a Deep RL agent for choosing low-level heuristic at each step for the CVRP. This work also suffers from the generalizability to other types of optimization problems due to the elements of the Deep RL agent that are specific to the CVRP problem. Additionally, in this approach the training process of the agent is designed in such a way that the agent is only focused on intensification rather than diversification. Thus, the diversification in their framework is done by a rule-based escape approach rather than giving the RL agent freedom to find the balance between diversification and intensification, which could lead to better results.

To the best of our knowledge previous work on this topic either suffer from a lack of generalizability in approach when it comes to other problems in the domain or they do not take advantage of the learning mechanism and representation power of Deep RL. In this work we seek to address these issues by introducing DRLH.

3. DRLH

In this section, we present the DRLH, a hyperheuristic framework to solve combinatorial optimization problems.

Our proposed hyperheuristic framework uses an RL agent for the selection of heuristics. This process improves on the ALNS framework of Ropke & Pisinger (2006) by leveraging the RL agent's decision making capability in choosing the next heuristic to apply on the solution in each iteration. The pseudocode of DRLH is illustrated in Algorithm 1.

Algorithm 1: DRLH.

Function `Deep Reinforcement Learning Hyperheuristic`
- Generate an initial solution x with objective function of $f(x)$ **(see section 3.5)**
- H=Generate_heuristics() **(see section 3.1)**
- $x_{best} = x$, $f(x_{best}) = f(x)$
- **Repeat**
  - $x' = x$
  - choose $h \in H$ based on policy $\pi(h|s,\theta)$ **(see section 3.3)**
  - Apply heuristic h to x'
  - **if** $f(x') < f(x_{best})$ **then**
    - $x_{best} = x$
  - **end**
  - **if** *accept*(x', x) ***(see section 3.3)***, **then**
    - $x = x'$
  - **end**
- **Until** *stop-criterion met* ***(see section 3.4)***
- **return** x_{best}

3.1. Generating heuristics

The heuristic generation process follows the steps in Algorithm 2. The set H consists of all possible heuristics that can be applied on the solution x at each iteration. The general method for obtaining these heuristics is to combine a removal and an insertion operator. Furthermore, additional heuristics can also be placed in H that do not share the characteristic of being a combination of removal and insertion operators. In the following, we present one example set of H for the problem types considered for this paper.

Algorithm 2: Generation of the set of heuristics H.

Function `Generate_heuristics`
- H={};
- **foreach** *removal operator* $r \in \mathcal{R}$ **do**
  - **foreach** *insertion operator* $j \in \mathcal{I}$ **do**
    - Create a heuristic h by combining r and j;
    - $H = H \cup h$;
  - **end**
- **end**
- **foreach** *additional heuristic* $c \in \mathcal{C}$ **do**
  - $H = H \cup c$;
- **end**
- **return** H

3.2. Sample set of heuristics

Each heuristic $h \in H$ is a combination of a removal and an insertion operator presented in Tables 1 and 2. Furthermore, one additional intensifying heuristic is also added to H. In each iteration, a heuristic $h \in H$ is applied on the incumbent solution x with cost of $f(x)$ and generates a new solution x' with cost of $f(x')$. For our sample set of heuristics, H has the size of $|H| = 29$ (7 removals $\times$ 4 insertions + 1 additional).

3.2.1. Removal operators $\mathcal{R}$

The set of all removal operators $\mathcal{R}$ are provided in Table 1. Seven removal operators are implemented, five of which are focused on inducing diversification through a high degree of randomness denoted by *Random* in their name. For intensification purposes, we define the operator "*Remove_largest_*$\mathcal{D}$" which uses the metric *Deviation* $\mathcal{D}$. We define the deviation D_i as the difference in cost with and without $element_i$ in the solution, and thus "*Remove_largest_*$\mathcal{D}$" removes the elements with the largest D_i. Finally, "*Remove_*τ" operator selects a number of consecutive elements in the solution and removes them.

3.2.2. Insertion operators $\mathcal{I}$

Table 2 lists the set of insertion operators $\mathcal{I}$ used. A total of 4 insertion operators are utilized to place the removed elements in a suitable position in solution x'. Operator "*Insert_greedy*" places each removed element in the position which obtains the minimum total cost of the new solution $f(x')$. Operator "*Insert_beam_search*" performs beam search with a search width of 10 for inserting each removed element. Beam search keeps track of the 10 best combinations of positions after inserting each removed element in the solution and inserts the elements in the best combination of positions that obtain the minimum $f(x')$ in the search space. The "*Insert_by_variance*" operator calculates the variance of the ten best insertion positions for each of the removed elements. Then the elements are ordered from high to low variance and inserted back into the solution with the "*Insert_greedy*" operator. Finally, operator "*Insert_first*" places each removed element randomly in the first feasible position found in the new solution.

3.2.3. Additional heuristic $\mathcal{C}$

Unlike in ALNS where only removal and insertion operators are used, our framework can also make use of standalone heuristics that share neither of the these types of characteristics. An example of one such additional heuristic, "*Find_single_best*", is responsible for generating the best possible new solution from the incumbent by changing one element. This heuristic calculates the cost of removing each element and re-inserting it with "*Insert_greedy*", and applies this procedure on the solution x for the element that

Table 1
List of all removal operators.

Name	Description
Random_remove_XS	Removes between 2–5 elements chosen randomly
Random_remove_S	Removes between 5–10 elements chosen randomly
Random_remove_M	Removes between 10–20 elements chosen randomly
Random_remove_L	Removes between 20–30 elements chosen randomly
Random_remove_XL	Removes between 30–40 elements chosen randomly
*Remove_largest_*$\mathcal{D}$	Removes 2–5 elements with the largest $\mathcal{D}_i$
*Remove_*τ	Removes a random segment of 2–5 consecutive elements in the solution

Table 2
List of all insertion operators.

Name	Description
Insert_greedy	Inserts each element in the best possible position
Insert_beam_search	Inserts each element in the best position using beam search
Insert_by_variance	Sorts the insertion order based on variance and inserts each element in the best possible position
Insert_first	Inserts each element randomly in the first feasible position

achieves the minimum cost $f(x')$. "*Find_single_best*" is the only additional heuristic that is used in the proposed sample set of heuristics, H.

3.3. Acceptance criteria and stopping condition

We use the acceptance criterion $accept(x', x)$ used in simulated annealing (Kirkpatrick, Gelatt, & Vecchi, 1983). This acceptance criterion depends on the difference in objective value between the incumbent x and the new solution x' denoted as $\Delta E = f(x') - f(x)$ together with a temperature parameter T that is gradually decreasing throughout the search. A new solution is always accepted if it has a lower cost than the incumbent, $\Delta E < 0$. In addition, worse solutions are accepted with probability $e^{-|\Delta E|/T}$.

To determine the initial temperature T_0 we accept all solutions for the first 100 iterations of the search and keep track of all the non-improving steps, $\Delta E > 0$. Then, we calculate the average of these positive deltas $\overline{\Delta E}$ in order to get:

$$T_0 = \frac{\overline{\Delta E}}{\ln 0.8} \tag{1}$$

To decrease the temperature we use the cooling schedule of Crama & Schyns (2003), and the search terminates after a certain number of iterations has been reached.

3.4. Deep RL agent for selection of h

In a typical RL setting, an agent is trained to optimize a policy π for choosing an action through interaction with an environment. At each time step (iteration) t, the agent chooses an action A_t and receives a scalar reward R_t from the environment indicating how good the action was. State S_t is defined as the information received at each time step from the environment based on the agent's choice of action A_t from a set of possible actions. Thus, a stochastic policy π for the agent is defined as

$$\pi(a|s) = Pr\{A_t = a|S_t = s\}. \tag{2}$$

One such type of policy is the parameterized stochastic policy function in which the probability of action selection is also conditioned on a set of parameters $\theta \in \mathcal{R}^d$. As a result, Eq. (2) is redefined as

$$\pi(a|s, \theta) = Pr\{A_t = a|S_t = s, \theta_t = \theta\}. \tag{3}$$

in which θ_t represents the parameters at time step t (Sutton & Barto, 2018). In our setting, the policy π is a MultiLayer perceptron (MLP), which is a class of non-linear function approximation (Goodfellow, Bengio, & Courville, 2016). In this scenario, the aim is to obtain the optimal policy π^* by tuning θ which represents the weights of the MLP network.

The training process for an RL agent is illustrated in Algorithm 3. For training the weights of the MLP, we follow the policy gradient method of PPO introduced in Schulman et al. (2017). In order to generalize to different variations of an optimization problem, the training process is done for a number of problem instances (episodes) with each instance corresponding to a different set of attributes of the problem. Each instance is optimized for a certain number of iterations (time steps) and at the end of each episode the policy parameters θ are updated until we obtain the optimal policy. Once the training process is complete, the optimal policy π^* is used to solve unseen instances in the test sets.

Algorithm 3: Training the Deep RL agent.

```
Result: π* optimal policy
Start with random setting of θ for a random policy π;
for e ← 1 to episodes do
    Receive initial state S_1;
    for t ← 1 to steps do
        choose and perform action a ∈ A_t according to
        π(a|s, θ);
        Receive R_t = v and s ∈ S_{t+1} from the environment
    end
    Optimize the policy parameters θ according to PPO
    (Schulman et~al., 2017).
end
```

As mentioned above, three main properties of the RL agent which are used to obtain the optimal policy π^* for solving the intended problem are the *state representation*, the *action space*, and the *reward function*. These parameters dictate the training process and decision making capability of the agent and are therefore essential for obtaining good solutions to optimization problems. Moreover, in our proposed approach, these properties are set to be independent of the type of problem which helps this approach generalize to many types of combinatorial optimization problems. The state representation contains the information about the current solution and the overall search state, and is shown to the agent at each step in order to guide the agent in the action selection process. The action space consists of a set of interchangeable heuristics that can be selected at each time step by the agent. Finally, the reward function guides the learning of the agent during

Table 3
A list of all features used for the state representation.

Name	Description
reduced_cost	The difference in cost between the previous & the current solutions
*cost_from_*min	The difference in cost between the current & the best found solution
cost	The cost of the current solution
min_cost	The cost of the best found solution
temp	The current temperature
cs	The cooling schedule (α)
no_improvement	The number of iterations since the last improvement
index_step	The iteration number
was_changed	1 if the solution was changed from the previous, 0 otherwise.
unseen	1 if the solution has not previously been encountered in the search, 0 otherwise.
last_action_sign	1 if the previous step resulted in a better solution, 0 otherwise.
last_action	The action in previous iteration encoded in 1-hot.

training and should be designed in a way that helps the agent optimize the objective of the problem. In the following, we explain the choice for each of these properties.

3.4.1. State representation

The state consists of a set of useful features for guiding the agent to select the best action/heuristic at each iteration in the search. We have prioritized general state features that are independent of the specifics of the problem being solved. In other words, the state representation is easily applicable to many optimization problems of different domains. Table 3 lists all the state features used by the agent.

The state features *cost* and *min_cost* together with *index_step* allow the agent to know approximately how well it is doing during the search. This becomes apparent if *cost* and *min_cost* are higher than their average values during training with respect to *index_step*. These state features primarily help at a macro-level by making the agent stick to a high-level strategy of heuristic selection throughout the search. *cost_from_*min, *temp*, *cs* and *no_improvement* inform the agent about how likely a new solution is to be accepted. These state features help the agent know how much intensification/diversification is appropriate at that step. For instance if it should try to escape a local optima or if it should focus on intensification. The last five state features; *reduced_cost*, *was_changed*, *unseen*, *last_action_sign* and *last_action* inform the agent about the immediate changes from the previous solution to the current solution. In particular, *reduced_cost* shows the difference in cost between the previous and current solution. *was_changed* indicates if the solution was changed from the previous step to the current step. *unseen* indicates whether the current solution was encountered before during the search. Finally, *last_action_sign* indicates if the solution improved or worsened from the previous step, and *last_action* indicates the action that was used in the previous step. Together these five features give information about what action the agent selected in the previous step and the result of that action. This helps the agent make decisions at a micro-level and is particularly useful as the agent can avoid selecting deterministic or semi-deterministic heuristics such as *Remove_largest_*$\mathcal{D}$, *Insert_by_variance* and *Find_single_best* twice in a row if the first time did not lead to any improvement, because then it is less likely, if at all, to work the second time on the same solution. This is particularly important for *Find_single_best* which is a fully deterministic heuristic and produces the same result if applied for two consecutive iterations.

3.4.2. Action

The actions in our setting for the agent are the same as the set of heuristics H, i.e, $A_t = H$. At each iteration of the DRLH (c.f., Algorithm 1), a heuristic h is selected and applied on the solution by the agent. Therefore the policy function π in Eq. (3) is redefined as

$$\pi(h|s,\theta) = Pr\{A_t = h|S_t = s, \theta_t = \theta\}. \tag{4}$$

3.4.3. Reward function

A good reward function needs to balance the need for gradual and incremental rewards while also preventing the agent from exploiting the reward function without actually optimizing the intended objective (also known as reward hacking Amodei et al., 2016). For our framework, we propose a reward functions that has the above property. We refer to this as R_t^{5310}, the formula for which is

$$R_t^{5310} = \begin{cases} 5, & \text{if } f(x') < f(x_{best}) \\ 3, & \text{if } f(x') < f(x) \\ 1, & \text{if } accept(x', x) \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

R_t^{5310} is inspired from the scoring mechanism that is applied in the ALNS framework for measuring the performance of each heuristic in a segment. This reward function encourages the agent to find better solutions than the current one as this gives a high reward. In addition it also gives a small reward if it finds a slightly worse solution that manages to get accepted by the acceptance criterion. This property of the function in turn motivates the agent to use diversifying operators when it is no longer able to improve upon the current solution. Moreover, other reward functions were considered for the framework which take the step-wise improvement of the solution as well as the amount of improvement into account. Further experiments on these reward functions demonstrate that the R_t^{5310} proved to be more stable and faster to train compared to the others (results in Appendix A). Furthermore, given the fact that R_t^{5310} comes from the original scoring function of ALNS in Ropke & Pisinger (2006), we use the same function for our Deep RL agent and ALNS for an equal comparison.

3.5. Solution representation and initial solution

For all the problems described in Section 4, the solution is represented as a permutation of orders/calls/jobs on each of the available vehicles/machines. Additionally, for the PDP and PDPTW, each call should be in the solution twice, one time for each of the pickup and the delivery elements respectively, and no call can be present in multiple vehicles, as the same vehicle has to both pick up and deliver the call.

The initial solutions for all of the problems are created by inserting all the orders/calls/jobs into the vehicles/machines using the *insert_greedy* operator from Table 2. For each of the problems and each test instance, DRLH, ALNS and URS start with the same initial solution for a fair comparison.

4. Problem sets

We consider four sets of combinatorial optimization problems as examples of problems that can be solved using DRLH. These problems are the Capacitated Vehicle Routing Problem (CVRP), Parallel Job Scheduling Problem (PJSP), Pickup and Delivery Problem (PDP) and Pickup and Delivery Problem with Time Windows (PDPTW).

4.1. CVRP

The Capacitated Vehicle Routing Problem is one of the most studied routing problems in the literature. It consists of a set of N orders that needs to be served by any of the M number of vehicles. Additionally, there is a depot in which the vehicles travel from and return to when serving the orders. Following the previous work, the number of vehicles in this particular problem is not fixed, but is naturally limited to $M = \{1, \ldots, N\}$. Meaning that the maximum number of vehicles that can be utilized is N and the minimum number is 1. Usually the number of vehicles used will fall somewhere in between depending on which number results in the best solution. Each order has a weight W_i associated to it, and the vehicles have a maximum capacity. The sequence of orders that a vehicle visits after leaving the depot before returning to the depot is referred to as a *tour*. There needs to be a minimum of one tour and a maximum of N tours. The combined weight of the orders in a tour can not exceed the maximum capacity of the vehicle, and so several tours are often needed in order to solve the CVRP problem. The objective of this problem is to create a set of tours that minimize the total distance travelled by all the vehicles that are serving at least one order.

4.2. PJSP

In the Parallel Job Scheduling Problem, we are given N jobs and M machines. Each of the machines operate with a different processing speed, and so the time required to complete job i on machine m is $T_{i,m}$. Each job has a *due time* associated with it, and if a job is finished after its due time, a delay is calculated for that job. The delay for job i is the difference in time between the due time and the actual finishing time of job i, and can never be lower than 0. The objective of the problem is to find a sequence of jobs to complete on each of the machines in order to minimize the total delay of all the jobs.

4.3. PDP

In Pickup and Delivery Problem we are given N calls and a single vehicle with a maximum capacity. Each call has a weight, a pickup location, and a delivery location, and is served when the order is transported by the vehicle from the pickup to the delivery location. The objective of the problem is to minimize the traveling distance of the vehicle while serving all the calls and not carrying more than the maximum capacity at any point.

4.4. PDPTW

In pickup and delivery problem, we are given a set of calls. A call consists of an origin and a destination and an amount of goods that should be transported. A heterogeneous fleet of vehicles are serving the calls, picking up goods at their origins and delivering them to their destinations. Time windows are assigned to each call at origins and destinations. Pickups and deliveries must be within the associated time windows. In the event of early arrival of a vehicle to a node before the start of the time window, the mentioned vehicle must wait until the beginning of the time window before being able to perform the pick up or delivery. A vehicle is never allowed to arrive at a node after the end of the time window. Additionally, a service time is considered for each time a call gets picked up or delivered, i.e., the time it takes a vehicle to load or deliver the goods at each node. For each call, a set of feasible vehicles is determined. Each vehicle has a predetermined maximum capacity of goods as well as a starting terminal in which duty of the vehicle starts. Moreover, a start time is assigned to each vehicle indicating the time that the vehicle leaves its starting terminal. The vehicle must leave its start terminal at the starting time, even if a possible waiting time at the first node visited occurs. The goal is to construct valid routes for each vehicle, such that time windows and capacity constraints are satisfied along each route, each pickup is served before the corresponding delivery, pickup and deliveries of each call are served on the same route and each vehicle only serves calls it is allowed to serve. The construction of the routes should be in such a way that they minimize the cost function. There is also a compatibility constraint between the vehicles and the calls. Thus, not all vehicles are able to handle all the calls. If we are not able to handle all calls by our fleet, we have to outsource them and pay the cost of not transporting them. For more details, readers are referred to Hemmati, Hvattum, Fagerholt, & Norstad (2014).

5. Experimental setup

In this section, we explain the baseline methods, process of hyperparameter selection, and dataset generation methods used for evaluation of the DRLH framework.

5.1. Experimental environment

The computational experiments in this paper were run on a desktop computer running a 64-bit Ubuntu 20.04 operating system with a AMD Ryzen 5 3600 processor and 32GB RAM.

5.2. Baseline models

Four baseline frameworks are chosen to compare with DRLH. Three of these methods use the same approach as DRLHin selecting a heuristic from the same set of heuristics at each iteration with the difference being in selection strategy. The last baseline uses a trained Deep RL agent to build a route by selecting a node at each step. The details of the baselines are presented in the following.

5.2.1. Adaptive large neighborhood search (ALNS)

As our approach is improving on the ALNS algorithm, this method is chosen as a baseline for performance comparison. This framework measures the performance of each heuristic using a scoring function for a certain number of iterations, referred to as a *segment*. At the end of each segment, the probability of choosing a heuristic during the next segment is updated using the aggregated scores of each heuristic in the previous segment. The extent to which the scores of the previous segment should contribute to updating the weights is controlled by the *reaction factor*.

There is a trade-off between speed and stability when choosing the values of the segment size and the reaction factor. Longer segments mean less frequent updates of the weights, but may increase the quality of the update. Similarly, a low reaction factor means that the weights can take longer to reach their desired values, but may also prevent sudden unfavorable changes to the weights due to the stochastic nature of the problem.

5.2.2. Uniform random selection (URS)

As a simpler approach to the selecting heuristics in each iteration, this method selects the heuristic randomly from H with equal probabilities.

5.2.3. Tuned random selection (TRS)

We introduce another baseline to our experiments which is refered to as TRS. For this method, we tuned the probabilities of selecting heuristics using the method of IRace (López-Ibáñez, Dubois-Lacoste, Pérez Cáceres, Birattari, & Stützle, 2016). The package "IRace" applies iterative F-Race to tune a set of parameters in an optimization algorithm (heuristic probabilities in our method) based on the performance on the training dataset.

5.2.4. Attention Module (AM) based Deep RL heuristic

We also consider the AM method of Kool et al. (2019) which achieved state-of-the-art results among the Deep RL based method for solving combinatorial optimization problems. This method uses the Deep RL agent combined with deep attention representation learning to build the solution at each step in a constructive manner using problem specific features from the environment. As a result, when applied on new problems, a new set of features as well as a problem specific representation learning scheme need to be defined. For example, the time window and vehicle incompatibility constraints were not mentioned in the original paper and for that reason we can not solve the difficult problem of PDPTW with this framework.

5.3. Hyperparameter selection

The hyperparameters for the Deep RL agent determine the speed and stability of the training process and also the final performance of the trained model. A small learning rate will cause training to take longer, but the smaller updates to the neural network also increase the chance of a better final performance once the model has been fully trained. Because the training process is done in advance of the testing stage, we opt for a slow and stable approach in order to train the best models possible. The hyperparameters of Deep RL agent for the experiments are listed in Table 4.

In order to decide on the hyperparameters for DRLH, some initial experiments were performed on the PDP problem (as the simple baseline problems compared to others) on a separate validation set to see which combinations performed best. The resulting set of hyperparameters have been applied for all experiments in this paper. Our motivation for doing so is that we wanted to test the generalizability of the framework in terms of the hyperparameters as well as the performance on different problems. By tuning the hyperparameters on a simpler problem and applying them to all other problems of all sizes and variations, we tried to avoid overtuning DRLH for every separate problem to keep the evaluation fair for the baseline methods and make sure that the advantage of our approach is in the decision making approach not the choice of hyperparameters for each problem. Moreover, this adds to the generalizability trait of the framework that does not require hyperparameter selection for each specific problem. Based on our experiments we found that these set of hyperparameters work very well across all the problem variations that we tested. It is likely that these hyperparameters can work for any underlying combinatorial optimization problem, as the hyperparameters for DRLH are related to the high-level problem of *heuristic selection*, which stays the same, regardless of what the underlying combinatorial optimization problem actually is. In the case of ALNS, we apply the same set of optimized hyperparameters that are suggested by Hemmati et al. (2014), which is optimized for solving the benchmark of PDPTW.

5.4. Dataset generation

For all the problem variations we generate a distinct training set consisting of 5000 instances, and a distinct testing set consisting of 100 instances. Additionally, for PDPTW we also utilize a known set of benchmark instances for testing (Hemmati et al., 2014).

5.4.1. CVRP

CVRP data instances are generated in accordance with the generation scheme of Nazari et al. (2018), Kool et al. (2019), but we also add two bigger problem variations. Instances of sizes $N = 20$, $N = 50$, $N = 100$, $N = 200$ and $N = 500$ are generated where N is the number of orders. For each instance the depot location and node locations are sampled uniformly at random from the unit square. Additionally, each order has a size associated with it defined as $\hat{\gamma} = \gamma_i / D_N$ where γ_i is sampled from the discrete set of $\{1, \ldots, 9\}$, and the normalization factor D_N is set as $D_{20} = 30$, $D_{50} = 40$, $D_{100} = 50$, $D_{200} = 50$, $D_{500} = 50$, for instances with N orders, respectively.

5.4.2. PJSP

For the PJSP we generate instances of sizes $N = 20$, $N = 50$, $N = 100$, $N = 300$ and $N = 500$ where N is the number of jobs and using $M = \lfloor N/4 \rfloor$ machines. Job i's required processing steps PS_i are sampled from the discrete set of $\{100, 101, \ldots, 1000\}$, and machine m's speed S_m, in processing steps per time unit, is sampled from $\mathcal{N}(\mu, \sigma^2)$ with $\mu = 10$, $\sigma = 30$, and the speed is rounded to the nearest integer and bounded to be at least 1. From there we get that the time required to process job i on machine m is calculated as $\lceil PS_i / S_m \rceil$.

5.4.3. PDP

For this problem, PDP data instances of sizes $N = 20$, $N = 50$, and $N = 100$ are generated where N is the number of nodes based on the generation scheme of Nazari et al. (2018), Kool et al. (2019). For each instance the depot location and node locations are sampled uniformly at random in unit square. Half of the nodes are pickup locations whereas the other half is the corresponding delivery locations. Additionally, each call has a size associated with it defined as $\hat{\gamma} = \gamma_i / D_N$ where γ_i is sampled from the discrete set of $\{1, \ldots, 9\}$, and the normalization factor D_N is set as $D_{20} = 15$, $D_{50} = 20$, $D_{100} = 25$, for each problem with N number of nodes respectively.

5.4.4. PDPTW

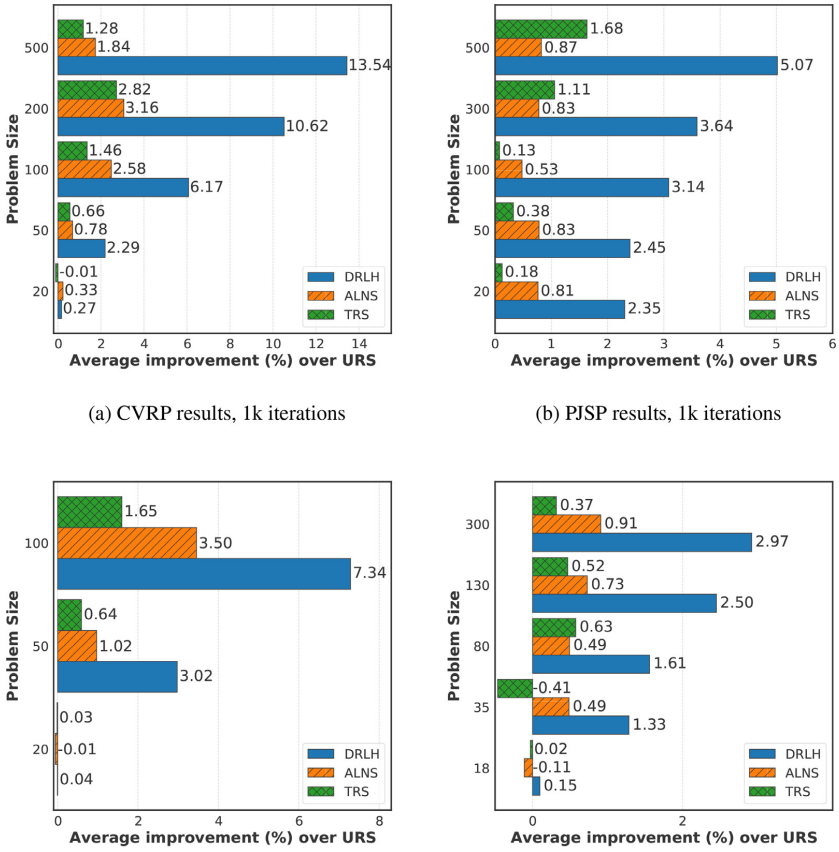
For the PDPTW we use instances with different combinations of number of calls and number of vehicles, see Table 5. For generating the training set and the 100 test instances, we use the provided instance generator of Hemmati et al. (2014). Additionally, we

Table 4
The hyperparameters used during training for the Deep RL agent of DRLH.

Hyperparameter	Value
Learning rate	1e−5
Batch size	64
First hidden layer size	256
Second hidden layer size	256
Discount factor	0.5

Table 5
Properties of different variations of the PDPTW instance types.

#Calls	#Vehicles	#Vehicle types
18	5	3
35	7	4
80	20	2
130	40	2
300	100	2



(a) CVRP results, 1k iterations

(b) PJSP results, 1k iterations

(c) PDP results, 1k iterations

(d) PDPTW results, 1k iterations

Fig. 1. Performance of DRLH on the generated test set.

use benchmark instances of Hemmati et al. (2014) for the remaining results. The benchmark test set consists of some instances of each variation, which are solved 10 times during testing in order to calculate the average best objective for each instance. Previous work by Homsi, Martinelli, Vidal, & Fagerholt (2020) have found the global optimal objectives for these instances, and we use these optimal values in order to calculate the *Min Gap (%)* and *Avg Gap (%)* to the optimal values for instances with 18, 35, 80 and 130 calls. Additionally, we also generate and test on a much larger instance size of 300 calls where we do not have the exact global optimal objectives, but instead use the best known values found by DRLH with 10,000 iterations to calculate the *Min Gap (%)* and *Avg Gap (%)*.

6. Results

In this section, we present the results of different experiments on the performance of DRLH. In the first experiment (Section 6.1), we set the number of iterations of the search to 1000 to compare the quality of the best found objective by each algorithm over a limited number of iterations for different problem sizes in the test set. In the next experiment (Section 6.2), we increase the number of iterations for all the methods and compare their performance when enough iterations are provided to fully explore the problem space. We also report the results on the benchmark of Hemmati et al. (2014) instances (Section 6.3). In order to demonstrate another advantage of using DRLH, we conduct an experiment with increased number of heuristics to illustrate the dependence of each framework on the performance of individual heuristics when the number of heuristics exceeds a certain number (Section 6.4). Additionally, we report the convergence speed and the training and inference time of each framework on instances of each problem (sections 6.5 and 6.6). Next, to gain insight into the reason behind the superiority of DRLH compared to the state of the art, we provide some figures and discuss the difference in strategy behind choosing a heuristic between DRLH and ALNS (Section 6.7). Finally, we compare the performance of DRLH, with a Deep RL heuristic approach (Section 6.8). Additional experiments and results regarding the reward function, convergence speed, and dependency of DRLH on the size of the problem can be found in Appendix.

6.1. Experiment on generated test set

For this experiment, each method was evaluated on a test set of 100 generated instances for each of the problems introduced in Section 4. Figure 1(a) shows the improvement in percentage that

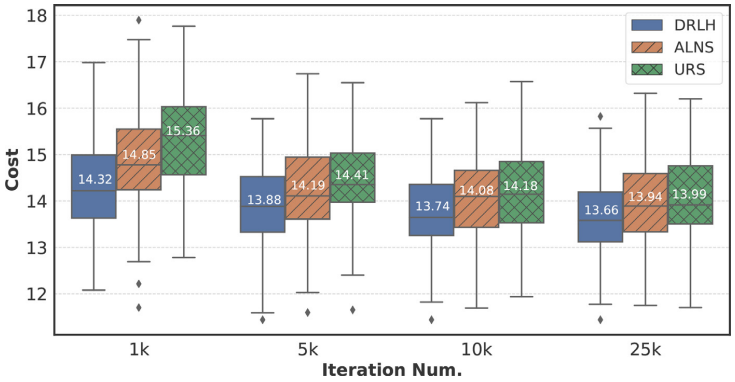


Fig. 2. Boxplot results for different iterations of PDP100.

Table 6
Average results for PDPTW instances with mixed call sizes after 1000 iterations.

		DRLH			ALNS			URS		
#C	#V	Min Gap (%)	Avg Gap (%)	Time (s)	Min Gap (%)	Avg Gap (%)	Time (s)	Min Gap (%)	Avg Gap (%)	Time (s)
18	5	**0.00**	**0.18**	32	0.00	0.46	25	0.00	0.40	12
35	7	2.67	**5.78**	9	3.45	7.08	36	**2.46**	6.40	27
80	20	**3.04**	**4.85**	37	3.64	6.51	98	4.62	7.23	100
130	40	**3.44**	**4.66**	100	4.00	6.24	186	4.85	6.71	176
300	100	**2.40**	**3.15**	637	3.10	5.04	599	5.29	6.51	398

using DRLH, ALNS, and TRS have over using URS on CVRP instances of different sizes. We see that DRLH is able to outperform all the baselines for all the instance sizes except for the smallest size. There is also a clear trend that shows how DRLH becomes increasingly better compared to other methods on larger instance sizes. Figure 1(b) shows a similar result for the PJSP problem. We see that DRLH is able to outperform the other methods for all of the instance sizes tested. Compared to the previous results, we see that the degree of improvement on larger instance sizes is less prominent for DRLH, but we also see that ALNS does not perform noticeably better on larger instance sizes at all. Because of that we still see a clear separation in performance between DRLH and ALNS on larger instance sizes that seem to grow with larger instance sizes. Finally, we observe a similar trend for PDP and PDPTW as for the other problems, which can be seen in Fig. 1(c) and (d), respectively. From this figure we see that DRLH outperforms ALNS and URS on all instance sizes tested and that performance difference tends to increase with larger instance sizes.

6.2. Experiment on increased number of iterations

Figure 2 shows that the number of iterations for improving the solution affects the minimum costs found for all the methods. We see that DRLH outperforms the baselines when tested for 1000, 5000, 10,000 and 25,000 iterations, and that the percentage difference between DRLH, ALNS and URS gets smaller as the number of iterations grows larger. Intuitively this makes sense as all three methods are getting closer to finding the optimal objectives for the test instances, and more iterations for improving the solution during the search makes the choices of which heuristics to select less sensitive compared to searching for a smaller number of iterations.

6.3. Experiment on the PDPTW benchmark dataset

In this section, we report results for PDPTW on the benchmark test set shown in Tables 6, 7 and 8 for 1000, 5000 and 10,000 iterations, respectively. We see from the tables that DRLH outperforms ALNS and URS on all of the tests on average, showing that it can find high quality solutions and has a robust average performance. Furthermore, we can see that the performance difference between DRLH and the baselines increases on bigger instances, meaning that DRLH scales favorably to the size of the problem, making it more viable for big industrial-sized problems compared to ALNS and URS.

We have also included the average time in seconds for optimizing the test instances. Note that the difference in time-usage is not directly dependent on the framework for selecting the heuristics (DRLH, ALNS, URS), but rather on the difference in time-usage of the heuristics themselves. This means that if all the heuristics used the same amount of time, then there would not be any time difference between the frameworks. However, because there is a relatively large variation in the time-usage between the different heuristics, we see a considerable variation between the frameworks as they all have different strategies for heuristic selection.

6.4. Experiment on the increased pool of heuristics

In addition to the set of heuristics mentioned in Section 3.1 we have also created an extended set of heuristics (see list in Appendix B). In total this extended set consists of 142 heuristics. Figure 3 shows the average gap of using the extended set compared to using the standard set for each of DRLH, ALNS and URS on PDPTW. The extended set obtains worse results on average compared to the standard set, but there is an interesting difference between the performance hit of DRLH, ALNS and URS when comparing the results of the extended set and the standard set. We see from Fig. 3 that DRLH is relatively unaffected by increasing the number of available heuristics (being only 0.02% worse on average), while ALNS and URS are performing much worse when using the extended set, and ALNS is hit especially hard. A likely reason for this is that there are too many heuristics to accurately explore

Table 7
Average results for PDPTW instances with mixed call sizes after 5000 iterations.

		DRLH			ALNS			URS		
#C	#V	Min Gap (%)	Avg Gap (%)	Time (s)	Min Gap (%)	Avg Gap (%)	Time (s)	Min Gap (%)	Avg Gap (%)	Time (s)
18	5	**0.00**	**0.00**	56	0.00	0.11	159	0.00	0.01	64
35	7	1.02	**2.95**	218	**0.78**	3.24	207	1.26	3.49	141
80	20	**1.76**	**3.25**	201	2.11	4.04	503	2.54	4.14	471
130	40	**2.10**	**3.14**	530	2.51	3.93	837	2.91	4.09	767
300	100	**0.48**	**1.15**	2580	1.01	2.35	2062	2.07	2.99	2352

Table 8
Average results for PDPTW instances with mixed call sizes after 10,000 iterations.

		DRLH			ALNS			URS		
#C	#V	Min Gap (%)	Avg Gap (%)	Time (s)	Min Gap (%)	Avg Gap (%)	Time (s)	Min Gap (%)	Avg Gap (%)	Time (s)
18	5	**0.00**	**0.00**	219	0.00	0.02	338	0.00	0.00	102
35	7	**0.67**	**2.02**	182	0.78	2.66	410	0.68	2.77	289
80	20	**1.80**	**2.95**	321	2.03	3.33	757	2.17	3.36	972
130	40	**1.93**	**2.84**	877	2.38	3.34	1307	2.56	3.37	1609
300	100	**0.00**	**0.64**	4630	0.55	1.89	4120	1.46	2.18	4203

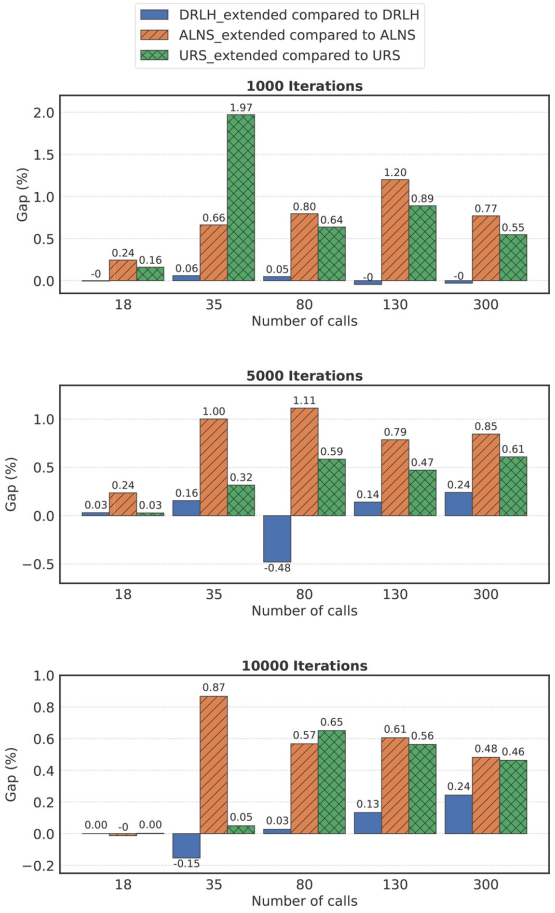


Fig. 3. Results of an Increased Pool of Heuristics.

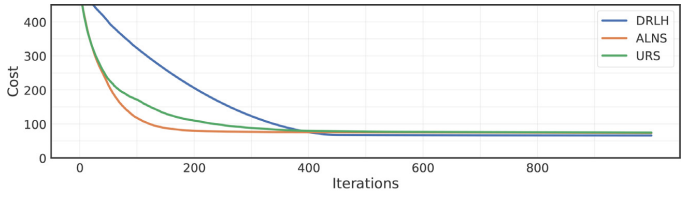
all of them during the search in order to identify the best heuristics and take advantage of them during the search.

An important conclusion from this result (albeit one that needs further empirical proof) is that when using DRLH, it is possible to supply it with a large number of heuristics and let DRLH identify the best ones to use. This is not possible for ALNS and consequently it is often necessary to spend time carrying out prior experiments with the aim of finding a small set of the best performing heuristics to include in the final ALNS model. This also resonates with the conclusion of Turkeš et al. (2021), who argue that the performance of ALNS benefits more from a careful a priori selection of heuristics, than from an elaborate adaptive layer. Considering that prior experiments can be quite time consuming, using DRLH can lead to a simpler development phase where heuristics can be added to DRLH without needing to establish their effectiveness beforehand, and not having to worry whether adding them will hurt the overall performance. Should a heuristic be unnecessary, then DRLH will learn to not use it during the training phase.

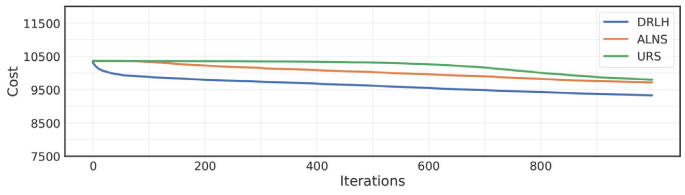
In addition to DRLH having a simpler development phase, an increased (or more nuanced) set of heuristics also has a larger potential to work well for a wide range of conditions, such as for different problems, instance sizes and specific situations encountered in the search. In other words, reducing the set of heuristics could negatively affect the performance of ALNS, but much less so for DRLH. Some heuristics work well only in specific situations, and so removing these "specialized" heuristics due to their poor average performance gives less potential for ALNS to be able to handle a diverse set of problem and instance variations compared to DRLH, which learns to take advantage of any heuristic that performs well in specific situations. Of course, these claims are based on a limited number of experiments and should be validated in a broad range of (future) experiments.

6.5. Average performance results

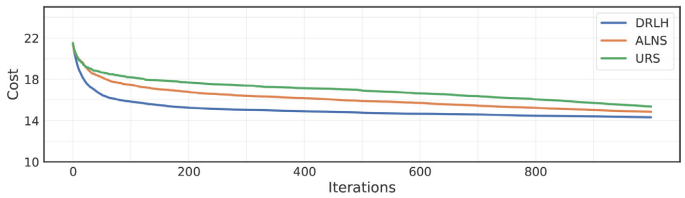
In this section, we explore the speed and characteristics of the performance of DRLH, ALNS and URS on the different problems. Fig. 4 shows that DRLH is able to quickly find better solutions compared to ALNS and URS for all the problems. Although for CVRP, DRLH takes a little bit longer initially, but ultimately reaches a much lower average minimum cost before the convergence of all three methods start to stagnate. For all the problems, DRLH is able



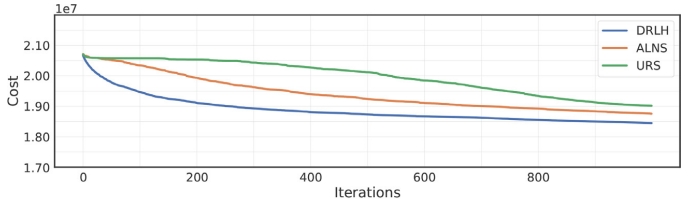
(a) CVRP, 500, 1k



(b) PJSP, 500, 1k



(c) PDP, 100, 1k



(d) PDPTW, 300, 1k

Fig. 4. Average performance of DRLH, ALNS and URS on each of the problems.

to reach a better cost after less than 500 iterations than what ALNS is able to reach after 1000 iterations. With the exception of the CVRP problem, DRLH is also extremely efficient in the beginning of the search, reaching costs in only 100 iterations that takes ALNS approximately 500 iterations to match. We refer to Appendix C for a complete collection of performance plots for all the problems that we have tested.

6.6. Training and inference time needed for each problem

Tables 9 and 10 report the time needed for training and solving for the instances of each problem, respectively. The main difference between DRLH and the baselines is the approach to decision making when it comes to choosing the next heuristic. This decision making process, on itself, does not add much overhead on the computational time of the methods. The main difference in the speed of these methods is the speed of the operators that they choose. This means that in some cases DRLH chooses operators that are faster or slower compared to baseline which results in lower or higher computational time. Therefore, when it comes to computational time, there is not much difference between these methods. This can also be shown in Table 10, in which in some cases DRLH is faster than the other two baselines and in some cases it is slower. It should be noted that the execution time of the operators can be improved if implemented carefully or using a faster programming language, e.g., C. However, the main focus of the paper is to improve the hyperheuristic approach of choosing the next heuristic at each step, not the execution time.

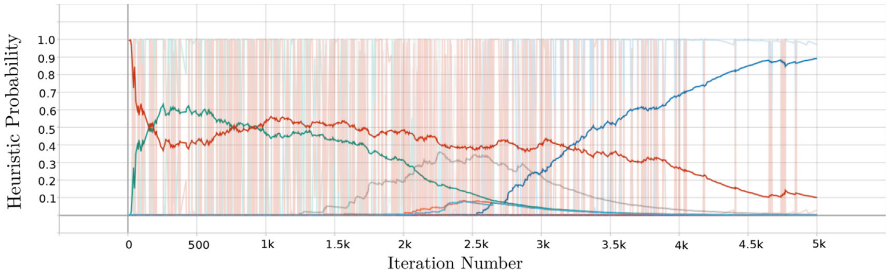
6.7. Comparison between heuristic selection strategies

Figure 5 demonstrate the probability of selecting heuristics at each step of the search for DRLH and ALNS in which each line corresponds to the probability of one heuristic at every step of the search. The “micro-level” heuristic usage of DRLH means that DRLH is able to drastically change the probabilities of selecting heuristics from one iteration to the next by taking advantage of the information provided by the search state, see Fig. 5(a) and (b). This is in contrast to the “macro-level” heuristic usage of ALNS where the probabilities of selecting operators only are updated at the beginning of each segment, meaning that the decision

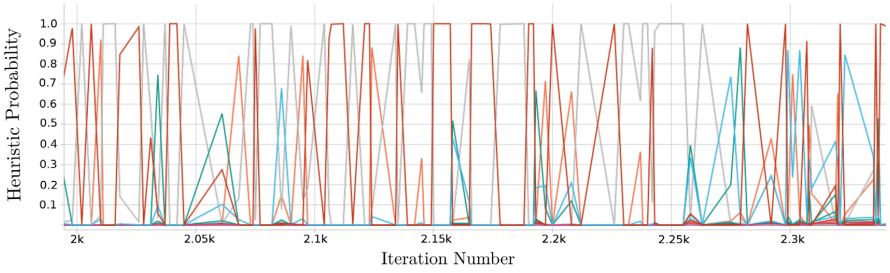
Table 9
Training time for DRLH on different problems.

Problem	Size	#Iterations	#Training Instances	Total training time (s)	Average time per instance (s)
CVRP	20	1k	1000	4586.85	4.59
CVRP	50	1k	1000	12394.3	12.39
CVRP	100	1k	1000	36330.0	36.33
CVRP	200	1k	100	8618.64	86.19
CVRP	500	1k	50	26483.2	529.66
PJSP	20	1k	1000	28233.7	28.23
PJSP	50	1k	1000	35552.1	35.55
PJSP	100	1k	500	16576.8	33.15
PJSP	300	1k	100	19758.1	197.58
PJSP	500	1k	100	79975.3	799.75
PDP	20	1k	500	1868.66	3.74
PDP	50	1k	100	2160.65	21.61
PDP	100	1k	100	12875.3	128.75
PDPTW	18	1k	600	25340.2	42.23
PDPTW	35	1k	600	12154.9	20.26
PDPTW	80	1k	500	20704.4	41.41
PDPTW	130	1k	100	8595.9	85.96
PDPTW	300	1k	90	53657.5	596.19

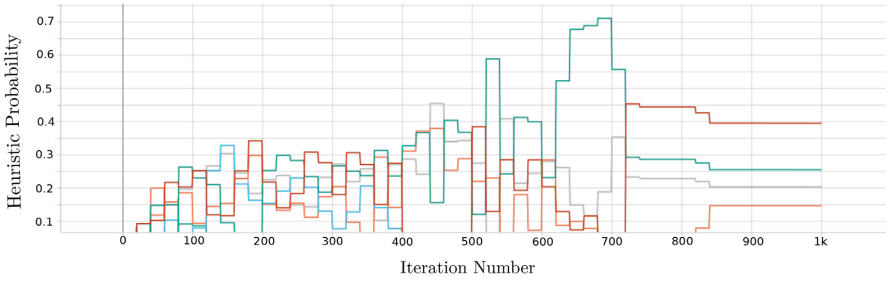
Deep Reinforcement Learning Hyperheuristic



(a) Smoothed probabilities of selecting heuristics for DRLH.



(b) Actual probabilities of selecting heuristics for DRLH, zoomed in between iteration 2000-2300.



(c) Actual probabilities of selecting heuristics for ALNS.

Fig. 5. Example of the probability of selecting heuristics for DRLH and ALNS.

Table 10
Average Time (seconds) required for solving the test instances for each method DRLH, ALNS and URS.

Problem	Size	#Iterations	DRLH	ALNS	URS
CVRP	20	1k	4.08	11.58	7.75
CVRP	50	1k	11.58	35.17	23.52
CVRP	100	1k	34.28	99.58	50.65
CVRP	200	1k	102.76	221.22	94.07
CVRP	500	1k	621.74	664.54	238.86
PJSP	20	1k	20.37	18.06	5.65
PJSP	50	1k	30.69	41.84	15.9
PJSP	100	1k	57.05	76.15	34.0
PJSP	300	1k	199.37	237.58	110.81
PJSP	500	1k	453.92	462.34	195.67
PDP	20	1k	3.89	4.17	1.85
PDP	50	1k	31.4	20.15	9.93
PDP	100	1k	159.86	79.58	45.86
PDPTW	18	1k	32.61	23.85	9.18
PDPTW	35	1k	10.67	29.75	21.18
PDPTW	80	1k	34.82	71.27	68.33
PDPTW	130	1k	110.67	139.45	132.32
PDPTW	300	1k	500.9	438.65	361.39

making of ALNS within a single segment is random according to the locked probabilities for that segment, see Fig. 5(c). Depending on the problem and available heuristics to select, there might exist exploitable strategies and patterns for heuristic selection, such as heuristic(s) that: work well when used together, work well for escaping local minima, work well on solutions not previously encountered during the search. Using DRLH, these types of exploitable strategies can be automatically discovered without the need for specially tailored algorithms designed by human experts. We refer to one such exploitable strategy found by DRLH on our problems with our provided set of heuristics as minimizing "wasted actions". We define a wasted action as the selection of a deterministic heuristic (in our case *Find_single_best*) for two consecutive unsuccessful iterations. The reason that this action is "wasted" is because of the deterministic nature of the heuristic, which makes it so that if the solution did not change in the previous iteration, then it is guaranteed not to change in the following iteration as well. Even though we have not specifically programmed DRLH to utilize this strategy, it becomes clear by examining Table 11 that the DRLH has picked up on this strategy when learning to optimize micro-level heuristic selection. Table 11 shows that the number of wasted actions for DRLH is almost non-existent for most problem variations. ALNS on the other hand ends up with far more wasted actions than DRLH, even though ALNS also uses *Find_single_best* much more seldom on average. Figure 5(c) shows how the heuristic probabilities for ALNS remain locked within the segments, making it impossible for ALNS to exploit strategies such as minimizing wasted actions which relies on excellent micro-level heuristic selection such as what DRLH demonstrates.

6.8. Performance comparison with AM deep RL heuristic

For this experiment, we ran the AM method of Kool et al. (2019) on our test datasets for the CVRP problem. The trained models and the implementation of the models needed to solve the problem have been provided publicly by the authors of this paper. The dataset generation procedure for both our work and the AM paper follow the work of Nazari et al. (2018). As a result, the models are well fit to be evaluated on our test set. For their method we considered three different approaches : *Greedy, Sample_128* and *Sample_1280*. In the greedy approach, at each step the node with the most probability is chosen. In the sampling approach, 128 and 1280 different solutions are sampled based on the probability of each node at each step. We test these methods for sizes $n = 20, 50, 100$ of the CVRP problem. The time and resources (Graphical Processing Units) needed to train the AM method for sizes larger than 100 scales exponentially due to heavy calculations needed for their representation learning method. Therefore, we only solve this problem for the mentioned instance sizes.

Figure 6 illustrates the comparison of performance of our method with the AM method of Kool et al. (2019). As shown in

Table 11
The percentage of wasted actions of the total number of deterministic heuristics selected, averaged over the test set for each problem.

(a) CVRP

		Wasted Actions (%)	
#Orders	#Iterations	DRLH	ALNS
20	1k	3.37	26.55
50	1k	0.00	23.98
100	1k	1.22	19.48
200	1k	0.00	23.43
500	1k	0.01	25.15

(b) PJSP

		Wasted Actions (%)	
#Jobs	#Iterations	DRLH	ALNS
20	1k	0.00	20.82
50	1k	0.86	24.57
100	1k	0.00	24.80
300	1k	0.00	24.85
500	1k	0.00	24.50

(c) PDP

		Wasted Actions (%)	
#Calls	#Iterations	DRLH	ALNS
20	1k	6.82	31.53
50	1k	0.00	29.00
100	1k	0.00	28.01
100	5k	0.02	30.62
100	10k	0.00	33.86
100	25k	0.00	32.69

(d) PDPTW

		Wasted Actions (%)	
#Calls	#Iterations	DRLH	ALNS
18	1k	0.00	21.68
35	1k	0.00	28.65
80	1k	0.00	24.50
130	1k	0.00	19.60
300	1k	0.00	17.90
18	5k	0.00	30.88
35	5k	0.00	36.26
80	5k	0.00	27.49
130	5k	0.00	26.98
300	5k	0.00	26.10
18	10k	0.25	37.82
35	10k	0.00	36.60
80	10k	0.00	32.41
130	10k	0.08	29.67
300	10k	0.00	26.10

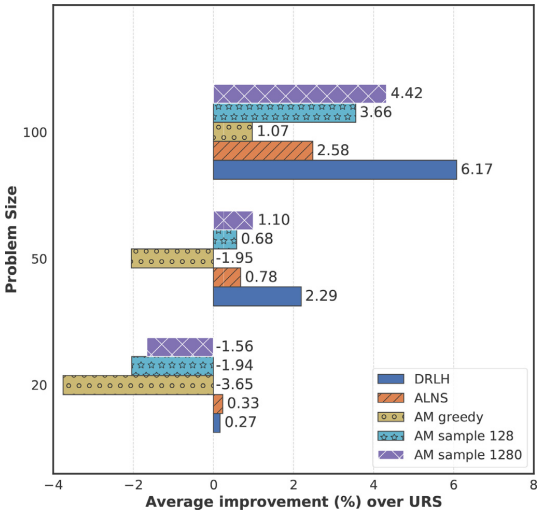


Fig. 6. Comparison of DRLH with the Deep RL method of Kool et al. (2019) (AM) on test instances of CVRP.

the figure, AM is not able to outperform the baseline of URS in the size 20 with any of the sampling methods. Regarding size 50 of the problem, in the greedy approach it still falls behind URS. However, given enough samples, AM manages to perform better than ALNS in some instances of the CVRP problem. On the other hand, our method of DRLH outperforms this approach in every single instance size as well as being able to handle different problems without any significant change in the code which is not the case for the method of Kool et al. (2019).

7. Concluding remarks

For quite some time now, it has increasingly become evident that the fields of machine learning and (heuristic) optimization can mutually benefit from an integration. On the one hand, recent advances in optimization can support the development of advanced machine learning methods, since these methods generally solve an optimization problem (e.g., what is the optimal subset of features from a data set that predict a certain outcome). This paper addressed the mirror issue: how can optimization approaches benefit from an integration of machine learning methods. We demonstrated that applying a well-known machine learning approach to the selection of low-level operators in a metaheuristic framework results in a robust mechanism that can be used to improve the performance of a heuristic on a broad range of optimization problems. We believe that approaches like the one presented in this paper have the potential to make the development of a powerful heuristic less dependent on the knowledge of an experienced developer with a deep insight into the structure of the specific problem being solved, and may therefore be instrumental in the integration of metaheuristics ideas into general purpose software packages. Our proposed DRLH, a general framework for solving combinatorial optimization problems, utilizes a trained Deep RL agent to select low-level heuristics to be applied on the solution in each iteration of the search based on a search state consisting of features from the search process. In our experiments, we solved four combinatorial optimization problems (CVRP, PJSP, PDP, and PDPTW) using our proposed approach and compared its performance with the baselines of ALNS and URS. Our results show that DRLH is able to select heuristics in a way that achieves better results in less number of iterations for almost all of the problem variations compared to ALNS and URS. Furthermore, the performance gap between DRLH and the baselines is shown to increase for larger problem sizes, making DRLH a suitable option for large real-world problem instances. Additional experiments on an extended set of heuristics show that DRLH is not negatively affected when selecting from a large set of available heuristics, while the performance of ALNS is much worse in this situation. Enriching or refining the state representation with additional information is possible with very little effort. We have experimented with adding problem-dependent information into the state representation and seen that this gives even better results than sticking with the simple chosen state representation. Yet once we start to introduce problem-dependent structure and constraint information into the state representation we lose some of the generality that we strive for with DRLH as we would have to separately engineer a different state representation for each new problem. For this reason we deem this outside of the scope of this paper and leave this area open for future work.

Future research should provide more empirical evidence for the superiority of DRLH over ALNS by applying this novel hyperheuristic to different problems. A potential direction for improving the model in the future is designing a reward function that is both stable and takes into account the difference of objective value at each iteration of the search. Initial experiments on alternative reward functions have shown promising results (see Appendix A), but are time-consuming to train and not very stable compared to the R^{5310} reward function that we have used in this paper.

Appendix A. Experiments on different reward functions

A1. R_t^{PM}

$$R_t^{PM} = \begin{cases} 1, & \text{if } f(x') < f(x) \\ -1, & \text{otherwise} \end{cases} \tag{A.1}$$

The R_t^{PM} reward function focuses more heavily on intensification by punishing any action choice that does not directly improve upon the current solution. This causes the agent to favor intensifying heuristics more strongly than R_t^{5310}. However, because the PPO framework leverages the *discounted future rewards* as opposed to only the immediate reward for training the agent, even the R_t^{PM} can cause the agent to select heuristics with a high likelihood of immediate negative reward if it sets it up for more positive rewards in future iterations.

Figure A.1 illustrates the distribution of minimum costs found on the PDP of size 100 test set after 1000 and 10,000 iterations for two different versions of DRLH, trained with reward functions R_t^{5310} and R_t^{PM} respectively. The model trained with R_t^{5310} achieves a lower median and quantile values for both iteration variations, compared to the model trained with R_t^{PM}. This makes the R_t^{5310} reward function more reliable to perform relatively better, and we therefore decided to use the R_t^{5310} reward function in this paper.

Table A.1
Average results for PDPTW instances with mixed call sizes after 1000 iterations.

		DRLH with R_t^{5310}		DRLH with R_t^{MC}	
#C	#V	Min Gap (%)	Avg Gap (%)	Min Gap (%)	Avg Gap (%)
18	5	0.00	0.18	**0.00**	**0.11**
35	7	2.67	5.78	**1.48**	**3.65**
80	20	**3.04**	4.85	3.15	**4.39**
130	40	3.44	4.66	**2.99**	**4.33**
300	100	2.40	3.15	**2.28**	**3.00**

Table A.2
Average results for PDPTW instances with mixed call sizes after 10,000 iterations.

		DRLH with R_t^{5310}		DRLH with R_t^{MC}	
#C	#V	Min Gap (%)	Avg Gap (%)	Min Gap (%)	Avg Gap (%)
18	5	**0.00**	**0.00**	0.00	0.13
35	7	0.67	**2.02**	**0.42**	2.32
80	20	**1.80**	**2.95**	2.55	3.87
130	40	**1.93**	**2.84**	2.20	3.04
300	100	**0.00**	**0.64**	1.12	1.88

A2. R_t^{MC}

$$R_t^{MC} = \begin{cases} \dfrac{f(x_{best}) - f(x')}{f(x_{best})} \end{cases} \tag{A.2}$$

The R_t^{MC} is a reward function that more directly correlates with the intended objective of minimizing the cost of the best found solution, and to achieve this as quickly as possible. Instead of focusing on rewarding actions that directly improve the solution, this reward function is subject to the performance of the entire search process up to the current step, putting a greater emphasis on acting quickly and selecting heuristics that have a greater impact on the solution. The challenge with using this reward function compared to reward functions such as R_t^{5310} and R_t^{PM} is that there is an inherent delay between when a good heuristic is selected and when the reward function gives a good reward. This makes it more difficult to train an agent using this reward function, making training times much longer and less stable than with the R_t^{5310} reward function.

Having said that, the potential upside of using this reward function is very promising, and results in Table A.1 show that R_t^{MC} is able to outperform the R_t^{5310} reward function on 1k iteration searches. However, the agents were unable to learn effectively for larger number of iterations such as 10k (Table A.2), and so results for this shows that R_t^{MC} performs worse than R_t^{5310} on 10k iteration searches. A potential reason for why the R_t^{MC} agents were unable to learn well on 10k iteration searches is that the amount of improving iterations are much less frequent, making the feedback signal from the R_t^{MC} reward function even more delayed and high variance. Another potential reason is that the training required in order to solve 10k iteration searches likely needed more training than what was possible to carry out for our experiments due to time constraints with the experiments. We encourage future work on improving the integration of the R_t^{MC} reward function into the framework of DRLH as it likely has a lot of potential.

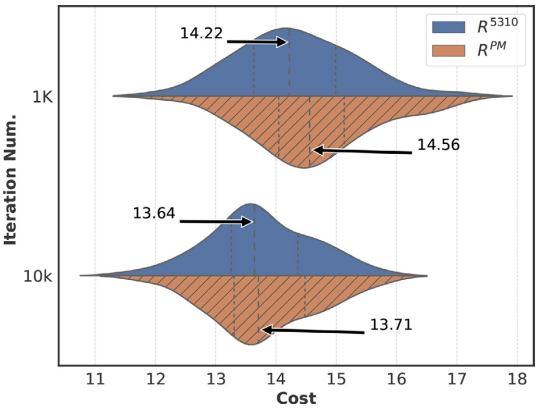


Fig. 7. Comparison of the two reward functions.

Appendix B. Extended set of heuristics

Tables A.3, A.4 and A.5 list the extended set of heuristics built up from 14 removal operators, 10 insertion operators and 2 additional heuristics, for a total of $14 \times 10 + 2 = 142$ total heuristics, using the generation scheme of Algorithm 2. Most of these heuristics only use problem-independent information, but some of them rely on problem-dependent information specific to the PDPTW problem.

Table A.3
List of extended removal operators.

Name	Description
Random_remove_XS	Removes between 2–5 elements chosen randomly
Random_remove_S	Removes between 5–10 elements chosen randomly
Random_remove_M	Removes between 10–20 elements chosen randomly
Random_remove_L	Removes between 20–30 elements chosen randomly
Random_remove_XL	Removes between 30–40 elements chosen randomly
Random_remove_XXL	Removes between 80–100 elements chosen randomly
Remove_largest_D_S	Removes 5–10 elements with the largest $\mathcal{D}_i$
Remove_largest_D_L	Removes 20–30 elements with the largest $\mathcal{D}_i$
Remove_τ	Removes a random segment of 2–5 consecutive elements in the solution
Remove_least_frequent_S	Removes between 5–10 elements that has been removed the least
Remove_least_frequent_M	Removes between 10–20 elements that has been removed the least
Remove_least_frequent_XL	
Remove_one_vehicle	Removes all the elements in one vehicle
Remove_two_vehicles	Removes all the elements in two vehicle

Table A.4
List of extended insertion operators.

Name	Description
Insert_greedy	Inserts each element in the best possible position
Insert_beam_search	Inserts each element in the best position using beam search
Insert_by_variance	Sorts the insertion order based on variance and inserts each element in the best possible position
Insert_first	Inserts each element randomly in the first feasible position
Insert_least_loaded_vehicle	Inserts each element into the least loaded available vehicle
Insert_least_active_vehicle	Inserts each element into the least active available vehicle
Insert_close_vehicle	Inserts each element into the closest available vehicle
Insert_group	Identifies the vehicles that can fit the most of the removed elements and inserts each elements into these
Insert_by_difficulty	Inserts each element using *Insert_greedy* ordered by their difficulty, which is a function of their compatibility with vehicles, strictness of time windows,size and more.
Insert_best_fit	Inserts each element into the vehicle that is the most compatible with the call.

Table A.5
List of extended additional heuristics.

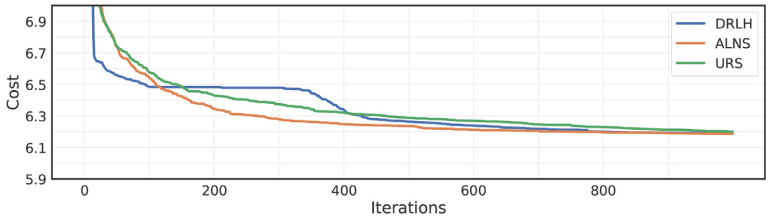
Name	Description
Find_single_best	Calculates the cost of removing each element and re-inserting it with *Insert_greedy*, and applies this procedure on the solution x for the element that achieves the minimum cost $f(x')$.
Rearrange_vehicles	Removes all of the elements from each vehicle and inserts them back into the same vehicles using *Insert_beam_search*

Appendix C. Additional performance plots

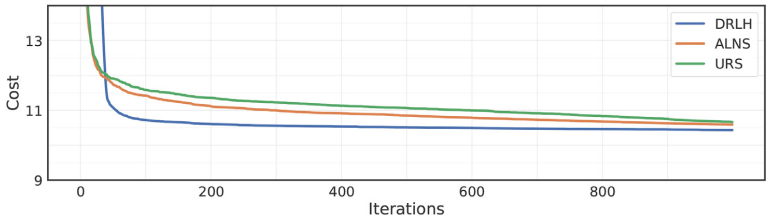
Figures 8, 9, 10 and 11 show the performance of DRLH, ALNS and URS averaged over the test set for all the problems that we have tested. These show that DRLH usually reaches better solutions more quickly than ALNS and URS, as well as ending up with better solutions overall.

Appendix D. Experiment on the cross size training scheme

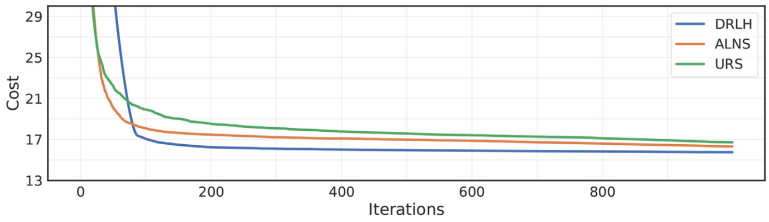
In this experiment, in the training phase, an instance of a specific problem with different size is solved by DRLH in each episode. This training scheme is referred to as Cross Size (CS) training. During test time, the trained model solved the test instances that were used in Section 6.1 as well as test instances of slightly different sizes that were seen during training. As seen in Fig. 12, it is possible to train one model that can handle many different variations of instance sizes quite well. Moreover, as shown in Fig. 12(e)–(h), the model does not specifically overfit on the specific instance sizes included in the training when evaluated on slightly different test data. This means that the DRLH_CS generalizes very well, even to sizes higher than any of the ones included in the training.



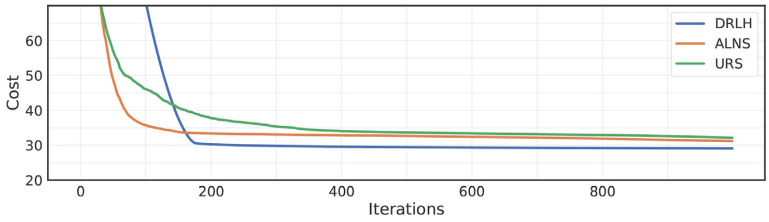
(a) CVRP, 20, 1k



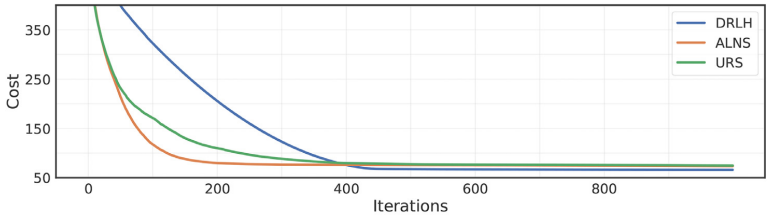
(b) CVRP, 50, 1k



(c) CVRP, 100, 1k

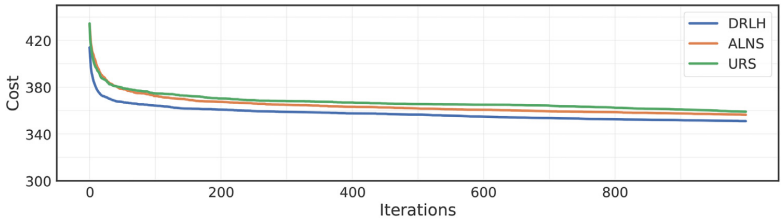


(d) CVRP, 200, 1k

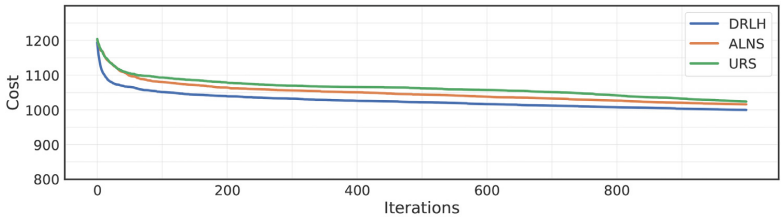


(e) CVRP, 500, 1k

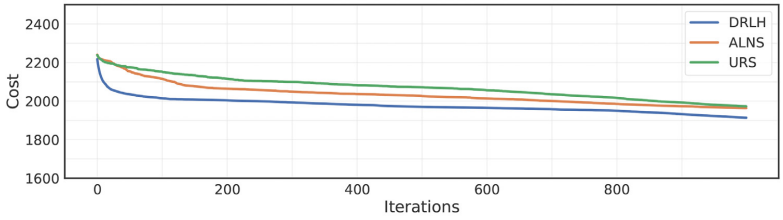
Fig. 8. Average performance of DRLH, ALNS and URS on CVRP.



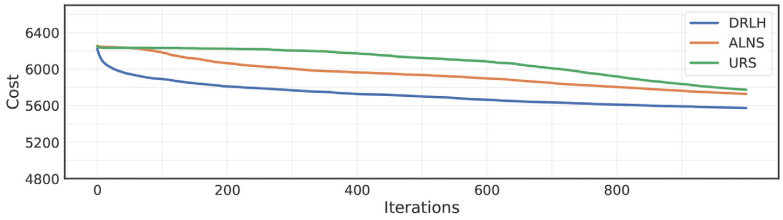
(a) PJSP, 20, 1k



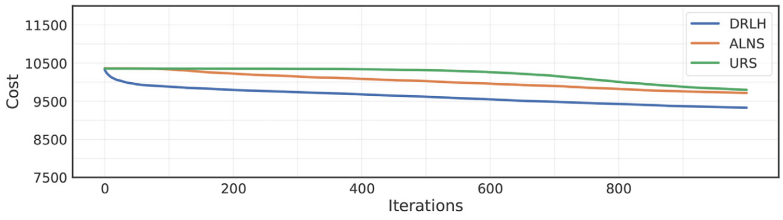
(b) PJSP, 50, 1k



(c) PJSP, 100, 1k

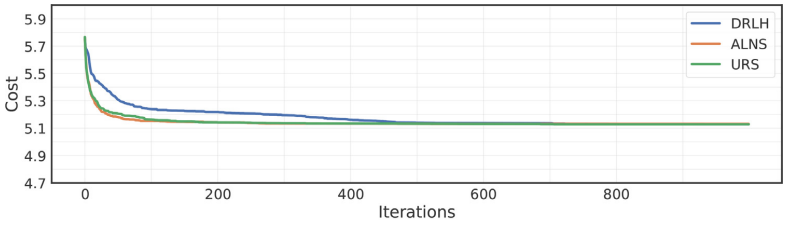


(d) PJSP, 300, 1k

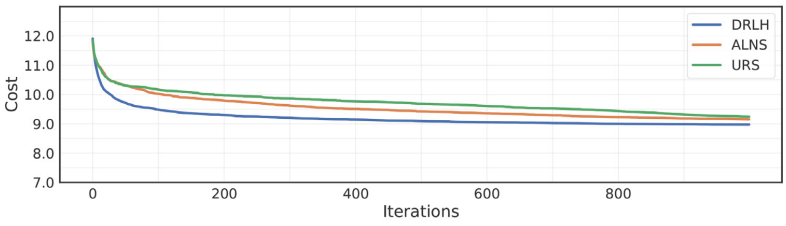


(e) PJSP, 500, 1k

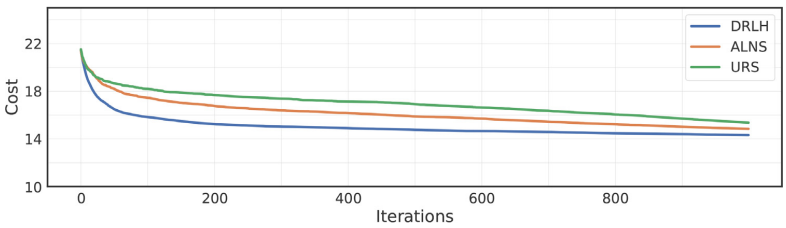
Fig. 9. Average performance of DRLH, ALNS and URS on PJSP.



(a) PDP, 20, 1k

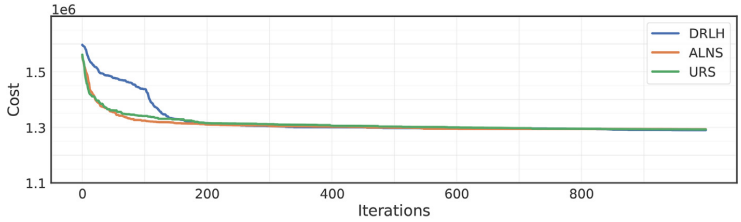


(b) PDP, 50, 1k

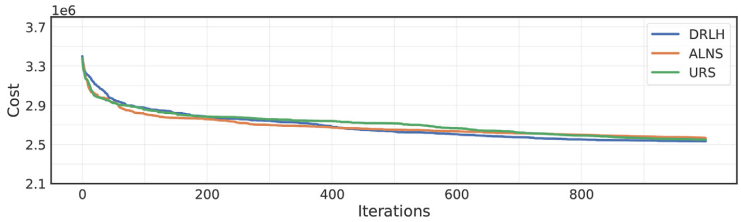


(c) PDP, 100, 1k

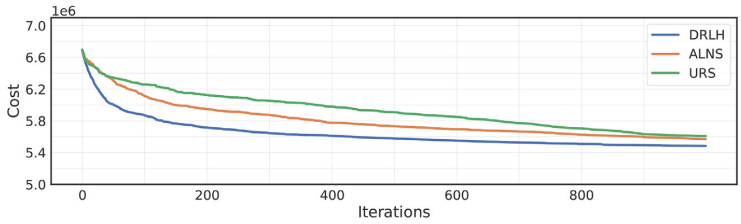
Fig. 10. Average performance of DRLH, ALNS and URS on PDP.



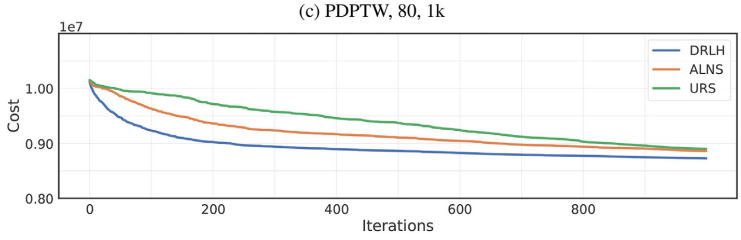
(a) PDPTW, 18, 1k



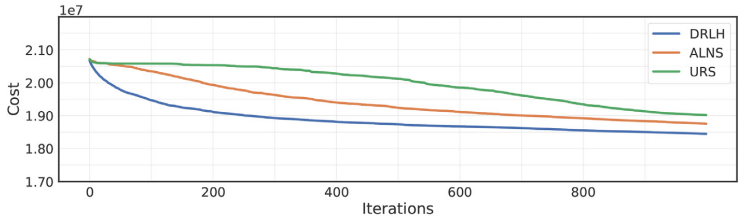
(b) PDPTW, 35, 1k



(c) PDPTW, 80, 1k

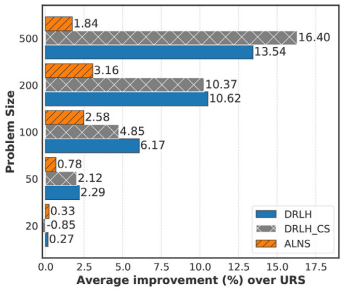


(d) PDPTW, 130, 1k

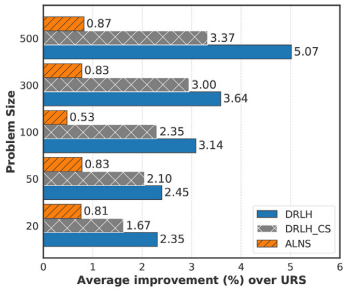


(e) PDPTW, 300, 1k

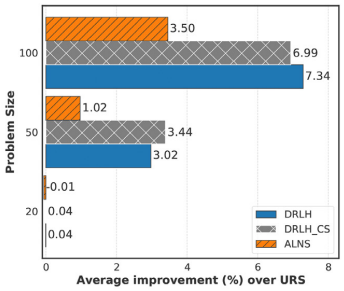
Fig. 11. Average performance of DRLH, ALNS and URS on PDPTW.



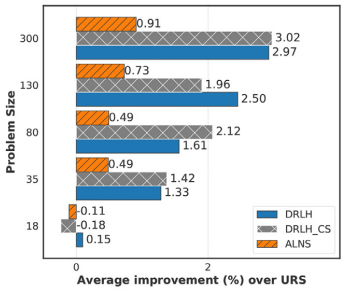
(a) CVRP results, same sizes as training



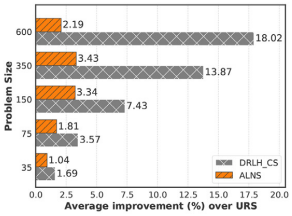
(b) PJSP results, same sizes as training



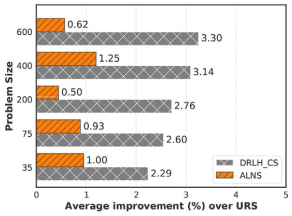
(c) PDP results, same sizes as training



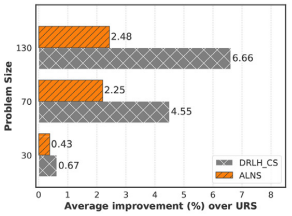
(d) PDPTW results, same sizes as training



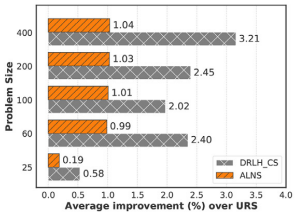
(e) CVRP results, unseen sizes in training



(f) PJSP results, unseen sizes in training



(g) PDP results, unseen sizes in training



(h) PDPTW results, unseen sizes in training

Fig. 12. Performance of DRLH with Cross Size (CS) training scheme on different problem sizes with 1k iterations.

References

Aksen, D., Kaya, O., Sibel Salman, F., & Özge Tüncel (2014). An adaptive large neighborhood search algorithm for a selective and periodic inventory routing problem. *European Journal of Operational Research, 239*(2), 413–426. https://doi.org/10.1016/j.ejor.2014.05.043.

Amodei, D., Olah, C., Steinhardt, J., Christiano, P. F., Schulman, J., & Mané, D. (2016). Concrete problems in AI safety. *CoRR*. http://arxiv.org/abs/1606.06565.

Asta, S., & Özcan, E. (2014). An apprenticeship learning hyper-heuristic for vehicle routing in hyflex. Orlando, Florida. https://doi.org/10.1109/EALS.2014.7009505.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Woodward, J. R. (2010). *A classification of hyper-heuristic approaches*. In M. Gendreau, & J.-Y. Potvin (Eds.) (pp. 449–468). Boston, MA: Springer US.

Chen, C., Demir, E., & Huang, Y. (2021). An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and delivery robots. *European Journal of Operational Research, 294*(3), 1164–1180. https://doi.org/10.1016/j.ejor.2021.02.027.

Chen, X., & Tian, Y. (2019). Learning to perform local rewriting for combinatorial optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems: vol. 32*. Curran Associates, Inc.. https://proceedings.neurips.cc/paper/2019/file/131f383b434fdf48079bff1e44e2d9a5-Paper.pdf

Cowling, P., Kendall, G., & Soubeiga, E. (2001). A hyperheuristic approach to scheduling a sales summit. In E. Burke, & W. Erben (Eds.), *Practice and theory of automated timetabling iii* (pp. 176–190). Berlin, Heidelberg: Springer Berlin Heidelberg.

Crama, Y., & Schyns, M. (2003). Simulated annealing for complex portfolio selection problems. *European Journal of Operational Research, 150*(3), 546–571. https://doi.org/10.1016/S0377-2217(02)00784-1. Financial Modelling

Demir, E., Bektaş, T., & Laporte, G. (2012). An adaptive large neighborhood search heuristic for the pollution-routing problem. *European Journal of Operational Research, 223*(2), 346–359. https://doi.org/10.1016/j.ejor.2012.06.044.

Dokeroglu, T., Sevinc, E., Kucukyilmaz, T., & Cosar, A. (2019). A survey on new generation metaheuristic algorithms. *Computers & Industrial Engineering, 137*, 106040. https://doi.org/10.1016/j.cie.2019.106040.

Friedrich, C., & Elbert, R. (2022). Adaptive large neighborhood search for vehicle routing problems with transshipment facilities arising in city logistics. *Computers & Operations Research, 137*, 105491. https://doi.org/10.1016/j.cor.2021.105491.

Goodfellow, I. J., Bengio, Y., & Courville, A. (2016). *Deep learning*. Cambridge, MA, USA: MIT Press. http://www.deeplearningbook.org

Grangier, P., Gendreau, M., Lehuédé, F., & Rousseau, L.-M. (2016). An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. *European Journal of Operational Research, 254*(1), 80–91. https://doi.org/10.1016/j.ejor.2016.03.040.

Gullhav, A. N., Cordeau, J.-F., Hvattum, L. M., & Nygreen, B. (2017). Adaptive large neighborhood search heuristics for multi-tier service deployment problems in clouds. *European Journal of Operational Research, 259*(3), 829–846. https://doi.org/10.1016/j.ejor.2016.11.003.

Hemmati, A., & Hvattum, L. M. (2017). Evaluating the importance of randomization in adaptive large neighborhood search. *International Transactions in Operational Research, 24*(5), 929–942. https://doi.org/10.1111/itor.12273.

Hemmati, A., Hvattum, L. M., Fagerholt, K., & Norstad, I. (2014). Benchmark suite for industrial and tramp ship routing and scheduling problems. *INFOR: Information Systems and Operational Research, 52*(1), 28–38. https://doi.org/10.3138/infor.52.1.28.

Homsi, G., Martinelli, R., Vidal, T., & Fagerholt, K. (2020). Industrial and tramp ship routing problems: Closing the gap for real-scale instances. *European Journal of Operational Research, 283*(3), 972–990. https://doi.org/10.1016/j.ejor.2019.11.068.

Hottung, A., & Tierney, K. (2019). Neural large neighborhood search for the capacitated vehicle routing problem. *CoRR*. http://arxiv.org/abs/1911.09539.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680. https://doi.org/10.1126/science.220.4598.671.

Kool, W., van Hoof, H., & Welling, M. (2019). Attention, learn to solve routing problems!

Laborie, P., & Godard, D. (2007). Self-adapting large neighborhood search: Application to single-mode scheduling problems. In *Proceedings MISTA-07: Vol. 8*. Paris

Li, Y., Chen, H., & Prins, C. (2016). Adaptive large neighborhood search for the pickup and delivery problem with time windows, profits, and reserved requests. *European Journal of Operational Research, 252*(1), 27–38. https://doi.org/10.1016/j.ejor.2015.12.032.

Lu, H., Zhang, X., & Yang, S. (2020). A learning-based iterative method for solving vehicle routing problems. In *International conference on learning representations*. https://openreview.net/forum?id=BJe1334YDH

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives, 3*, 43–58. https://doi.org/10.1016/j.orp.2016.09.002.

Nazari, M., Oroojlooy, A., Snyder, L., & Takac, M. (2018). Reinforcement learning for solving the vehicle routing problem. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems: vol. 31*. Curran Associates, Inc.. https://proceedings.neurips.cc/paper/2018/file/9fb4651c05b2ed70fba5afe0b039a550-Paper.pdf

Özcan, E., Misir, M., Ochoa, G., & Burke, E. (2010). A reinforcement learning - great-deluge hyper-heuristic for examination timetabling. *International Journal of Applied Metaheuristic Computing, 1*, 39–59.

Pisinger, D., & Ropke, S. (2019). Large neighborhood search. In *Handbook of metaheuristics* (pp. 99–127). Springer.

Ropke, S., & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science, 40*(4), 455–472. https://doi.org/10.1287/trsc.1050.0135.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*. abs/1707.06347. http://dblp.uni-trier.de/db/journals/corr/corr1707.html#SchulmanWDRK17

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher, & J.-F. Puget (Eds.), *Principles and practice of constraint programming — CP98* (pp. 417–431). Berlin, Heidelberg: Springer Berlin Heidelberg.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. Cambridge, MA, USA: A Bradford Book.

Turkeš, R., Sörensen, K., & Hvattum, L. M. (2021). Meta-analysis of metaheuristics: Quantifying the effect of adaptiveness in adaptive large neighborhood search. *European Journal of Operational Research, 292*(2), 423–442. https://doi.org/10.1016/j.ejor.2020.10.045.

Tyasnurita, R., Özcan, E., Shahriar, A., & John, R. (2015). *Improving performance of a hyper-heuristic using a multilayer perceptron for vehicle routing*. Exeter, UK, http://eprints.nottingham.ac.uk/id/eprint/45707

Graphic design: Communication Division, UiB / Print: Skipnes Kommunikasjon AS

uib.no

ISBN: 9788230865255 (print)
9788230851630 (PDF)