From Algorithm Selection to Generation using Deep Learning

Mohamad Alissa

School of Computing
Edinburgh Napier University

A thesis submitted in partial fulfilment of the requirements of Edinburgh Napier University, for the award of
Doctor of Philosophy.

January 2022

To my lovely family ... my parents, my sister and my brothers

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

- External examiner Prof. Jim Smith
- Internal examiner Dr. Dimitra Gkatzia
- Director of studies Dr. Kevin Sim
- Second supervisor Prof. Emma Hart

Mohamad Alissa
January 2022

Acknowledgements

My thanks to Doctor Kevin Sim and Professor Emma Hart for their guidance and help throughout the period of study culminating in this thesis. Thanks to Dr Neil Urquhart and Professor Ben Paechter who acted as 2nd and 3rd supervisors at the start of the PhD. Also thanks to Doctors Michael Lones and Marta Vallejo for proof reading the abstract, introduction and conclusion chapters of the thesis. To my parents, my sister and my brothers my thanks for their encouragement, support, patience and understanding over the past years.

Abstract

Algorithm selection and generation techniques are two methods that can be used to exploit the performance complementarity of different algorithms when applied to large diverse sets of combinatorial problem instances. As there is no single algorithm that dominates all others on all problem instances, *algorithm selection* automatically selects an algorithm expected to perform best for each problem instance. Meanwhile, *algorithm generation* refers to combining different algorithms in a manner that allows the resulting method to improve the efficacy of a pool of algorithms.

This thesis examines algorithm selection and generation within a single streaming problem domain, that is Bin-Packing, where novel approaches are proposed and evaluated on large problem sets. This research starts with presenting a novel feature-free approach to select the best performing heuristic by capturing the sequential information implicit in a streaming instance and using this as direct input to two Deep Learning (DL) models, Long-Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU), to learn a mapping from an instance to an algorithm. Results obtained using the proposed approach show that the performance of the feature-free selectors significantly outperforms the performance of both the single best solver and the classical feature-based approach using well-known Machine Learning (ML) classifiers when applied to large sets of diverse problem instances. Next, a more radical approach is proposed: bypass algorithm selection altogether by training encoder-decoder LSTM using solutions obtained from a set of algorithms to directly predict a solution from the instance data behaving as an automatically generated algorithm. Experiments conducted on large datasets using problem batches of varying sizes show that the generated algorithm is able to accurately predict solutions, particularly with small batch sizes. Finally, the thesis develops the proposed encoder-decoder approach by introducing a novel neural approach for generating algorithms, in which a neural network acts as an algorithm by generating decisions. Two architectures are evaluated, an encoder-decoder LSTM and a feed-forward Neural Network (NN), and trained using the decisions output from existing algorithms on a large set of instances. Experiments show that the new generated algorithms are capable of solving a subset of instances better than the well-known bin-packing algorithms, and hence they can significantly improve the overall performance when they are added to a pool of algorithms.

Table of contents

List of Figures

List of Tables

Chapter 1

Introduction

“Life is a matter of choices, and every choice you make makes you.”
—John C. Maxwell
"Remember you always have the choice to create your own choice"
—Mohamad Alissa

While most aspects of human life involve decisions-making, (naturally) humans aim for the best choices in daily life tasks, e.g. what is the best movie to watch tonight?, what is the most enjoyable book to read?, what food to eat at lunch?, how to reach a specific destination with less time and effort . . . etc. Having only one option or a clear preference makes it easier to choose but having many choices tends to be difficult, especially when there are bad options. Also, these best choices can not be best always as the best choice now (given different aspects such as the mood, weather, time of the day . . . etc) might be the worse choice in a different situation. Friends (who should be experts in the related situation) can help to either *select* what is the "best" choice among others or *generate* a completely new choice that might outperform the others. However, experts might not always be available.

Computer science is no different in this respect, and it is an important part of these human life aspects, such as what are called "Algorithm Selection" and "Algorithm Generation". These terms address similar kinds of decisions where the choice here is an algorithm to be selected or generated. The Algorithm Selection Problem (ASP) was introduced firstly by Rice, an American mathematician and computer scientist [1], over forty years ago. ASP has many applications, such as meta-learning (or "learning-to-learn") [2, 3]. The Algorithm Generation Problem (AGP) can be dated back to the 1940s [4]. It has many applications, such as the automatic generation of computer programs like intelligent agent programs [5] and music programs [6]. Both are considered effective ways to solve many hard problems including Combinatorial Optimization Problems (COPs) [7–9].

[Algorithm Selection and Generation]

Since different algorithms behave differently on different problem instances, no single algorithm dominates all others on all problem instances (i.e. performance complementarity phenomena). Given a set of problem instances and algorithms, algorithm selection informally can be defined as the task to select the algorithm with the best behaviour for each problem instance given a set of features extracted from the instances. However, these features are often not intuitive and are hard to be derived. In this sense, the expert presence is required to design and extract a set of relevant features correlated with the algorithm's performance. This often requires post-generation analysis to select the best set of features.

Algorithm selection relies on a pool of algorithms to address a COP. Improving the pool with new algorithms can improve the overall performance, one way to add new algorithms to the pool is by designing them. However, manually designing algorithms (i.e. with the presence of an expert) is hard, tedious and time-consuming. One way to avoid having to hand-craft an algorithm is by automatically generating it from a pool of hand-designed algorithms (exploiting the performance complementarity phenomena), often with using features extracted from problem instances. In this sense, algorithm selection and generation have suffered from extra tasks (burdens), including the need to define features to solve an optimisation problem.

[Streaming Optimisation Problems]

In addition to the issues associated with feature-deriving to build an algorithm selector or to generate an algorithm automatically, a further challenge arises concerning the need to deal with streaming data within combinatorial optimisation. Many practical applications (e.g. in scheduling and packing) fall into this category in which items/jobs arrive in a continual stream and should be packed/assigned as they arrive. In these potentially large streams, the order of streaming data points cannot be influenced and the underlying distribution of the data points in the stream can change over time. Due to these factors, the statistical approaches for defining features for streaming data are complex and challenging since these features should capture the sequential information contained in the stream to be informative. Furthermore, the streaming decisions are highly interdependent, e.g. the decision for packing/assigning an item/operation may influence the decision of other items/operations in packing and scheduling problems. Therefore, there is a need for algorithm selection and generation approaches that consider the historical data in making such decisions in streaming problems.

[Artificial Intelligence for Automated Algorithm Selection and Generation]

Research over the last decades focused on building feature-based algorithm selectors using Artificial Intelligence (AI) techniques such as machine learning. Similarly, algorithm generation research makes use of techniques such as genetic programming and other traditional machine learning approaches relying on domain-related hand-designed features as input. Most of these AI techniques do not consider the sequential information that is implicit in the streaming problem instances in domains such as bin-packing.

Given the recent advances of deep learning, in particular, Recurrent Neural Networks with Long-Short Term Memory (RNN-LSTM) [10] or Gated Recurrent Units (RNN-GRU) [11], in achieving groundbreaking results in applications where the input is formatted as time-series data or in domains where sequences have specific orderings but without any explicit notion of time and without defining features *a priori* [12]. Examples include video and image recognition, natural language processing, music generation and speech recognition [13–15]. The research presented in this thesis focuses on algorithm selection and generation to address the streaming version of an optimisation problem, namely bin-packing, using this sophisticated AI technique, deep learning.

This research is conducted within the realm of the bin-packing domain, specifically the streaming variant of the fixed capacity One-Dimensional Bin-Packing Problem (1D-BPP). BPP belongs to the broader class of cutting-stock and packing problems which occur frequently in real-world applications including container loading, job scheduling on multiple processors, distributing processing tasks in cloud computing and memory allocation [16]. This problem is proven to be an NP-hard problem [17, 18], i.e. it is often impossible to solve the problem instances in any reasonable time. Thus, often approximation algorithms are used to produce near-optimal solutions in a satisfactory time. In all the approaches proposed here, the underlying algorithm space is restricted to simple approximation algorithms (i.e. constructive deterministic heuristics), allowing for greater confidence when analysing the performance of the proposed approaches than could be inferred by using sets of stochastic heuristics. It is expected that the proposed approaches in this thesis could easily be transferred to other problem domains.

This thesis focuses on investigating the benefits to be gained by using deep learning for algorithm selection and generation to exploit the complementary performance of sets of deterministic heuristics. Both algorithm selection and generation approaches are presented and evaluated on large datasets of evolved and randomly generated problem instances. The evolved datasets are newly generated in which each instance has a distinct best-solver, and thus the pools of heuristics exhibit complementary performance.

The following section outlines the main research questions that have driven the research conducted and presented in this thesis.

1.1 Research Questions

This section presents the research questions that are investigated in this thesis. It should be noted that “to what extent” phrase implies an accuracy metric to predict the best performing algorithm or the generated solution, however other metrics can be used to measure the quality of the end-results, e.g. number of bins.

- **Question 1:** To what extent can sequential deep learning (RNN-LSTM/GRU) be used to build a feature-free algorithm selector that selects the best performing algorithm from a pool of well-known deterministic algorithms relying only on the sequential information implicit in the streaming problem instances?
- **Question 2:** To what extent can sequential deep learning feature-free algorithm selector outperform classical machine learning feature-based algorithm selectors to solve a streaming optimisation problem?
- **Question 3:** To what extent can deep learning (encoder-decoder LSTM) be used to generate algorithms automatically that can output solutions for streaming problem instances directly?
- **Question 4:** To what extent can a pool with neural generated algorithms be improved over a pool of human-designed algorithms to solve a streaming optimisation problem?

1.2 Thesis Contribution

This thesis introduces novel approaches in the fields of algorithm selection and generation using a single problem domain, that of bin-packing. The main contributions of this thesis are listed below:

- The introduction of four new sets of benchmark problem instances totalling 16,000 instances for the streaming version of 1D-BPP domain, described in Chapter 3. Each dataset has 4,000 instances, and contains 1,000 instances uniquely best solved by *exactly one* of four algorithms in a pool used in the algorithm selection problem. The datasets were generated using an Evolutionary Algorithm (EA), which maximises the performance difference between the target algorithm and the other algorithms used in the selection problem. Thus these datasets occupy different parts of the performance space w.r.t the considered heuristics that exhibit complementary performance.

- The introduction of a novel feature-free algorithm selection approach (described in Chapter 4) using deep learning RNN-LSTM [10] or Gated Recurrent Units (GRU) [11], that relies only on the sequential information inherent in a streaming problem instance to select the best performing algorithm.
- A novel application of deep learning, namely encoder-decoder LSTM [11, 19], as a generative approach for creating constructive heuristics for the bin-packing problem (described in Chapter 5). Unlike other approaches in the literature, this approach considers the sequential information implicit in the streaming problem instances and avoids the need to derive high-quality, relevant features *a priori*. Thus, this approach outputs solutions directly from the problem instance information.
- A novel neural approach using an encoder-decoder LSTM or a feed-forward neural network [20] to generate constructive heuristics for the bin-packing problem with limited available bins (described in Chapter 6). This approach builds upon the one proposed in Chapter 5 to handle an unbounded stream with stochastic arrival of individual items which are packed on arrival. It also examines all the available bins and uses dynamic information regarding bins-state (i.e. the current candidate solution) to close full bins when necessary and open new ones if required. This approach directly outputs decisions relying on the item to be packed and the current bins-state.

1.3 Publications Resulting from the Period of Study

The following conference papers and journal articles, listed in chronological order, were published during the period of study resulting in this thesis. It is worth mentioning that the first published paper has been awarded the *best paper award* in the ECOM track in GECCO19.

- **Alissa, Mohamad**, Kevin Sim, and Emma Hart. Automated algorithm selection: from feature-based to feature-free approaches. *Journal of Heuristics*, 2021. [Under Review].
- **Alissa, Mohamad**, Kevin Sim, and Emma Hart. A neural approach to generation of constructive heuristics. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 1147–1154. IEEE, 2021.
- **Alissa, Mohamad**, Kevin Sim, and Emma Hart. A deep learning approach to predicting solutions in streaming optimisation domains. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 157–165, 2020.

- **Alissa, Mohamad**, Kevin Sim, and Emma Hart. Algorithm selection using deep learning without feature extraction. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 198–206. ACM, 2019.

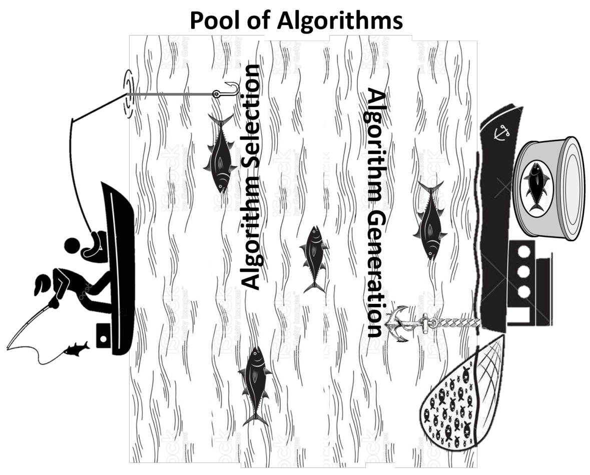
1.4 Thesis Layout

The remainder of the thesis is structured as follows:

- Chapter 2 Background: Introduces the fields of algorithm selection and generation, mostly concentrating on the literature approaches that have been applied to the bin-packing problem.
- Chapter 3 Benchmark Problem Instances: Introduces and analyses the newly generated bin-packing problem instances used throughout this thesis.
- Chapter 4 Automated Algorithm Selection: A novel feature-free algorithm selection approach is proposed using deep learning (RNN-LSTM or GRU) to predict the best performing heuristic from a pool of heuristics relying only on the sequential information contained in the problem instance. This chapter also compares this feature-free approach with traditional feature-based methods using machine learning techniques with a set of hand-designed features.
- Chapter 5 A Deep Learning Approach to Predicting *Solutions* in Streaming Optimisation Domains: A novel generative heuristic approach using deep learning (encoder-decoder LSTM) is presented that is able to output solutions directly from problem instances without defining features *a priori*.
- Chapter 6 A Neural Approach to Generation of Constructive Heuristics: This chapter develops the approach proposed in Chapter 5 by introducing a novel neural approach (encoder-decoder LSTM or feed-forward neural network) to generating constructive heuristics for dealing with the streaming bin-packing problem with limited bins available that directly output decisions to pack items accounting for current bins-state.
- Chapter 7 Conclusions: summarises the contributions of this thesis to the research field, discusses findings, provides answers to the posed research questions and outlines potential avenues for further research.

Chapter 2

Background



2.1 Introduction

Performance Complementarity [7, 25]: is the phenomenon of having different problem instances best solved using different algorithms in the field of Combinatorial Optimisation Problem (COP). There is no single algorithm that dominates all others on all problem instances. Instead, different algorithms with complementary strengths perform best on different subsets of problem instances. This phenomenon has been observed for all NP-hard optimisation problems including scheduling [26] and planning problems, Constraint Satisfaction Problem (CSP) [27], propositional Satisfiability (SAT) [28], the Travelling Salesperson Problem (TSP) [29]. Also, this phenomenon can be observed in a wide range of continuous optimisation problems such as machine learning (meta-learning, i.e. "*learning about learning*").

Rather than using only one algorithm, using a pool of complementary algorithms is one effective way to solve a given combinatorial optimisation problem. The performance complementarity of the pool of algorithms can be exploited in various ways: parallel algorithm portfolios/ensembles; algorithm schedules/pipelines and per-instance/per-set algorithm selection/generation/configuration (or what is called "automated algorithm design"). These concepts are quite close and, unfortunately, there is potential for confusion. Parallel algorithm portfolios refer to applying an ensemble of algorithms in parallel to solve a given problem instance and the produced solution of the entire portfolio is the best solution achieved by any of the component algorithms at any given time [30–32]. Algorithm scheduling refers to ordering the algorithms in a sequence where a certain algorithm at a specific time is applied [33, 34].

Automated algorithm design is the research that concerns automatically design search algorithms or solvers which are able to solve COPs or problem instances. The taxonomy of automated algorithm design has been formulated and classified into three lines of research [35, 36]: (i) Automated algorithm configuration; (ii) Automated algorithm selection and (iii) Automated algorithm composition (or generation). These three lines of research are different ways of automating algorithm design. Per-instance/per-set algorithm selection is the task of selecting the best performing algorithm to solve a given problem instance or a set of problem instances [7]. Algorithm generation aims to combine existing algorithms so that the resulting method outperforms these existing algorithms, this task of generating algorithms automatically is an important sub-field of the Hyper-Heuristic (HH) community, described in Section 2.5. Algorithm configuration refers to finding the best set of algorithm parameters that achieves optimal performance for a problem instance or set of instances [37–40]. It is worth mentioning that combining some of these concepts to address problems more effectively is an active area of research [41].

The scope of the thesis is restricted to per-instance algorithm selection and algorithm generation. It is worth noting the relationship between algorithm selection and generation. Since the algorithm selection relies on a pool of complementary algorithms, improving the pool with new algorithms can improve the overall performance, by adding new algorithms to the pool is by generating them automatically. In the remainder of the thesis, the term algorithm selection refers to per-instance algorithm selection and whenever algorithm design is mentioned, it refers to algorithm generation (unless otherwise stated).

This chapter motivates this thesis research by firstly introducing the automated algorithm selection problem in Section 2.2 and automated algorithm generation problem in Section 2.3, before introducing bin-packing problem as an Np-hard problem in Section 2.4. Some of the algorithm selection and generation approaches that have been applied to the BPPs are reviewed in Section 2.5 and Section 2.6 respectively. Finally, sequential deep learning architectures are introduced in Section 2.7 before reviewing some of the deep learning approaches to address a COP in Section 2.8. Subsequent chapters provide additional specific background information that is relevant to the presented approaches.

2.2 Automated Algorithm Selection Problem

Algorithm selection - the process of selecting the best performing algorithm to solve a given problem instance - is motivated by the potential to exploit the complementary performance of different algorithms on sets of diverse problem instances. However, determining the best-performing algorithm for an unseen instance has been shown to be a complex problem that has attracted much interest from researchers over the decades [8, 7, 42]. Algorithm selection has been intensely explored over the last decades, leading to significant state of the art advances in solving an increasing number of discrete combinatorial problems including bin-packing problems [43], the Job Shop Scheduling problem (JSSP) [44], TSP [45, 46], SAT [47], graph colouring [48] and AI planning. Algorithm selection also holds a lot of promise for improving performance in continuous and mixed continuous/discrete optimisation problems [49, 50].

Originally formulated by Rice [1], the per-instance ASP can be defined as:

Definition 2.2.1 (per-instance Algorithm Selection Problem (ASP))**.** *"Given a set $\mathscr{I}$ of instances of a problem P, a set $\alpha = \{\alpha_1, \ldots, \alpha_n\}$ of algorithms for P and a metric $m : \alpha \times \mathscr{I} \to \mathbb{R}$ that measures the performance of any algorithm $\alpha_n \in A$ on instance set $\mathscr{I}$, construct a selector S that maps any problem instance $I \in \mathscr{I}$ to an algorithm $S(I) \in A$ such that the overall performance of S on $\mathscr{I}$ is optimal according to metric m" [7].*

A schematic of the ASP is shown in Figure 2.1 [1]. In Rice's definition:

- The problem space $\mathscr{P}$ represents a potentially infinite sized set of instances for the problem domain.
- The feature space $\mathscr{F}$ describes a set of characteristics derived using feature extraction from $\mathscr{P}$.
- The algorithm space $\mathscr{A}$ is the set of algorithms available for the problem domain.
- The performance space $\mathscr{Y}$ maps each algorithm to a set of performance metrics [48].

The objective is to identify a mapping between $\mathscr{P}$ and $\mathscr{A}$ that maximises $\mathscr{Y}$. Although Rice's framework is a useful approach for describing ASP, it provides no advice about the mapping from problem space $\mathscr{P}$ to the feature space $\mathscr{F}$, and it clearly shows that the effectiveness of the algorithm selection process for solving a particular problem domain relies on the quality of the problem's features [46, 51].

Definition 2.2.2 (Heuristic)**.** A heuristic is a "rule of thumb" which is derived based on human intuition for a particular domain. A heuristic can be a deterministic or stochastic

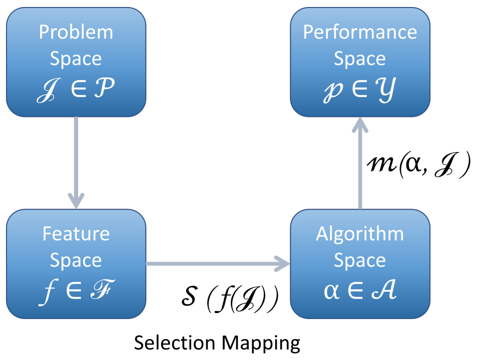

Problem Space $\mathcal{J} \in \mathcal{P}$
Performance Space $p \in \mathcal{Y}$
$m(\alpha, \mathcal{J})$
Feature Space $f \in \mathcal{F}$
$S(f(\mathcal{J}))$
Algorithm Space $\alpha \in \mathcal{A}$
Selection Mapping

Fig. 2.1 Schematic of the Algorithm Selection Problem [1]

constructive procedure to build a solution or a stochastic perturbative (i.e. local search) operator to improve an existing solution. The direct outputs of a heuristic are decisions that formulate a complete solution to the problem being tackled [52–54].

Definition 2.2.3 (Virtual Best Solver (VBS))**.** For a finite set of problem instances $\mathscr{I}$, a fixed set of heuristics $\mathscr{H}$ and a single performance metric m, the Virtual Best Solver (VBS) is defined as a perfect mapping between $\mathscr{I}$ and $\mathscr{H}$.

Definition 2.2.4 (Single Best Solver (SBS))**.** For a finite set of problem instances $\mathscr{I}$, a fixed set of heuristics $\mathscr{H}$ and a single performance metric m, The Single Best Solver (SBS) is the heuristic $\in \mathscr{H}$ that achieves the best performance over $\mathscr{I}$.

A common way of evaluating the algorithm selection systems is by comparing them against the two baselines defined above, SBS and VBS. Where SBS denotes what can be achieved without algorithm selection and VBS makes perfect decisions and chooses the best-performing algorithm on each instance without any overhead (e.g. extracting features) [55, 56].

Based on the selector decision time, ASP systems can be classified as follows:

- Offline algorithm selection: where algorithm selectors are built using a collection of training instances before being applied to new problem instances.

- (Dynamic) online algorithm selection: which adapts an algorithm selector as it solves a series of problem instances.

The offline trained selectors can be saved and reapplied to replicate the solutions attained due to the deterministic nature of the heuristics used in the pool. The online algorithm selection is outwith the focus of this thesis.

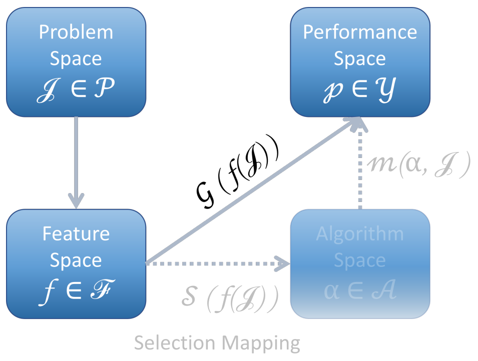

Problem Space $\mathcal{J} \in \mathcal{P}$
Performance Space $p \in \mathcal{Y}$
$\mathcal{G}(f(\mathcal{J}))$
$m(\alpha, \mathcal{J})$
Feature Space $f \in \mathcal{F}$
Algorithm Space $\alpha \in \mathcal{A}$
$\mathcal{S}(f(\mathcal{J}))$
Selection Mapping

Fig. 2.2 Schematic of the Algorithm Generation Problem

2.3 Automated Algorithm Generation Problem

Algorithm Generation - the process of generating a new algorithm to solve a given set of problem instances - is motivated by the fact that there is no single algorithm that dominates all others on all problem instances. Thus, potentially exploiting the complementary performance of different generated algorithms to solve sets of diverse problem instances. However, hand-crafting algorithms is hard, tedious and time-consuming. Alternatively, generating algorithms automatically from a pool of hand-designed algorithms (exploiting the Performance Complementarity Phenomena) has attracted much interest from researchers over the decades. Algorithm generation has been intensely explored over the last decades, leading to significant state of the art advances in solving an increasing number of discrete combinatorial problems including bin-packing problems [57], the JSSP [58], TSP [59] and SAT [60]. AGP can be defined as:

Definition 2.3.1 (Algorithm Generation Problem (AGP))**.** *"Given a set $\mathscr{I}$ of instances of a problem P, a set $\alpha = \{\alpha_1, \dots, \alpha_n\}$ of algorithms (or their basic components) for P, construct an algorithm $\mathscr{G}$ that is built using a set of $\alpha \in A$ to solve any problem instance $I \in \mathscr{I}$ such that the overall performance of $\mathscr{G}(\mathscr{I})$ is optimal and outperform any of the individual $\alpha_n \in A$ according to metric m: $\mathscr{G} \times \mathscr{I} \to \mathbb{R}$ that measures the performance of the generated algorithm $\mathscr{G}$ on instance set $\mathscr{I}$".*

A schematic of the AGP is shown in Figure 2.2, the spaces are as described in the previous ASP section 2.2. The objective is to generate an algorithm $\mathscr{G}$ (using a set of low-level algorithms $\alpha \in A$) that has maximum performance $\mathscr{Y}$ on an instance set $\mathscr{I}$. Similar to ASP, it is clear that the effectiveness of the algorithm generation process for solving a particular problem domain relies on the quality of the problem's features. Similar to ASP, based on the generation process, AGP systems can be classified as follows:

- Offline algorithm generation: The generation takes place by gathering knowledge in the form of rules or programs, from a set of representative training instances before solving unseen problem instances, the generated algorithms using this way are called reusable algorithms.

- Online algorithm generation: The generation takes place while the algorithm is solving an instance of a problem, the generated algorithms using this way are called disposable algorithms.

The offline generated heuristics can be saved and reapplied to replicate the solutions obtained due to the deterministic nature of the heuristics used in the pool. This provides a higher level of assurance in the performance of the generated heuristics as opposed to the variable solution quality achieved with stochastic heuristics. The online generation is outside the concern of this research.

As mentioned before, both ASP and AGP have been used in a wide range of optimisation problems. The next section describes the bin-packing problem as a standard optimisation problem and some of its constructive deterministic heuristics.

2.4 One Dimensional Bin-Packing Problem

The Bin-Packing Problem (BPP) is a standard combinatorial optimisation problem that belongs to the broader class of cutting-stock and packing problems which occur frequently in real-world applications. These applications include industrial manufacturing, container loading, job scheduling on multiple processors, distributing processing tasks in cloud computing and memory allocation [16]. In all these applications, the general objective of BPP is to pack/assign a number of items/tasks into the minimum number of resources, e.g. containers, bins or machines. This problem has been researched intensely over the last century with dozens if not hundreds of publications [61–69] and thus it is an ideal domain to use as an example to explore the Algorithm Selection and Algorithm Generation, Sections 2.2 and 2.3, respectively. The abbreviation BPP is used throughout the remainder of this thesis in relation to the one dimensional class of bin-packing problem, unless stated otherwise, that is defined below.

Definition 2.4.1 (One-Dimensional Bin-Packing Problem (1D-BPP))**.** The problem is well known to be NP-hard [17, 18]. The objective of the One Dimensional Bin-Packing Problem (1D-BPP) is to find a packing which minimises the number of containers, *b*, of fixed capacity *C* required to accommodate a set of *n* items with weights $\omega_j : j \in \{1,\dots,n\}$ falling in the range $1 \leq \omega_j \leq C, \omega_j \in \mathbb{Z}^+$ whilst enforcing the constraint that the sum of weights in any bin does not exceed the bin capacity *C*. The lower and upper bounds on *b*, $(b_l$ and $b_u)$ respectively, are given by Equation 2.1. Any heuristic that does not return empty bins will produce, for a given problem instance, *I*, a solution using b_I bins where $b_l \leq b_I \leq b_u$ [63].

$$b_l = \left\lceil \frac{1}{C} \sum_{j=1}^{n} \omega_j \right\rceil, \; b_u = n \tag{2.1}$$

The general objective of BPP is to pack/assign a number of items/tasks into the minimum number of resources, e.g. containers, bins or machines. In practical contexts, many variants of BPP arise based on dimensionality and the availability of the items that make up the problem instance. Regarding the dimensionality, bin-packing in the literature has been classified into the following:

- One-Dimensional: in this group only one variable is considered that is the size of the items to be packed and thus minimise the number of bins required to pack the items [16].
- Two-Dimensional: It is the generalisation of the one-dimensional problem. In this group, the width and the height of the items are variables so the free space of the container is minimised [16]. This version has many real-world applications such

as cutting glass, wood or metal and packing in the context of transportation or warehousing [70, 67].

- Three-Dimensional: the width, height and depth are variables and thus mainly the volume of the containers is targeted to be minimised [16]. Many real-world applications exist for this version such as air cargo application [71].

Another way to classify BPPs is using the availability of the BPP items. It can be classified into:

- Online or streaming: in the online problems, the items arrive as stream (one by one) to be packed immediately, knowledge about the future items is not known and the order of the items can not be changed. This version can be considered as streaming BPP [72] when the decision can be delayed at the end of checking all the problem instance items, however, the order of these items still can not be changed. The decisions of online/streaming algorithms are irrevocable, i.e. once the item is packed in a bin then it is not possible to move it to another bin [73]. The online/streaming BPP variant is a very active field in the community [74].

- Offline: all the items are known before starting the packing, so the items ordering plays no role [72].

Theoretically, the containers used to pack the items that make up an instance of BPP are available (i.e. remain open) for the duration of this procedure, i.e. whilst there are still items remaining to be packed. This later classification (i.e. the availability of the BPP items) affects the choice of the algorithms (heuristics) to solve the problem. The next subsection describes some of the online and offline heuristics that are used to solve BPP.

2.4.1 BPP Approximation Algorithms

As BPP is an NP-hard problem; hence polynomial-time exact optimisation algorithm is unlikely to be found for its solution. So approximation algorithms are used to generate a near-optimal solution as these do not guarantee an optimal solution for every instance within polynomial time [72, 17]. There have been numerous studies over the decades that have investigated simple approximation algorithms for the online and streaming variants of the BPP [75, 76] such as these deterministic constructive heuristics:

– **First Fit (FF):** Places each item into the first feasible bin that will accommodate it.

– **Best Fit (BF):** Places each item into the feasible bin that minimises the residual space.

– **Worst Fit (WF):** Places each item into the feasible bin with the most available space.

– **Next Fit (NF):** Places each item into the current bin.

NF is different to the other 3 online algorithms in that it only ever considers the most recently opened bin. If an item cannot fit in the current bin that bin is closed and removed from the problem. Streaming and online algorithms are similar in the sense of dealing with one item at a time, i.e. item by item. Online algorithms must make decisions immediately, i.e. pack the items directly as they arrive, while streaming algorithms can postpone such decisions to the very end [77]. Also, some studies used many offline deterministic constructive heuristics such as:

– **Best Fit Descending (BFD)** [66]: The offline version of BF where BFD sorts the unpacked BPP items in descending order, then packs each item in turn into the open bins with the less free space.

– **First Fit Descending (FFD)** [66]: The offline version of FF where FFD sorts the unpacked BPP items in descending order, then packs each item in turn into the first open bin that has free space for it. Bins are traversed in the same order as they are opened.

For all the algorithms listed, if no feasible bin is available to accommodate the next item then it is placed into a newly opened bin. The supply of bins is unlimited and all items must be packed. It is worth mentioning that all the listed above heuristics are hand-designed heuristics, i.e. hand-crafted by experts.

These heuristics will be used to (a) evaluate the performance of the different algorithm selectors in Chapter 4, (b) compare the solutions of these hand-designed heuristics with the solutions predicted using the automatically generated heuristics in Chapter 5 and (c) compare these hand-designed heuristics performance with the performance of the automatically generated heuristics that are developed in Chapter 6.

2.4.2 Performance Metrics

The performance of an algorithm α on instance I refers to the quality of the solution which is produced using algorithm α. The BPP solution is how to pack the items that make up the BPP instance in bins. The quality of this solution can be measured using different metrics such as the number of bins. The number of bins is a natural measure to examine solutions generated using different heuristics, however, it is not a precise measure alone as many heuristics can produce different solutions with the same number of bins

[78]. A more precise metric is Falkenauer's fitness function [79] given in equation 2.2. Falkenauer's fitness function rewards the solutions with free space that is restricted to as few bins as possible rather than having this free space sparse across all bins. In other words, Falkenauer's fitness emphasises providing as large free space as possible in fewer bins. This metric seems a sensible choice as it rewards heuristics that fill bins early on.

$$Fitness = \frac{1}{b}\sum_{i=1}^{b}(\frac{fill_i}{C})^k \tag{2.2}$$

Throughout this thesis, Falkenauer's fitness equation is used with k = 2 as it was originally presented in the literature. With k > 1, the equation provides a distinction between different solutions to the same problem instance that use the same number of bins. Using Falkenauer's fitness, solutions are rewarded if any free bin capacity is restricted to as few bins as possible. C is the bin capacity, $fill_i$ is the sum of the item sizes in bin_i and b is the number of bins used. Both of these metrics (i.e. number of bins used and Falkenauer's fitness) will be used to examine the solutions generated using the different approaches proposed in this research.

The performance of an algorithm α on instance I is denoted by $\alpha(I)$. $OPT(I)$ is the optimal solution for that instance. The worst-case performance ratio (WCPR) of α is defined as the smallest real number $r(\alpha) > 1$ such that $\frac{\alpha(I)}{OPT(I)} \leq r(\alpha)$ for all possible instances. The WCPR of NF is known to be 2 [76] and it was recently concluded after many theoretical studies that the WCPR of FF and BF is $\frac{17}{10}$ [80]. The listed heuristics in the previous section have different performances on different benchmark datasets [52]. Through the literature, BF has been shown to have the best possible average and worst-case performance over all possible problems [81].

Algorithm selection and generation approaches observed in the literature can be broadly split into two main categories:

- Top-down approaches [35]: Automated algorithm selection belongs to this type where the human knowledge provides a template or a grammar (i.e. a structural basis) and the selection process attempts to select the best possible algorithm based on this structure.

- Bottom-up approaches [82–87]: Automated algorithm generation or composition belongs to this type where heuristics are crafted using little human insights and heavily relying on automatically discovered knowledge. The scope of this type has been traditionally restricted to the design of simple heuristics by flexibly composing heuristics components such as using Hyper-Heuristics (HHs), Section 2.5.

Also, depending on the learning techniques used, automated algorithm selection and generation approaches observed in the literature can be broadly split into the following:

- Approaches with unsupervised learning: These have been used for a while to select or generate heuristics automatically in the context of HHs. Traditionally, these approaches use Genetic Programming (GP) [88] and its variations to select or learn the most effective heuristic for an optimisation problem with using objective function and without providing the expected outputs for the input features, e.g. GP has to discover both the heuristic structures as well as the corresponding parameters [89]. While this type of learning does not require labelled data, it takes a longer learning time and it is harder for the learner to converge.

- Approaches with supervised or imitation learning [90]: These approaches use learning algorithms such as machine learning or deep learning to select or learn a heuristic using pairs of inputs and desired outputs (oracle, even though it is not necessarily optimal). The learner (i.e. the selector or the generated heuristic) is not trained to optimise a performance measure, but to blindly mimic the expert who provides the expected behaviour to the ML/DL model. Apart from having labelled data is a hard task and not always available, the performance of the learned selectors or heuristics using supervised learning is limited to what the expert provided, especially when the expert is not optimal. However, this type of learning is faster than the other two types and easier for the learner to converge.

- Approaches with Reinforcement Learning (RL) (or learning through experience) [91]: Without assuming any expert knowledge, the selector or the heuristic is learned through trial and error with a reward signal. These approaches are used when the expert knowledge is not satisfactory and it is required to find better ways of making decisions. This type of learning has a well-known problem, namely exploration and exploitation dilemma [92]. If exploration is not sufficient, the learning process may get stuck around poor solutions or solutions which do not generalise well are found. Similar to unsupervised learning, defining a reward signal might be not an easy task where different reward shapes can value different accomplishments. Despite these drawbacks, the learner using RL can potentially outperform any expert, at the cost of a much longer learning time.

The RL is outside the review of this thesis. So far this point in this chapter, ASP and AGP have been described, then the bin-packing problem has been presented as one standard COP with its heuristics. The next section reviews some of the literature research of algorithm selection on the bin-packing problem.

2.5 Algorithm Selection Approaches

A comprehensive review of different approaches towards algorithm selection can be found in a number of survey papers in this active area of research [7, 8, 42]. Over the literature, ASP has been addressed as either a regression, clustering or classification problem. ASP as a regression problem aims to build a regression model (S, in Rice's notation) to predict the run-time performance $\mathscr{Y}$ of the algorithms $\mathscr{A}$ using the problem characteristics $\mathscr{F}$, especially when dealing with computationally hard problems $\mathscr{P}$, such as CSP [93–95], SAT [96, 97] and TSP [98]. As a clustering problem, the problem instances are clustered based on some extracted features and all the cluster instances are best solved using one heuristic, then the un-seen problem instance is assigned to one of these clusters based on a distance metric, the meta-learning domain is one example of this type [99, 100]. ASP as a classification problem aims to map characteristics $\mathscr{F}$ of problem instances to algorithms themselves $\mathscr{A}$. The research presented in this thesis focuses on ASP as a classification problem. Some of the most relevant ASP approaches are described here, grouped by the approach type.

Before diving into ASP approaches from the literature, the hyper-heuristics field that includes both algorithm selection and generation concepts is introduced briefly.

[Hyper-Heuristics]

Definition 2.5.1 (Hyper-Heuristic (HH))**.** "an automated methodology for selecting or generating heuristics to solve computational search problems [54]."

The rationale behind Hyper-Heuristics is to combine low-level human-designed heuristics in a manner which allows the generated heuristic to outperform any of the individual heuristics when solving a combinatorial optimisation problem. According to [54], HHs can be classified into selective and generative. *Selective HH* methods select pre-defined low-level heuristics in new sequences to construct or improve an existing solution where the aim is to find good sequences of low-level heuristics.

Low-level heuristics are categorised as constructive heuristics, which are usually used to create an initial solution to a problem, or perturbative (local search) heuristics which are iteratively applied to improve an existing initial solution created either randomly or by using a constructive heuristic. In this sense, hyper-heuristics are classified as being selection constructive, selection perturbative, generation constructive or generation perturbative [54, 101]. The next section reviews some of the selection constructive HHs approaches and the generative HHs is described in Section 2.6.

2.5.1 Feature-based Selective HHs Approaches

Typical approaches to ASP is to identify features using expert knowledge, and then use a learning method to find the best performing algorithm(s) for a problem instance from its feature-profile. However, identifying features that are significant in determining performance is complex, usually requires hand-crafting [102, 103, 46, 104], and often is not intuitive. Often the approach must also be combined with a *feature-reduction* method to simplify learning, e.g Principal Component Analysis (PCA) [48, 105], and understand the correlations between features and algorithm performance.

López-Camacho et al. [105] studied ASP in the packing domain using a wide range of 23 features and six heuristics within an evolutionary selective HH framework. They studied the correlation between the structure of 1D- and 2D-BPP instances and the performance of the solvers using PCA [106] as a knowledge discovery method. After the feature-reduction, a subset of nine features is used that is strongly correlated with the heuristics performance (most of them are related to 2D-BPP), including means and standard deviation (std) of the item sizes. They analysed the distribution of feature values across the PCA map and their analysis suggested that there are indeed correlations between instance characteristics and heuristic performance.

Given a problem instance and a set of pre-defined low-level heuristics, selective HHs aim to select and apply an appropriate low-level single heuristic at each decision point based on a set of features that can be derived from the current instance state each time a heuristic is applied. Selective HHs aim to build a solution incrementally, thus this process is repeated until the task is over, e.g. all bin-packing items are packed and a complete solution has been constructed.

A similar approach was proposed by Ross et al. [107] to use a selective HH in the 1D-BPP. A learning classifier system XCS [108] was used to learn rules that map a set of problem-states to specific heuristics. The problem state is defined using the percentage of items in each of four ”natural” ranges relating to item size, given as a ratio of the bin capacity, namely small, medium, large and huge. The approach is tested using a large set of BPP instances and a set of eight heuristics. Their results showed that the evolved rule-set can find an optimal solution in over 78% of cases. An approach that tries to avoid having to hand-craft good features was described in [109] who evolve the *parameters* of a feature design method for 1D bin-packing problems to that best improve the performance of k-Nearest Neighbours (KNN) classifier [20]. As expected from a HH approach, their developed system was able to achieve results substantially better than any individual heuristic on the unseen problem instances.

In addition to the issues associated with feature-deriving, a further challenge arises with respect to the need to deal with streaming data within combinatorial optimisation, particularly given the many practical applications that fall into this category (e.g. in scheduling and packing). The next section reviews some of the literature approaches to deal with streaming problems.

2.5.2 ASP with Streaming Problems

Although feature-based approaches have been shown to work well in domains in which there is no sequential information associated with an instance description[1], domains in which data arrives in a continual stream are more challenging. Statistical approaches to defining features for streaming data are complex, and developing algorithm selectors to tackle streaming data poses considerable challenges due to potentially large streams, the fact that the order of data points cannot be influenced and that the underlying distribution of the data points in the stream can change over time. A recent survey article describing the state-of-the-art in algorithm selection [50] highlighted a pressing need to develop automated algorithm selection methods that are capable of learning in the context of streaming data. A supervised-learning approach was used by van Rijn et al. [110, 111] to predict which classifier performs best on a (sub)stream. Unsupervised learning approaches such as stream-clustering have been used to identify, track and update clusters over time [112, 113]. However, due to the huge space of parameter and algorithm combinations, clear guidelines on how to set and adjust them over time are lacking [114–116].

Recently, machine learning algorithms have gained some traction in the ASP field due to their ability to learn from extremely large datasets in a reasonable time. The next section reviews the most relevant supervised ML-based approaches with ASP.

2.5.3 ASP with ML-based Approaches

One of the most common approaches to ASP is to identify features using expert knowledge, and then train machine learning methods to predict the best performing algorithm(s) for a problem instance from its feature-profile [7]. The input of these techniques essentially is a vector of hand-crafted features and the output is the predicted best performing heuristic. Mao et al. [117] proposed a heuristic performance predictor using a neural network trained on a large set of instances of variable-sized 1D bin-packing problems using 16 features as input. Their prediction system has achieved up to 72% validation accuracy to select the best performing heuristic that can generate a better quality bin-packing solution.

Cruz-Reyes et al. [118] have proposed a methodology to solve ASP in the domain of 1D-BPP. Their methodology relies on data collected from past experience to characterise

[1]although it could be argued that some sequential information is implicit in those approaches just mentioned that dynamically calculate problem state and use this to select heuristics [109, 107].

algorithm performance and it is divided into three phases: initial training, prediction, and training with feedback. The output of the training phase is a trained model that relates the problem characteristics to the algorithms' performance — this model is used to predict the best algorithm for a new given instance in the prediction phase. The new solved instances are then incorporated into the knowledge base to improve the selection quality. They used five deterministic heuristics and two non-deterministic algorithms with 1D-BPP. Three machine learning methods are compared — Discriminant Analysis (DA) [119], a decision tree (a classical model in machine learning) to build the selectors and a Self-Organizing Map (SOM) [120] to implement the selection system with feedback. Five features were used as input. Their method obtained 76% accuracy with DA and 81% accuracy with a decision tree to select the best algorithm. Also, the accuracy increased from 78.8% with initial training up to 100% when using SOM with feedback and the number of problem characteristics was the minimum.

So far this point in this chapter, ASP as one way to exploit the complementary performance has been described. Also, the most relevant ASP approaches have been reviewed. Rather than selecting the best performing algorithm from a pool of complementary algorithms, the next section describes generating a new algorithm from a pool of existing algorithms as another way to exploit the performance complementarity.

2.6 Algorithm Generation Approaches

As mentioned earlier in Section 2.1, the performance complementarity of a pool of heuristics can be exploited in many ways such as algorithm selection and algorithm generation. In the previous section, ASP with some traditional ASP works have been reviewed and showed that ASP relies on a pool of heuristics to select the best performing heuristic for a given combinatorial optimisation problem instance. Improving this pool by adding good heuristics may lead to better overall performance. However, as mentioned before, COPs are NP-hard problems and thus state-of-the-art algorithms rely on hand-crafted heuristics. Designing such heuristics is hard, tedious and time-consuming. Therefore, there is a need to generate heuristics automatically, then these generated heuristics can be used to enrich a pool of heuristics to solve a COP. This section reviews the most relevant approaches in the AGP literature, grouped by the approach type.

As described in the previous Section 2.5, HHs are high-level strategies that operate on a search space of low-level heuristics which in turn operate on the solution space. *Generative HH* methods produce new low-level heuristics from components or building blocks of human-designed heuristics where the aim is to explore the heuristics space (or a space of heuristic components) to generate good heuristics [54, 121].

2.6.1 Feature-Based Generative HHs approaches

The aim of *generation constructive HH* is to produce new low-level constructive heuristics, rather than designing them manually based on human intuition, which is a time-consuming and laborious process [122, 101]. Automating this process reduces the man-hours involved in deriving low-level heuristics and may lead to the induction of new constructive heuristics that humans would not think of [101].

For a while GP [88] with its variations have held the monopoly with respect to generation constructive hyper-heuristics including: tree-based GP [123–126, 122], grammar-based GP [60, 122], gene expression programming [127, 128] and grammatical evolution [129, 130, 83]. This employs the high level methodology of selection, combination and mutation to evolve a population of constructive heuristics for robust problems [121]. Some studies have investigated other techniques for this purpose including, genetic algorithms [131, 132], single-node genetic programming [133] and artificial immune systems combined with genetic programming [134, 58]. The next sub-section reviews some of the generation constructive HH approaches.

[Deriving Features from the Problem Instances]

As explained earlier in this chapter, deriving features is not an easy task. Typical approaches to generation constructive heuristics using GP rely on high-quality features that are derived from the problem instance. Brownlee et al. [57, 135] have used ten 1D-BPP features that are related to the distribution of item sizes within each instance and performance features to analyse the relationship between the training data and automatic design of algorithms. They investigated the distributions of values for features over the instances in benchmark sets, and how these distributions relate to the performance of algorithms built by automatic design of algorithms. They concluded that high variation in some of these features, including mean, standard deviation and maximum of item size, is a strong indicator for good fitting to the training instances and to achieve good performance for automatic design of algorithms.

Another generative constrictive HHs approach that does not explicitly rely on feature identification and extraction was proposed by [133, 134] for the offline version of 1-D BPP. Here, a GP-based system (in specific Single Node Genetic Programming (SNGP) [136, 137]) continuously generates novel heuristics which are maintained in an ensemble, and samples multiple problem instances from the environment. Heuristics that "win" an instance (perform best) are maintained. This was shown to rapidly produce solutions and generalise over the problem space, but required a greedy method of actually selecting between generated heuristics and hence does not fit with the classical ASP definition.

[Deriving Features from Both the Problem Instance and the Current Candidate Solution]

On the other hand, approaches to generation constructive heuristics using GP rely on high-quality features that are derived from both the problem instance and the current candidate solution such as the approach proposed by Burke et al. [138]. Focusing more on generating general constructive heuristics that can address one-, two-, or three-dimensional knapsack and bin-packing problems, Burke et al. [138] proposed a tree-based GP methodology to evolve constructive heuristics using a set of 18 benchmark knapsack and bin-packing datasets. Each combination of item, orientation and corner, in every bin, is evaluated by the generated heuristic. The generated heuristic returns a score for each combination and the actual allocation performed is the one that receives the maximum score. They used a set of 8 features as terminals including the volume of the item, the value of the item (for the knapsack problem), three waste features (XY, XZ, YZ) indicates how good a fit the item is to the corner under consideration, and the relevant coordinate (X, Y, Z) of the corner. Falkenauer function is used as a fitness function in BPP. Their results showed that the generated heuristics are highly competitive with the state of the art human-created heuristics and meta-heuristics. They showed that without sacrificing the quality of the results, they were able to automatically create a general packing heuristic that can solve any packing problem.

The task of identifying appropriate features that correlate to algorithm performance is far from trivial in many domains: in some domains, specifying features is not intuitive, and it can be difficult to create a sufficient number to train a model such as bin-packing domain, while in others in which there are many features, it is necessary to invoke feature-selection methods to choose appropriate features such as the scheduling domain[139, 58, 123, 124, 122, 140]. In this domain, HHs are used to generate dispatching rules automatically that support the sequencing decisions of the jobs on the available machines. Here, the terminal set of GP consists of global or system attributes, job attributes, and machine attributes, and the function set includes basic operators that are used in existing heuristics [89]. [122] listed a set of promising job and shop features that have been commonly used in the development of effective dispatching rules in the literature. This work emphasised the importance to choose the features carefully as it should match the objective function the generated heuristic is evolved for. Otherwise, the search space could be either too restrictive or unnecessarily large, which both hinder the HH ability to generate effective heuristics. Also, it highlighted the problem of using the features with their basic or aggregated form and how this restricts the hyper-heuristic in its search for a better priority function and thus affects the performance of the generated heuristics (i.e. dispatching rules). [89] highlighted the challenge of the real-world production systems with many technical constraints, many attributes need to be considered to construct effective heuristics and thus the search space of scheduling heuristics can be very large.

Deriving features is not only a hard task, it requires post-generation analysis to determine and select features that are more relevant and important for evolving dispatching rules using GP. Removing the irrelevant features can lead to improving the GP-evolved heuristics significantly as the features should match the objective function used in the investigated approach [141]. Many works in the scheduling domain conducted feature analysis and they concluded that the quality of the generated heuristics relies greatly on the quality and relevance of the features derived from both problem instances and the candidate solution. [142] have analysed the importance of the features that appear in the priority functions of their evolved dispatching rules by leaving out each one of them one-by-one without the respective feature and examine the performance of these dispatching rules. Their analysis showed that the effectiveness of the generated dispatching rules relies substantially on some features more than the others. This sheds the light on the importance of selecting and constructing suitable features automatically (using GP) to build effective hyper-heuristics as have been investigated in other works such as [143]. Generally, these challenges of extracting and analysing features can be mentioned for other domains such as CSP [60] and TSP [59].

It is worth mentioning that there is a recent trend to design general methodologies that are applicable to solve across many combinatorial optimization problems, e.g. Ochoa et al. [144] proposed a Hyper-heuristic Flexible framework (HyFlex) for developing cross-domain search methodologies that are implemented four COPs, namely maximum satisfiability, 1D-BPP, permutation flow shop and personnel scheduling.

Similar to ASP, in addition to the issues associated with feature-deriving, a further challenge arises with respect to the need to deal with streaming data within combinatorial optimisation, particularly given the many practical applications that fall into this category (e.g. in scheduling and packing). The next section reviews some of the generative heuristics literature approaches to deal with streaming problems.

2.6.2 AGP with Streaming Problems

Real-world streaming problems (e.g. packing or production lines) typically appear in a dynamic environment with stochastic events such as container fullness or machine breakdowns and random item/job arrivals. This requires flexible responses to the changes in the conditions and constraints. The packing domain has attracted much previous attention: for example, generation constructive HH using tree-based GP approaches are particularly prevalent in the 1-D online BPP [145, 146], which take into account the size of the current item, the full capacity and the load of a bin. These attributes have been reduced to the size of the current item and the residual capacity of a bin (i.e. how much space is remaining in the bin) later in [78]. The research conducted by [145] shows that a simple GP tree can be used to discover human-designed heuristics such as first-fit by examining each bin in turn and place the item in the first suitable bin, while the research in [78, 146] evolved heuristics whose performance was comparable with best-fit by examining all of the bins and place the item in the bin which receives the maximum score. Also, the research in [146] sheds the light on the trade-off between the performance and generality of the generated heuristics and their robustness to new problems, where the choice of the training instances (categorised according to the item size distribution) is vital in the area of automatic heuristic generation.

Other techniques have also been employed to solve the 1-D online BPP. For example, genetic algorithms have been used to evolve low-level constructive heuristics in the form of policy matrix [131, 132]. Depending on the residual space of the bin and the item size, a policy matrix indicates the weight for packing an item in a bin where the item is packed in the bin with the highest weight. This research shows that the generated heuristics are specialised to the distribution of item sizes and outperform the existing human-designed heuristics.

The streaming decisions might be highly interdependent, e.g. the decision for packing/assigning an item/operation may influence the decision of other items/operations in packing and scheduling problems. Therefore, approaches that consider the historical data in making such decisions are needed to deal with streaming problems. In the field of scheduling, since it is hard to find a dispatching rule that is capable of providing optimal decisions for all situations, a sequential GP-based approach has been proposed by Nguyen et al. [147] to address order acceptance and scheduling problems. Rather than generating a single priority rule using the existing GP method, their proposed GP method generates, using a terminal set of eight scheduling features, a set of priority rules from optimal scheduling decisions at different decision moments. Unlike the existing GP methods, the fitness of a rule depends on how well the rule performs at each decision point rather than on the final objective values of the schedule. Thus, these generated rules can cope with different situations and also compensate each other when they are used later in a new forward construction heuristic (FCH). FCH aims to choose which rule to be used in a certain situation by estimating the effectiveness of rules at a decision moment using a backbone rule. Their results show that the proposed forward heuristics are significantly better than the generated heuristics using the existing GP methods, they are competitive with existing meta-heuristics and able to deal with large problem instances. To some extent, this approach is considered as a combination of generative and selective HH approaches where the FCH orders a list of complementary generated rules. However, this approach requires intensive problem domain knowledge, e.g. optimal decisions for training and specialised construction procedure, which may not be always available.

Most of the GP-based HHs approaches presented so far used unsupervised learning to learn the most effective heuristics for optimisation problems. The next section describes some of the generation constructive heuristics approaches using machine learning that learns new heuristics with providing the expected output and hence they are considered as supervised learning approaches.

2.6.3 AGP with ML-Based Approaches

The application of machine learning to address the discrete optimisation problems can be dated back to the 1980s and 1990s [148]. However, this area of research is left nearly inactive at the beginning of this century with very limited success is ultimately achieved. Therefore, these optimisation problems (as NP-hard problems) have traditionally been solved using heuristic methods [149]. Recently, Machine Learning for Combinatorial Optimisation (MLCO) has become a trending research topic. There is a growing trend adopting modern data-driven approaches to solve combinatorial optimisation problems that achieve better and faster results [150].

A recent survey [9] about ML with COPs has distinguished between two types of generating heuristics automatically, using learning algorithms such as ML or DL. (i) The generated heuristic is trained to output solutions directly from the input instances ("End to end learning"), thus calling the heuristic once for each problem instance. (ii) Building a heuristic that can be frequently called, by a master algorithm controls the high-level structure, to assist in lower-level decisions, thus using repeatedly the same heuristic to make the same type of decisions. Both of these types can use supervised and reinforcement learning.

[Generated Heuristic to Produce Solutions Directly]

Approaches for generating heuristics to produce solutions directly from instances can be found in the load planning domain. Larsen et al. [151] used NNs to solve stochastic load planning problem (LPP) [152] where containers are loaded optimally onto double-stack rail-cars, i.e. this is kind of similar to the offline bin-packing problem with the size and weight constraints. The load planning problem can be defined as "Given a set of containers and a set of rail-cars, determine the subset of containers to load and the exact way of loading them on a subset of rail-cars". The aim is to reduce the overall cost of containers left behind and partially loaded rail-cars. The solution is determined by the characteristics of individual containers and rail-cars. Containers are characterised by their length, height, standardised type, content and weight. Rail-cars are characterised by the weight, capacity, tare of each platform and by the specific loading capabilities associated with their standardised type.

Focusing more on predicting the solution directly under incomplete information (i.e. when container weights are still unknown) and rather than propose a heuristic solution for the LPP, [151] used two classical NN models (one for classification and the other for regression) to predict descriptions of solutions for the discrete stochastic version of this optimization problems. Their LPP version depends on features of both rail-cars (e.g. weight holding capacity and length of slots) and containers (e.g. weight and size). and the problem is to predict the solution without knowledge of the container weights since this information is not available at the time of booking.

Mainly, they trained NN models using only as input problem instances with missing information and the targets being obtained from solving the integer linear programming (ILP) formulation of this problem using commercial solvers with the full problem instances. They used features such as total numbers of available rail-cars of each type and the number of available containers of each length. Both of these models have been trained using supervised learning. They compare the performance of the trained models against logistic regression, linear regression and two simple greedy heuristics solutions.

The results show that they were able to predict solutions with high accuracy and in very short computing time (fraction of a second) comparing to solve a single deterministic instance with a traditional solver (seconds to minutes). Their results show that regression feed-forward neural network presented the best performance overall and could generalise reasonably well to harder instances. However, this approach generates solutions with intermediate-level of details, i.e. it just shows which containers will be assigned and does not provide to which rail-car. In other words, their solution shows the number of rail-cars of each type that are used in the solution and the number of containers of each size that are assigned.

[Generated Heuristic to Produce Decisions Directly]

Approaches that call a trained ML model to make low-level decisions rather than solutions can be found in the scheduling domain. [153, 154] used a decision tree to learn new scheduling rules from existing schedules obtained by some simple dispatching rules. Their newly learned rules improved significantly upon the underlying scheduling rules rather than mimicking existing practice. While [155] have described a framework to learn decision tree-based scheduling rules from high-quality schedules produced using Tabu Search [156]. Other ML techniques have been used such as logistic regression [157] and artificial neural networks [158, 159, 44, 160, 161, 142]. These approaches used NNs to learn a priority function that can then be used as part of a heuristic, e.g., a dispatching rule, to solve other problem instances. These approaches do not rely on a pool of low-level heuristics (or their basic blocks) where the ML models outputs are optimal schedules that are generated using e.g. EA. The generated heuristic then serve as a scheduler to reproduce the optimal schedules as closely as possible. All these approaches still rely on domain-related hand-designed features as input to the NNs and they all emphasised how critical the feature selection task is to generate effective dispatching rules.

Most of the ASP and AGP works presented so far use techniques that rely on high quality hand-designed relevant features and/or do not consider the sequential information implicit in the problem instances in the online/streaming problems, e.g. online BPP or scheduling. As described so far, the extracted features should be correlated with the algorithm performance in the case of algorithm selection and match the objective function in the case of algorithm generation. Also, since the streaming decisions are highly interdependent, thus techniques that are able to consider and learn from the historical data in making such decisions are needed to deal with streaming problems. The next chapter reviews techniques from the deep learning field that tackles sequential data specifically.

2.7 Sequential Deep Learning Architectures

As described in the previous sections, streaming problems are challenging and traditional approaches to select or generate algorithms to deal with streaming instances are not effective enough. Most of the research in the literature is used techniques without considering the sequential data that is implicit in the problem instances. Also, these approaches rely greatly on hand-designed features which are not easy to derive and require further processing like features-selection. Even though some of these approaches use sophisticated machine learning techniques such as neural networks, they are not able to deal with sequential-data in the streams let alone relying heavily on hand-designed features. Therefore, there is a need to find a technique that can deal with sequential-data and avoid using features.

One solution to dealing with sequential data can be found in the field of deep learning, where the use of recurrent neural networks with long short-term memory [10] or gated recurrent units [11] to classify sequential data has become widespread in recent years. Deep learning has achieved groundbreaking results in applications where the input is formatted as time-series data or in domains where sequences have specific orderings but without any explicit notion of time [12]. Examples including video and image recognition, natural language processing, music generation and speech recognition [13–15]. From the learning scheme perspective, generally, deep learning is very flexible and can be used in supervised, unsupervised or reinforcement learning.

2.7.1 Recurrent Neural-Networks

A recurrent neural-network is a deep learning method designed to learn from sequence data. RNNs are one of the two most common architectures described under the umbrella term "Deep Learning", namely RNNs and Convolutional Neural Network (CNN) [162]. They differ from feed-forward neural networks [120] due to the presence of cyclic connections from each layers' output to the next layers input, with feedback loops returning to the previous layer (Fig 2.3-a) [163, 12]. This structure prevents traditional back-propagation from being applied since there is not an endpoint where the back-propagation can stop. Instead, Back Propagation Through Time (BPTT) is applied: the RNN structure is unfolded to several neural networks with certain time steps and then the traditional back-propagation is applied to each one of them (Fig 2.3-b) [163]. RNNs are specifically designed to learn from sequence data where temporal information explicit in the order of sequences is used to identify relationships between the data and the expected outputs from the network. More precisely, RNNs operate on sequence data by sharing parameters across different sequence steps where the same neural network block (i.e., with the same architecture and parameter values) is applied at each step of the sequence [9].

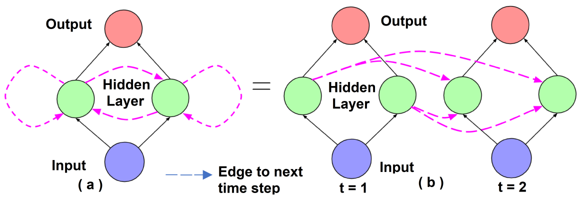
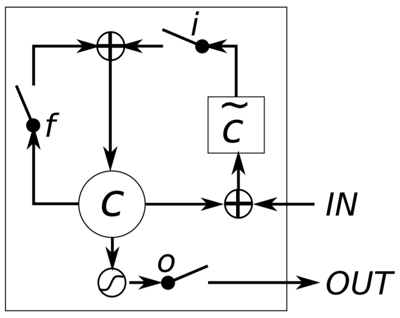


Fig. 2.3 (a) a Simple RNN and (b) an example of unfolded RNN with two time steps [12]

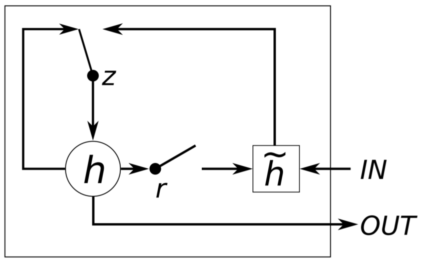
[LSTM and GRU]

Specialised RNNs are known as a long short-term memory [10] and gated recurrent unit [11] have been shown to be efficient and effective in learning long-term dependencies from sequences of ordered data. LSTM neural networks incorporate additional gates that can retain, retrieve and forget information (i.e. the network states) over long periods of time [13]. These gates are simply a combination of addition, multiplication and non-linear functions [164]. Three main gates are used in the LSTMs input, output and forget gates and three main states input, hidden and internal cell states. Internal states are the ”memory” of the LSTM block, hidden states represent values that come from the previous time step, and the input state is the result of the linear combination between the hidden state and the input of the current time step. In the LSTM network, the classic neurons in the hidden layer are replaced by memory blocks. Fig 2.4a shows that the LSTM’s input comes from the network through the input gate and the only outputs from the LSTM to the rest of the network emanate from the output gate multiplication. The input gate determines how much of the new memory content is added to the memory cell, the output gate modulates how much of the internal state would be exposed to the external network (higher layers and the next time step), while the forget gate defines how much of the existing memory is forgotten. GRU is a recent variation on LSTM with only two gates, update and reset gates which decide what information should be passed to the output. The update gate decides how much of the past information would be passed to the future while the reset gate determines how much to be forgotten. Also, GRU does not use the internal state and instead uses the hidden state to transfer information through the time steps [165]. A more comprehensive description of the rapidly expanding field of DL, which has many competing, but no prevalent architectures, is outwith the scope of this study. The reader is referred to [166] for a deep learning textbook.

RNNs have been successfully used to learn over sequences for more than three decades [167]. However, RNNs are not able to deal with certain complicated problems such



(a) Long Short-Term Memory



(b) Gated Recurrent Unit

Fig. 2.4 Graphical illustration of (a) LSTM, where i, f and o are the input, forget and output gates, respectively. c and c~ denote the memory cell and the new memory cell content. (b) GRU, where r and z are the reset and update gates, and h and h~ are the hidden stat and the new hidden stat, respectively [165].

as sequence-to-sequence problems [11, 19], where the input and the output can have different sequences lengths. One solution has been proposed recently by [11, 19] called encoder-decoder in which uses one RNN (LSTM or GRU) to map an input sequence to an embedding, i.e. a fixed-sized vector, and another (possibly the same) RNN to map the embedding to an output sequence. The next section describes this deep learning topology, encoder-decoder LSTM.

2.7.2 Encoder-Decoder LSTM

Encoder-decoder LSTM [11, 168] is a deep learning architecture dedicated to solving what is commonly known as a sequence-to-sequence problem (or seq2seq). These problems are challenging due to the fact that the number of items in the input and output sequences can be arbitrary and even from different spaces (e.g. in example statistical machine translation [169]). Seq2Seq problems have been tackled in other domains using encoder-decoder LSTM that has proven very effective and achieved state-of-the-art performance. Encoder-decoder LSTM has received much attention in the natural-language domain in tasks such as text-translation which generate a translated sentence directly from an input consisting of a stream of words [11, 19]. Other example applications including question answering [170], image caption [171] and summarization [172].

An encoder-decoder LSTM architecture contains two models as shown in Fig 2.5. The **encoder** is used for mapping an input sequence into a vector of fixed dimensionality, and the **decoder** for outputting a predicted sequence based on the encoder output. The encoder-decoder LSTM uses a training technique known as *teacher forcing* [173]. This has been shown to train RNNs quickly and effectively where the decoder model receives the ground truth output $y_{(t)}$ as input at time t + 1 as shown in Fig 2.5. This approach

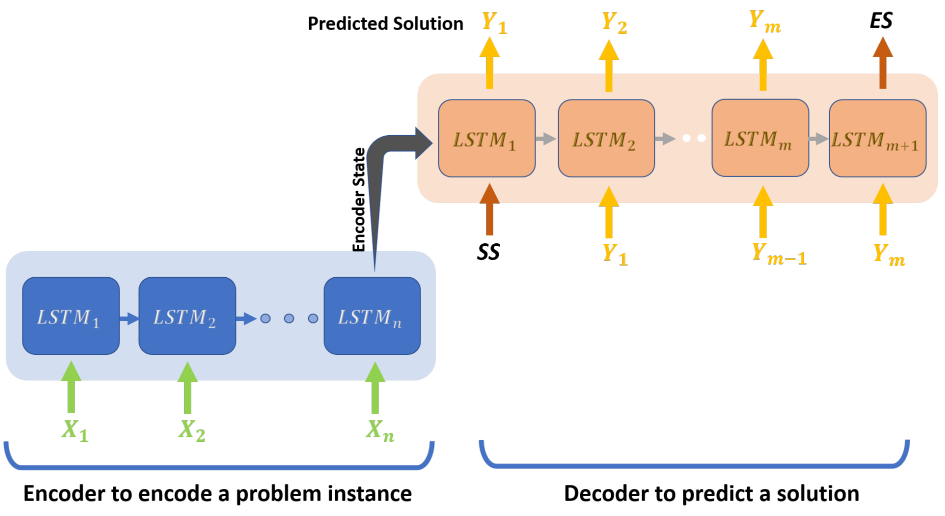


Fig. 2.5 The training process of the encoder-decoder LSTM

was originally described and developed as an alternative technique to BackPropagation Through Time (BPTT) for training RNNs architecture that have lacking hidden-to-hidden connections [174, 173]. A detailed explanation of the model is outside the scope of this chapter, the reader is referred to the original Google paper [19] and watching Stanford University lecture [175] for more details. However, a brief outline of how it works is provided below.

Training

Fig 2.5 provides a conceptual overview of how the encoder-decoder LSTM works. During training, the encoder produces a vector that represents the input sequence (window of words or items) to initialise the state of the decoder. Triggered by the encoder's state, the decoder produces each output in the target sequence in a one-shot manner (i.e. rather than recursively), as the entire target sequence is known during training.

Testing

Similar to training, the encoder provides a fixed vector from the input. As shown in Fig 2.6, the main difference with the testing phase is that the whole target sequence is not known beforehand, so the decoder will be called recursively for each output until the end of the target sequence. Using the encoder's state and the starting sequence value (SS), the decoder starts to predict the first output in the target sequence and feed the first predicted output with the first decoder unit's cell state back to the next time step unit. This process will end when reaching the maximum size of the target sequence. If the network predicts a target sequence before reaching the maximum size then it will predict the End

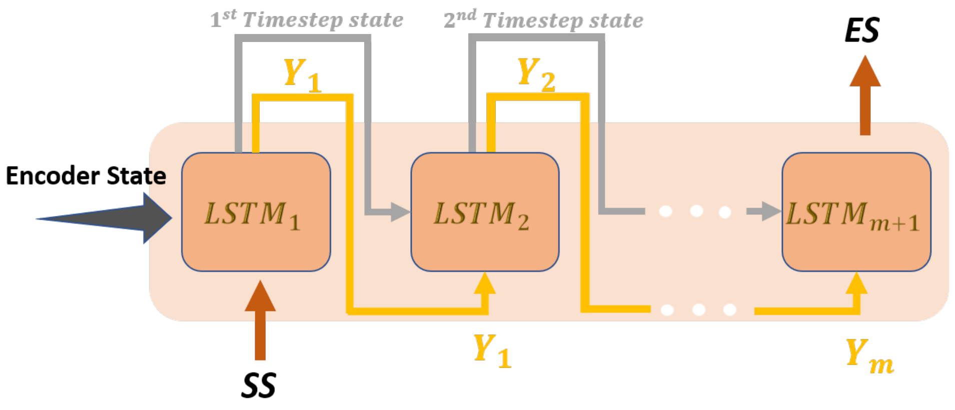


Fig. 2.6 The testing process of the decoder

Sequence (ES) value until reaching the max target sequence size. ES refers to the end of the sequence and it is an unused value to make it out of the input range.

The process of prediction starts when the encoder initiates the decoder state with a vector representing the problem instance. Given the encoder state, alongside a trigger to indicate the start of a sequence, the first decoder unit tries to predict the first output of the target sequence, to provide the first decoder unit state. The first decoder unit state, along with the first predicted output in the target sequence then feedback into the second decoder unite to predict the second targeted output, thus providing the second decoder unit state and so on. Note that once the target sequence is predicted, the network will predict ES until the end of the sequence is reached.

2.8 Deep Learning with Combinatorial Optimisation Problems

Very few approaches have been proposed using deep learning (especially with supervised learning) for ASP and AGP, here some of these works are reviewed.

2.8.1 Deep Learning for ASP

Recently, deep learning algorithms have gained some traction in the ASP field due to their ability to learn from extremely large datasets in a reasonable time. To eliminate the arduous task of manually designing features, Loreggia et al. [176] proposed a deep learning approach to automatically derive features in SAT [177] and CSP [178] assuming that any problem instance can be expressed as a text document. Unlike previous works e.g. [48, 105] that derive features from features automatically using PCA, their approach automatically derives features from a visual representation of the problem instances (i.e. converting the text files into grey-scale square images), which can be used to train a CNNs to predict the best solver for the instance. Although their approach obtained better results than the SBS, it was not able to outperform the approaches that use regular manually crafted features.

A recent conference paper, Seiler et al. [179] addressed the ASP in the Euclidean TSP domain, using an evolved (and balanced) dataset (1000 instances) with two TSP solvers. They compared a feature-based approach using four different classical ML classifiers to a feature-free approach using deep learning CNNs. Due to the large TSP-related feature sets, they conducted data analysis and automatic feature selection to choose an adequate set of 15 most relevant features. Their results show that the feature-based approach improved over the SBS performance but still quite far away from the performance of the oracle-like VBS. The feature-free approach matches the performance of the quite complex classical ML approaches, despite being solely based on the raw visual representation of the TSP instances.

2.8.2 Deep Learning for AGP

Recently some researchers start to investigate using deep learning to solve COPs, Vinyals et al. [180] have introduced a new neural architecture named Pointer Net (Ptr-Nets) as the first attempt of using supervised deep learning for solving a combinatorial optimisation problem, such as the planar TSP. Pointer network is a specific attention mechanism motivated by the neural encoder-decoder sequence model (described in section 2.7.2) to solve sequence-to-sequence combinatorial problems where the size of the output dictionary depends on the length of the input sequence. It learns the conditional probability of an output sequence with elements that are discrete tokens corresponding to positions in an input sequence. However, the problems that have been tackled in [180] are graph-type problems and can be argued that their inputs are not a temporal sequence (it is more a spatial sequence), e.g. set of Cartesian coordinates for TSP. Their research shows that Ptr-Nets can be used to learn approximate solutions to TSP using training examples of inputs and desired outputs given by an approximate solver. Similar to [180], [181] have used supervised pointer network to solve TSP as part of their research. However, their results were not as good as the results presented in [180]. They conclude that learning from optimal tours is harder for supervised pointer networks due to subtle features that the model cannot figure out only by looking at given supervised targets.

Although it is outwith the scope of this thesis, it is worth mentioning that currently RL for combinatorial optimization has become a trending research topic. Deep RL has been used to learn a policy rather than a heuristic for domains such as BPP [149, 182–184], TSP [181, 185, 186], JSSP [150, 187, 188] and vehicle routing [189, 190]. More related works in this active area of research can be found in these surveys [9, 191, 192].

2.9 Summary

Algorithm *selection* refers to the task of selecting the best performing heuristic from a pool of existing heuristics to solve a combinatorial optimisation problem. This pool of heuristics can be improved by adding new good heuristics that are *generated* automatically and thus improve the overall performance of the pool. This chapter reviewed the research related to algorithm selection and generation with a variety of approaches.

Both algorithm selection and generation have suffered from extra tasks (burdens) including the need to define features. Feature design is not an easy task, not intuitive and requires expert input. Often it requires post-generation analysis to determine the best set of features to the task at hand as the features should be high-quality, relevant and important for the task, e.g. features should be correlated with the algorithm performance in the case of algorithm selection and match the objective function in the case of algorithm generation.

These burdens become more challenging when building algorithm selectors or generating algorithms automatically in domains with streaming nature including bin-packing and scheduling where items/jobs arrive in a continual stream and should be packed/assigned as they arrive. The statistical approach to extract features from streaming problem instances needs to derive features that capture the sequential information contained in the stream in order to be informative. The streaming decisions are highly interdependent, e.g. the decision for packing/assigning an item/operation may influence the decision of other items/operations in packing and scheduling problems. Therefore, approaches that consider the historical data in making such decisions are needed to deal with streaming problems. While typical approaches in the algorithm selection and generation literature used techniques, such as machine learning and genetic programming, relying greatly on high-quality extracted features as input and without considering the sequential information in the stream. One solution to address these burdens can be found in the field of deep learning, where the use of RNN-LSTM or RNN-GRU to classify sequential data or make sequential decisions has become widespread in recent years.

The key points emerging from this review that have guided the research presented in the remainder of this thesis include:

- Most of the research related to algorithm selection and generation rely on hand-designed features. However, deriving features is hard, complex task and requires further processing like feature selection.

- Most of these approaches use techniques that don't consider the sequential information that is implicit in the streaming problem instances.
- There is a dominant use of GP and its variations to select or generate heuristics automatically.
- In general, there is lacking to use the sophisticated techniques including supervised deep learning in these two areas of research, i.e. algorithm selection and generation.

As will be seen in the subsequent chapters, the research presented in this thesis differs substantially from the previous work described in that it abandons the need to derive features from problem instances, circumventing the associated issues. Also, the proposed approaches make use of deep learning techniques, in particular LSTM or GRU, to capture and learn the sequential information when learning an algorithm selector or generate a new heuristic automatically for streaming problems. In this thesis, the underlying algorithm space is restricted to constructive deterministic heuristics that provide a thorough analysis of the merits of the proposed approaches without being influenced by the stochasticity in the case of using the perturbative heuristics. All this research is in the realm of the 1D-BPP. It is expected that the proposed approaches in this thesis could easily be transferred to other problem domains.

The remainder of this thesis concentrates on addressing the research questions outlined in Chapter 1 in the streaming version of 1D-BPP. Chapter 4 proposes a novel feature-free algorithm selection approach using a RNN-LSTM or RNN-GRU that relies only on the sequential information inherent in a streaming problem instance to select the best performing algorithm. Chapter 5 proposes a novel approach to generating constructive heuristics automatically using an encoder-decoder LSTM that is able to predict solutions directly from problem instances (i.e. without deriving features). Finally Chapter 6 develops the approach proposed in Chapter 5 by proposing a neural approach using an encoder-decoder LSTM or a feed-forward neural network that is able to output decisions to address streaming 1D-BPP with limited bins.

Since instance diversity, in regards to heuristics performance, plays a crucial role in developing good algorithm selectors that exploit the complementary performance, the next chapter 3 describes preparing the bin-packing problem instances that are used for the approach proposed in Chapter 4.

Chapter 3

Benchmark Problem Instances

3.1 Introduction

This chapter describes the BPP instances that are used to build feature-free algorithm selectors proposed in Chapter 4. There are many sets of benchmark problem instances available in the literature for the domain of 1D bin-packing. However, relying on well-studied benchmark instances, or randomly generated instances, limits the ability to study ASP [46]. Instance diversity plays a crucial role for ASP in terms of learning from the instances' characteristics [193–195]. As noted in [42], while benchmark problems are suitable for comparing different algorithm's performances, they are often unsuitable for investigating the ASP due to being similar in structure and often tailored towards the abilities of specific solvers, e.g. Falkenauer instances [196, 197]. Furthermore, Ross [198] discusses specific examples of heuristics that have been shown to be 'friendly' on many benchmark bin-packing problems that are generally considered to be hard, but that produce very poor results on easy problems. As a practical illustration, consider the following: taking 5 well-researched datasets totalling 1370 instances from the literature [79, 199], it is noticed that if an instance is treated in the order that the item-size data is published, FF is the dominant heuristic solving 955 instances better than any of the other heuristics. BF wins on 414 instances and WF wins on 1. The success of FF on many benchmarks is due to the fact that many of the published benchmarks provide instance data in the order found in the best-known solution, hence FF simply recreates that solution. If the item orders are randomised then BF, as expected, becomes the single best solver winning on 926 instances. FF wins on 381, WF wins on 61 and NF wins on 2. Clearly using these relatively small sets of benchmark problems would lead to dramatically unbalanced datasets and hinder an investigation into the ASP with DL and ML. To counter this, one solution is to generate instances that serve the aim of the research.

3.2 Contribution

The introduction of four new sets of benchmark problem instances totalling 16,000 instances for the streaming version of the 1D-BPP domain. Each dataset has 4,000 instances, and contains 1,000 instances uniquely solved best by *exactly one* of four algorithms in a pool used in the algorithm selection problem. The datasets were generated using an Evolutionary Algorithm (EA) [200] which maximises the performance difference between the target algorithm and the other algorithms used in the selection problem. Thus these datasets exhibit complementary performance.

Through the literature, many works in different domains have generated their datasets and the next section reviews some of these works.

3.3 Study the Correlation Between Solver Performance and Instance Characteristics

Due to the limitation of an algorithm selectors' ability to generalise over new problems which are not similar to those used during their construction, and in order to construct informative benchmark sets that provide the required diversity to build selectors, several researchers [46, 193, 201, 202, 179, 203] have focused on generating problem instances that are either hard or easy for specific solvers, or that maximise the performance difference between given solvers. These instances can provide insights into the correlation between performance differences of the solvers and instance characteristics. For instance using an EA, Smith-Miles and van Hemert [46], Smith-Miles et al. [193] evolved TSP instances that are easy or hard to solve for a certain algorithm, and also *uniquely* hard or easy for a given algorithm within a portfolio containing two heuristics. Bossek and Trautmann [201, 202] have used an evolutionary approach to evolve TSP instances with maximal performance difference of two heuristics in both directions, i.e. instances are easier for solver A but much harder for solver B as well as the opposite case. Seiler et al. [179] and Bossek et al. [203] have recently evolved TSP instances that are diverse in both performance and feature spaces using EA with respect to two TSP solvers. Gao et al. [194] generated TSP instances using EAs that are hard or easy to be solved by a given solver and diverse with respect to different features of the TSP (i.e. feature-based diversity). Neumann et al. [195] used a discrepancy-based evolutionary diversity optimization approach to construct diverse sets of images as well as TSP instances.

3.4 Description of Datasets Used

A streaming version of a 1-d bin-packing problem is considered in which items arrive in a stream, one at a time, and *must* be packed in the order that they arrive. In the specific version that is considered here, all items to be packed are known before packing starts (i.e. they constitute a fixed-length batch) but the order that items in the batch are presented to the packing heuristics is fixed and cannot be changed. In contrast to other types of packing problems, the sequence cannot be re-ordered to find an ordering that provides an optimal packing with respect to a given heuristic. The function of the algorithm selection method is therefore to select a heuristic to apply to pack the entire batch, considering the items in the fixed order given. As described in Chapter 2, there have been numerous studies over the decades that have investigated the performance of simple approximation algorithms for the streaming variant of the BPP [75, 76]. A pool of four simple approximation algorithms (i.e. heuristics) is considered for the approach described in Chapter 4. These heuristics are selected from the literature specifically designed for this variation of the BPP [204]: BF, FF, WF and NF, described in Chapter 2.

These are the datasets used to investigate the feature-free algorithm selection approach proposed in Chapter 4. These datasets were created by Dr Kevin Sim, while the analysis were conducted by the author of this thesis. The next section describes the datasets creation process and the datasets analysis:

- Evolved Datasets DS(1-5).
- Random Datasets RDS(1-8).

3.4.1 Description of Evolved Datasets

As mentioned in the Introduction Section 3.1, using either benchmark or random instances will result in unbalanced datasets with respect to the performance of the best-solving heuristics. Therefore, in order to address this problem and create balanced data sets for ASP investigation, EA is used to generate four datasets, each with 4,000 instances and using the bounds and distributions shown in Table 3.1. Each dataset comprises of four sub-classes of 1,000 instances that are uniquely solved best (according to Falkenauer's performance Equation 3.1) by one of the four algorithms under investigation. Algorithm 1 describes how each sub-class of instances is generated. For each instance, a population of candidate sequences is initialised with n_{items} and weights randomly drawn from the desired distribution. The EA then attempts to evolve the order of items such that the performance of the target algorithm α_{target} exceeds the performance of the other algorithms $\in \mathscr{A}$.

Unlike in [205], who used a binary representation and evolved the weights of items, an integer representation is used: a chromosome is simply the sequence of n_{items} defining

Table 3.1 Dataset Generation Parameters. Bin Capacity is fixed at 150

Dataset	n_{items}	Lower - Upper Bounds	Distribution $\mathscr{D}$
DS1	120	[40-60]	Gaussian
DS2	120	[20-100]	Uniform
DS3	250	[40-60]	Gaussian
DS4	250	[20-100]	Uniform
DS5	(120,250)	[20-100, 40-60]	(Uniform, Gaussian)

that instance. Item weights are randomly initialised at the start, then do not change. Only the order of items is evolved, preserving the desired weight distribution.

Algorithm 1 EA Pseudo Code

```
Require: $\mathscr{A} =$ :The set of algorithms
Require: $\alpha_{target} =$ :The target algorithm
Require: $\mathscr{D}_{target} =$ :The target distribution
Require: $\mathscr{I} = \emptyset$ :The set of instances being evolved
Require: $n_{instances} =$ : The number of instances to generate
Require: $n_{items} =$ : The number of items in each instance
 1: repeat
 2:     $population \leftarrow$ initialise($popSize$, $n_{items}$, $\mathscr{D}_{target}$)
 3:     for $i \leftarrow 1, maxIter$ do
 4:         $parent_1 \leftarrow$ select($population$)
 5:         $parent_2 \leftarrow$ select($population$)
 6:         $child \leftarrow$ crossover($parent_1$, $parent_2$)
 7:         $child \leftarrow$ mutate($child$)
 8:         $child_{fitness} \leftarrow$ evaluate($child$)
 9:         $population \leftarrow$ replace($child$)
10:     end for
11:     if $evaluate(population_{best}) > 0$ then
12:         $\mathscr{I} \leftarrow \mathscr{I} + population_{best}$
13:     end if
14: until $|\mathscr{I}| = n_{instances}$
15: return $\mathscr{I}$
```

In the pseudo-code shown, the target algorithm α_{target} is the algorithm $\alpha \in \mathscr{A}$ that this method is attempting to evolve instances for, such that the solution fitness (defined by Equation 3.1) using α is better than the solutions produced by all the other algorithms $\in \mathscr{A}$. $\mathscr{D}_{target}$ is the distribution from Table 3.1 that is used to initialise the $population$ (line 2). When using a Gaussian distribution, item weights are generated using a Gaussian distribution with standard deviation $(ub - lb) \div 2$ and mean $lb + (ub - lb) \div 2$. Items generated with weight $w < lb$ or $w > ub$ are discarded. The lower and upper bounds on item weights are given in Table 3.1. For each data set, Algorithm 1 is run once for each

$\alpha_{target} \in \mathscr{A}$ using $n_{instances=}$ 1,000 resulting in 4 balanced data sets of 4,000 instances where for each data set 1,000 instances are solved best by each $\alpha \in \mathscr{A}$.

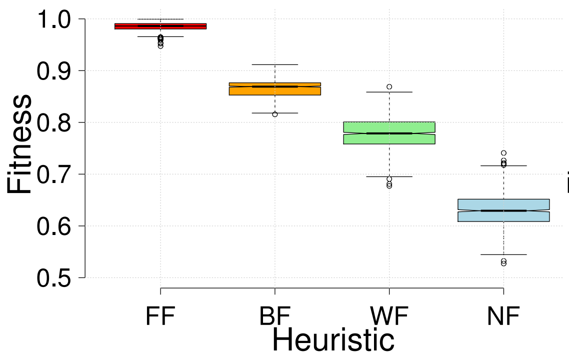
$$Fitness = \frac{1}{b}\sum_{i=1}^{b}(\frac{fill_i}{C})^k \tag{3.1}$$

As describes in Chapter 2, Falkenauer's performance metric [79] (Equation 3.1) is used to measure the quality of a solution produced by an individual algorithm $\alpha \in \mathscr{A}$. In this study k is fixed at 2. C is the bin capacity which is fixed at 150, $fill_i$ is the sum of the item sizes in bin_i and b is the number of bins used. The evaluation function used in Algorithm 1 assigns a score to each algorithm using Equation 3.1 and then orders the algorithms from best to last $\alpha_{1..4}$. The fitness of a chromosome is then $evaluation = fit(\alpha_{target}) - fit(\alpha_2)$. i.e. The fitness of a chromosome turns positive when the target algorithm has the best fitness on that instance. The population size is fixed at 500. Tournament selection is used with a tournament size of 2. Crossover is a single-point crossover applied with 99% probability and mutation (applied with 2% probability) simply swaps the order of two items in the chromosome. The child produced replaces the worst member of the population if it has higher fitness. The algorithm is run for $maxIter = 1X10^6$ iterations. At the end of that time, the population best is kept only if the target algorithm is the best performing algorithm on that instance.

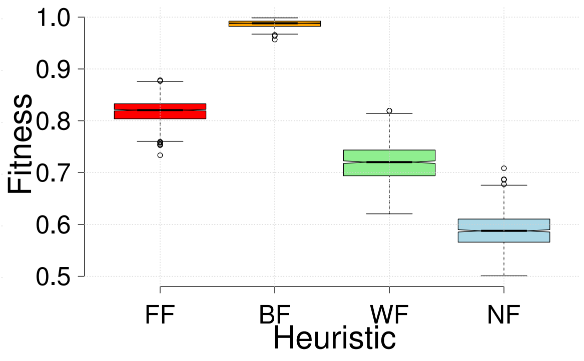
[The Evolved Datasets Analysis]

Each of the Figures 3.1 a-c shows the performance of all algorithms on the subset of 1,000 instances from DS2 identified in the caption by the target class. In all cases, the target algorithm significantly outperforms the other algorithms. Figure 3.1d shows that for NF the difference in performance between the target class and the other algorithm appears less pronounced although it is still highly significant (comparing BF and NF using a paired Wilcoxon Signed-Rank Test [206] gives a $p - value$ of $2.2e^{-16}$). NF is known to have the lowest WCPR of all the algorithms investigated and the task of evolving these instances required many more attempts than for the other algorithms, taking around 3 times the computational effort to find 1,000 instances for each data set. Plots for the other data sets show similar patterns but are omitted due to space limitations.

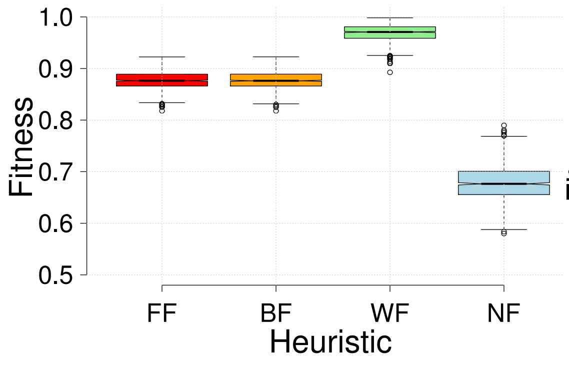
Figure 3.2 shows the performance of all algorithms on the full set of 4,000 instances in DS2. Best Fit is the Single Best Solver across all 4,000 instances (this is the same for the other data sets). The plot labelled VBS shows the Virtual Best Solver i.e. the performance of the Oracle that greedily selects the best algorithm for each instance. The objective of the ASP is to achieve the same performance as the VBS.



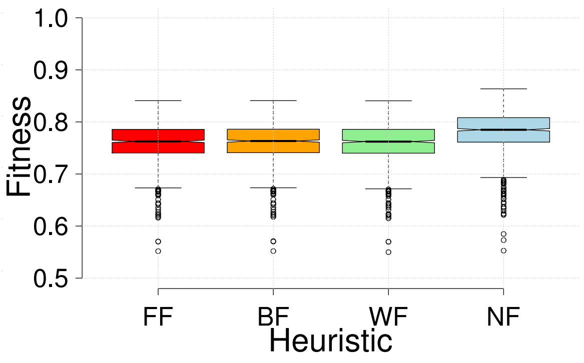
(a) α_{target} = First Fit



(b) α_{target} = Best Fit



(c) α_{target} = Worst Fit



(d) α_{target} = Next Fit

Fig. 3.1 Each plot shows how the 4 heuristics perform on the sub-class of 1000 instances evolved for each target heuristic (DS2 120 items U[20, 100])

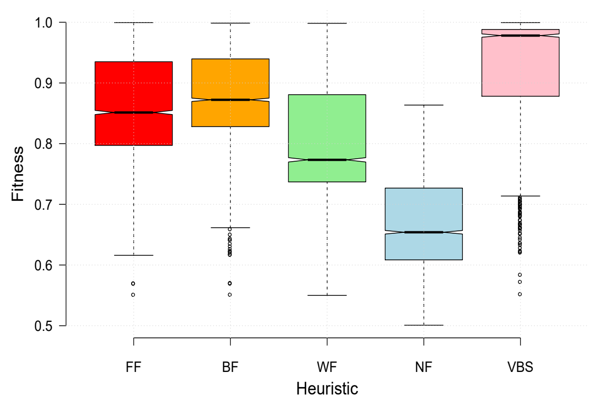
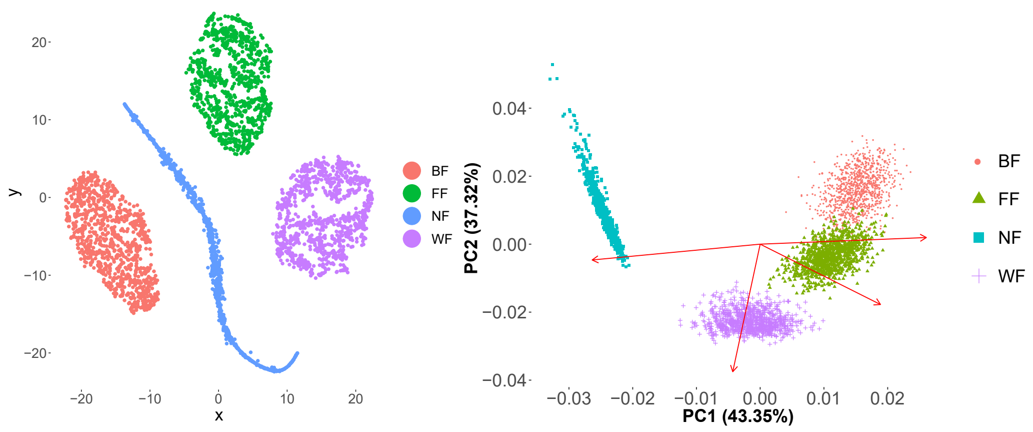


Fig. 3.2 DS2 Performance



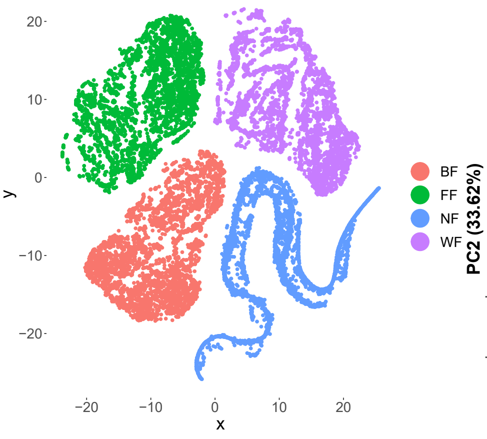
(a) t-SNE Plot (TW = 0.99) **(b)** PCA Plot

Fig. 3.3 t-SNE and PCA plots of the performance of all heuristics on all 4,000 problem instance from DS2

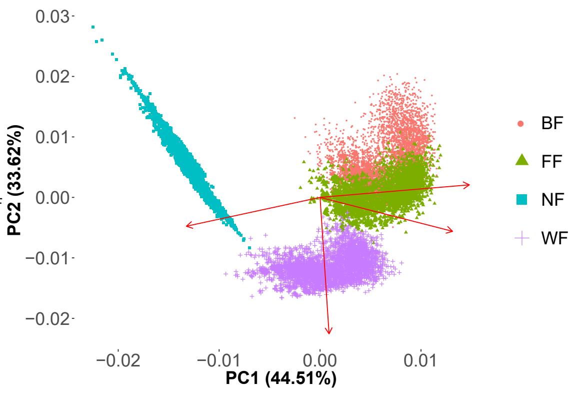
To further illustrate the diversity in performance on the evolved instances, linear and non-linear dimensionality reduction techniques have been used to visualise the performance space, namely PCA [207] and t-Distributed Stochastic Neighbour Embedding (t-SNE) [208] techniques, respectively. In order to assess the *trustworthiness* of the t-SNE plots, the metric proposed in [209] is used. Conceptually, the trustworthiness score captures errors in the preservation of neighbourhoods or proximity relationships in the new projection of the data, and is quantified by the metric defined by [209] on a scale of 0 to 1, where 1 represents maximum trustworthiness. The trustworthiness value TW is noted on each of the plots presented in the figures.

Figure 3.3 shows the performance of each algorithm on DS2 reduced from 4-dimensional performance space to 2 dimensions using t-SNE and PCA. Clearly, a set of instances has been successfully generated that cover the performance space from the perspective of the algorithms and the metric investigated. Figure 3.4 emphasises this showing t-SNE and PCA plots of the performance of each algorithm on the full set of 16,000 instances reduced from 4 to 2 dimensions.

From PCA plots 3.3b and 3.4b, BF and FF are the closest in terms of their eigenvectors but are clearly still disparate. NF is isolated as the worst-performing heuristic — as expected due to its poor WCPR. If the orders of instances are shuffled to destroy the evolved sequences and replot with PCA (Figure 3.5b) it is clear that there is now no identifying structure apparent in the PCA plot that a selection method might exploit. On the randomised instances, BF wins for 10875 instances, FF wins on 4586, WF wins on 539 and NF fails to win on any instance. Although PCA failed to identify a useful structure of this randomised version of the data, t-SNE (Figure 3.5a) successfully differentiates between the performance of the different heuristics (a reason might be t-SNE is a non-linear technique that works by minimising the distance between the points). Based on these plots, it seems using t-SNE is more suitable for the data in this research as a dimensionality reduction technique and thus it will be used instead of PCA.



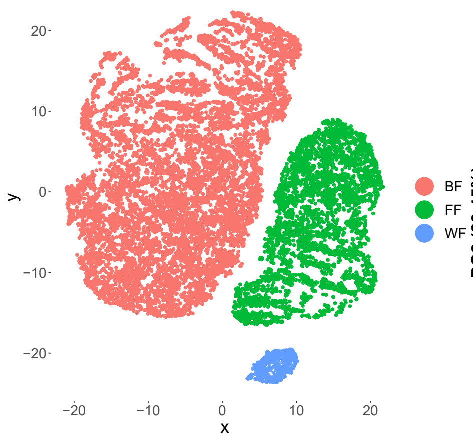
(a) t-SNE Plot (TW = 0.99)



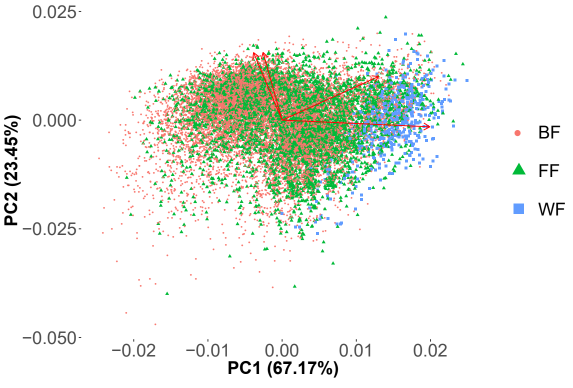
(b) PCA Plot

Fig. 3.4 t-SNE and PCA plot of the performance of all heuristics on all 16,000 evolved problem instances

A further dataset is created by combining instances selected from all four datasets just described (identified as DS5 shown in Table 3.1). This facilitates an investigation into whether the feature-based and feature-free models generalise across a mixed set of instances of different lengths with item weights drawn from different probability distributions and bounds. DS5 contains 4,000 instances with 1,000 instances selected from each dataset DS1-DS4. For each dataset, 250 instances were selected at random for each class (FF, BF, WF and NF), resulting in a balanced dataset containing equal numbers of instances from each class and each distribution. Table 3.1 provides a description of



(a) t-SNE Plot (TW = 0.97)



(b) PCA Plot

Fig. 3.5 t-SNE and PCA plot of the performance of all heuristics on all 16,000 instances after the ordering of items is randomised

each dataset. These datasets are available for other researchers working in the field of ASP to compare approaches[2].

3.4.2 Description of Random Datasets

1,000 new instances were randomly generated from each of the distributions that define datasets DS(1-4), denoted **R**DS(1-4). As just noted, these random instances are not evolved to be tailored for targeted heuristics. A further dataset RDS5 combines the instances in RDS(1-4). In addition, three additional datasets RDS(6-8) are created with new distributions, each containing 4,000 instances and 250 items per instance. RD6 is generated from a Poisson distribution with a mean equal to 50, as this has been previously suggested as a natural alternative to normal distributions for generating BPP instances [210–212]. RDS(7,8) are generated from a Weibull distribution [213] with shape parameter k equal to 0.454 and 1.044 respectively, as this distribution has been cited in the literature as more closely modelling the structure of real-world BPP benchmarks [210, 214].

Table 3.2 shows how many of the randomly generated instances are best solved by each of the 4 heuristics. It is immediately clear that a) the BF heuristic wins the majority of instances in all datasets while the NF heuristic fails to win a single instance in any dataset. The WF heuristic only wins instances in three random datasets RDS(1,3,5): all of these instances were generated from a Gaussian distribution with range [40-60]. The table also highlights that on the same datasets, the BF and FF heuristic tie as winners on large numbers of instances. A possible explanation for this is that the narrow range of values allowed results in similar instances. Fig 3.6 shows the t-SNE plot of the performance

[2]https://github.com/Kevin-Sim/BPP.

space using the different heuristics on RDS5, it is worth noting that this plot is similar to Fig 3.5a that shows the performance space of the randomised version of the evolved instances.

Table 3.2 Number of instances are best solved by each heuristics for RDS(1-8)

	FF	BF	NF	WF	BF = FF
RDS1	216	587	0	45	152
RDS2	265	735	0	0	0
RDS3	327	643	0	10	20
RDS4	178	822	0	0	0
RDS5	986	2787	0	55	172
RDS6	617	3382	0	1	6
RDS7	1407	2593	0	0	1
RDS8	1028	2972	0	0	0

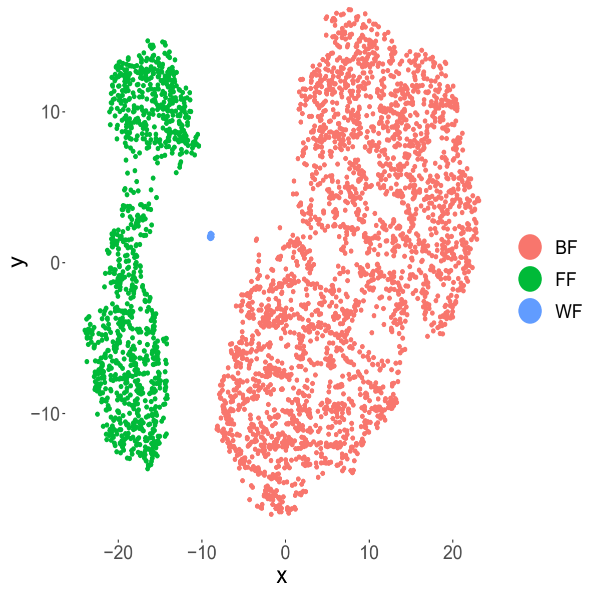


Fig. 3.6 t-SNE plot (TW = 0.94) of the performance of all heuristics on all 4,000 random instances from RDS5

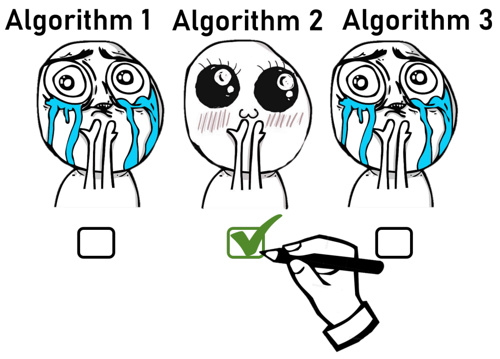
All datasets are significantly skewed toward BF: 73% of the instances are best solved by BF, 25% solved best using FF, and 2% using WF. However, these 25% FF instances produce results that are only very slightly higher than BF in terms of performance, so they can be solved using BF with minimal loss in performance.

3.5 Conclusions

Using either benchmark or random instances will result in unbalanced datasets with respect to the performance of the best-solving heuristics which hinders an investigation into the ASP using deep- and machine learning techniques. Therefore, this chapter introduced new and large streaming bin-packing datasets that are designed to facilitate the training when investigating the feature-free algorithm selection approach described in the next Chapter 4. This chapter made use of an evolutionary approach which maximises the Falkenauer's fitness performance difference between the target algorithm and the other algorithms used in the selection problem. Four datasets were generated for this purpose, each with 4,000 instances, in which each dataset contains exactly 1,000 instances uniquely solved best by each of the four heuristics (BF, FF, WF and NF) considered in the ASP pool. This provides balanced datasets that occupy different parts of the performance space w.r.t the heuristics chosen which in turn exhibit complementary performance. Having complementary performance is crucial to building good algorithm selectors as will be seen in the next Chapter 4. Also, 8 randomly generated datasets were introduced for investigating the generalisation ability of the proposed approach in the next chapter. All these datasets were analysed here, furthermore, the next chapter will provide more analysis in the light of the proposed feature-free ASP approach results.

Chapter 4

Automated Algorithm Selection: from Feature-Based to Feature-Free Approaches



As explained in Chapter 2, ASP refers to the task of automatically selecting an algorithm from a given pool. Although intensely studied over recent years, the vast majority of previous work in the ASP domain follows the cycle first outlined by Rice [1], shown in figure 2.1, in which an important first step is the extraction of a feature-set. However, as Rice himself, noted: “*The determination of the best (or even good) features is one of the most important, yet nebulous, aspects of the algorithm selection problem.*”. Extracting good features that correlated to algorithm performance remains a complex and hard task [102, 103, 46, 104].

This chapter proposes a novel feature-free approach for algorithm selection, applicable to optimisation domains in which there is implicit sequential information encapsulated in the data, e.g., in streaming bin-packing. Specifically, two types of recurrent neural networks are trained to predict a packing heuristic in streaming bin-packing, selecting from four well-known heuristics. As input, the RNN methods only use the sequence of item-sizes. This contrasts to typical approaches to algorithm selection (presented in Chapter 2) which require a model to be trained using domain-specific instance features that need to be first derived from the input data. In this research, restricting the algorithm space to a set of constructive deterministic heuristics allows for greater confidence when analysing the performance of the algorithm selectors than could be inferred by using sets of stochastic heuristics.

4.1 Contribution

The proposed approach provides the first example of applying a recurrent-neural network as an algorithm selection to data which has sequential characteristics. Although such networks have demonstrated ground-breaking performance on a variety of tasks that include image captioning, language translation and handwriting recognition [12], their applicability has not been exploited within the ASP domain. The contributions of this chapter are as follows:

- A novel feature-free algorithm selection approach using a recurrent neural network with either LSTM or GRU that avoids the need to identify features through training, using only the sequential information defining a problem instance.
- An extensive comparison of the feature-free approaches to feature-based approaches, using a wide range of features as input to six well-known classifiers, each tuned to ensure optimised performance.
- An investigation of the capability of both methods to generalise over new datasets with different characteristics to those on which they were trained.
- New insights into the structure of the feature-space and performance-space of a broad range of datasets from the 1D bin-packing domain that shed some light on the likely ability of a classifier to accurately perform algorithm selection.

4.2 Background and Motivation

A common approach to tackling the ASP is to treat it as a classification problem where each instance is described in terms of a vector of hand-designed features, and an instance's class indicates the best performing algorithm. Although there have been a number of successful studies using this method, e.g. [43, 215, 47, 50], the task of identifying appropriate features that correlate to algorithm performance is far from trivial in many domains: in some domains, specifying features is not intuitive, and it can be difficult to create a sufficient number to train a model, while in others in which there are many features, it is necessary to invoke feature-selection methods in order to choose appropriate features [45, 48] as the noisy and uninformative features prevent the selection techniques making intelligent decisions [176].

Feature-design is even more complex in domains in which the data has sequential characteristics. For example, in streaming bin-packing [216, 217] and streaming job-shop scheduling problems [44, 218], items/tasks arrive in a stream (one at a time) and have to be packed/assigned to a container/machine exactly in the sequence that they arrive. In such cases, it would be appropriate to derive features that capture the sequential information contained in the stream in order to be informative, but deriving such features is even more challenging than in the cases mentioned above.

As described in Chapter 2, one solution to dealing with sequential data can be found in the field of deep learning, where the use of RNN-LSTM [10] or RNN-GRU [11] to classify sequential data has become widespread in recent years; example applications include text classification [219], scene-labelling [220] and time-series classification [221]. Such networks directly use a sequence of data as input (e.g the size of the next item to be packed in bin-packing). In this sense they are ‘feature-free’ in that it is not necessary to derive auxiliary features from the data to train the network. The addition of the LSTM/GRU to the network enables a model to learn the long-term context or dependencies between symbols present in an input sequence, and also handles variable-length sequences of information. Therefore, this chapter proposes that an RNN-LSTM or RNN-GRU could be used as a *feature-free* classification technique to perform algorithm selection in optimisation domains in which there is sequential data[3]. In contrast to some previous research which *extends* Rice’s diagram to encapsulate a broader agenda relating to the relative power of algorithms (e.g. Smith-Miles et al. [48]), the proposed method shrinks Rice’s diagram through bypassing the feature extraction block. To be clear, in the context of this thesis, the term *feature-free* refers to the use of raw input data defining a problem instance as input to a selector, where there is no pre-processing of data required or a need to define and derive features *a-priori* from the problem data. This therefore addresses the associated issues outlined above. Also, this research restricts the algorithm space to a pool of deterministic heuristics, however, this document uses the phrase "algorithm selection" as it is the familiar phrase of this problem in the literature.

This chapter compares two RNN architectures, LSTM and GRU, to six different classical machine learning techniques that use derived features as input to provide better insight into the relative merits of deep learning RNN vs classical machine learning.

[3]in fact, it is possible to re-cast some optimisation problems that do not contain sequential information in sequence form; more details in the Conclusion.

An extensive evaluation is conducted using 5 benchmark datasets. Following this, the trained models are evaluated on four new datasets generated from distributions that are suggested in the literature to be more indicative of real-world instances [210]. Finally, this chapter conducts an analysis of the instance-space, feature-space and performance-space investigated by projecting the data into a low-dimensional space that can be easily visualised. This provides new insights into the structure of instances in each dataset, and sheds light on why some spaces are more likely to facilitate classification than others. The approach proposed in this chapter is considered offline algorithm selection where the selectors are trained offline before testing them on unseen before problem instances. The BPP scenario considered in this ASP approach has been described in the previous Chapter 3, however, the next section presents it again as a reminder.

4.3 Streaming 1D-BPP Problem Instances

To remind the reader, in the previous Chapter 3, new and large datasets totalling 16,000 instances for the streaming version of the 1D-BPP domain have been produced via an evolutionary approach in which each instance is solved best by *exactly one* of four heuristics in a pool used in the algorithm selection problem, showing that the new instance elicits diverse behaviour with respect to the performance space. These datasets were created in accordance with the commonly accepted view in the literature that in order to develop better algorithm selection methods, datasets should exhibit complementary performance with respect to the solvers in question [7]. These benchmark BPP datasets DS(1-5) and RDS(5-8) are used in the research presented in this chapter.

4.4 Models: A Deep Learning Model and a set of Classical Machine Learning Models for Algorithm Selection

As outlined in Chapter 2, conventional machine learning techniques used for the ASP tend to consider vectors of features as input to classical machine learning models. Candidate features typically describe spatial or statistical characteristics, with little consideration to ordering or sequential information describing an instance. On the one hand, this ignores potentially valuable sequential information that could improve algorithm selection, while on the other, limits the applicability of standard approaches on streaming data where features need to be calculated dynamically.

This chapter proposes a model applicable to domains in which data has a fixed ordering that explicitly considers the ordering as input to an algorithm selection technique, that uses a method borrowed from the deep learning literature. Deep learning has achieved groundbreaking results in applications where the input is formatted as time-series data or in domains where sequences have specific orderings but without any explicit notion of time [12]. Examples including video and image recognition, natural language processing, music generation and speech recognition [13–15].

4.4.1 Deep Learning Feature-Free Model

As RNNs are specifically designed to learn from data where sequential information explicit in the order of the stream is used to identify relationships between the data (a sequence of bin-packing items) and the expected outputs from the network, i.e. the predicted packing heuristic from the pool. In this chapter, a specialised RNNs is known as a LSTM [10] and GRU [11] are used where they have been shown to be efficient and effective in learning long-term dependencies from sequences of ordered data, as described in Chapter 2.

4.4.2 Feature-based machine learning models

Six well-known classical machine learning techniques [20] from the literature are evaluated: Neural Network (NN); Decision Trees (DT); Random Forest (RF); Naive Bayes (NB); k-Nearest Neighbours (KNN) and Support Vector Machine (SVM). Each is used in its standard form, parameterised using grid-search and Bayesian optimisation as described in section 4.5. The success of ASP methods that rely on feature extraction depends critically upon the chosen features [51] and on the classification method used to learn the correlation between the features and best-performing heuristic. The next subsection outlines a set of 10 features that are used in the research presented in this chapter.

4.4.3 Definition of features

The traditional method of dealing with ASP is to hand-design features and then extract them from the instance data. Here, a set of features collated from multiple papers in the literature is used where algorithm selection methods have been applied to bin-packing [118, 105, 57], discussed in Chapter 2. The features are given below:

1. Mean item size divided by bin capacity C [57, 105, 118, 117];
2. Standard Deviation (std) in the item sizes divided by bin capacity C [57, 105, 118, 117];
3. Maximum item size divided by bin capacity C [57, 117];
4. Minimum item size divided by bin capacity C [57, 117];

5. Median item size divided by bin capacity C [57];
6. Maximum item size divided by minimum item size [57];
7. The ratio of small items of size $\omega \leq C/4$ [107];
8. The ratio of medium items of size $C/4 < \omega \leq C/3$ [107];
9. The ratio of large items of size $C/3 < \omega \leq C/2$ [107];
10. The ratio of huge items of size $\omega > C/2$ [107].

It should be clear that these features do not provide any information about the sequential characteristics implicit in the data-stream that describes each instance (i.e. the order of items to be packed). Rather, they describe statistical properties relating to the distribution of the item sizes. The ten features were extracted for each of 16,000 instances contained in DS(1-5) and used as input to the classical machine learning models.

4.5 Experiments and Results

Both the Keras[4] and the Sklearn [5] libraries are used to implement the models used in the DL and classical ML experiments. A Keras implementation of LSTM and GRU is used in "sequence-to-one mode" where input is an ordered list of item weights and output is a "one-hot" encoding using 4 bits to identify the best heuristic (1000 = BF, 0100 = FF, 0010 = NF, 0001 = WF). The Sklearn library is used to implement the classical ML models, with the exception of the NN model which is implemented using Keras using a one-hot encoding output. All the classical models take 10 features as input as defined in section 4.4.3. Experiments are conducted on Google Colab[6] with Tensor Processing Unit (TPU) run-time used to execute the experiments. A preliminary empirical investigation was conducted to tune both the LSTM and NN architectures and hyper-parameters. Also, Fig A.1 in the appendixes shows a schematic description for the GRU model (LSTM has a similar schematic) with its layers and the trainable parameters number. The "Adam" optimiser [222] was selected due to its reported accuracy, speed and low memory requirements.

Due to the time limitation and the large number of hyper-parameters for LSTM/-GRU and NN, preliminary investigations (i.e. manually/empirically) were undertaken to

[4]https://github.com/fchollet/keras.
[5]https://github.com/scikit-learn/scikit-learn.
[6]https://colab.research.google.com/notebooks/welcome.ipynb.

optimise the LSTM[7] and NN approaches. LSTM and GRU experiments used 300 and 700 learning iterations for DS(1,2) and DS(3-5) respectively, since the longer instances were found to require more learning iterations. On the other hand, both grid search and Bayesian optimisation are used to optimise the remaining traditional machine learning techniques to choose the best set of hyper-parameters for each ML model. Tables A.1 and A.2 in the appendix show the range of hyper-parameters evaluated and the final selected parameters for each of the LSTM, GRU, NN and ML models.

This chapter conducts independent experiments using each of the 5 datasets to train and test the LSTM, GRU and classical ML models to predict the best heuristic for each instance. Each dataset was shuffled (i.e. the order of the instances in each set was randomised) and then split into training (80%) and test (20%) sets while maintaining the balance in the size of each target class (each test set has 800 instances where 200 are solved best by each heuristic). Each model was trained using 10-fold cross-validation using 10% of the training instances to verify each fold. The best model from each experiment was then tested on the corresponding test set comprising 800 unseen instances[8].

The best model from the 10 trained models is selected for evaluation on the test set based on the accuracy to select the best performing heuristics on the validation set. The best model selected from the training models is evaluated on an unseen test set in terms of the four following aspects (these aspects also described in details in the next section 4.6),

1. Comparison of algorithm classification accuracy, i.e. the percentage of instances classified correctly.
2. Comparison of selectors' performance to SBS and VBS using Falkenauer's performance metric (Equation 2.2).
3. Comparison of selectors' solution quality using the number of used bins b' by the predicted heuristic.
4. Evaluate the ability to generalise based on the above criteria with a wide range of different unseen datasets.

In addition to evaluating the accuracy, the total number of bins in a solution is also reported, summed over all instances in a dataset, when the heuristic returned by the model for each instance is used to create the solution. As the number of bins per instance in a given dataset can vary widely, to avoid issues that occur when summing data that has

[7]GRU has not been tuned: the best LSTM hyper-parameters are used with GRU since LSTM and GRU are very similar.

[8]A better way might be to re-train on the full training set after assessing the generalising error using cross-validation.

different scales, the total number of bins is normalised according to equation 4.1 and sum the normalised values.

$$PercentageOfBins = \frac{b' - b}{b}, \text{where } b = \left\lceil \sum_{j=1}^{n} (\frac{\omega_j}{C}) \right\rceil, \text{ and } b \leq b' \leq 2b \tag{4.1}$$

As described in Chapter 2 in Section 2.4.2, the worst-case performance ratio for the heuristics considered is equal to double the lower bound, then Equation 4.1 returns a value between 0 and 1.

Friedman test [223] followed by Nemenyi post-hoc test [224] with 5% confidence level are used to evaluate significance in a pairwise fashion for all comparisons of Falkenauer's performance and bins used. Friedman test has been chosen since it is a distribution-free test (non-parametric version of ANOVA) used to determine whether group of dependent samples (i.e. sets of performance or number of bins for the same instances) were selected from populations having the same distribution. Nemenyi post-hoc test has been chosen to calculate pairwise comparisons for sets of data that differ significantly based on results from conducting Friedman test.

4.6 Results

This section discusses in details the four aspects that are used to evaluate each best model selected from the training models on an unseen test set.

4.6.1 Accuracy of Algorithm Selection

As mentioned in section 4.5, 10-fold cross-validation is used to train the models taking 10% of the training instances to verify each fold. Table A.3 in the appendix shows results achieved on the *validation sets*[9] used during training for the DL models (LSTM and GRU) and all 6 ML models. Table 4.1 shows the results achieved on each test set with the best model obtained from training. The classification accuracy is reported as an indicator of the LSTM's, GRU's and ML's predictive abilities. Significance is calculated in a pairwise fashion using Friedman test followed by Nemenyi post-hoc test between the DL techniques and VBS, SBS and best of the ML techniques for each dataset, and between the best ML technique and the SBS. p-values are shown in table A.4 in the appendix. Table 4.2

[9]Significance has not been tested between the folds results since the same hyper-parameters are used for each fold. It might be required to check the significance when using cross-validation for hyper-parameter tuning which is not the case here.

additionally shows the comparison between the LSTM, GRU, the best ML technique and the SBS over the different test sets.

Table 4.1 Classification accuracy of the best model in each experiment from the test set, values in *italic* indicate the best ML results and values in **bold** the best overall result per dataset. Significance at the 5% confidence level is indicated as follows (using Falkenauer's performance): $*$ indicates that the DL model performed significantly worse than the VBS; $\diamond$ indicates that the DL model performed significantly better than the SBS; § indicates that the DL model performed significantly better than the best ML model; $\bullet$ indicates that the best ML model performed significantly better than the SBS.

	DL		ML					
Dataset	LSTM	GRU	NN	DT	RF	SVM	NB	KNN
DS1	**82.38%**$^{*\diamond§}$	81.38%$^{*\diamond§}$	65.38%	63.38%	*67.50%*$^{\bullet}$	66.13%	61.63%	64.50%
DS2	91.50%$^{*\diamond§}$	**92.75%**$^{\diamond§}$	56.88%	54.50%	57.75%	*58.63%*$^{\bullet}$	55.63%	54.50%
DS3	**83.38%**$^{*\diamond§}$	82%$^{*\diamond§}$	*67.37%*$^{\bullet}$	67.37%	65.50%	66.50%	54.25%	64.13%
DS4	95.75%$^{\diamond§}$	**96.25%**$^{\diamond§}$	72.13%	67.50%	73%	*73%*$^{\bullet}$	64.25%	66%
DS5	83.38%$^{*\diamond§}$	**85.37%**$^{*\diamond§}$	61.62%	61%	*64.12%*$^{\bullet}$	53.25%	43.12%	57.38%

Results obtained from deep learning LSTM and GRU approaches are significantly better than the results using classical ML techniques. It has been observed that the DL results on DS2 and DS4 are better than on the other datasets suggesting that *the sequential* correlations are easier to find in the datasets comprised of item weights generated from a wider range of values following a uniform distribution. It might be that the instances that are evolved with items in the range [40-60] may have fewer distinct patterns (i.e. sequential information) than instances generated with items in the range [20-100], regardless of the length of the instances. It is well known that problems with an average weight of $\frac{C}{3}$ are more difficult to solve [79] and it is interesting that problems with those characteristics are more difficult to classify using LSTM or GRU, i.e. instances from DS(1,3) with item weights generated from a narrow range of values [40,60] which is an average of one-

Table 4.2 The comparison between the LSTM, the best ML technique and the SBS over the different test sets in terms of performance using the Friedman test followed by Nemenyi post-hoc test with 5% confidence level. The $\uparrow$ means the left-hand approach of the given pair has a better median; $+$ indicates significance ($\ll 5\%$) and $-$ indicates no significance ($\gg 5\%$).

	Falkenauer's Performance				
	LSTM-SBS	LSTM-ML	GRU-SBS	GRU-ML	ML-SBS
DS1	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$
DS2	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$
DS3	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$
DS4	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$
DS5	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$

third of bin capacity 150. Although all the ML results show relatively poor performance compared to the DL methods, it does comparatively better, in most cases, on the two longer datasets DS3 and DS4 than the shorter ones DS1, DS2 on the validation and test sets. It might be that the longer instances result in more distinct values for the extracted features, hence increasing the ability of the ML techniques to classify correctly.

Although training the LSTM and GRU on the longer instances requires significantly more learning iterations before the models converge[10], it is interesting to note that for both distributions the results obtained by the LSTM and GRU on the longer instances (e.g. DS4) exceed those reported on the datasets with smaller numbers of items (e.g. DS2). It has been conjectured that the longer instances provide the DL models with more sequential information, hence increasing the ability to determine patterns in the item sequences. The ML results partially concur with the LSTM and GRU results in this respect, i.e. results on DS3 are more accurate than DS1 (apart from RF, NB and KNN) and those for DS4 are more accurate than for DS2. The results of the LSTM and GRU experiments conducted on the combined DS5 set show intermediate results with accuracy between that achieved on the experiments on instances with uniform distribution and those with Gaussian distribution. These models are able to generalise over instances sampled from all of the problem lengths and the different weight distributions investigated without any apparent loss of precision. In contrast, the ML experiments conducted on the combined DS5 sets show worse results than most of the other experiments. This demonstrates that the feature-based approach is weak in its ability to generalise over instances with different characteristics and highlights its reliance on the quality of the designed features, in contrast to the LSTM and GRU approaches.

Table 4.3 presents confusion matrices of the LSTM, GRU and best ML techniques extracted from the experiments conducted on the test set DS5, the rest of the confusion matrices are shown in table A.6 in the appendix. In most cases, the DL models were most frequently confused when attempting to classify the sequences identified as being solved best by FF and BF. It is interesting to note that the analysis in the previous chapter 3 showed that these two algorithms are extremely close in terms of their two largest principal components in a space defined by a 4-d vector containing the performance metric of each of the 4 heuristics for each instance. Similarly, instances labelled as NF appear to be the easiest to identify and correspondingly are the most isolated in the performance-space. As noted previously in Chapter 3, it appears that the patterns shown by conducting a PCA of the performance space are correlated with the ability of LSTM to identify the best performing algorithm from the raw instance sequences. On the other hand, the ML models are frequently confused between FF and BF (similarly to LSTM and GRU), but the ML models are also confused between FF and WF. In terms of the accuracy to classify NF,

[10]Convergence has been assessed manually, i.e. without using early stopping.

Table 4.3 The Confusion Matrix of the LSTM, GRU and Best ML models in experiments on DS5 from the test set

	LSTM				GRU				RF			
Heuristic	BF	FF	NF	WF	BF	FF	NF	WF	BF	FF	NF	WF
BF	160	36	0	4	162	37	0	1	145	39	1	15
FF	48	133	0	19	45	136	0	19	46	94	13	47
NF	0	0	198	2	0	2	195	3	9	25	147	19
WF	2	13	9	176	2	4	4	190	12	40	21	127

ML results partly concur with the DL results, in that NF instances are easier to classify only in DS1 and 3.

Purely out of curiosity and for comparison purposes, the feature-free proposed approach is compared to conventional machine learning techniques using the ordered list of item weights as input. The purpose of this is to investigate whether conventional ML classifiers can learn anything from feature-free input. 10-fold cross-validation is conducted on DS4 using only the item sizes information defining the original instances (i.e. without knowledge-driven feature extraction) directly supplying all item sizes at once to two of the classical ML techniques: the Multi-Layer Perceptron (MLP) and the Random Forest (RF) classifiers. The ML techniques were used directly from Weka [225] without altering any of the default parameters. The RF achieved 67.55 % accuracy. MLP was equally successful, achieving 67.08% accuracy. Although better than expected, results show that a DL approach that has been designed to work with sequential data provides more informative results.

4.6.2 Comparison to SBS and VBS

Figure 4.1 shows cumulative distribution plots over the test sets of 800 instances from each DS(1-5) in terms of Falkenauer's fitness: the plots show the percentage of instances that are solved with a distance d_p of the oracle-like VBS (the perfect mapping) given the solver predicted by a model. The distance d_p is calculated as:

$$d_p = (VBS_{Falkenauer'sfitness} - Selector_{Falkenauer'sfitness}) \geq 0, \text{ and } d_p = 0 \text{ is the optimal} \tag{4.2}$$

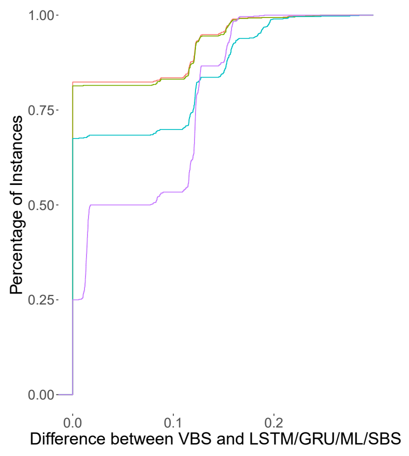
For all experiments the DL selectors significantly improve on both the SBS (BF for all datasets) and the classical ML techniques, and are very close to the VBS, P-values are shown in table A.4 in the appendix. Results show that the RNN-LSTM/GRU solves between 81.5%-97.1% of the instances within 5% of the VBS performance, compared to 50%-62.2% using the SBS. This represents a 27.5% to 35% improvement on DS(1-5). The best ML technique solves 68%-81.5% instances with a 5% performance difference in all the datasets which is an 11.5%-19.3% improvement over the SBS. Using either LSTM or GRU, between 99.6% and 100% of the instances can be solved within 30% of VBS for DS(1-5): in contrast, using the best ML technique, this is only true on DS(1,3). From the figure, it is clear that the best ML technique can only solve 100% of instances if a performance difference of 40% is considered in DS(2,4,5). It is interesting to note that although the SBS solves a lower percentage of instances than both the DL and the best ML technique if a threshold of 5% of the VBS performance is considered, it manages to solve all the instances at 20% of the VBS in all the datasets. Although the ML techniques improve over the SBS in most cases, they are not as good as the LSTM/GRU and are not as close to the VBS as the DL methods.

It is noticeable from the evolved instances in Chapter 3 that the performance of the heuristics is skewed towards FF and BF, i.e. these heuristics are either the best or second-best choice for all instances. This means that even if the selector misclassified most of the NF and WF instances by choosing FF or BF, it will still achieve high performance. Therefore, as well as comparing the two approaches based on the performance metric and classification accuracy, they are additionally compared in terms of the number of bins used to pack a set of items to get a clearer understanding of their relative performance.

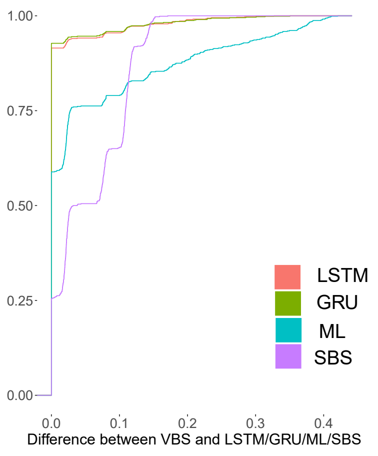
4.6.3 Evaluation of Solution Quality

The overall objective of the BPP is to minimise the number of bins used to pack a set of items. Ultimately, the number of containers defines the cost of any real-world solution. Table 4.4 shows the number of bins required to pack all 800 test instances for each DS(1-5) and contrasts this against the lowest possible number of bins used by the VBS and the number of bins needed using the algorithms predicted by LSTM, GRU and traditional ML selectors. The LSTM and GRU use between 1.2% and 2.3% fewer bins than the SBS and between 0.2% and 1.4% more than the VBS. On DS4, GRU uses over 1000 bins fewer than the SBS and only 192 (0.2%) more than the VBS which uses 83,823 bins. While the ML techniques use between 1.6% more bins and up to 0.9% fewer bins than the SBS and 1.6% to 4.5% more than the VBS.

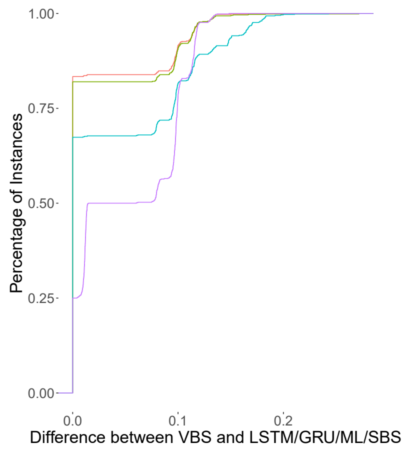
After normalising the number of bins b' used by the predicted algorithm (described at the end of section 4.5), Friedman test followed by Nemenyi post-hoc test are used to evaluate significance in a pairwise fashion for all comparisons of bins. Table A.4 in the appendices shows the p-values of b' in a pairwise fashion to the bins used by the SBS and the VBS. Based on these p-values, Table 4.5 shows the comparison between the LSTM, GRU the best ML technique and the SBS over the different test sets in terms of bins. Figure 4.2 shows the cumulative distribution plots over the test sets of 800 instances from



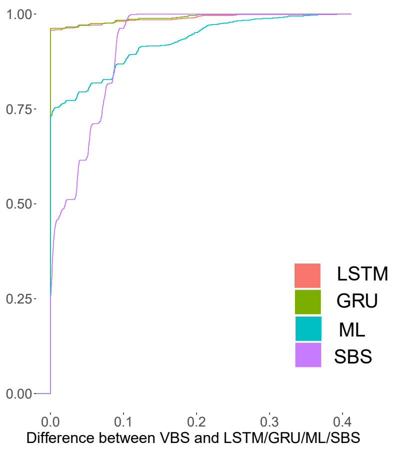
(a) DS1 - Falkenauer's Fitness



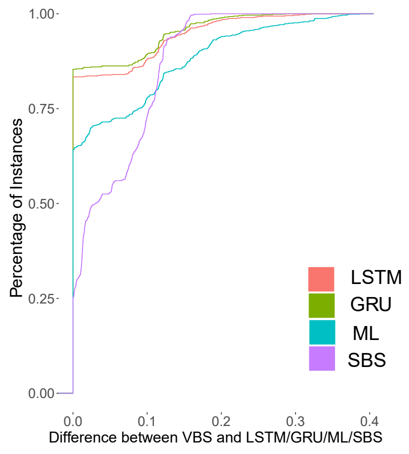
(b) DS2 - Falkenauer's Fitness



(c) DS3 - Falkenauer's Fitness



(d) DS4 - Falkenauer's Fitness



(e) DS5 - Falkenauer's Fitness

Fig. 4.1 Cumulative distribution plots over the test sets of 800 instances of each DS(1-5) to evaluate LSTM and GRU predictors VS classical ML techniques, SBS and VBS on performance space using Falkenauer's fitness metric

each DS(1-5) in terms of the number of bins, where the difference d$_b$ is calculated as:

$$d_b = (Selector_{usedbins} - VBS_{usedbins}) \geq 0, \text{ and } d_b = 0 \text{ is the best.} \tag{4.3}$$

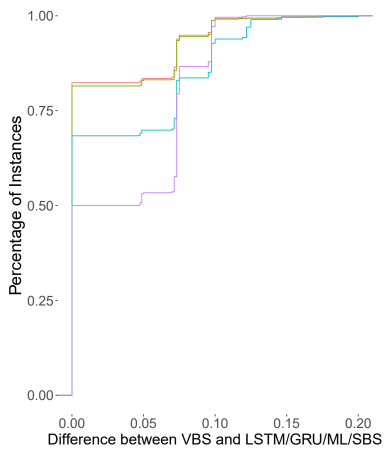
In terms of the percentage of instances that are solved by each method, LSTM/GRU solves 81%-98.2% within 5% VBS while the best ML technique solves 69%-86% and 53%-89% by SBS for DS(1-5). For higher values of differences from the VBS, the results depend on the dataset used. On DS(1,3), 99.1%-99.8% of the instances are solved by LSTM/GRU compared to 94% solved by ML within 10% of VBS and both techniques solve all the instances within 20% VBS. On DS(2,4,5), 99.3%-100% instances solved by LSTM/GRU while 92%-98.2% by best ML technique within 20% VBS that solves all the instances within 40% VBS. The SBS manages to solve 99.6% to all instances within 10% VBS on DS(1,3) and within 20% VBS on DS(2,4,5).

Table 4.4 Total Bins required to pack instances in the test set for the LSTM predictor, GRU predictor, classical ML techniques, traditional heuristics and VBS, values in *italic* indicate the best ML results and values in **bold** the best overall result per dataset and BF is the SBS over all the datasets.

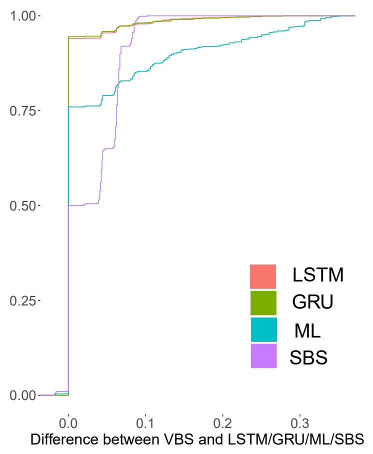
	Heuristics					DL		ML					
DS	FF	BF(SBS)	WF	NF	VBS	LSTM	GRU	NN	DT	RF	SVM	NB	KNN
DS1	34815	34722	35357	38357	33439	**33908**	33929	34417	34552	*34379*	34423	34489	34427
DS2	41494	41062	43031	47961	39929	40133	**40128**	41735	41593	41349	*41317*	41062	41502
DS3	71839	71741	73312	79498	69638	**70359**	70441	*71264*	71341	71410	71345	71562	71425
DS4	85677	85166	89435	100439	83823	84061	**84016**	85241	85982	85455	*85349*	85750	85736
DS5	58536	58242	60345	66672	56772	57525	**57371**	58683	58689	*58273*	58773	58577	58481

Table 4.5 The comparison between the LSTM, the best ML technique and the SBS over the different test sets in terms of number of bins using the Friedman test followed by Nemenyi post-hoc test with 5% confidence level. For each given pair, the $\uparrow$ means the first approach's median is better, $+$ means there is significance, $\downarrow$ means the first approach's median is worst and $-$ means there is no significance.

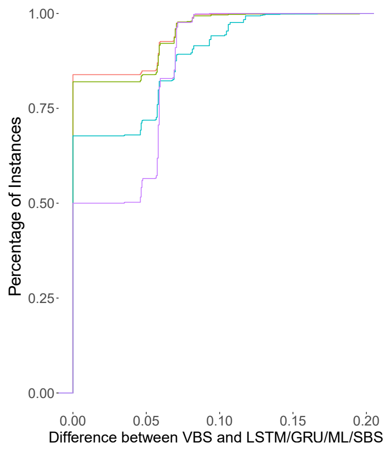
	Bins				
	LSTM-SBS	LSTM-ML	GRU-SBS	GRU-ML	ML-SBS
DS1	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$
DS2	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$
DS3	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$
DS4	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow -$
DS5	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow +$



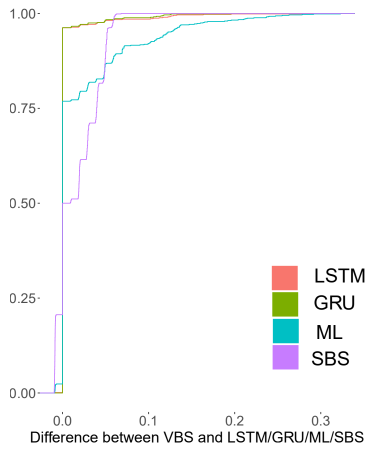
(a) DS1 - Number of Bins



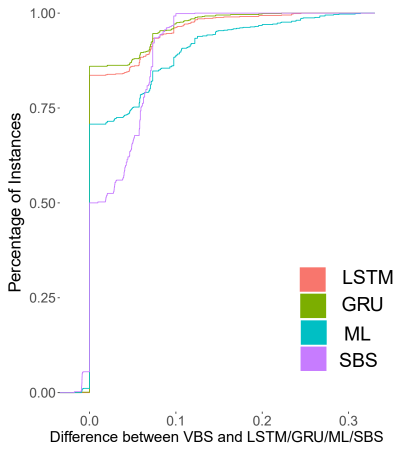
(b) DS2 - Number of Bins



(c) DS3 - Number of Bins



(d) DS4 - Number of Bins



(e) DS5 - Number of Bins

Fig. 4.2 Cumulative distribution plots over the test sets of 800 instances of each DS(1-5) to evaluate LSTM and GRU predictors VS classical ML techniques, SBS and VBS on performance space using number of bins metric

4.6.4 Evaluation of the ability to generalise

In the previous section, the novel feature-free and the traditional feature-based approaches of ASP were compared using balanced instance sets tailored for targeted heuristics. This section aims to investigate how well the trained models produced from these approaches generalise to completely new instances, generated at random from the same distributions as the evolved datasets. Hence, there is no a-priori knowledge of which heuristic will best solve them — or whether there is even a uniquely best solver for each instance. The datasets RDS(5-8), described in Chapter 3 are used in this set of experiments. Features are extracted from these datasets exactly as described in section 4.4.3 in order to train the classical ML models.

Table 4.6 shows the classification accuracy obtained from each model on RDS(5-8), while figure 4.3 shows the cumulative distribution plots of the performance difference between the VBS and the predicted heuristic of each of the SBS, LSTM, GRU and best ML selectors in terms of both Falkenauer's fitness and the number of bins for 800 randomly chosen instances from dataset RDS5 (to be consistent with the previous experiments). The same pattern is observed on all the random datasets and hence the remainder of the results are omitted. It is clear that although the *classification accuracy* results are very poor in comparison to those obtained on the evolved datasets, in fact, the performance in terms of Falkenauer's fitness of the LSTM and GRU is very close to the VBS, where 80.5%, 89% of the instances are solved within 5% of VBS using the LSTM and GRU respectively on RDS5. The accuracy is affected by the fact that FF and BF have very similar performance, therefore it is easy to predict the incorrect heuristic; on the other hand, as their performance is so similar, in terms of actual bins needed to pack the items, confusing these two heuristics makes little difference to overall performance. From the classical models, the Naive-Bayes method also obtains very good performance(the best ML technique on RDS5) where it solves 65.5% of the instances within 5% of the VBS in terms of Falkenauer's fitness. It is interesting to note that the best ML technique solves all the instances within 20% of the VBS superseding a bit the used DL models where they solve 98.8% of the instances in terms of the number of bins on RDS5.

Table 4.6 The classification accuracy obtained from each model on RDS(5-8) on a test set of 4000 unseen instances

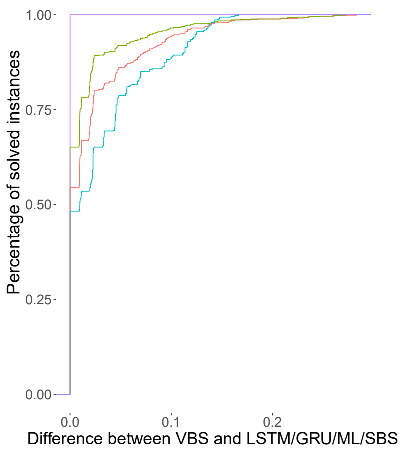
	DL		ML					
	LSTM	GRU	NN	DT	RF	SVM	NB	KNN
RDS5	31.32%	39%	18.78%	21.02%	24.03%	21.86%	33.49%	28.10%
RDS6	4%	24.9%	5.1%	4.4%	4.9%	0.02%	0.62%	8.1%
RDS7	27.5%	22.7%	17.8%	17.9%	21.3%	11%	19.9%	21.3%
RDS8	21.8%	22.1%	8.6%	11%	12.6%	12.2%	28.5%	11%

Table 4.7 (constructed based on p-values of tables A.4 and A.5 in the appendices) conducts statistical testing to evaluate three hypotheses:

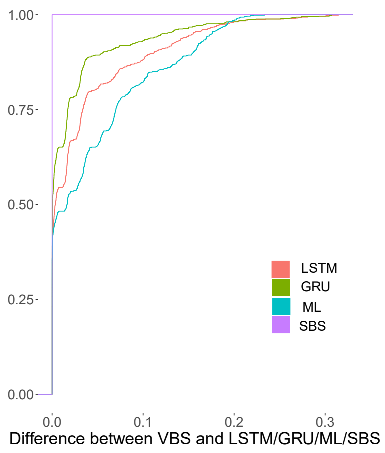
- $\mathscr{H}_0(1)$: the LSTM/GRU and best ML method produce equal results with respect to a) Falkenauer's fitness metric b) total bins utilised.
- $\mathscr{H}_0(2)$: the LSTM/GRU and SBS produce equal results with respect to a) Falkenauer's fitness metric b) total bins utilised.
- $\mathscr{H}_0(3)$: the best ML method and SBS produce equal results with respect to a) Falkenauer's fitness metric b) total bins utilised.

Friedman test obtains p-values = $0 \ll 5\%$, i.e. reject the null hypothesis that the median performance is the same for all the methods mentioned. Then the p-values of Nemenyi post-hoc test determine exactly which methods have different medians. In terms of Falkenauer's fitness metric, using GRU, in two out of four of the randomly generated datasets RDS(5,6), $\mathscr{H}_0(1)$ can be rejected and median testing shows that the GRU outperforms the best ML method, while it can not be rejected on RDS(7). Using LSTM, $\mathscr{H}_0(1)$ can be rejected only on RDS7 and it outperforms the best ML method, while it can not be rejected on RDS(5,6). On RDS8, the null hypothesis is rejected but the ML method is superior to LSTM/GRU. However, on all four random datasets, both $\mathscr{H}_0(2)$ and $\mathscr{H}_0(3)$ are rejected: both the ML/DL methods are out-performed by the single best-solver. In terms of the number of bins, then the same pattern is observed w.r.t the SBS. $\mathscr{H}_0(1)$ can be rejected in two out of four random datasets with the LSTM and GRU providing superior results. $\mathscr{H}_0(1)$ can not be rejected on RDS(6) (for LSTM) and RDS(7) for GRU. Similar to Falkenauer's fitness, on RDS8, the null hypothesis is rejected but the ML method is superior to LSTM/GRU. For clarity, this table also repeats the results obtained on the evolved datasets, to indicate the stark contrast between the results obtained.

In summary, the results presented indicate that a deep learning method is clearly superior to using a classical prediction method trained with extracted features. Furthermore, it has been inferred that the choice of the deep method itself has no significant effect (as shown in Table 4.8), i.e. it is the switch to a learning method that captures sequential information that provides the gain in performance. Most of the cases in Table 4.8 (which is created based on the p-values in Tables A.4 and A.5) have no significant difference between the LSTM and GRU performance on the test set in terms of Falkenauer's Fitness and number of bins used. Thus, the subsequent chapters in the thesis will use the original LSTM network to investigate the proposed approaches.



(a) RDS5 - Number of Bins



(b) RDS5 - Falkenauer's Fitness

Fig. 4.3 Cumulative distribution plots over 800 random instances from *R*DS5 to evaluate LSTM and GRU predictors VS classical ML techniques, SBS and VBS using number of bins and Falkenauer's fitness metrics

Table 4.7 The comparison between the feature-free approach using LSTM/GRU, the feature-based approach using traditional ML techniques and the SBS over the different datasets in terms of performance and number of bins using Friedman test followed by Nemenyi post-hoc test with 5% confidence level. For a given pair of tests, the ↑ means the first approach's median is better, + means there is significance, ↓ means the first approach's median is worst and − means there is no significance.

	Falkenauer's Performance					Bins				
	LSTM-SBS	LSTM-ML	GRU-SBS	GRU-ML	ML-SBS	LSTM-SBS	LSTM-ML	GRU-SBS	GRU-ML	ML-SBS
DS1	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +
DS2	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +
DS3	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +
DS4	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ −
DS5	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +	↑ +
RDS5	↓ +	↑ −	↓ +	↑ +	↓ +	↓ +	↑ +	↓ +	↑ +	↓ +
RDS6	↓ +	↑ −	↓ +	↑ +	↓ +	↓ +	↑ −	↓ +	↑ +	↓ +
RDS7	↓ +	↑ +	↓ +	↑ −	↓ +	↓ +	↑ +	↓ +	↓ −	↓ +
RDS8	↓ +	↓ +	↓ +	↓ +	↓ +	↓ +	↓ +	↓ +	↓ +	↓ +

Table 4.8 The comparison between the feature-free approaches using LSTM and GRU over the different datasets in terms of performance and number of bins using the Friedman test followed by Nemenyi post-hoc test with 5% confidence level. For a given pair of tests, the $\uparrow$ means the first approach's median is better, $+$ means there is significance, $\downarrow$ means the first approach's median is worst and $-$ means there is no significance and $\Longleftrightarrow$ means both approaches have same median.

	Falkenauer's Performance	Bins
	LSTM-GRU	LSTM-GRU
DS1	$\uparrow -$	$\Longleftrightarrow -$
DS2	$\downarrow -$	$\Longleftrightarrow -$
DS3	$\uparrow -$	$\Longleftrightarrow -$
DS4	$\Longleftrightarrow -$	$\Longleftrightarrow -$
DS5	$\downarrow -$	$\downarrow -$
RDS5	$\downarrow +$	$\downarrow +$
RDS6	$\downarrow +$	$\downarrow +$
RDS7	$\uparrow +$	$\uparrow +$
RDS8	$\uparrow -$	$\Longleftrightarrow -$

4.7 Analysis and Discussion

Referring to the spaces in Rice's diagram (Fig 2.1) and to the commonly accepted view in the literature [50] that in order to develop better algorithm selection methods, datasets should exhibit complementary performance with respect to the solvers in question, this section evaluates the datasets' *diversity* with respect to three elements: (1) the *feature-space* created by extracting the 10 described features from each instance; (2) the *instance-space* defined by the fixed order of item-sizes defining an instance; (3) the *performance-space*, defined by a four-dimensional vector of the Falkenauer fitness metric obtained by each of the 4 heuristics applied to the instance. These spaces are visualised using t-SNE technique [208] for non-linear dimensionality reduction. In order to assess the *trustworthiness* of the t-SNE plots, the metric proposed in [209] is used. Conceptually, the trustworthiness score captures errors in the preservation of neighbourhoods or proximity relationships in the new projection of the data, and is quantified by the metric defined by [209] on a scale of 0 to 1, where 1 represents maximum trustworthiness. The trustworthiness value TW is noted on each of the plots presented in the figures.

4.7.1 Insights from performance space

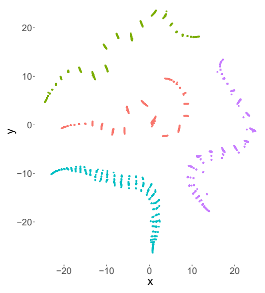
In figure 4.4, visualisations using t-SNE on two datasets are provided. The first is DS3 (see table 3.1), which was specifically evolved to be diverse with respect to the *performance* space. For contrast, this subsection also visualises a dataset in which instances are generated at random, but using the same descriptors as DS3 in terms of length and distribution. Instances are coloured according to the heuristic that obtains the best result on the instance. In the case of the randomly generated instances, BF wins on 643 instances, FF 327, WF 10 and NF fails to win on any instances, therefore only 3 colours are visible on these diagrams.

It is clear from figure 4.4b that the performance-space is diverse with respect to the 4 heuristics from the evolved dataset with TrustWorthiness TW=0.99. Also, the randomly generated instances (Fig 4.4d) exhibit some separability (but less diversity) at least between two classes (TW=0.85). Although the performance-space is much as expected and underlines the need to evolve separable instances for training, these plots give no insight into whether LSTM/ML models might be able to classify using the instances/features data as input.

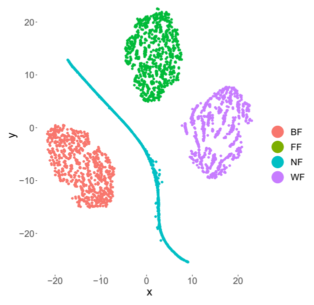
4.7.2 Insights from Feature-space and Instance-space

Figures 4.4a and 4.4c visualise the feature-space of the two sets of instances using t-SNE. There is an observable structure in the feature-space of hand-designed features using the non-linear embedding technique t-SNE with both sets, TW = 0.97 and 0.90 for DS3 and RDS3 respectively. This non-linear projection might explain why the ML techniques, which are linear techniques, cannot discriminate between the instances using these features. Also, depending on the datasets used e.g. DS4 (Fig 4.5d) and RDS6 (Fig 4.5b), there are multiple distinct clusters per heuristic which might make it harder for the ML methods to map a heuristic to multiple different parts of the space.

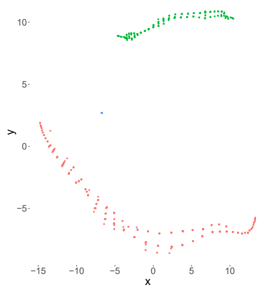
These figures suggest that a fruitful line of future work would be to move towards automatic feature-design methods such as auto-encoders [226] and parametric t-SNE [227]. The feature-space shows that it is possible to *auto-design* features from *hand-designed features* (i.e. with the presence of expert input) and create separability using a non-linear approach e.g. t-SNE. On the other hand, the instance-space shows it is possible to auto-design features from the *raw* instance data but this gives less separability than the "features from features" method. Although both methods suggest the possibility of training a feature-based method for ASP with appropriate effort in creating useful auto-designed features, the LSTM/GRU works *without* any effort in this respect. Note that t-SNE is also applied to the instance-space of the respective datasets using distance-measures explicitly designed to capture the distance between *sequences*. However, a projection has not been obtained in which the trustworthiness TW was greater than 0.62. Hence, these plots have been omitted from the thesis. However, it is interesting to remark that LSTM/GRU (which uses only instances as input) can discriminate between the instances even though the t-SNE can not find any discernible separability in the instance-space.



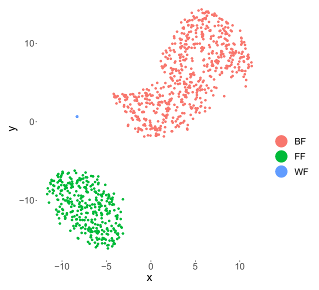
(a) Feature space DS3 (TW=0.97)



(b) Performance space DS3 (TW=0.99)

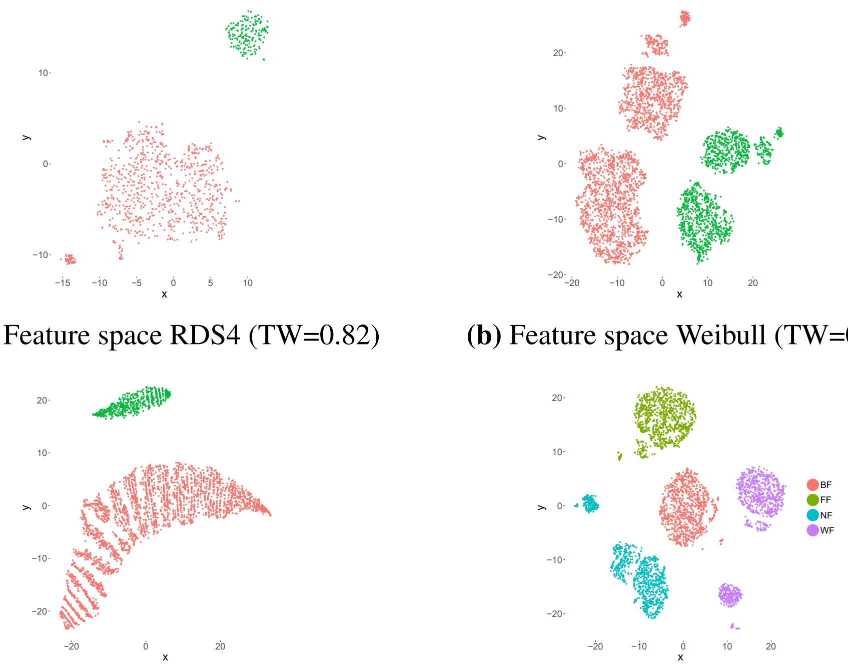


(c) Feature space *R*DS3 (TW=0.90)



(d) Performance space *R*DS3 (TW=0.85)

Fig. 4.4 t-SNE Visualisation of Feature space and Performance space. Each point represents an instance, coloured by the heuristic that best solves it. *First row*: DS3 — instances evolved by [24] with 250 items generated from a Gaussian distribution with lower/upper bounds 40-60. *Second row*: instances generated at random, with same length and from same distribution as DS3. Datasets are indicated as *R*DS3.



(a) Feature space RDS4 (TW=0.82) (b) Feature space Weibull (TW=0.92)

(c) Feature space Poisson (TW=0.97) (d) Feature space DS4 (TW=0.90)

Fig. 4.5 t-SNE of Instance space and Feature space for the randomly generated datasets (RDS4, Weibull, Poisson) and for the evolved dataset DS4 as comparison

4.7.3 Generalisation

In general, the results show that the randomly generated instances are not very discriminatory, as has previously been reported for other problems [228, 46]. The proposed approach findings concurred with [228] and [46] that randomly generated instances have limitations in terms of diversity and therefore, they are often quite restrictive as a means of developing new ASP methods. Fig 4.5 shows how the feature space moves from less structured instances to more structured ones. Table 4.7 provides an overall summary that clearly illustrates the comparison between the feature-free approach using LSTM/GRU, the feature-based approach using traditional ML techniques and the SBS over the different all datasets in terms of Falkenauer's fitness performance and the number of bins. It is clear that the novel approach using LSTM or GRU proposed in this chapter outperforms the classical approach using ML techniques in the majority of cases.

4.8 Conclusions

This chapter has described a novel approach to algorithm selection for sequential optimisation problems that exhibit an ordering with respect to the elements of the problem and how they should be dealt with. Unlike most ASP techniques, the approach does *not* require the design and selection of features to describe an instance. Two deep-neural networks (RNN-LSTSM and RNN-GRU) were trained using the sequence of items representing an instance directly as input to predict the best algorithm to solve the instance. This chapter has compared this feature-free approach with traditional feature-based approaches using ten hand-designed features and six classical ML techniques. Both the novel and the traditional approaches were thoroughly evaluated on 5 different large datasets, exhibiting different numbers of items and different distributions of item-sizes. All classifiers were trained using a large database of instances in which each instance has a distinct best-solver, previously described in Chapter 3.

On the artificially generated datasets using EA[11] (described in Chapter 3), the accuracy of the LSTM and the GRU models ranges from 82-96% while the accuracy of the ML models ranges from 43%-73%, depending on the dataset used. In terms of the percentage of the instances that are solved using DL predictors within a small difference of the VBS Falkenauer's performance ($\leq 5\%$), results show between 27.5%-35% improvement over the Single Best Solver (SBS), depending on the dataset used. On the other hand, the best ML predictor ranges from 11.5%-19% improvement over the SBS. This is the first time that such an approach has been used, and represents a significant step forward in algorithm selection for problems with sequence information, where the difficulties associated with defining suitable features and selecting from large sets of potential features are well understood.

It is worth mentioning that the proposed approach can also be applied to domains that do not naturally have sequence information through artificially transforming them into sequences. For example, in the TSP domain, an instance is defined by a set of coordinates while a solution is a sequence (ordered series of visits). In order to use the proposed approach, a TSP map could be scanned in a fixed pattern providing a sequence of coordinates representing the order of the cities that appear on a map, implicitly encapsulating spatial information. Sequences produced in this manner could then be used to train an LSTM or GRU.

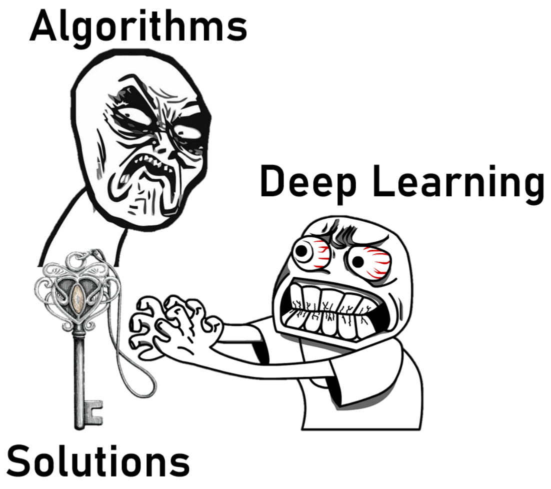
The analysis also revealed that a promising line of future work would be to move towards automatic feature design methods using non-linear techniques such as parametric t-SNE. Results found that automatically designing features from hand-designed features provides a good separability in feature-space. On the other hand, auto-designing features from the raw instance data using t-sne provides less separability than applying t-sne directly to the hand-designed features. Note however that although these auto-designed features can be used to train feature-based methods for ASP, the proposed approach works *without* any effort in this respect.

[11]These datasets have been generated using EA to have maximum performance difference of the heuristics under investigation in the performance space which might be not the case in the real-world datasets.

As described so far in this thesis, the vast majority of previous work in the ASP domain follows the cycle first outlined by Rice [1], shown in figure 2.1, in which an important first step is the designing of a feature-set then determining algorithms and the final step is to learn a mapping between the feature-space $\mathscr{F}$ and the algorithm-space $\mathscr{A}$. In this chapter, the first step (i.e. feature designing) has been bypassed. In addition to bypass the feature designing step, the next chapter bypasses the need to select the best performing algorithm by predicting the best solution directly from the problem information. Having this shortcut in the ASP diagram transfers the research from algorithm selection into algorithm generation. The next chapter introduces a preliminary study of a novel approach to generate heuristics automatically using deep learning.

Chapter 5

A Deep Learning Approach to Predicting *Solutions* in Streaming Optimisation Domains



In the field of combinatorial optimisation, per-instance algorithm selection remains a challenging problem, particularly with respect to streaming problems such as packing or scheduling. As shown in Chapter 2, the vast majority of previous work in the ASP domain follows the cycle first outlined by Rice [1], shown in figure 5.1, in which an important first step is the designing of a feature-set then determining algorithms and the final step is to learn a mapping between the feature-space $\mathscr{F}$ and the algorithm-space $\mathscr{A}$. The previous chapter 4 addressed both the issue of feature-extraction and that of streaming data in proposing a deep learning approach (RNN-LSTM or RNN-GRU) that learned from implicit sequential information present in the problem instances to predict the best performing heuristic. The approach was "feature-free", i.e. it did not require the design and selection of features to describe an instance. The approach was shown to provide promising results in the domain of 1D-BPP.

The proposed approach in this chapter goes beyond that of the previous chapter in further pursuing the goal of abandoning the need to derive features in describing a deep learning model that directly predicts a *solution* from the instance data rather than the algorithm that best solves it. This radical approach bypasses algorithm selection altogether by training a deep learning model, encoder-decoder LSTM, using solutions obtained from a pool of heuristic algorithms[12]. Bypassing algorithm selection transfers the research from algorithm selection into algorithm generation where the trained deep learning models behave as heuristics that are generated automatically using solutions obtained from a pool of hand-designed heuristics from the literature. To validate the concept, experiments were conducted using a simplified version of the packing problem in which items arrive in batches where the generated heuristic produces solutions by making implicitly decisions per batch as a distinct problem.

5.1 Contribution

The major contribution of this chapter is to describe a radical approach that bypasses algorithm selection altogether by training a deep learning model using solutions obtained from a pool of heuristic algorithms to directly predict a *solution* from the instance-data. The new method is rigorously evaluated on a streaming bin-packing problem by:

1. Conducting an investigation of the ability of an encoder-decoder LSTM to behave as a generated heuristic that directly predicts an optimised solution from a stream of data describing a problem-instance.

2. Conducting an investigation of the scalability of the proposed approach with respect to the length of the stream.

3. Conducting an investigation of the accuracy of the trained model (i.e. generated heuristic) with respect to the diversity of the solution-space used to train the model.

4. Conducting an investigation of the capability of the generated heuristics to generalise to new datasets containing randomly generated instances with similar characteristics to those on which they were trained.

[12]It should be clear that the approach still requires greedy selection of the best heuristic in the pool for each problem instance in order to collect the solutions. However, once the models are trained on the solutions, then there is no need for algorithms.

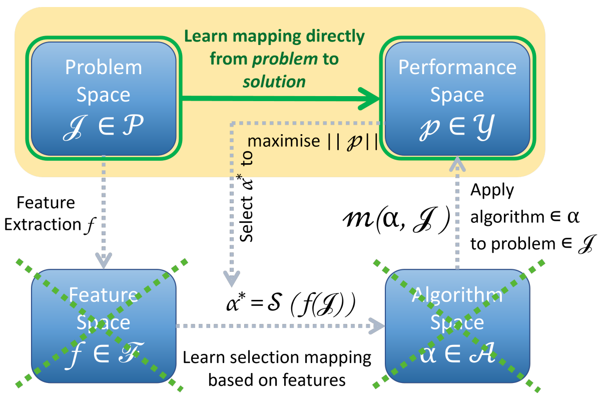


Fig. 5.1 The traditional schematic of the Algorithm Selection Problem [1], with proposed modifications shown in green

5.2 Background and Motivation

For a representative set of problem instances $\mathscr{I}$, the algorithm selection process normally includes three main tasks: (1) extracting useful features f; (2) determining/designing good algorithms(s); (3) selecting an appropriate mapping technique $S\,(f(\mathscr{I}))$. However, as shown in previous Chapter 4, it is well known that obtaining useful features is not an easy task and often requires expert input which is not always available [102, 103, 46, 104]. This is even more challenging with problems that have a sequential nature, for instance, job-shop scheduling [44, 218] and streaming bin-packing [216, 217], where tasks/items continually arrive to be assigned to a machine or packed in a particular bin in the same sequence that they arrive. In such cases, features should capture the sequential information present in the stream, which cannot be achieved using classical statistical approaches for feature definition. Step (2) (determining/designing algorithms) is generally more straightforward, particularly given recent advances in the field of automated algorithm design [40], for instance using genetic-programming or grammatical-evolution to design new algorithms and heuristics. However, the final step of learning a mapping between the feature-space $\mathscr{F}$ and the algorithm-space $\mathscr{A}$ is also non-trivial, usually involving feature-selection and tuning of a classification model.

Although the difficulties associated with feature-selection can be partially mitigated by the approach proposed in the previous Chapter 4 to bypass the feature-designing task by capturing the sequential information implicit in a stream and using this as direct input to a deep learning algorithm to learn a mapping from *instance to algorithm*, here a more radical solution is proposed: train a deep learning algorithm to *directly predict the solution to an instance from the instance-data itself.* The approach is illustrated in Figure 5.1, which shows Rice's original diagram modified according to this proposal. Rather than constructing a feature-space from the instance-data and then mapping this to the algorithm-space, the proposed approach learns a mapping directly from problem to solution for a given performance metric. The model is *trained* using solutions obtained by greedy selection of the best algorithm from the algorithm space $\mathscr{A}$ (which does not require the construction of the feature-space $\mathscr{F}$). The approach represents a paradigm-shift in the way in which algorithm selection is approached, in removing the need to select an algorithm altogether, and instead directly returning the best solution.

This shortcut in the traditional Rice Diagram moves the research from ASP into algorithm generation. As described in Chapter 2, approaches to generate heuristics automatically using ML or DL can be divided into (from the generated heuristics output) i) approaches to output solutions directly from the problem instance at once and ii) approaches to output decisions iteratively at each decision point when dealing with the problem instance. The proposed approach in this chapter follows the former type where the trained model behaves like an automatically generated heuristic that produces solutions directly from the problem instances.

Learning this direct mapping takes inspiration from the natural-language processing field in which an encoder-decoder LSTM[13] model has been demonstrated to give state-of-the-art performance on tasks such as text translation [11, 19], image captioning [229], conversational modelling [230], learning to execute programs [231] and movement classification [232], as described in Chapter 2. Unlike Vinyals et al. [180] who used a pointer network, which is an encoder-decoder LSTM with attention mechanism, to predict solutions directly (the optimal visited cities sequence) from spatial sequences as input, e.g. set of cartesian coordinates in TSP domain, the proposed approach predicts solutions from sequences (i.e. items to be packed in the order they arrive). Before describing how encoder-decoder has been adapted for use as a solution predictor, the next section describes the streaming BPP scenario considered in this chapter.

5.3 Streaming 1D-BPP Problem Instances

To prove the proposed approach is viable, a simplified version of a streaming problem is considered in the domain of bin-packing, in which items arrive in fixed-size *batches*. Once a batch is packed, the next batch is treated as a new sub-problem. Items have to be packed into a set of empty bins. Two datasets DS(1,2) that are described in Chapter 3 are used which have 120 items per instance that are initialised with item sizes drawn from two different distributions as shown in table 5.1. Note that these instances were specifically evolved to maximise the difference between the fitness of the best-solving

[13]Long Short Term Memory

heuristic and the next best heuristic, therefore each subset of instances that are best-solved by a particular heuristic are likely to have similar characteristics.

Table 5.1 Data set details. Bin Capacity is fixed at 150 [24]

DS	n_{items}	Lower - Upper Bounds	Distribution	total
DS1	120	[40-60]	Gaussian	4000
DS2	120	[20-100]	Uniform	4000

For the specific type of streaming problem considered, each instance is split into batches of size 6, 12 and 18 items per batch, resulting in 80,000, 40,000 and 24,000[14] batches respectively per dataset. The duplicate batches are deleted where those are found only using DS1 with a batch size of 6 items ($\approx 1,000$ cases). Then each batch is solved using each of n chosen heuristics (depending on the experiment). The solution with the best fitness according to the Falkenauer fitness [79] given in equation 2.2 is added to a training set. To generate solutions to these batches to use as training data for the model, this chapter considers the four hand-designed heuristics that were used in the previous Chapter 4 (BF, FF, NF and WF) which pack one item per time-step in the order they arrive (i.e. online heuristics), plus two additional hand-designed heuristics Best-Fit-Descending (BFD) and First-Fit-Descending (FFD) [105] from the literature that consider each batch in its entirety (i.e. offline heuristics), as described in Chapter 2.

For reference, Table 5.2 shows the percentage of the solutions solved best by each heuristic under investigation on DS(1,2) with window sizes 6, 12 and 18. The table also highlights the SBS, i.e the heuristic that achieves the best performance in solving solutions than any other heuristic in the pool. BF and BFD are the SBS in DS1[15] and 2 respectively, regardless of batch size. The next section provides a brief outline of how encoder-decoder has been adapted for use as a solution predictor with a numerical example.

5.4 Encoder-Decoder LSTM Architecture for Solution Prediction

This section describes the proposed model for providing a mapping from instance-data directly to a solution. Predicting a solution for a given problem instance is what is commonly known as a sequence-to-sequence problem (or seq2seq). As described in Chapter 2, these problems are challenging due to the fact that the number of items in the

[14] 120 items $\div$ 18 items per batch = 6.66 so 6 batches per instance are considered with batch size of 18 items $\times$ 4,000 instances = 28,000 batches.

[15] Although BF wins fewer solutions than the BFD on DS1 using batches of 6 and 12 items, BF achieves the best overall performance.

Table 5.2 Percentage of solutions solved best by each heuristic H per dataset and per batch size (BS), e.g. DS1-6H-6 means using DS1, six hand-designed heuristics and batch size of 6 items. Values in **bold** show the single-best-solver in the related dataset

DS-#H-BS	H1	H2	H3	H4	H5	H6
	BF	FF	NF	WF	BFD	FFD
DS1-6H-6	**34.85%**	0.03%	0.14%	2.08%	62.89%	0.00%
DS1-6H-12	**39.77%**	0.33%	2.14%	7.75%	50.02%	0.00%
DS1-6H-18	**44.60%**	1.00%	3.28%	13.58%	37.54%	0.00%
DS2-6H-6	32.63%	1.21%	2.09%	1.08%	**62.98%**	0.02%
DS2-6H-12	12.95%	1.69%	2.87%	2.24%	**80.16%**	0.09%
DS2-6H-18	7.72%	1.91%	1.93%	2.89%	**85.48%**	0.07%

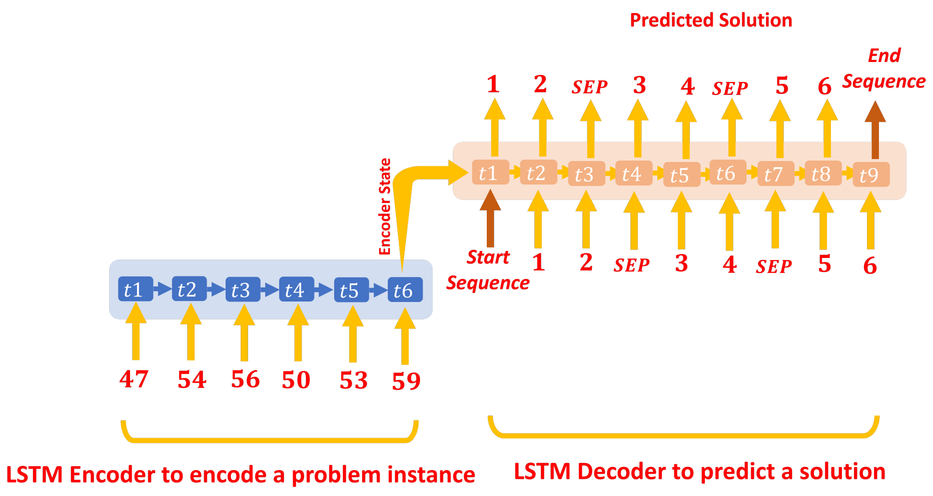
input and output sequences can be arbitrary and even from different spaces (e.g. statistical machine translation). However, Seq2Seq problems have been tackled in other domains using a deep learning architecture known as an encoder-decoder LSTM [11, 168] that has proven very effective and achieved state-of-the-art performance.

As described in Chapter 2, an encoder-decoder LSTM architecture contains two models. The **encoder** is used for mapping an input sequence (batch of items) into a vector of fixed dimensionality, and the **decoder** for outputting a predicted sequence (a solution) based on the encoder output. The whole process of encoder-decoder training and testing has been explained in Chapter 2. Training input data is represented as a fixed-length list of item-sizes in a batch and the provided output data is a fixed-length list of item-indexes. For a batch of size, e.g. $n = 6$, the network output is of the form shown below, which indicates which item in the batch is placed in which bin:

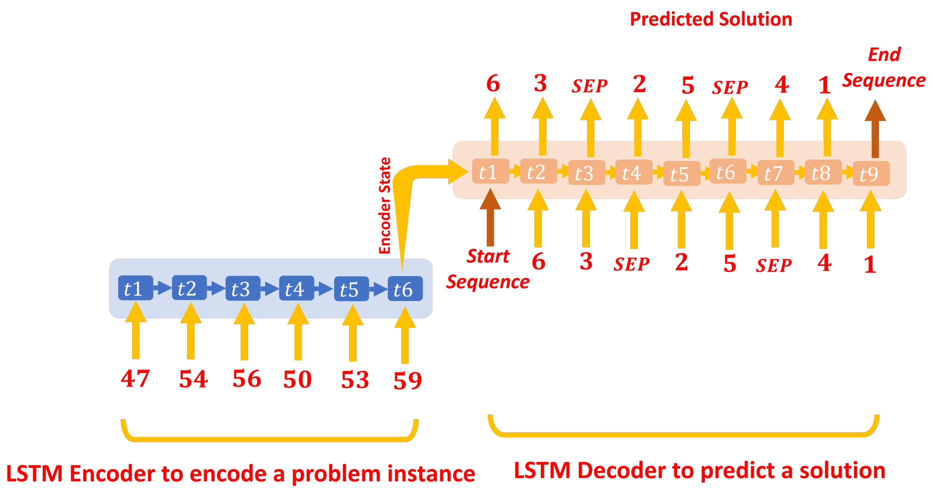
- Batch of items = [47, 54, 56, 50, 53, 59]
- Solution = [6, 3, *Sep*, 2, 5, *Sep*, 4, 1, *ES*]

The output is interpreted as follows: *place the 6th and 3rd items in bin 1, the 2nd and 5th items in bin 2, and the 4th and 1st items in bin 3.* A *separator* value is used to indicate breaks between bins. ES refers to the end of the sequence. Any values outside the range 1 to n (where n is the batch-size) could be chosen to represent the separator and ES. Here ES = 152 and Sep = 0 are selected; ES is a large value to make it out of the batch-size range, and the difference between the ES and Sep values is maximised to make it easier for the network to learn. Once the deep learning model is trained, the model behaves like a generated heuristic that is able to predict a solution for a given batch of items. Fig 5.2 shows two numerical examples of how the predicted solution would be when the model trained on solutions collected from either i) only online hand-designed heuristics (Fig 5.2a) or ii) a combination of online and offline hand-designed heuristics (Fig 5.2b). Fig

5.2 shows how the generated heuristic behaves differently depending on the solutions that are used for training. Also, the figure shows the *teacher forcing* technique to train this kind of networks as has been described in Chapter 2.



(a) A Generated heuristic from solutions produced by only online hand-designed heuristics



(b) A Generated heuristic from solutions produced by combination of online and offline hand-designed heuristics

Fig. 5.2 The trained deep learning models behave as generated heuristics to predict solutions directly from a batch of six bin-packing items

5.5 Experiments and Results

The Keras functional API [16] is used as an implementation of the LSTM where the input is a list of item weights and the output is a solution. The input and output are both one-hot

[16]https://github.com/fchollet/keras

encoded [168]. As the predicted solutions vary in length due to the variable number of separator tokens used to delineate bins, solutions are padded to a fixed length by repeatedly appending the value End sequence ES = "152". Table B.1 in the appendixes shows the input and output length of the encoder-decoder after the padding process. Experiments are conducted on Google Colab[17] with GPU run-time used to execute the experiments. A preliminary empirical investigation was conducted to tune the encoder-decoder LSTM architecture and hyper-parameters using the ranges shown in table B.2 in the appendixes. Also, Fig B.1 in the appendixes shows a schematic description of the encoder-decoder LSTM model (used in experiment #1 in Table 5.3 as an example). The "Adam" optimiser [222] was used in all tuning experiments due to its reported accuracy, speed and low memory requirements. Note that further preliminary investigation considered various options for the representation of the output other than that described in the previous section. These included predicting the list of items per bin at each time step and predicting the size of the item to be packed per time step. However, based on these early experiments, it was clear that predicting the item's index per time step using separators to delineate bins provided the most promising results.

Each dataset was split into a training set (80%) and a test set (20%). Training sets were created for DS(1,2) using batches of size (6, 12 and 18), i.e. 64,000, 32,000 and 19,200 batches respectively. Similarly, testing sets were created for DS(1,2) using the same batches sizes, i.e. 16,000, 8,000 and 4,800 batches respectively. Each training set contains a list of input-output pairs in which the input is a list of item-sizes in the batch, and the output is a solution represented as a list of indices of items, with a separator delineating bins. For training, the solution associated with an input is the best solution found from greedily applying H heuristics, according to Falkenauer's fitness function. Experiments were conducted that varied both the size and composition of the pool of H heuristics used to select a solution. All experiments were repeated ten times. For each experiment, the model that provides the lower error from the training phase is saved. The trained models are tested on the test set, and additionally on two new random datasets each of which includes 800 randomly chosen instances (i.e. 16,000, 8,000 and 4,800 batches for 6, 12 and 18 batch sizes respectively) from datasets RDS(1,2) (described in Chapter 3) that are created from the same properties of the original datasets, see Table 5.1.

To evaluate the results, the following metrics are considered:

- **Accuracy:** The percentage of solutions that are exactly the same as the solutions in the test set.

[17]https://colab.research.google.com/notebooks/welcome.ipynb

- **Validity:** The percentage of solutions that are *valid* in the sense that (1) all items in the batch are represented exactly once in the solution and (2) no constraints regarding bin-capacity are broken.
- **Bilingual Evaluation Understudy (BLEU):** This metric is commonly used in the natural language processing field for evaluating the quality of translated text [233]. It measures the overlap between the predicted solution and the ground truth, returning a value between 0 and 1 indicating the similarity between 0%(0) and 100%(1).
- **LSTM >BSS:** How many valid solutions are *better* than the Best Solver Solution (BSS) (in terms of the Falkenauer fitness metric or number of bins used). This metric provides insight into whether the model is 'creative' in the sense that it can produce novel, high-quality solutions not generated by the heuristics used in training.
- **LSTM <BSS:** How many valid solutions are worse than the Best Solver Solution (BSS) (in terms of the Falkenauer fitness metric or number of bins used).

5.6 Results

This section describes the results to the questions posed in the introduction in turn.

5.6.1 Encoder-Decoder LSTM Performance

The first experiment conducted uses the simplest possible training set in which only one heuristic in the pool is used to create solutions for a batch, i.e. the model only has to learn a mapping between instance and the solutions generated by a single specific heuristic. This is motivated by the idea that a given heuristic follows the same pattern in solving a set of instances, and therefore provides a simpler task for the model to learn. the BFD heuristic is used in these experiments as it wins the majority of instances in each dataset (Table 5.2). Results are shown as experiment #1 in Tables 5.3 and 5.4.

The results show that the encoder-decoder LSTM model can be used as a learner (or an automatically generated heuristic) to predict an accurate solution for a given problem instance obtaining 97% and 88% accuracy for test sets of DS(1,2) respectively. One notices that the results from experiments on DS2 (Table 5.4) are lower than the ones on DS1 (Table 5.3). One reason might be that the DS2 instances are created using a wider range of item-size values [20, 100] compared to the DS1 instances which have item-sizes in the narrower range [40, 60]. This wider range makes the problem instances

Table 5.3 The mean and std over ten runs on the test sets from DS1 using models trained using different combinations of the heuristics H[1-6] and different window or batch sizes in terms of Falkenauer's performance and number of bins used

							Falkenauer's performance		Number of Bins	
#	Window size	#Heuristics	Heuristics	Acc	Valid	BLEU	LSTM >BSS	LSTM <BSS	LSTM >BSS	LSTM <BSS
1	6	1	H5	97.69% (+/- 0.53%)	98.66% (+/- 0.49%)	98.24% (+/- 0.38%)	2 (+/- 2)	140 (+/- 21)	0 (+/- 0)	25 (+/- 12)
2	6	2	H[1, 5]	91.41% (+/- 0.27%)	95.89% (+/- 0.40%)	92.83% (+/- 0.25%)	10 (+/- 4)	564 (+/- 56)	1 (+/- 1)	83 (+/- 18)
3	6	2	H[1, 4]	98.85% (+/- 0.55%)	99.40% (+/- 0.55%)	99.18% (+/- 0.40%)	5 (+/- 3)	66 (+/- 42)	1 (+/- 1)	12 (+/- 6)
4	6	4	H[1-4]	98.76% (+/- 0.68%)	99.29% (+/- 0.52%)	99.13% (+/- 0.46%)	3 (+/- 1)	64 (+/- 45)	1 (+/- 1)	16 (+/- 14)
5	6	6	H[1-6]	90.71% (+/- 1.12%)	95.63% (+/- 0.71%)	92.05% (+/- 0.94%)	10 (+/- 2)	633 (+/- 60)	1 (+/- 1)	79 (+/- 18)
6	12	4	H[1-4]	63.00% (+/- 1.01%)	80.31% (+/- 2.08%)	79.89% (+/- 0.59%)	67 (+/- 13)	1222 (+/- 180)	3 (+/- 1)	185 (+/- 51)
7	18	4	H[1-4]	27.56% (+/- 1.49%)	48.51% (+/- 1.69%)	60.75% (+/- 0.96%)	50 (+/- 5)	919 (+/- 52)	2 (+/- 2)	191 (+/- 25)

Table 5.4 The mean and std over ten runs on the test sets from DS2 using models trained using different combinations of the heuristics H[1-6] and different window or batch sizes in terms of Falkenauer's performance and number of bins used

							Falkenauer's performance		Number of Bins	
#	Window size	#Heuristics	Heuristics	Acc	Valid	BLEU	LSTM >BSS	LSTM <BSS	LSTM >BSS	LSTM <BSS
1	6	1	H5	88.12% (+/- 0.43%)	94.48% (+/- 0.47%)	91.78% (+/- 0.26%)	7 (+/- 3)	937 (+/- 44)	0 (+/- 0)	169 (+/- 20)
2	6	2	H[1, 5]	68.47% (+/- 0.72%)	87.10% (+/- 0.89%)	77.58% (+/- 0.48%)	29 (+/- 7)	2436 (+/- 112)	1 (+/- 1)	273 (+/- 41)
3	6	2	H[1, 4]	85.59% (+/- 0.51%)	93.14% (+/- 0.61%)	89.95% (+/- 0.36%)	46 (+/- 7)	1087 (+/- 38)	1 (+/- 1)	229 (+/- 24)
4	6	4	H[1-4]	84.42% (+/- 0.79%)	92.57% (+/- 1.29%)	89.12% (+/- 0.50%)	25 (+/- 5)	1189 (+/- 88)	0 (+/- 0)	233 (+/- 36)
5	6	6	H[1-6]	66.17% (+/- 0.51%)	86.59% (+/- 0.56%)	75.90% (+/- 0.37%)	26 (+/- 6)	2686 (+/- 91)	1 (+/- 1)	278 (+/- 29)
6	12	4	H[1-4]	25.21% (+/- 2.35%)	53.41% (+/- 2.89%)	58.51% (+/- 1.64%)	105 (+/- 13)	2089 (+/- 125)	1 (+/- 1)	478 (+/- 44)
7	18	4	H[1-4]	4.14% (+/- 0.62%)	20.31% (+/- 2.62%)	41.63% (+/- 1.17%)	33 (+/- 4)	736 (+/- 95)	1 (+/- 1)	253 (+/- 48)

more difficult for the encoder to summarise and thus potentially the model requires more encoder layers.

98% and 94% of the predicted solutions are *valid* in DS1 and 2 respectively. On both datasets, the BLEU metric returns values in between the accuracy and validity values. Perhaps surprisingly, a very small number of 'creative' solutions are produced, i.e solutions that outperform the hand-designed heuristic used to generate the training set in terms of Falkenauer's performance metric and the number of bins used; however, the models (i.e. generated heuristic) also produce some solutions that are lower in performance than the BSS.

5.6.2 Impact of Training with Diverse Heuristics

Experiments #2 to #5 in Tables 5.3 and 5.4 aim to study how the various metrics are influenced by the diversity of the heuristics used to create the training set, i.e. whether training on a dataset created using multiple diverse heuristics makes it harder or easier to learn an effective model. 4 different hand-designed heuristic pools are considered:

- (BF, BFD): this pool contains two heuristics that have very different characteristics (the former packing a single item in the order of appearance, the latter sorting the entire batch before packing).
- (BF, WF): these heuristics have similar characteristics, both packing a single item.
- (BF, FF, NF, WF): all the heuristics that pack a single item in order of appearance.
- (BF, FF, NF, WF, BFD, FFD): all heuristics.

Note that as (BF, FF) produce very similar results these experiments did not test this combination (with the same applying to (BFD, FFD)). Also, the combination (BF, NF) is not evaluated given that NF solves only a few of the instances.

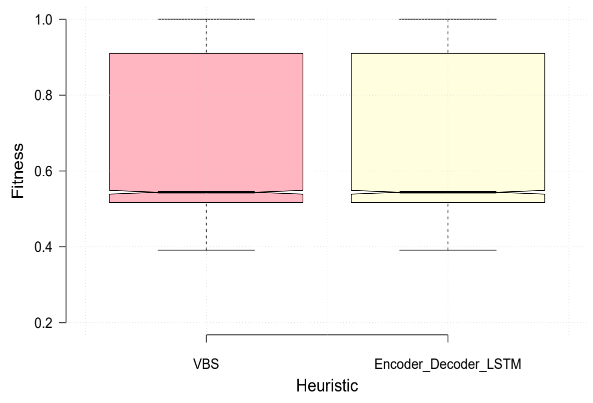
These experiments show that training with solutions created from diverse heuristics that have different characteristics in terms of the methods they use to create a solution reduces the ability of the learner (generated heuristic) to predict accurate and valid solutions (e.g. comparing experiment #2 to #1) — a learner with more memory units is likely required to cover this diversity. In contrast, solutions that are created using hand-designed heuristics with similar characteristics (experiments #3,4) obtain similar results to experiment #1.

From the evaluation metrics perspective, these experiments show that accuracy and validity appear correlated. The decrease in accuracy for experiment #2 (c.f. experiments #1,3,4) is accompanied by a small increase in the number of *new* solutions found that are better than any produced by the heuristics used to create the test data[18]. However, this also appears to be accompanied by a significant increase in the number of solutions obtained that are worse than the best solver used to create the test data.

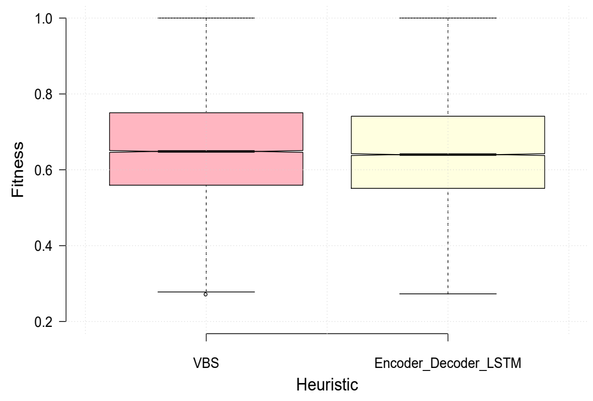
Fig 5.3 shows the performance distributions achieved by the VBS (i.e. the greedy selection of the solution created using the best heuristic for each batch) and by the proposed encoder-decoder LSTM learner for all valid solutions on the test sets DS(1,2), using batch-sizes of 6 items and considering a pool of 4 heuristics (i.e. experiments #4). For these two experiments, the trained learner obtains a very similar performance to the VBS.

Table 5.5 shows the comparison based on the p-values (in Table B.3 in the appendixes) obtained from applying a Wilcoxon Signed-Rank Test [206] with 5% confidence level to evaluate significance in a pairwise fashion of the encoder-decoder LSTM approach to VBS. Falkenauer's performance metric (equation 2.2) and the number of bins are used to evaluate solutions quality. The null hypothesis that the VBS and encoder-decoder LSTM

[18]Obviously a model that generates novel solutions that are better than those contained in the test set must have reduced accuracy as this metric measures the percentage of solutions that are identical to those in the test-set.



(a) DS1 using 4H and batch of 6 items, i.e. experiment #4 in Table 5.3



(b) DS2 using 4H and batch of 6 items, i.e. experiment #4 in Table 5.4

Fig. 5.3 Evaluating encoder-decoder LSTM predictor VS VBS for the test set from DS(1,2)

achieve the same results on DS1 in terms of the number of bins cannot be rejected, while the null hypothesis is rejected otherwise. It should be noted that Wilcoxon Signed-Rank Test is a rank-sum test (not median tests) and thus it is possible for the ranks to differ but the medians to be the same.

Table 5.5 The comparison between encoder-decoder LSTM and VBS for the test set from DS(1,2) (experiments #4 in Tables 5.3 and 5.4) in terms of Falkenauer's performance and number of bins using Wilcoxon Signed-Rank Test. For LSTM-VBS pair of test, the $\downarrow$ means the first approach's median is worst, $-$ means there is no significance and $\Longleftrightarrow$ means both approaches have same median.

	Falkenauer's performance	Number of bins used
DS1	$\downarrow +$	$\Longleftrightarrow -$
DS2	$\downarrow +$	$\Longleftrightarrow +$

5.6.3 Effect of batch-size

Experiments #4, #6 and #7 in tables 5.3 and 5.4 aim to analyse the effect of the batch size on the solution prediction. These experiments use batches of size 6, 12 and 18 items per instance[19]. These experiments are trained using solutions obtained from the pool of 4 heuristics [1-4] given that this setting returned the best result in the previous experiments.

The results show that the larger batch-size in both DS(1,2) reduces the ability of the learner to produce accurate and valid solutions: a more complex encoder(i.e. more layers) is required to address this. However, although the accuracy is decreased with the larger batches, the results show a corresponding increase in the models' creativity (especially in terms of Falkenauer's performance metric), i.e. its ability to produce better solutions than the heuristics it was trained on. As in previous experiments, this is also accompanied by a significant increase in solutions which are worse than BSS however. Unlike in experiments #2 to #5, the Bleu metric has better values than accuracy and validity in experiments #6 and #7 which means although the solutions do not map exactly to those produced by the training heuristics, there is good overlap with the original solutions.

5.6.4 Generalisation to new instances

The final series of experiments investigates the ability of the trained models (generated heuristics) from the experiments in Tables 5.3 and 5.4 to generalise to a new dataset of unseen instances. These instances are generated at random from a distribution with the same parameters as DS1, DS2 respectively. However, recall that the instances in DS1, DS2 were specifically evolved to maximise the difference between the fitness of the best-solving heuristic and the next best heuristic, so are likely to have particular characteristics that are not apparent when a random ordering of items is used to generate the instance.

Tables 5.6 and 5.7 show the results of testing the previously trained models on new unseen instances. Experiments #1 to #6 in these tables show that the models trained on the non-random datasets DS1 and DS2 have excellent ability to generalise over the new problem instances RDS1 and RDS2, obtaining results that only deviate by a maximum of 2% in terms of accuracy when compared to the previous results presented in tables 5.3 and 5.4. Experiments #7 and #8 using the larger batch-size show accuracies that are reduced by up to 7% in comparison to the non-random datasets, although it is noted that these models performed relatively poorly on the original datasets.

In general, the results presented here concur with what Vinyals et al. [180] concluded in their research using pointer network to predict solutions (i.e. optimised tours) for TSP instances. Similar to the work here, they found that the trained pointer network would

[19]These sizes are chosen as on average each bin can fit two or three items

Table 5.6 The mean and std results of the trained models on RDS1 (120-N-40-60-*Random*) using different combinations of the heuristics H[1-6] and different window or batch sizes in terms of Falkenauer's performance and number of bins used

							Falkenauer's performance		Number of Bins	
#	Window size	#Heuristics	Heuristics	Acc	Valid	BLEU	LSTM >BSS	LSTM <BSS	LSTM >BSS	LSTM <BSS
1	6	1	H5	97.19% (+/- 0.32%)	98.39% (+/- 0.36%)	97.83% (+/- 0.21%)	1 (+/- 2)	171 (+/- 36)	0 (+/- 0)	26 (+/- 9)
2	6	2	H[1, 5]	88.91% (+/- 0.50%)	94.42% (+/- 0.61%)	90.73% (+/- 0.36%)	6 (+/- 4)	683 (+/- 56)	0 (+/- 0)	66 (+/- 19)
3	6	2	H[1, 4]	98.79% (+/- 0.53%)	99.37% (+/- 0.57%)	99.11% (+/- 0.38%)	9 (+/- 2)	66 (+/- 37)	1 (+/- 1)	10 (+/- 5)
4	6	4	H[1-4]	98.58% (+/- 0.57%)	99.17% (+/- 0.47%)	98.98% (+/- 0.38%)	5 (+/- 2)	72 (+/- 35)	1 (+/- 1)	10 (+/- 9)
5	6	6	H[1-6]	88.62% (+/- 0.63%)	94.36% (+/- 0.49%)	90.27% (+/- 0.53%)	9 (+/- 3)	731 (+/- 60)	1 (+/- 1)	56 (+/- 8)
6	12	4	H[1-4]	60.42% (+/- 1.19%)	76.94% (+/- 2.96%)	78.88% (+/- 0.69%)	67 (+/- 7)	1178 (+/- 154)	5 (+/- 2)	137 (+/- 38)
7	18	4	H[1-4]	23.79% (+/- 2.46%)	43.24% (+/- 3.36%)	59.84% (+/- 1.25%)	51 (+/- 5)	852 (+/- 54)	3 (+/- 2)	174 (+/- 16)

Table 5.7 The mean and std results of the trained models on RDS2 (120-U-20-100-*Random*) using different combinations of the heuristics H[1-6] and different window or batch sizes in terms of Falkenauer's performance and number of bins used

							Falkenauer's performance		Number of Bins	
#	Window size	#Heuristics	Heuristics	Acc	Valid	BLEU	LSTM >BSS	LSTM <BSS	LSTM >BSS	LSTM <BSS
1	6	1	H5	88.84% (+/- 0.56%)	94.21% (+/- 0.63%)	92.44% (+/- 0.33%)	6 (+/- 3)	793 (+/- 58)	0 (+/- 0)	114 (+/- 19)
2	6	2	H[1, 5]	66.81% (+/- 0.83%)	85.39% (+/- 1.13%)	76.87% (+/- 0.64%)	27 (+/- 6)	2449 (+/- 661)	1 (+/- 1)	176 (+/- 31)
3	6	2	H[1, 4]	85.34% (+/- 0.70%)	92.22% (+/- 0.57%)	90.17% (+/- 0.50%)	62 (+/- 5)	989 (+/- 45)	0 (+/- 0)	157 (+/- 21)
4	6	4	H[1-4]	83.69% (+/- 0.95%)	91.02% (+/- 1.45%)	88.98% (+/- 0.53%)	30 (+/- 4)	1074 (+/- 91)	0 (+/- 0)	152 (+/- 29)
5	6	6	H[1-6]	63.82% (+/- 0.64%)	84.41% (+/- 0.70%)	74.71% (+/- 0.46%)	25 (+/- 5)	2751 (+/- 106)	1 (+/- 1)	186 (+/- 29)
6	12	4	H[1-4]	18.67% (+/- 1.79%)	45.36% (+/- 3.18%)	54.78% (+/- 1.32%)	105 (+/- 14)	2010 (+/- 134)	1 (+/- 1)	443 (+/- 44)
7	18	4	H[1-4]	2.57% (+/- 0.43%)	13.97% (+/- 2.10%)	39.58% (+/- 1.08%)	27 (+/- 5)	518 (+/- 79)	1 (+/- 1)	175 (+/- 32)

sometimes output an invalid tour — for instance, it would repeat two cities or decided to ignore a destination. This procedure was relevant for wider pointer network, i.e. the number of cities >20, where at least 10% of TSP instances would not produce any valid tour; and this is similar to the results presented in Section 5.6.3 in which the larger the batch-size the worse the accurate and the valid solutions.

5.7 Conclusions

This chapter has proposed a radical approach which uses a deep learning model, encoder-decoder LSTM, to directly predict a solution from instance-data. This bypasses the typical processes of ASP first defined by Rice [1] in which features need to be derived from instances and mapped to an algorithm-space via a learner of some kind. The trained deep learning models can be considered as automatically generated heuristics that can make decisions implicitly in order to generate the solution.

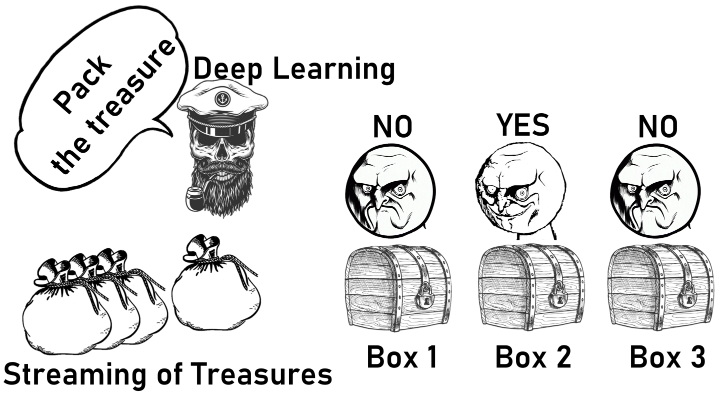
In order to show the potential benefits of pursuing this line of research more deeply, this chapter has evaluated the approach on a simple type of streaming problem in the packing domain in which items arrive in batches, and each batch must be packed before moving to the next. This chapter has provided the first description of a model that is capable of predicting a solution in the bin-packing domain and then studied two questions in depth. The first concerns the influence of the stream length on performance, while the second aims to understand whether the diversity of the heuristics in a pool used to create the training data influences performance.

Experiments showed that the proposed approach is able to predict very accurate solutions with smaller batches, particularly using solutions produced by a pool of heuristics with similar characteristics. Also, the models trained on the non-random datasets have excellent ability to generalise over the new datasets containing randomly generated instances with similar item size distributions to those on which they were trained. Furthermore, and perhaps surprisingly, all of the trained models were able to produce a small number of novel solutions that had better performance according to the Falkenauer fitness metric and the number of bins than any of the heuristics used to train the system.

Clearly, the work presented here only represents the first steps into this new paradigm. The next chapter introduces further work to develop the approach to deal with a more general version of streaming problems and to investigate feeding dynamic information about the current state of partial solutions into the networks. Despite this, this chapter suggests that the method provides a promising new direction for research within generating algorithms automatically that is ripe for further exploration.

Chapter 6

A Neural Approach to Generation of Constructive Heuristics



Both algorithm selection methods and selective hyper-heuristic methods rely on a pool of complementary heuristics. Improving the pool with new heuristics can improve performance, however, designing new heuristics can be challenging. Methods such as genetic programming have proved successful in automating this process in the past. Typically, these make use of problem state-information and existing heuristics as components.

The previous chapter introduced a preliminary study for a novel approach to generate constructive heuristics automatically to solve a simple version of the streaming bin-packing problem (i.e. short streams) by training a deep learning model on solutions that are generated from a set of heuristics. This chapter develops the previous AGP approach to handle a general version of streaming bin-packing problem instances and to feed dynamic information about the current state of partial solutions into the deep learning model as input. This chapter proposes a novel *neural* approach for generating constructive heuristics, in which a neural network acts as a heuristic by generating decisions. Two architectures are evaluated, an encoder-decoder LSTM and a feed-forward neural network. Both are trained using the *decisions* output from existing hand-designed heuristics from the literature on a large set of instances. Unlike the AGP approach proposed in the previous chapter, this AGP approach can handle an unbounded stream with stochastic arrival of individual items which must be packed immediately in strict order (one by one) and using a limited number of bins. It also examines all the available bins and uses dynamic information regarding bins-state to close full bins when necessary and open new ones if required. These trained neural models themselves essentially act as heuristics, directly outputting decisions. Thus, they are considered reusable heuristics that can be applied to new similar BPP instances with the same item size distribution.

This AGP approach is tested by evaluating the contribution of the new heuristics to a pool created by combining them with a set of existing well-known heuristics in the field of bin-packing on a large set of instances with an increasing number of available bins. The resulting pool can be used either with an algorithm selector or a selective hyper-heuristic. Many methods exist for both the former and the latter and therefore the evaluation is constrained to the contribution of the heuristics to the pool.

6.1 Contribution

The major contribution of this chapter is to describe a novel neural approach to generating constructive heuristics using an encoder-decoder LSTM or NN that directly outputs decisions taking into account the size of the current item to be packed and the current state of open bins. The new AGP method is rigorously evaluated on a streaming bin-packing problem with a limited number of bins by:

- Conducting an investigation of the scalability of the proposed approach with respect to the number of bins available for packing.
- Establishing a comparison of the new pools of heuristics that include encoder-decoder LSTM and/or a feed-forward neural network heuristics to the baseline pool of three constructive hand-designed heuristics from the literature, in order to determine whether the new generated heuristics provide significant overall improvement.
- Conducting an investigation of the capability of the proposed approach to generalise to new datasets containing randomly generated instances with similar characteristics to those on which they were trained.

This is the first time that such an approach has been used for the streaming 1D-BPP with a limited number of bins available, and provides an alternative method for generating heuristics that can be used in algorithm selection or selective hyper-heuristics. Although the method is evaluated in this instance on examples from the bin-packing domain, it is expected that the proposed approach should easily generalise to other streaming domains such as the Block Relocation Problem (BRP) [234] or dynamic Job-Shop Scheduling [235].

6.2 Background and Motivation

As mentioned in Chapter 2, given a large set of problem instances in a combinatorial optimisation domain, it is well-known that the performance of any given heuristic will vary from instance to instance. Hence, a pool of heuristics that have a complementary performance within the instance space is likely to be beneficial. This performance complementarity can be exploited in multiple ways including algorithm selection problem: given a set of heuristics, an *algorithm selector* [7] can be used to choose the best heuristic for a particular instance. Alternatively, *hyper-heuristic* [54] can combine low-level human-designed heuristics in a manner which allows the resulting method to outperform any of the individual heuristics when solving a combinatorial optimisation problem. Both approaches rely on a pool of quality heuristics. Adding new heuristics to the pool can improve performance across a set of instances, particularly if a new heuristic is diverse w.r.t the existing heuristics.

While hand-crafted heuristics can be selected from existing literature, some research communities have focused on new automated methods to generate new heuristics, e.g. generative hyper-heuristic, machine and deep learning communities (described in Chapter 2). Most of these methods rely on a pool of existing heuristics which become *components* of new generated heuristics, e.g. exist as nodes in a tree evolved using GP-based HH. As described in Chapter 2, the generated heuristic using machine or deep learning is either called once to produce a solution for a given problem instance (similar to the proposed approach in Chapter 5) or called repeatedly to make decisions at each decision step while dealing with the problem instance. Unlike the AGP approach proposed in Chapter 5, the generated heuristics using the approach in this chapter can be called repeatedly to make a decision for each item in the streaming problem instance. This chapter develops the previous chapter approach in which it can handle an unbounded stream with stochastic arrival of individual items which are packed on arrival taking into account the size of the current item to be packed and the current state of open bins. While the networks in the previous chapter approach were trained on solutions, neural network heuristics in this approach are trained to learn from the *decisions* generated by a set of low-level heuristics — this is in stark contrast to typical HH methods for the heuristic generation that combine low-level heuristics into new heuristics. Unlike generative HH approaches that search in the space of possible heuristics that may be suitable for solving a problem, the proposed approach navigates the space of possible decisions that create a solution for a given problem instance.

The approach is applied to solving streaming bin-packing problems: domains which incorporate *streaming data* — data points that arrive in a continual stream, which may be large and potentially unbounded, and in which the order of data points cannot be influenced — which still pose considerable challenges for existing methods. Many real-world problems also have an additional constraint in that the items arriving in the stream have to be dealt with by limited resources, e.g. packing items from a conveyor into trucks in a holding area, or stacking problems (common in shipping and steel industries [234, 236]) in which large items are moved by crane between arrival, holding and delivery stacks, each with a fixed capacity.

There are of course different methods for decision-making in a dynamic environment such as Monte Carlo Tree Search [237] also likely to be able to solve the problem at hand. However, as described in section 6.4, the problem at hand is a sequence-to-sequence problem. Given the fact that encoder-decoder LSTM topologies have been applied successfully to solve sequencing problems in the natural language processing field [11, 19], the driving motivation is to investigate whether they can also be used to generate heuristics to solve a combinatorial optimization problem. The next section describes the version of streaming bin-packing and the data used in the research presented in this chapter.

6.3 Streaming 1D-BPP Problem Instances

The following streaming bin-packing problem scenario is considered (Fig 6.1): *"Given a production line where the packing/stacking is carried out by a fixed robot arm/crane at the end of the line, the items (e.g. steel slabs) arrive one by one to be packed/stacked into a certain set of containers. When the item doesn't fit in any container, then the container with the lowest free space is closed and a new one is opened to pack/stack the item into."*

A subset of 900 bin-packing instances is used from datasets DS(1,2) described in Chapter 3, each of which has 120 items and is initialised with item sizes drawn from two different distributions. In order to solve this as a streaming instance, the items are considered to arrive in the order defined in each instance. From these instances, two balanced datasets DS(1,2) are defined each with 450 instances as shown in Table 6.1 as follows. Each instance is solved best by one of the heuristics under investigation (according to the Falkenauer fitness function [79] given in equation 2.2). The heuristics

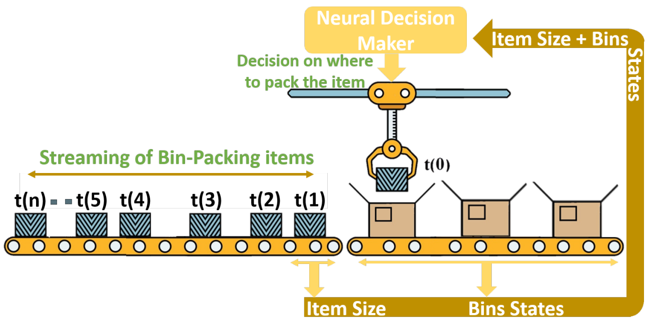

Neural Decision Maker
Item Size + Bins States
Decision on where to pack the item
Streaming of Bin-Packing items
t(0)
t(n) t(5) t(4) t(3) t(2) t(1)
Item Size
Bins States

Fig. 6.1 The Packing Process

considered are — BF; FF and WF, described in Chapter 2. Thus, 150 instances are solved best by each one of them.

Each row of training data describes the *input* to the network as [*Item0*, $BinState_0$, $BinState_1$, $BinState_2$] and the related *output* as sequence of actions as $[A_{bin0}, A_{bin1}, A_{bin2}]$: a *bin state* is defined as the residual capacity of each bin (i.e. how much space is remaining in the bin) and an action A_{bin} specifies whether or not an item should be placed in a bin, or whether or not a bin should be closed and the item packed into a new one. This process of extracting this data is explained in the following steps:

1. Label each *instance* with the best performing heuristic, as described above.
2. Split the data set into 300 instances for training and 150 instances for testing purposes.
3. Concatenate all the training instances to create a long training stream of 36,000 items (300x120); this is repeated for the test set, resulting in a test stream with 18,000 items. Label each *item* with the heuristic H_{item} that best solved the instance the item came from.
4. For each item i in the training stream, determine the current bin states BS, and denote the input data as [*item*, *BS*].
5. Apply H_{item} which determines which bin the item will be placed in (assuming there are a fixed number of bins b available at any one time, each with bin capacity $C =$ 150). Assign an action sequence A to the item as the desired output, e.g. [0,0,1] indicating the item is placed in the 3rd bin.

The duplicates in these inputs (item size + bin states) and outputs (actions) have been checked and very few rows have duplicates in inputs and conflicted outputs, between 0.02%-0.18% depending on the dataset and the number of bins used (see Table C.1 in the appendixes), thus these duplicates are unlikely to affect the training of the neural models.

Table 6.1 Data sets details [24]. Bin Capacity is fixed at 150

DS	total	n_{items}	Lower - Upper Bounds	Distribution
DS1	450	120	[40-60]	Gaussian
DS2	450	120	[20-100]	Uniform

The next section explains the proposed neural approach to generate constructive heuristics automatically.

6.4 The Neural Approach For Decision Making

This proposed method generates a heuristic that makes a decision per-item as it arrives in the stream[20] (arrival items size are unknown in advance). Two types of neural models are trained. The first uses a sequential-based architecture, namely an encoder-decoder LSTM. This is compared to a classic feed-forward neural network. These models take the size of the current item from a stream and the current state of open bins *BS* as input and output a decision: i.e. whether the item should be packed in a bin, or whether a bin should be closed and the item packed into a new one. Therefore, the trained models act as *constructive heuristics* in determining where an item is packed. The models are trained using *decisions* generated from each of a set of low-level heuristics using a large set of instances as training data.

In this sense, the proposed approach aims to make a sequence of actions per bin based on an item size and a sequence of current bins states, following which the item is packed accordingly, as presented in Fig 6.1. The decision is the non-zero action and this is checked for *validity* (e.g. to ensure an item can fit in a bin, explained in section 6.5), then the approach packs the item and updates the current bin states. Dynamic information regarding the current bin states is fed back into the network to pack the next item in the stream (Fig 6.2). Three possible actions are considered:

- No packing in this bin (denoted by 0).
- Pack the item in this bin (denoted by 1).
- Close this bin, open a new one and pack the item in it (denoted by 2).

Fig 6.3 shows two numerical examples of how the predicted decision would be using either i) encoder-decoder LSTM (Fig 6.3a) or ii) classical neural network (Fig 6.3b) when packing an item into one out of three available bins (details about these two models is presented in the next section). Also, the figure shows the *teacher forcing* technique to train this kind of networks as has been described in Chapter 2.

[20]An alternative would be to consider a sliding window and consider batches as distinct sub-problems

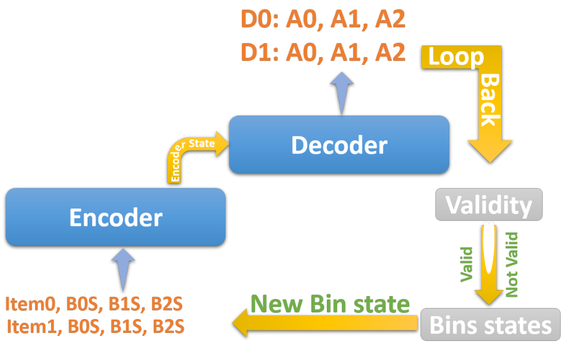


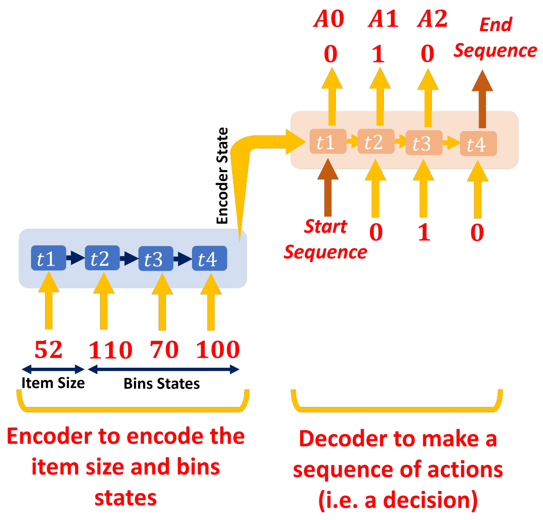
Fig. 6.2 The workflow of the encoder-decoder LSTM decisions making

6.5 Experiments and Results

Keras functional API[21] is used for the LSTM implementation, where the input is the size of an item to be packed and a sequence of the current bin states, and the output is a sequence of actions per bin. The input and output are both one-hot encoded. A preliminary empirical investigation was conducted to tune the encoder-decoder LSTM architecture and hyper-parameters using the ranges shown in Table C.2 in the appendixes. Also, Fig C.1 in the appendixes shows a schematic description of the encoder-decoder LSTM model with its layers and the trainable parameters number. The "Adam" optimiser [222] was used in all tuning experiments due to its reported accuracy, speed and low memory requirements. Using Keras, a classical neural network is also implemented where the input/output are the same as in LSTM approach except that they are not one-hot encoded, and output actions are represented using two neurons. For example, the last NN layer includes 6 neurons in the experiments with 3 bins and 40 neurons with 20 bins, where (0,0) refers to no packing; (1,0) for packing and (0,1) for closing the bin and open new one to pack the item in. The hyper-parameters of both the LSTM and NN are shown in Table C.2 in the appendixes. All experiments are conducted on Google Colab[22] with GPU run-time used to execute the experiments.

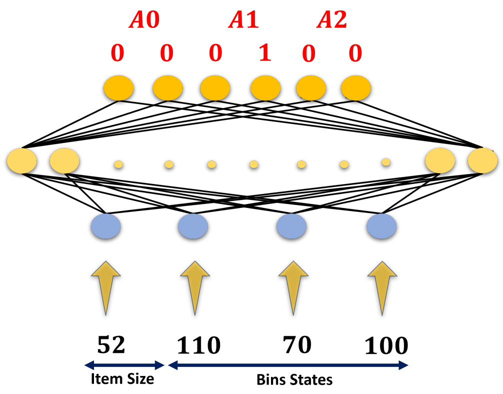
As previously described in section 6.3, each dataset was split into a training set (67%) and test set (33%). These sets were created for two scenarios, each using a different number of bins (3 and 20). Each training set contains a list of input-output pairs in which the input is item size and list of the current bin states, and the output is a list of actions for each bin. Each experiment was repeated ten times, thus produces 20 different LSTM- and NN-generated heuristics. For each experiment, the model that provides the lowest error from the training phase is saved then the trained models are tested on the testsets, and additionally on two new random datasets each of which includes 150 randomly chosen

[21]https://github.com/fchollet/keras
[22]https://colab.research.google.com/notebooks/welcome.ipynb



A0 A1 A2
0 1 0
End Sequence
Encoder State
t1 t2 t3 t4
Start Sequence
0 1 0
52 110 70 100
Item Size
Bins States
Encoder to encode the item size and bins states
Decoder to make a sequence of actions (i.e. a decision)

(a) A Generated heuristic using encoder-decoder LSTM



(b) A Generated heuristic using NN, the rest of the connections are deleted for simplicity

Fig. 6.3 The trained deep learning models behave as generated heuristics to predict sequence of actions per bin for packing an item in one out of three available bins

instances from datasets RDS(1,2) (described in Chapter 3) that are created from the same properties of the original datasets, see Table 6.1.

As mentioned before, the LSTM model outputs a sequence of actions per bin. The decision is defined as the non-zero action; the item is packed into the bin indicated by the '1' in the action sequence. As the decoder predicts an action per bin recursively, the LSTM approach might output a sequence of conflicting actions (e.g. the item should be packed in multiple bins). In this case, the tie is broken by applying the first non-zero action then correct all subsequent actions to 0 so that there is only one non-zero action. Despite the fact that using an architecture that can produce conflicting actions comes with a cost associated with their repair, in practice, this rarely happens: for instance, for DS2 using 20 bins only 1% of actions are conflicted per instance. NN approach predicts a decision directly (i.e. non-recursively) and thus only one non-zero action is output, i.e. there are no conflicting actions. Now after having a decision (the non-zero action), it is possible that the LSTM and NN can output an invalid decision, e.g. attempting to overfill a bin or not to place the item in any bin. Therefore a correction to these cases is applied, ensuring a valid decision is always produced. These special cases and the corrections applied are listed in Table C.3 in the appendixes. It is worth noting that this technique of dealing with invalid decisions is common in the literature related to generating heuristics with constraints [145, 146, 78, 131, 132].

The quality of the decisions made is determined by considering a sequence of $n = 120$ decisions as a batch, and measuring the quality of the overall solution created for the entire batch as a result of applying each decision. The Virtual Best Solver (VBS) (i.e. the baseline oracle) is defined as the best of the three low-level heuristics (BF, FF and WF) used to create the training data. 'Best' is defined in terms of the Falkenauer fitness metric or number of used bins, depending on the experiment. Also, **Validity:** is defined as the percentage of decisions per batch that are *valid* in the sense that (1) there is at least one decision with value 1 or 2 (i.e. the item is packed) (2) no constraints regarding bin-capacity are broken.

6.6 Results

This section describes the experiments results as presented in Tables 6.2 to 6.5. The section presents experiments to (A) Establish how often the encoder-decoder LSTM and NN generated heuristics produce valid decisions; (B) Compare the generated heuristics performance to the individual constructive heuristics performance; (C) Gain insight into how much benefit the generated heuristics bring to the baseline pool of heuristics and (d) Investigate the generalisation ability of the proposed approach over new random instances.

Table 6.2 Mean/std of the validity results obtained from 10 experiments of LSTM and NN approaches on 150 testing instances from DS1 and DS2

#	LSTM	NN
DS1-3bins	99.87% (+/- 0.03%)	99.59% (+/- 0.26%)
DS1-20bins	97.79% (+/- 0.56%)	82.04% (+/- 1.95%)
DS2-3bins	99.37% (+/- 0.15%)	99.54% (+/- 0.25%)
DS2-20bins	94.14% (+/- 0.49%)	76.51% (+/- 3.19%)

6.6.1 Returning Valid Decisions

Tables 6.2 shows that when using 3 bins both LSTM- and NN-generated heuristics obtain similar results in terms of generating valid decisions. It is clear that the NN heuristics fail to handle larger numbers of bins (20), generating many more invalid decisions. The poor performance of the standard neural network with the long sequences of bin states is not surprising, as a standard architecture cannot learn the mapping between the sequential information implicit in the bins states and the sequence of actions. Furthermore, the results show that the LSTM heuristics make valid decisions (that is, do not require correction for not packing an item or breaking bin-capacity) the majority of the time: [97%-99%] and [94%-99%] for DS1 and 2 respectively. The DS2 results in more invalid decisions than DS1. This is possibly due to a broader range of item sizes [20, 100] that are used to build DS2 instances (DS1 ranges from[40, 60]): the encoder may have more difficulty summarising instances with a wide range of values, potentially requiring more encoder layers for the model.

6.6.2 Comparing Neural-Generated Heuristics To Human-Designed Heuristics Performance

The VBSs and their heuristics are presented in Figure 6.4. The baseline **VBS(1)** includes the human-designed constructive heuristics (BF, FF and WF); **VBS(2) and (3)** expand VBS(1) with the best-of-run (in terms of valid decisions) generated heuristic using LSTM and NN respectively; and **VBS(4)** includes the best-of-run generated LSTM and NN heuristics. Tables 6.3 shows the number of instances that are uniquely best solved using the VBSs for DS(1,2) in terms of both Falkenauer's performance and the number of bins used. The uniquely best heuristic is the heuristic that outperforms the other heuristics in a pool to solve a given instance, i.e. it has the best performance and without being equal

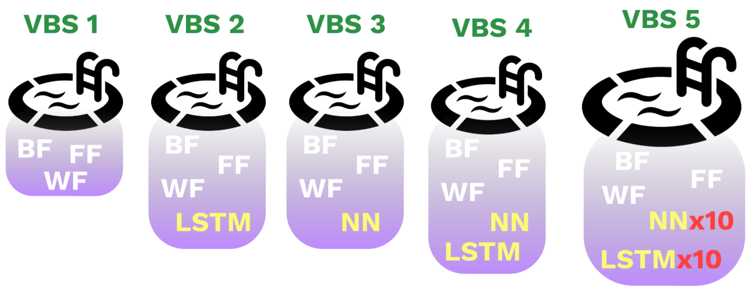


Fig. 6.4 The heuristics used in each VBS

to any other heuristic performance. One would notice that the numbers of instances in Table 6.3 (especially in terms of the number of bins) do not sum up to the number of test instances (150 instances). Since these experiments only aim to evaluate the unique contribution of each heuristic, the number of instances that are solved equally using heuristics in a pool is not presented.

As balanced datasets are used in terms of Falkenauer's performance (described in section 6.3), the heuristics in VBS(1) solve the same number of instances each in DS(1,2) however, these numbers can be different in terms of bins used. In the experiments using 3 bins from Falkenauer's performance perspective, the generated heuristics solve a good number of instances and sometimes exceed those solved by the well-known human-designed heuristic BF. The LSTM heuristic solves 31% of the test instances (47 instances) uniquely best outperforming BF that solves 29% (44 instances) in DS1. The LSTM heuristics are considered as first- and second-best performance heuristics in the VBS(2) solving uniquely 26% and 31% of the testsets on DS(1,2) respectively while NN heuristics are considered second-best performance in VBS(3) solving 26% and 27% of test instances on DS(1,2) respectively. In terms of the number of bins, many instances are solved equally using the heuristics in the pools however the new generated heuristics in VBSs(2 and 3) are considered second- and third-best performance. Also, extending the baseline VBS(1) to VBS(4) by adding the new heuristics, LSTM and NN heuristics solve very few instances equally and better than the original heuristics, i.e. 2 and 3 instances from DS1 and 2 respectively.

In the experiments using 20 bins, it is clear that the NN heuristics fail to handle larger numbers of bins (20), failing to solve any instance with performance better than the other methods. While LSTM heuristics solve 35 and 14 in DS1 and DS2 respectively from Falkenauer's performance perspective. In general, the new heuristics solve fewer instances in DS2 comparing to DS1. This is possibly due to a broader range of item sizes [20, 100] that are used to build DS2 instances (DS1 ranges from[40, 60]): the encoder in the LSTM

Table 6.3 The number of instances in testsets from DS(1,2) that are uniquely best solved using the heuristics per VBS in terms of both Falkenauer's performance and number of bins used.

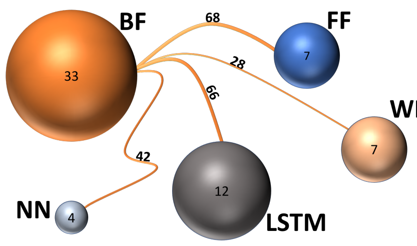
DS1- 3 bins	Falkenauer's Performance						Number of Bins					
	BF	FF	WF	LSTM	NN	LSTM=NN	BF	FF	WF	LSTM	NN	LSTM=NN
VBS(1)	50	50	50	—	—	—	42	7	16	—	—	—
VBS(2)	44	27	31	47	—	—	33	7	8	14	—	—
VBS(3)	49	39	20	—	41	—	41	7	8	—	7	—
VBS(4)	43	22	15	37	31	0	33	7	7	12	4	2
DS1- 20 bins												
VBS(1)	50	50	50	—	—	—	50	39	38	—	—	—
VBS(2)	48	32	31	35	—	—	48	22	26	11	—	—
VBS(3)	50	50	50	—	0	—	50	39	38	—	0	—
VBS(4)	48	32	31	35	0	0	48	22	26	11	0	0
DS2- 3 bins												
VBS(1)	50	50	50	—	—	—	43	16	6	—	—	—
VBS(2)	48	40	22	40	—	—	39	11	5	11	—	—
VBS(3)	49	35	27	—	39	—	38	12	5	—	7	—
VBS(4)	47	33	15	31	24	0	35	9	4	8	3	3
DS2- 20 bins												
VBS(1)	50	50	50	—	—	—	50	23	10	—	—	—
VBS(2)	50	50	36	14	—	—	50	23	9	3	—	—
VBS(3)	50	50	50	—	0	—	50	23	10	—	0	—
VBS(4)	50	50	36	14	0	0	50	23	9	3	0	0

approach may have more difficulty summarising instances with a wide range of values, potentially requiring more encoder layers for the model.

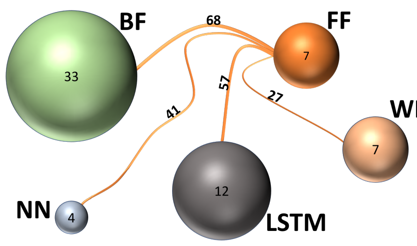
As an example to show the instances that are solved equally best, figure 6.5 shows the number of instances that are solved using VBS(4) heuristics on DS1 using 3 bins in terms of the number of bins used. Each heuristic is a bubble with a size refers to the number of instances uniquely best solved by that particular heuristic. The width of the edges between the bubbles refers to the number of instances solved equally using the two related heuristics. As it is complicated to show the different combinations of heuristics that solve the instances equally, the combinations are decomposed into pairs, e.g. if an instance is solved equally best using this set of heuristics (BF, FF and LSTM) then this instance is counted in (LSTM, BF), (LSTM, FF) and (BF, FF) edges.

6.6.3 Comparing The VBSs

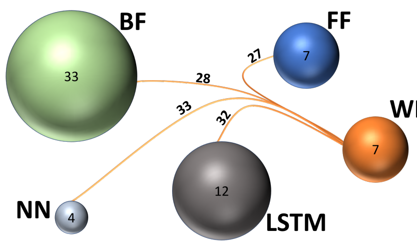
As described in the background section 6.2, the aim of the research presented in this chapter is to improve the overall performance by adding new generated heuristics to a pool of human-designed heuristics. Assuming a perfect per-instance algorithm selector using three, four and five heuristics, i.e. VBSs 1, 2/3 and 4 respectively. Table 6.4 shows the total performance from both Falkenauer's performance and the number of bins



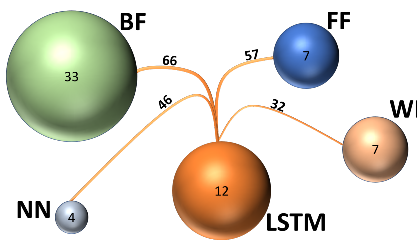
(a) DS1_3bins_BF - Number of Bins



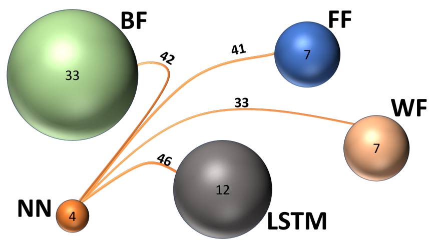
(b) DS1_3bins_FF - Number of Bins



(c) DS1_3bins_WF - Number of Bins



(d) DS1_3bins_LSTM - Number of Bins



(e) DS1_3bins_NN - Number of Bins

Fig. 6.5 The diagram illustrates the relative contribution of each heuristic to the pool. The size of a bubble indicates the number of instances uniquely solved by a heuristic. An edge between two heuristics indicates the number of instances on which the two heuristics are equal "winners", denoted by the edge-width and numeric indicator. The orange bubble represents the heuristic under consideration

perspectives on DS(1,2) using 3 and 20 bins. In general, adding the LSTM-generated heuristic to the original VBS improves the overall performance in most cases while adding the NN-generated heuristic improves the performance in the experiments using 3 bins only. Adding both the neural generated heuristics (LSTM and NN) leverage the complementary strength of both heuristics,

In order to exploit the performance complementarity of the different heuristics that cover different parts of the instance space, **VBS(5)** is created that includes all the heuristics, i.e. the human-designed constructive and all the neural generated heuristics (10 LSTM heuristics and 10 NN heuristics). A pool with more well-designed heuristics is expected to achieve significantly better results than a pool with a fewer number of heuristics. The results mainly back this up, as shown in Table 6.5 using 3 bins only (constructed based on p-values of table C.4 in the appendices). Ultimately, the number of containers (bins)

Table 6.4 The Total Falkenauer's performance and number of used bins per VBS on testsets (150 instances) from DS(1,2) using 3 and 20 bins, where the higher the Falkenauer value is better and the lower number of bins used is better

	Falkenauer's Performance					Number of Bins				
	VBS(1)	VBS(2)	VBS(3)	VBS(4)	VBS(5)	VBS(1)	VBS(2)	VBS(3)	VBS(4)	VBS(5)
DS1- 3 bins	123.826	124.402	124.186	124.646	**125.376**	6639	6624	6631	6619	**6601**
DS1- 20 bins	134.153	134.706	134.153	134.706	**135.516**	6358	6345	6358	6345	**6325**
DS2- 3 bins	118.251	118.669	118.545	118.805	**119.932**	8255	8243	8248	8240	**8202**
DS2- 20 bins	134.488	134.636	134.488	134.636	**134.703**	7634	7631	7634	7631	**7630**

determines the cost of any real-world solution. VBS(5) obtains the best performance improving over VBS(1) and saving 38 and 53 bins in the experiments using 3 bins for DS(1 and 2) respectively. Using 20 bins, VBS(5) uses 4 and 33 fewer bins than the baseline VBS(1) on DS(1 and 2) respectively. Although saving 4 bins could be seen as a small number of saving, those bins might be ships or planes and thus saving potentially millions of dollars. Table 6.5 shows the results obtained from applying a Friedman test followed by Nemenyi post-hoc test with 5% confidence level to evaluate significance in a pairwise fashion of the new VBSs to the baseline VBS.

Table 6.5 The comparison between the new VBSs and the baseline VBS over the different testsets (150 instances) from DS(1,2) in terms of Falkenauer's performance and number of bins using Friedman test followed by Nemenyi post-hoc test. For a given pair of VBS tests VBS(a,b), the $\uparrow$ means the first VBS's median is better, $\Longleftrightarrow$ means both VBSs have equal median, $+$ means there is significance, $-$ means there is no significance and Typical means both VBSs are typical and the added heuristic failed to outperform the constructive heuristics.

	Falkenauer's Performance					Number of Bins				
DS- Bins	VBS(2,1)	VBS(3,1)	VBS(4,1)	VBS(2,3)	VBS(5,1)	VBS(2,1)	VBS(3,1)	VBS(4,1)	VBS(2,3)	VBS(5,1)
DS1- 3 Bins	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\uparrow -$	$\uparrow +$	$\Longleftrightarrow -$	$\Longleftrightarrow -$	$\Longleftrightarrow -$	$\Longleftrightarrow -$	$\Longleftrightarrow +$
DS1- 20 Bins	$\uparrow +$	Typical	$\uparrow +$	$\uparrow +$	$\uparrow +$	$\Longleftrightarrow -$	Typical	$\Longleftrightarrow -$	$\Longleftrightarrow -$	$\Longleftrightarrow -$
DS2- 3 Bins	$\uparrow +$	$\uparrow -$	$\uparrow +$	$\uparrow -$	$\uparrow +$	$\Longleftrightarrow -$	$\Longleftrightarrow -$	$\Longleftrightarrow -$	$\Longleftrightarrow -$	$\uparrow +$
DS2- 20 Bins	$\Longleftrightarrow -$	Typical	$\Longleftrightarrow -$	$\Longleftrightarrow -$	$\Longleftrightarrow -$	$\Longleftrightarrow -$	Typical	$\Longleftrightarrow -$	$\Longleftrightarrow -$	$\Longleftrightarrow -$

6.6.4 Generalisation to new instances

Finally, experiments presented in Table 6.6 aim to evaluate the generalisation ability of the LSTM- and NN-generated heuristics to new datasets RDS(1,2) of unseen instances, 150 instances per dataset. These instances are created randomly using the same parameter distribution as DS1, DS2, respectively. Since these instances are randomly generated, so they are imbalanced solved using the heuristics in the VBS(1) in RDS(1,2) using 3 and 20 bins from Falkenauer's performance and the number of bins perspectives. It can be noticed that these instances are uniquely solved using FF and BF in most cases. The results of the generated heuristics have a similar pattern to the results using DS(1,2).

The LSTM and NN heuristics are considered the third-best performance heuristics in the VBS(2,3) on RDS(1,2) using 3 bins from Falkenauer's performance perspective. While they are considered second- and third-best performance heuristics on RDS(1,2) respectively using 3 bins from the number of bins perspective. It is interesting to note that NN-generated heuristic outperforms LSTM-generated heuristic in VBS(4) solving 12 instances comparing to 9 solved by LSTM heuristic on RDS1 using 3 bins. Similar to DS(1,2) results, in experiments with 20 bins, NN heuristics fail to handle larger numbers of bins (20), failing to solve any instance with performance better than the other methods. While LSTM heuristics solve only 10 instances in RDS1 and fail to solve any in RDS2 from Falkenauer's performance perspective.

Table 6.6 The number of instances in testsets from **R**DS(1,2) that are uniquely best solved using the heuristics per VBS in terms of both Falkenauer's performance and number of bins used.

RDS1- 3 bins	Falkenauer's Performance						Number of Bins					
	BF	FF	WF	LSTM	NN	LSTM=NN	BF	FF	WF	LSTM	NN	LSTM=NN
VBS(1)	64	64	22	—	—	—	51	3	1	—	—	—
VBS(2)	54	44	16	35	—	—	34	2	0	5	—	—
VBS(3)	62	52	16	—	18	—	46	3	0	—	5	—
VBS(4)	53	36	11	32	15	0	33	2	0	4	4	1
RDS1- 20 bins												
VBS(1)	70	69	10	—	—	—	48	14	2	—	—	—
VBS(2)	68	62	9	10	—	—	45	14	2	0	—	—
VBS(3)	70	69	10	—	0	—	48	14	2	—	0	—
VBS(4)	68	62	9	10	0	0	45	14	2	0	0	0
RDS2- 3 bins												
VBS(1)	100	50	0	—	—	—	84	7	0	—	—	—
VBS(2)	97	44	0	9	—	—	79	6	0	1	—	—
VBS(3)	96	37	0	—	13	—	77	5	0	—	2	—
VBS(4)	93	32	0	9	12	0	72	4	0	1	2	0
RDS2- 20 bins												
VBS(1)	98	52	0	—	—	—	78	7	0	—	—	—
VBS(2)	98	52	0	0	—	—	78	7	0	0	—	—
VBS(3)	98	52	0	—		—	78	7	0	—	0	—
VBS(4)	98	52	0	0	0	0	78	7	0	0	0	0

6.7 Conclusions

This chapter proposed a novel neural approach to generating constructive heuristics for dealing with streaming bin-packing problem that directly outputs decisions to pack items accounting for current bin states. Unlike typical methods to generate heuristics such as genetic programming that search the heuristics space for potential good heuristics, the proposed approach navigates the decision space for solving the problem instance. This chapter tested the approach by evaluating the contribution of the new generated heuristics to a pool created by combining them with a set of existing well-known heuristics in the bin-packing field with an increasing number of available bins. This chapter used two datasets including long streams drawn from two different distributions. The results showed that the expanded pool with the best-of-run generated heuristics brings improvement in 26%-31% of cases in terms of Falkenauer's performance using a small number of available bins. Also, since the different heuristics cover different parts of the instance space, adding all the generated heuristics leverages the performance complementarity strength and provides more improvement with saving up to 53 bins. Furthermore, this chapter provides the first example of using an encoder-decoder network with long-short-term memory as a generative heuristic technique for streaming bin-packing problem with limited resources.

Chapter 7

Conclusions

This chapter (i) summarises the work presented in preceding chapters of this thesis; (ii) provides answers to the research questions presented in Chapter 1; (iii) draws conclusions and discussion and (iv) presents an outlook on future work.

7.1 Summary

This thesis has investigated using deep learning, precisely recurrent neural networks with long-short term memory, for algorithm selection and generation to address the literature gaps in these two fields of the need to define features *a priori*, especially for dealing with streaming data. Generally, deep learning has proved the ability to extract features automatically in many domains with sequential nature, including video and image recognition, natural language processing, music generation and speech recognition [13–15]. This thesis extends these works by applying RNN-LSTM to the optimisation domain, and specifically domains in which there is a sequential aspect. The contribution of this thesis to the knowledge is summarised in Figure 7.1. Starting with traditional feature-based algorithm selection in the literature, moving on to the proposed feature-free approach in the second row. Then, the third row shows how bypassing algorithm selection moves the research into algorithm generation, followed by the proposed feature-free algorithm generation approaches in the last two rows.

The research presented in this thesis started with proposing a novel feature-free algorithm selection approach (Chapter 4) for sequential optimisation problems that exhibit an ordering w.r.t the elements of the problem and how they should be dealt with. Unlike most ASP techniques (first row in Fig 7.1), this approach does *not* require the design and selection of features to describe an instance, as shown in the second row in Fig 7.1. Two RNNs, with LSTM [10] or GRU [11], were trained using the sequence of items representing an instance directly as input to predict the best algorithm to solve the instance.

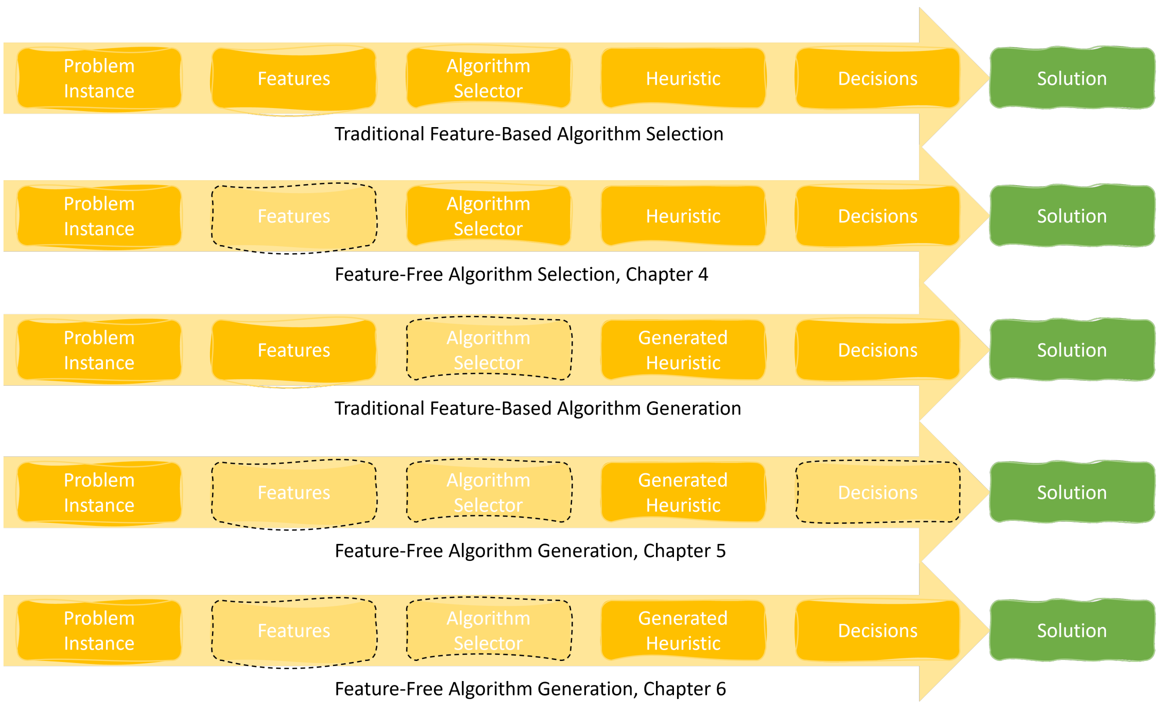


Fig. 7.1 From algorithm selection to algorithm generation using deep learning contribution. The dotted boxes refer to tasks that were bypassed by the approaches proposed in this thesis

Thus no need to design and select features to describe an instance. This chapter also compared this feature-free approach with traditional feature-based approaches using ten hand-designed features and six classical machine learning techniques. Both the novel and the traditional approaches were thoroughly evaluated on nine different large datasets (five evolved and four randomly generated datasets), exhibiting different numbers of items and different distributions of item sizes. The evolved datasets were generated in a way that each problem instance has a distinct best-solver.

The whole concept of algorithm selection consists of predicting the best performing algorithm for a given problem instance. Bypassing the need to select the best algorithm and predicting the solution directly instead, moves the research from ASP into AGP, as shown in the third and fourth rows of Fig 7.1. Chapter 5 proposed a novel approach using a deep learning model, encoder-decoder LSTM, to generate constructive heuristics automatically. Encoder-decoder LSTM was trained using solutions obtained from a set of heuristic algorithms to directly predict a *solution* from the instance data, i.e. without the need to define features *a priori*. The trained deep learning model can be considered an automatically generated heuristic that implicitly make decisions to generate the solution. This chapter studied two questions in depth. The first concerned the influence of the stream length on performance, while the second aimed to understand whether the diversity of the heuristics used to create the training data influences performance. This proposed approach was investigated on a simple type of streaming problem in the packing domain in which items arrive in batches, and each batch must be packed before moving to the next.

Finally, Chapter 6 developed the approach proposed in Chapter 5 by introducing a novel neural approach to generating constructive heuristics for dealing with an unbounded streaming bin-packing problem with limited available bins that directly output decisions to pack items accounting for current bins-state (i.e. the current candidate solution), as shown in the last row in Fig 7.1. Two architectures were evaluated, an encoder-decoder LSTM and a feed-forward neural network. Both were trained using the *decisions* output from existing hand-designed heuristics from the literature on a large set of instances. This chapter tested the approach by evaluating the contribution of the newly generated heuristics to a pool created by combining them with a set of existing well-known heuristics in the bin-packing field with an increasing number of available bins. This chapter used two datasets including long streams drawn from two different distributions.

7.2 Answers to Research Questions

This section presents the answers to the research questions that were investigated in this thesis. It should be noted that "to what extent" phrase implies an accuracy metric to predict the best performing algorithm or the generated solution, however other metrics can be used to measure the quality of the end-results, e.g. number of bins.

- **Question 1:** To what extent can sequential deep learning (RNN-LSTM/GRU) be used to build a feature-free algorithm selector that selects the best performing algorithm from a pool of well-known deterministic algorithms relying only on the sequential information implicit in the streaming problem instances?

  Chapter 4 investigated this particular question with a number of experiments on five newly generated datasets, each of which includes 4,000 instances. Each problem instance had a distinct best-solver exhibiting different numbers of items and different distributions of item sizes. On these artificially generated datasets[23], the results revealed very promising and accurate prediction of the best performing heuristics for un-seen problem instances, reaching 82%-96% prediction accuracy depending on the dataset used. These selectors were able to exceed the Single Best Solver (SBS), i.e. the single heuristic that achieves the best performance over the instance

[23]These datasets have been generated using EA to have maximum performance difference of the heuristics under investigation in the performance space which might be not the case in the real-world datasets.

set, and achieving comparable performance to the Virtual-Best Solver (VBS), i.e. the oracle or the perfect selector.

- **Question 2:** To what extent can sequential deep learning feature-free algorithm selector outperform classical machine learning feature-based algorithm selectors to solve a streaming optimisation problem?

  Chapter 4 compared the feature-free approach with traditional feature-based approaches using ten hand-designed features and six classical machine learning techniques. On the artificially generated datasets using EA, the accuracy of selecting the best performing heuristic using the LSTM and the GRU models ranges from 82%-96% while the ML models ranges from 43%-73%, depending on the dataset used. In terms of the percentage of the instances solved using DL predictors within a small difference of the VBS performance ($\leq 5\%$), results show between 27.5%-35% improvement over the SBS, depending on the dataset used. On the other hand, the best ML predictor ranges from 11.5%-19% improvement over the SBS. Thus, the DL selectors solved more instances than both the ML selectors and the SBS with a small difference of the VBS performance. On the new datasets used to evaluate the ability of the trained models to generalise, the feature-free approach outperforms feature-based in two out of four randomly generated datasets.

- **Question 3:** To what extent can deep learning (encoder-decoder LSTM) be used to generate algorithms automatically that can output solutions for streaming problem instances directly?

  Chapter 5 investigated this particular question. Experiments showed that the proposed approach was able to predict very accurate solutions with smaller batches, particularly using solutions produced by heuristics with similar characteristics. Furthermore, and perhaps surprisingly, all of the trained models were able to produce a small number of novel solutions that had better performance than any of the heuristics used to train the system.

- **Question 4:** To what extent can pool with neural generated algorithms be improved over a pool of human-designed algorithms to solve a streaming optimisation problem?

  Experiments in Chapter 6 showed that the expanded pool with the best-of-run generated heuristics produced a performance improvement over the pool composed of only hand-designed heuristics in 26%-31% of problem instances using a small number of available bins. Also, since the different heuristics cover different parts of the instance space, adding all the generated heuristics (i.e. not only the best-of-run generated heuristics) to a pool composed of hand-designed heuristics leverages the performance complementarity strength and provides more improvement with saving up to 53 bins. Thus, the implicit knowledge extracted by neural generated algorithms is complementary to the hard-won explicit expertise extracted by the experts.

7.3 Discussion

[Generalisation Ability to other Problem Domains]

Algorithm selection and generation have been studied extensively over the last decades. As shown in Chapter 2, both of them have suffered from the need to define features. Also, most of the previous approaches have not considered the sequential information contained in the streaming problem instances. The research conducted in this thesis is motivated by these literature gaps. This thesis proposed novel approaches using deep learning to alleviate the need to define features and learn from the sequential information of the streaming problem instances to build algorithm selectors or generate algorithms automatically.

Although the novel approaches proposed in this thesis were investigated only in one combinatorial optimisation problem domain (i.e. bin-packing), these approaches can be applied to other problem domains including, scheduling [26] and Travelling Salesperson Problem (TSP) [238]. Proposing a feature-free algorithm selection approach, Chapter 4, represents a significant step forward in algorithm selection for problems with sequence information, where the difficulties associated with defining suitable features and selecting from large sets of potential features are well understood. This approach has already been adopted and adapted by Seiler et al. [179] to work in the TSP domain using an evolved dataset with two TSP solvers. They compared a feature-based approach using four different classical ML classifiers to a feature-free approach using deep learning Convolutional Neural Network (CNN) [162]. Due to the large TSP-related feature sets, they conducted data analysis and automatic feature selection to choose an adequate set of 15 most relevant features. Their results show that the feature-based approach improved over the SBS performance but still quite far away from the performance of the oracle-like VBS. The feature-free approach matches the performance of the quite complex classical ML approaches, despite being solely based on a raw visual representation of the TSP instances. Although TSP is not an online or sequential problem, the work of Seiler et al. [179] borrows the key concept of the proposed method in Chapter 4, i.e. that the raw data defining an instance can be used without modification as input to a selection algorithm.

The proposed approaches in the thesis are not only suitable for the combinatorial optimisation problems, but they can also be applicable for continuous optimisation problems. The research in Chapters 4 and 5 inspired Friess et al. [239] to propose a pipeline that learns features capable of distinguishing different continuous optimisation problems within a latent space. Firstly, they generate samples uniformly random (i.e. training data) within the search space of continuous optimisation problems. Then they partition the search space using different neighbourhood-relationships (i.e. unstructured, cartesian and graph) that can be enforced through different unsupervised clustering techniques including, k-means [20], self-organised map [120] and growing neural gas [240], respectively. The output of the partitioning phase will be sets of partitions with different data formats including vectors, matrix and tensors depending on the clustering techniques used. After that, they train three different neural-based classifiers including classical neural network, CNN and Graph Neural Networks (GNNs) [241–243] to learn features capable of separating the different continuous optimisation problems within a latent space. Their results show that the approach is able to sufficiently separate different continuous optimization problems, especially with using the GNN that demonstrates the highest performance. They noticed that this approach bypasses the calculation of problem characteristics in the traditional algorithm selection framework [1] (described in Chapter 2) by encapsulating this step into a training procedure of neural network architectures.

[Domain-Independent Features-Based Approaches VS Feature-Free Approaches]

It is worth differentiating between feature-free algorithm selection approaches and approaches that rely on domain-independent features that are extracted automatically. Recently, Loreggia et al. [176] proposed a deep learning approach to automatically derive features in satisfiability [177] and constraint programming domains [178] assuming that any problem instance can be expressed as a text document. Their approach automatically derives domain-independent features from a visual representation of the problem instances (i.e. converting the text files into grey-scale square images), which can be used to train a conventional neural network to predict the best solver for the instance. Although their approach obtained better results than the SBS, it was not able to outperform the approaches that use regular manually crafted features.

[Some Limitations of the Proposed Approaches]

All the proposed approaches in this thesis used supervised learning either to build algorithm selectors or to generate algorithms without the need to define features *a priori*. Chapter 4 introduced a supervised feature-free ASP approach trained on labelled datasets that were produced using a greedy selection that determines the best performing heuristic for each problem instance. Chapters 5 and 6 proposed supervised approaches to automatically generate algorithms either using solutions or decisions collected from hand-designed heuristics (i.e. from an expert). Having labelled data is a hard task, time-consuming and not always available. Also, the performance of the learned heuristics using supervised fashion is limited to what the expert provided, especially when the expert is not optimal. The supervised generated heuristics only can expect to imitate and marginally outperform the expert, e.g. because the learner can reduce the variance of the answers across similarly-performing experts. The better the learning, the closer the performance of the learner to the expert's. Another drawback of this type is that the learned heuristic may not generalise well to unseen examples and small variations of the task. Thus, this type of learning might be used only when it is significantly faster than the expert to hand-designed a heuristic and/or the labelled dataset is available or easy to produce. Some drawbacks of using supervised learning to generate heuristics can be overcome with more advanced algorithms, including active learning [244] to query the expert as an oracle to improve the behaviour in uncertain states. An alternative to supervised learning would be using Reinforcement Learning (RL) [9], described in the next section.

Generally, deep learning has achieved near-human accuracy levels in various tasks including classification and prediction and has proved the ability to extract features automatically in wide ranges of domains. However, it is still considered a black-box since it suffers from interpretability problems [245]. This factor makes deep learning approaches not to be widely accepted in real-world applications yet. Some could argue that the approaches that rely on hand-designed features provide more human-understandable justifications. Some researchers in the community would scarify performance, such as accuracy in algorithm selection and solution quality in the algorithm generation to gain interpretable approaches. Recently, multiple research [246] have focused on visualising the trained deep learning models in attempts to extract some knowledge out of them for greater confidence when applying the deep learning-based approaches.

The next section describes some of the potential future work to develop the proposed approaches in this thesis.

7.4 Future Work

Some extensions have emerged while conducting the research presented in this thesis.

[Extending the Investigation to Other Searching Methods]

As described in Chapter 3, instance diversity, in regards to heuristics performance, plays a crucial role in developing good algorithm selectors that exploit the complementary performance. In this research, a vanilla evolutionary algorithm [200] is used to generate datasets that occupy different parts of the performance space w.r.t the considered heuristics that in turn exhibit complementary performance. Recently more sophisticated search algorithms have been used in the research w.r.t generating instance data such as novelty search [247]. Unlike objective-based search in evolutionary computation, novelty search ignores the ultimate objective of the search, thus expanding what can be achieved through evolutionary methods such as genetic programming. Novelty search can be used to find *distinct* problem instances that are better diversified through the performance space. Investigating the proposed approach on such informative datasets might produce algorithm selectors that are able to generalise well on real-world datasets.

[Expanding the Algorithm Space]

All the proposed approaches operated on a pool of deterministic constructive heuristics. Future work will extend the approaches to include larger and more complex sets of algorithms such as meta-heuristics, thus covering more of the instance space and exhibiting better complementary performance. Thus, the feature-free algorithm selectors will have the chance to choose between a wide range of different algorithms that might have a stochastic behaviour. In neural algorithm generation approaches, the generated heuristics will be evaluated when trained on solutions or decisions generated using a greater variety of optimisation algorithms, including meta-heuristic as well as hyper-heuristic approaches.

[Comparing to Other Methods of Heuristic Selection or Generation Approaches]

As described in Chapter 2, HHs research includes both algorithm selection and generation methods often using GP. As described in Chapter 2, GP-based HHs approaches are considered unsupervised learning approaches, while the approaches proposed in this thesis use supervised learning-based approaches. In this sense, this line of future work is a comparison between GP as an unsupervised learning method and deep learning with supervised learning methods.

[Using Reinforcement Learning]

As mentioned in the previous section, using supervised learning to develop an algorithm selector or to generate algorithms might be limited to what the expert provides. Thus a promising direction of future work might be using RL [91]. It is critical to understand the difference between learning a selector or heuristic using supervised learning (without a reward) where the expected behaviour is shown by an expert or oracle and learning a policy (i.e. a heuristic-like) using reinforcement learning through trial and error with a reward signal (without an expert). The learner using reinforcement learning is encouraged to quickly accumulate rewards that might be delayed. In comparison to an expert that would prefer one decision over others, the developed selectors or the generated heuristics using a reward signal (experience) can be flexible when multiple decisions are (almost) equally good. This type of learning has a well-known problem, namely exploration and exploitation dilemma [92] (e.g. having a meal in the best-known restaurant tried or exploring a new restaurant arbitrarily). If exploration is not sufficient, the learning process may get stuck around poor solutions or solutions which do not generalise well. Also, defining a reward signal might be not an easy task where different reward shapes can value different accomplishments. Despite these drawbacks, the learner using reinforcement learning can potentially outperform any expert, at the cost of a much longer learning time.

It is worth noting that often to solve a COP, it is a good idea to start with supervised learning (i.e. with human knowledge), then refine the generated heuristic or selector (i.e. a trained model) using reinforcement learning through experience and a reward signal. A successful example of this combination is the well known AlphaGo paper [248]. Deep reinforcement learning has become a trending research topic, and it has been used to learn a policy rather than a heuristic for many domains [9].

[Investigating on using other Sequential Deep Learning Architectures]

The core idea behind the proposed approaches is learning the sequential information implicit in streaming problems using a sequential deep learning model, LSTM. However, many sequential deep learning models can be used in the proposed approaches such as 1D convolutional neural networks [249], recursive neural networks [250] and many others especially in the natural language processing field [251, 252] such as attention mechanisms and transformer [253, 254]. Building algorithm selectors from these different sequential models leads to what is called "Meta-Algorithm Selection" [255] where the task aims to select an algorithm selector, which is then used to select the actual algorithm for solving a given problem instance.

[Interpretability]

Ultimately, the goal is to extract knowledge from the trained models of the proposed approaches to gain new insight into the correlation between problems characteristics and algorithms performance to build quality algorithm selectors and to generate new algorithms automatically. One track might be interpreting the deep learning models with visualisations to get insights into how the model processes the input data and makes predictions. A wide variety of different methods in different fields have been recently proposed to address this issue [256–261].

References

[1] John R Rice. The algorithm selection problem. In Morris Rubinoff and Marshall C. Yovits, editors, *Advances in Computers*, volume 15, pages 65 – 118. Elsevier, 1976.

[2] Pavel Brazdil and Christophe Giraud-Carrier. Metalearning and algorithm selection: progress, state of the art and introduction to the 2018 special issue, 2018.

[3] Lucas V Dias, Péricles BC Miranda, André CA Nascimento, Filipe R Cordeiro, Rafael Ferreira Mello, and Ricardo BC Prudêncio. Imagedataset2vec: An image dataset embedding for algorithm selection. *Expert Systems with Applications*, 180: 115053, 2021.

[4] David Lorge Parnas. Software aspects of strategic defense systems. *Communications of the ACM*, 28(12):1326–1335, 1985.

[5] Lee Spector. Automatic generation of intelligent agent programs. *IEEE Expert*, 12 (1):3–4, 1997.

[6] François Pachet and Pierre Roy. Automatic generation of music programs. In *International Conference on Principles and Practice of Constraint Programming*, pages 331–345. Springer, 1999.

[7] Pascal Kerschke, Holger H Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary computation*, pages 1–47, 2018.

[8] Lars Kotthoff. Algorithm selection for combinatorial search problems: A survey. In *Data Mining and Constraint Programming*, pages 149–190. Springer, 2016.

[9] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 2020.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.

[11] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[12] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[13] Alex Graves. Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*, pages 5–13. Springer, 2012.

[14] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and S. S. Iyengar. A survey on deep learning: Algorithms, techniques, and applications. *ACM Comput. Surv.*, 51(5): 92:1–92:36, September 2018. ISSN 0360-0300. doi: 10.1145/3234150.

[15] Sandro Skansi. *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. Springer, 2018.

[16] Nastaran Coleman and Pearl Wang. *Bin-Packing*, pages 116–126. Springer US, Boston, MA, 2013. ISBN 978-1-4419-1153-7. doi: 10.1007/978-1-4419-1153-7_75.

[17] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1979. ISBN 0716710447.

[18] Korte Bernhard and Jens Vygen. Combinatorial optimization: Theory and algorithms. *Springer, Third Edition, 2005.*, 2008.

[19] I Sutskever, O Vinyals, and QV Le. Sequence to sequence learning with neural networks. *Advances in NIPS*, 2014.

[20] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[21] **Alissa, Mohamad**, Kevin Sim, and Emma Hart. Automated algorithm selection: from feature-based to feature-free approaches. *Journal of Heuristics*, 2021. [Under Review].

[22] **Alissa, Mohamad**, Kevin Sim, and Emma Hart. A neural approach to generation of constructive heuristics. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 1147–1154. IEEE, 2021.

[23] **Alissa, Mohamad**, Kevin Sim, and Emma Hart. A deep learning approach to predicting solutions in streaming optimisation domains. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 157–165, 2020.

[24] **Alissa, Mohamad**, Kevin Sim, and Emma Hart. Algorithm selection using deep learning without feature extraction. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 198–206. ACM, 2019.

[25] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[26] Michael Pinedo. *Scheduling*, volume 29. Springer, 2012.

[27] Sally C Brailsford, Chris N Potts, and Barbara M Smith. Constraint satisfaction problems: Algorithms and applications. *European journal of operational research*, 119(3):557–581, 1999.

[28] Robert G Jeroslow and Jinchang Wang. Solving propositional satisfiability problems. *Annals of mathematics and Artificial Intelligence*, 1(1-4):167–187, 1990.

[29] Wansoo T Rhee. On the travelling salesperson problem in many dimensions. *Random Structures & Algorithms*, 3(3):227–233, 1992.

[30] Bernardo A Huberman, Rajan M Lukose, and Tad Hogg. An economics approach to hard computational problems. *Science*, 275(5296):51–54, 1997.

[31] Carla P Gomes and Bart Selman. Algorithm portfolios. *Artificial Intelligence*, 126 (1-2):43–62, 2001.

[32] Alex S Fukunaga. Genetic algorithm portfolios. In *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No. 00TH8512)*, volume 2, pages 1304–1311. IEEE, 2000.

[33] Marius Lindauer. *Algorithm selection, scheduling and configuration of Boolean constraint solvers*. PhD thesis, Universität Potsdam, Institut für Informatik, 2015.

[34] Marius Lindauer, Rolf-David Bergdoll, and Frank Hutter. An empirical study of per-instance algorithm scheduling. In *International Conference on Learning and Intelligent Optimization*, pages 253–259. Springer, 2016.

[35] Rong Qu, Graham Kendall, and Nelishia Pillay. The general combinatorial optimization problem: Towards automated algorithm design. *IEEE Computational Intelligence Magazine*, 15(2):14–23, 2020.

[36] Leonardo CT Bezerra, L Manuel, et al. Automatically designing state-of-the-art multi-and many-objective evolutionary algorithms. *Evolutionary computation*, 28 (2):195–226, 2020.

[37] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009.

[38] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka: Automatic model selection and hyperparameter optimization in weka. In *Automated Machine Learning*, pages 81–95. Springer, Cham, 2019.

[39] Manuel López-Ibánez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[40] Thomas Stützle and Manuel López-Ibáñez. Automated design of metaheuristic algorithms. In *Handbook of metaheuristics*, pages 541–579. Springer, 2019.

[41] Marius Lindauer, Holger Hoos, and Frank Hutter. From sequential algorithm selection to parallel portfolio selection. In *International Conference on Learning and Intelligent Optimization*, pages 1–16. Springer, 2015.

[42] Kate A. Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Comput. Surv.*, 41(1):1–25, 2009. doi: 10.1145/1456650.1456656.

[43] Joaquin Perez, Juan Frausto, Laura Cruz, Hector Fraire, Elizabeth Santiago, et al. A machine learning approach for modeling algorithm performance predictors. In *International Conference on Modeling Decisions for Artificial Intelligence*, pages 70–80. Springer, 2004.

[44] Gary R Weckman, Chandrasekhar V Ganduri, and David A Koonce. A neural network job-shop scheduler. *Journal of Intelligent Manufacturing*, 19(2):191–201, 2008.

[45] Pascal Kerschke, Lars Kotthoff, Jakob Bossek, Holger H Hoos, and Heike Trautmann. Leveraging tsp solver complementarity through machine learning. *Evolutionary computation*, 26(4):597–620, 2018.

[46] Kate Smith-Miles and Jano van Hemert. Discovering the suitability of optimisation algorithms by learning from evolved instances. *Annals of Mathematics and Artificial Intelligence*, 61(2):87–104, Feb 2011. ISSN 1573-7470. doi: 10.1007/s10472-011-9230-5.

[47] Marco Collautti, Yuri Malitsky, Deepak Mehta, and Barry O'Sullivan. Snnap: Solver-based nearest neighbor for algorithm portfolios. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 435–450. Springer, 2013.

[48] Kate Smith-Miles, Davaatseren Baatar, Brendan Wreford, and Rhyd Lewis. Towards objective measures of algorithm performance across instance space. *Computers & Operations Research*, 45:12 – 24, 2014. ISSN 0305-0548. doi: https://doi.org/10.1016/j.cor.2013.11.015.

[49] Pascal Kerschke, Mike Preuss, Simon Wessing, and Heike Trautmann. Low-budget exploratory landscape analysis on multiple peaks models. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 229–236. ACM, 2016.

[50] Pascal Kerschke and Heike Trautmann. Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. *Evolutionary computation*, 27(1):99–127, 2019.

[51] Kate Smith-Miles and Leo Lopes. Measuring instance difficulty for combinatorial optimization problems. *Computers & Operations Research*, 39(5):875–889, 2012.

[52] Kevin Sim. *Novel Hyper-heuristics Applied to the Domain of Bin Packing*. PhD thesis, Edinburgh Napier University, 2014.

[53] Nelishia Pillay. A generative hyper-heuristic for deriving heuristics for classical artificial intelligence problems. In *Advances in Nature and Biologically Inspired Computing*, pages 337–346. Springer, 2016.

[54] Edmund K Burke, Matthew R Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. A classification of hyper-heuristic approaches: revisited. In *Handbook of Metaheuristics*, pages 453–477. Springer, 2019.

[55] Marius Lindauer, Jan N van Rijn, and Lars Kotthoff. The algorithm selection competitions 2015 and 2017. *Artificial Intelligence*, 272:86–100, 2019.

[56] Marius Lindauer, Holger H Hoos, Frank Hutter, and Torsten Schaub. Autofolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, 53:745–778, 2015.

[57] Alexander Brownlee, John R Woodward, and Nadarajen Veerapen. Relating training instances to automatic design of algorithms for bin packing via features (detailed experiments and results). Technical report, University of Stirling, 2018.

[58] Emma Hart and Kevin Sim. A hyper-heuristic ensemble method for static job-shop scheduling. *Evolutionary computation*, 24(4):609–635, 2016.

[59] Gabriel Duflo, Emmanuel Kieffer, Matthias R Brust, Grégoire Danoy, and Pascal Bouvry. A gp hyper-heuristic approach for generating tsp heuristics. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 521–529. IEEE, 2019.

[60] Alejandro Sosa-Ascencio, Gabriela Ochoa, Hugo Terashima-Marin, and Santiago Enrique Conant-Pablos. Grammar-based generation of variable-selection heuristics for constraint satisfaction problems. *Genetic Programming and Evolvable Machines*, 17(2):119–144, 2016.

[61] Chanaleä Munien and Absalom E Ezugwu. Metaheuristic algorithms for one-dimensional bin-packing problems: A survey of recent advances and applications. *Journal of Intelligent Systems*, 30(1):636–663, 2021.

[62] Maxence Delorme, Manuel Iori, and Silvano Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1):1–20, 2016.

[63] Edward G Coffman, János Csirik, Gábor Galambos, Silvano Martello, and Daniele Vigo. Bin packing approximation algorithms: survey and classification. In *Handbook of combinatorial optimization*, pages 455–531. 2013.

[64] Gerhard Wäscher, Heike Haußner, and Holger Schumann. An improved typology of cutting and packing problems. *European journal of operational research*, 183 (3):1109–1130, 2007.

[65] JM Valério De Carvalho. Lp models for bin packing and cutting stock problems. *European Journal of Operational Research*, 141(2):253–273, 2002.

[66] EG Co man Jr, MR Garey, and DS Johnson. Approximation algorithms for bin packing: A survey. *Approximation algorithms for NP-hard problems*, pages 46–93, 1996.

[67] Paul E Sweeney and Elizabeth Ridenour Paternoster. Cutting and packing problems: a categorized, application-orientated research bibliography. *Journal of the Operational Research Society*, 43(7):691–706, 1992.

[68] Robert W Haessler and Paul E Sweeney. Cutting stock problems and solution procedures. *European Journal of Operational Research*, 54(2):141–150, 1991.

[69] Harald Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145–159, 1990.

[70] E Hopper and B Turton. A genetic algorithm for a 2d industrial packing problem. *Computers & Industrial Engineering*, 37(1-2):375–378, 1999.

[71] Célia Paquay, Sabine Limbourg, Michaël Schyns, and José Fernando Oliveira. Mip-based constructive heuristics for the three-dimensional bin packing problem with transportation constraints. *International Journal of Production Research*, 56 (4):1581–1592, 2018.

[72] Edward G Coffman, Gabor Galambos, Silvano Martello, and Daniele Vigo. Bin packing approximation algorithms: Combinatorial analysis. In *Handbook of combinatorial optimization*, pages 151–207. Springer, 1999.

[73] Joan Boyar, Shahin Kamali, Kim S Larsen, and Alejandro López-Ortiz. Online bin packing with advice. *Algorithmica*, 74(1):507–527, 2016.

[74] Henrik I Christensen, Arindam Khan, Sebastian Pokutta, and Prasad Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 24:63–79, 2017.

[75] D. Johnson, A. Demers, J. Ullman, M. Garey, and R. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

[76] Maxence Delorme, Manuel Iori, and Silvano Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1):1 – 20, 2016. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2016.04.030.

[77] Graham Cormode and Pavel Veselỳ. Streaming algorithms for bin packing and vector scheduling. *Theory of Computing Systems*, pages 1–27, 2020.

[78] Edmund K Burke, MR Hyde, Graham Kendall, and John R Woodward. The scalability of evolved on line bin packing heuristics. In *2007 IEEE Congress on Evolutionary Computation*, pages 2530–2537. IEEE, 2007.

[79] Emanuel Falkenauer and Alain Delchambre. A genetic algorithm for bin packing and line balancing. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 1186–1192. IEEE, 1992.

[80] György Dósa and Jiří Sgall. Optimal analysis of best fit bin packing. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, pages 429–441, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[81] Claire Kenyon et al. Best-fit bin-packing with random order. In *SODA*, volume 96, pages 359–364. Citeseer, 1996.

[82] José Antonio Vázquez-Rodríguez and Gabriela Ochoa. On the automatic discovery of variants of the neh procedure for flow shop scheduling using genetic programming. *Journal of the Operational Research Society*, 62(2):381–396, 2011.

[83] Edmund K Burke, Matthew R Hyde, and Graham Kendall. Grammatical evolution of local search heuristics. *IEEE Transactions on Evolutionary Computation*, 16(3): 406–417, 2011.

[84] Franco Mascia, Manuel López-Ibáñez, Jérémie Dubois-Lacoste, and Thomas Stützle. Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. *Computers & operations research*, 51:190–199, 2014.

[85] Marcelo de Souza and Marcus Ritt. Automatic grammar-based design of heuristic algorithms for unconstrained binary quadratic programming. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 67–84. Springer, 2018.

[86] Marie-Eléonore Marmion, Franco Mascia, Manuel López-Ibánez, and Thomas Stützle. Automatic design of hybrid stochastic local search algorithms. In *International workshop on hybrid metaheuristics*, pages 144–158. Springer, 2013.

[87] Manuel López-Ibánez, Marie-Eléonore Kessaci, and Thomas G Stützle. *Automatic design of hybrid metaheuristic from algorithmic components*. IRIDIA, Institut de Recherches Interdisciplinaires et de Développements en ..., 2017.

[88] John R Koza and John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[89] Su Nguyen, Yi Mei, and Mengjie Zhang. Genetic programming for production scheduling: a survey with a unified framework. *Complex & Intelligent Systems*, 3 (1):41–66, 2017.

[90] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50 (2):1–35, 2017.

[91] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[92] Daniella Laureiro-Martínez, Stefano Brusoni, Nicola Canessa, and Maurizio Zollo. Understanding the exploration–exploitation dilemma: An fmri study of attention control and decision-making performance. *Strategic management journal*, 36(3): 319–338, 2015.

[93] Eric Horvitz, Yongshao Ruan, Carla Gomes, Henry Kautz, Bart Selman, and Max Chickering. A bayesian approach to tackling hard computational problems. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 235–244. Morgan Kaufmann Publishers Inc., 2001.

[94] Kevin Leyton-Brown, Eugene Nudelman, and Yoav Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *International Conference on Principles and Practice of Constraint Programming*, pages 556–572. Springer, 2002.

[95] Kevin Leyton-Brown, Eugene Nudelman, Galen Andrew, Jim McFadden, and Yoav Shoham. A portfolio approach to algorithm selection. In *IJCAI*, volume 3, pages 1542–1543, 2003.

[96] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Hydra-mip: Automated algorithm configuration and selection for mixed integer programming. In *RCRA workshop on experimental evaluation of algorithms for solving problems with combinatorial explosion at the international joint conference on artificial intelligence (IJCAI)*, pages 16–30, 2011.

[97] Jonas Hanselle, Alexander Tornede, Marcel Wever, and Eyke Hüllermeier. Algorithm selection as superset learning: Constructing algorithm selectors from imprecise performance data. In *PAKDD (1)*, pages 152–163, 2021.

[98] Josef Pihera and Nysret Musliu. Application of machine learning to algorithm selection for tsp. In *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, pages 47–54. IEEE, 2014.

[99] Jun Won Lee and Christophe Giraud-Carrier. Automatic selection of classification learning algorithms for data mining practitioners. *Intelligent Data Analysis*, 17(4): 665–678, 2013.

[100] Daren Ler, Hongyu Teng, Yu He, and Rahul Gidijala. Algorithm selection for classification problems via cluster-based meta-features. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 4952–4960. IEEE, 2018.

[101] Nelishia Pillay and Rong Qu. *Hyper-heuristics: theory and applications*. Springer, 2018.

[102] Eugene Nudelman, Kevin Leyton-Brown, Holger H. Hoos, Alex Devkar, and Yoav Shoham. Understanding random sat: Beyond the clauses-to-variables ratio. In Mark Wallace, editor, *Principles and Practice of Constraint Programming – CP 2004*, pages 438–452, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-30201-8.

[103] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79 – 111, 2014.

[104] J. Pihera and N. Musliu. Application of machine learning to algorithm selection for tsp. In *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, pages 47–54, Nov 2014. doi: 10.1109/ICTAI.2014.18.

[105] Eunice López-Camacho, Hugo Terashima-Marín, Gabriela Ochoa, and Santiago Enrique Conant-Pablos. Understanding the structure of bin packing problems through principal component analysis. *International Journal of Production Economics*, 145 (2):488–499, 2013.

[106] Markus Ringnér. What is principal component analysis? *Nature biotechnology*, 26 (3):303, 2008.

[107] Peter Ross, Sonia Schulenburg, Javier G Marín-Blázquez, and Emma Hart. Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 942–948. Morgan Kaufmann Publishers Inc., 2002.

[108] Stewart W Wilson. Classifier fitness based on accuracy. *Evolutionary computation*, 3(2):149–175, 1995.

[109] Kevin Sim, Emma Hart, and Ben Paechter. A hyper-heuristic classifier for one dimensional bin packing problems: Improving classification accuracy by attribute evolution. In *International Conference on Parallel Problem Solving from Nature*, pages 348–357. Springer, 2012.

[110] Jan N van Rijn, Geoffrey Holmes, Bernhard Pfahringer, and Joaquin Vanschoren. The online performance estimation framework: heterogeneous ensemble learning for data streams. *Machine Learning*, 107(1):149–176, 2018.

[111] Jan N van Rijn, Geoffrey Holmes, Bernhard Pfahringer, and Joaquin Vanschoren. Algorithm selection on data streams. In *International Conference on Discovery Science*, pages 325–336. Springer, 2014.

[112] Matthias Carnein, Dennis Assenmacher, and Heike Trautmann. An empirical comparison of stream clustering algorithms. In *Proceedings of the Computing Frontiers Conference*, pages 361–366. ACM, 2017.

[113] Shufeng Gong, Yanfeng Zhang, and Ge Yu. Clustering stream data by exploring the evolution of density mountain. *Proceedings of the VLDB Endowment*, 11(4): 393–405, 2017.

[114] Stratos Mansalis, Eirini Ntoutsi, Nikos Pelekis, and Yannis Theodoridis. An evaluation of data stream clustering algorithms. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 11(4):167–187, 2018.

[115] Matthias Carnein and Heike Trautmann. Optimizing data stream representation: An extensive survey on stream clustering algorithms. *Business & Information Systems Engineering*, pages 1–21, 2019.

[116] Amineh Amini, Teh Ying Wah, and Hadi Saboohi. On density-based data streams clustering algorithms: A survey. *Journal of Computer Science and Technology*, 29 (1):116–141, 2014.

[117] Feng Mao, Edgar Blanco, Mingang Fu, Rohit Jain, Anurag Gupta, Sebastien Mancel, Rong Yuan, Stephen Guo, Sai Kumar, and Yayang Tian. Small boxes big data: A deep learning approach to optimize variable sized bin packing. In *Third IEEE International Conference on Big Data Computing Service and Applications, BigDataService 2017, Redwood City, CA, USA, April 6-9, 2017*, pages 80–89, 2017.

[118] Laura Cruz-Reyes, Claudia Gómez-Santillán, Joaquín Pérez-Ortega, Vanesa Landero, Marcela Quiroz, and Alberto Ochoa. Algorithm selection: From meta-learning to hyper-heuristics. In *Intelligent Systems*. IntechOpen, 2012.

[119] Joaquín Pérez, Rodolfo A Pazos, Juan Frausto, Guillermo Rodríguez, David Romero, and Laura Cruz. A statistical approach for algorithm selection. In *International Workshop on Experimental and Efficient Algorithms*, pages 417–431. Springer, 2004.

[120] Simon S Haykin et al. *Neural networks and learning machines/Simon Haykin.* New York: Prentice Hall,, 2009.

[121] Martin Hughes, Marc Goerigk, and Trivikram Dokka. Automatic generation of algorithms for robust optimisation problems using grammar-guided genetic programming. *Computers & Operations Research*, page 105364, 2021.

[122] Juergen Branke, Su Nguyen, Christoph W Pickardt, and Mengjie Zhang. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation*, 20(1):110–124, 2015.

[123] Nelishia Pillay. Evolving construction heuristics for the curriculum based university course timetabling problem. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 4437–4443. IEEE, 2016.

[124] Rushil Raghavjee and Nelishia Pillay. The effect of construction heuristics on the performance of a genetic algorithm for the school timetabling problem. In *Proceedings of the South African Institute of Computer Scientists and Information Technologists Conference on Knowledge, Innovation and Leadership in a Diverse, Multidisciplinary Environment*, pages 187–194, 2011.

[125] Kevin Sim and Emma Hart. A combined generative and selective hyper-heuristic for the vehicle routing problem. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 1093–1100, 2016.

[126] Ranulph Glanville, David Griffiths, Philip Baron, John H Drake, Matthew Hyde, Khaled Ibrahim, and Ender Ozcan. A genetic programming hyper-heuristic for the multidimensional knapsack problem. *Kybernetes*, 2014.

[127] Nasser R Sabar, Masri Ayob, Graham Kendall, and Rong Qu. Automatic design of a hyper-heuristic framework with gene expression programming for combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, 19(3): 309–325, 2014.

[128] Nasser R Sabar, Masri Ayob, Graham Kendall, and Rong Qu. A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. *IEEE transactions on cybernetics*, 45(2):217–228, 2014.

[129] Christopher Stone, Emma Hart, and Ben Paechter. Automatic generation of constructive heuristics for multiple types of combinatorial optimisation problems with grammatical evolution and geometric graphs. In *International Conference on the Applications of Evolutionary Computation*, pages 578–593. Springer, 2018.

[130] John H Drake, Nikolaos Kililis, and Ender Özcan. Generation of vns components with grammatical evolution for vehicle routing. In *European conference on genetic programming*, pages 25–36. Springer, 2013.

[131] Ender Özcan and Andrew J Parkes. Policy matrix evolution for generation of heuristics. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 2011–2018, 2011.

[132] Shahriar Asta, Ender Özcan, and Andrew J Parkes. Champ: Creating heuristics via many parameters for online bin packing. *Expert Systems with Applications*, 63: 208–221, 2016.

[133] Kevin Sim and Emma Hart. Generating single and multiple cooperative heuristics for the one dimensional bin packing problem using a single node genetic programming island model. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1549–1556, 2013.

[134] Kevin Sim, Emma Hart, and Ben Paechter. A lifelong learning hyper-heuristic method for bin packing. *Evolutionary computation*, 23(1):37–67, 2015.

[135] Alexander EI Brownlee, John R Woodward, and Nadarajen Veerapen. Relating training instances to automatic design of algorithms for bin packing via features. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 135–136, 2018.

[136] David Jackson. A new, node-focused model for genetic programming. In *European Conference on Genetic Programming*, pages 49–60. Springer, 2012.

[137] David Jackson. Single node genetic programming on problems with side effects. In *International Conference on Parallel Problem Solving from Nature*, pages 327–336. Springer, 2012.

[138] Edmund K Burke, Matthew R Hyde, Graham Kendall, and John Woodward. Automating the packing heuristic design process with genetic programming. *Evolutionary computation*, 20(1):63–89, 2012.

[139] Su Nguyen, Mengjie Zhang, Mark Johnston, and Kay Chen Tan. A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *IEEE Transactions on Evolutionary Computation*, 17(5):621–639, 2012.

[140] Rachel Hunt, Mark Johnston, and Mengjie Zhang. Evolving" less-myopic" scheduling rules for dynamic job shop scheduling with genetic programming. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 927–934, 2014.

[141] Yi Mei, Mengjie Zhang, and Su Nyugen. Feature selection in evolving job shop dispatching rules with genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 365–372, 2016.

[142] Jürgen Branke, Torsten Hildebrandt, and Bernd Scholz-Reiter. Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. *Evolutionary computation*, 23(2):249–277, 2015.

[143] Rachel Hunt. Genetic programming hyper-heuristics for job shop scheduling. 2016.

[144] Gabriela Ochoa, Matthew Hyde, Tim Curtois, Jose A Vazquez-Rodriguez, James Walker, Michel Gendreau, Graham Kendall, Barry McCollum, Andrew J Parkes, Sanja Petrovic, et al. Hyflex: A benchmark framework for cross-domain heuristic search. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 136–147. Springer, 2012.

[145] Edmund K Burke, Matthew R Hyde, and Graham Kendall. Evolving bin packing heuristics with genetic programming. In *Parallel Problem Solving from Nature-PPSN IX*, pages 860–869. Springer, 2006.

[146] Edmund K Burke, Matthew R Hyde, Graham Kendall, and John Woodward. Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1559–1565, 2007.

[147] Su Nguyen, Mengjie Zhang, and Mark Johnston. A sequential genetic programming method to learn forward construction heuristics for order acceptance and scheduling. In *2014 IEEE congress on evolutionary computation (CEC)*, pages 1824–1831. IEEE, 2014.

[148] Kate A Smith. Neural networks for combinatorial optimization: a review of more than a decade of research. *INFORMS Journal on Computing*, 11(1):15–34, 1999.

[149] Lu Duan, Haoyuan Hu, Yu Qian, Yu Gong, Xiaodong Zhang, Jiangwen Wei, and Yinghui Xu. A multi-task selected learning approach for solving 3d flexible bin packing problem. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, page 1386–1394, Richland, SC, 2019. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450363099.

[150] Runzhong Wang, Zhigang Hua, Gan Liu, Jiayi Zhang, Junchi Yan, Feng Qi, Shuang Yang, Jun Zhou, and Xiaokang Yang. A bi-level framework for learning to solve combinatorial optimization on graphs. *arXiv preprint arXiv:2106.04927*, 2021.

[151] Eric Larsen, Sébastien Lachapelle, Yoshua Bengio, Emma Frejinger, Simon Lacoste-Julien, and Andrea Lodi. Predicting solution summaries to integer linear programs under imperfect information with machine learning. *arXiv preprint arXiv:1807.11876*, 2018.

[152] Serena Mantovani, Gianluca Morganti, Nitish Umang, Teodor Gabriel Crainic, Emma Frejinger, and Eric Larsen. The load planning problem for double-stack intermodal trains. *European Journal of Operational Research*, 267(1):107–119, 2018.

[153] Sigurdur Olafsson and Xiaonan Li. Learning effective new single machine dispatching rules from optimal scheduling data. *International Journal of Production Economics*, 128(1):118–126, 2010.

[154] Xiaonan Li and Sigurdur Olafsson. Discovering dispatching rules using data mining. *Journal of Scheduling*, 8(6):515–527, 2005.

[155] Atif Shahzad and Nasser Mebarki. Learning dispatching rules for scheduling: a synergistic view comprising decision trees, tabu search and simulation. *Computers*, 5(1):3, 2016.

[156] Wojciech Bożejko and Mariusz Makuchowski. A fast hybrid tabu search algorithm for the no-wait job shop problem. *Computers & Industrial Engineering*, 56(4): 1502–1509, 2009.

[157] Helga Ingimundardottir and Thomas Philip Runarsson. Supervised learning linear priority dispatch rules for job-shop scheduling. In *International conference on learning and intelligent optimization*, pages 263–277. Springer, 2011.

[158] Ahmed El-Bouri, Subramaniam Balakrishnan, and Neil Popplewell. Sequencing jobs on a single machine: A neural network approach. *European Journal of Operational Research*, 126(3):474–490, 2000.

[159] Toru Eguchi, Fuminori Oba, and Toshiki Hirai. A neural network approach to dynamic job shop scheduling. In *Global Production Management*, pages 152–159. Springer, 1999.

[160] DA Koonce and S-C Tsai. Using data mining to find patterns in genetic algorithm solutions to a job shop schedule. *Computers & Industrial Engineering*, 38(3): 361–374, 2000.

[161] Toru Eguchi, Fuminori Oba, and Satoru Toyooka. A robust scheduling rule using a neural network in dynamically changing job-shop environments. *International Journal of Manufacturing Technology and Management*, 14(3-4):266–288, 2008.

[162] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[163] Haohan Wang and Bhiksha Raj. On the origin of deep learning. *arXiv preprint arXiv:1702.07800*, 2017.

[164] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA:, 2015.

[165] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[166] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[167] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[168] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. Dive into deep learning. *Unpublished draft. Retrieved*, 3:319, 2019.

[169] Holger Schwenk. Continuous space translation models for phrase-based statistical machine translation. In *Proceedings of COLING 2012: Posters*, pages 1071–1080, 2012.

[170] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28:1693–1701, 2015.

[171] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057. PMLR, 2015.

[172] Sumit Chopra, Michael Auli, and Alexander M Rush. Abstractive sentence summarization with attentive recurrent neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 93–98, 2016.

[173] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[174] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[175] Chris Manning Richard Socher. Machine translation and advanced recurrent lstms and grus. URL https://www.youtube.com/watch?v=QuELiw8tbx8&list=PL3FW7Lu3i5Jsnh1rnUwq_TcylNr7EkRe6&t=1190s.

[176] Andrea Loreggia, Yuri Malitsky, Horst Samulowitz, and Vijay A Saraswat. Deep learning for algorithm portfolios. In *AAAI*, pages 1280–1286, 2016.

[177] Yakir Vizel, Georg Weissenbacher, and Sharad Malik. Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, 103(11): 2021–2035, 2015.

[178] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.

[179] Moritz Seiler, Janina Pohl, Jakob Bossek, Pascal Kerschke, and Heike Trautmann. Deep learning as a competitive feature-free approach for automated algorithm selection on the traveling salesperson problem. In *International Conference on Parallel Problem Solving from Nature*, pages 48–64. Springer, 2020.

[180] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *arXiv preprint arXiv:1506.03134*, 2015.

[181] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[182] Hang Zhao, Qijin She, Chenyang Zhu, Yin Yang, and Kai Xu. Online 3d bin packing with constrained deep reinforcement learning. *arXiv preprint arXiv:2006.14978*, 2020.

[183] Ruizhen Hu, Juzhan Xu, Bin Chen, Minglun Gong, Hao Zhang, and Hui Huang. Tap-net: transport-and-pack using reinforcement learning. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020.

[184] Haoyuan Hu, Xiaodong Zhang, Xiaowei Yan, Longfei Wang, and Yinghui Xu. Solving a new 3d bin packing problem with deep reinforcement learning method. *arXiv preprint arXiv:1708.05930*, 2017.

[185] Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *arXiv preprint arXiv:1704.01665*, 2017.

[186] Wouter Kool, Herke Van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

[187] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 270–288. 2019.

[188] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning. *arXiv preprint arXiv:2010.12367*, 2020.

[189] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence V Snyder, and Martin Takáč. Reinforcement learning for solving the vehicle routing problem. *arXiv preprint arXiv:1802.04240*, 2018.

[190] Wouter Kool, Herke van Hoof, and Max Welling. Attention solves your tsp, approximately. *Statistics*, 1050:22, 2018.

[191] Junchi Yan, Shuang Yang, and Edwin R Hancock. Learning for graph matching and related combinatorial optimization problems. In *International Joint Conference on Artificial Intelligence*. York, 2020.

[192] Yunhao Yang and Andrew Whinston. A survey on reinforcement learning for combinatorial optimization. *arXiv preprint arXiv:2008.12248*, 2020.

[193] Kate Smith-Miles, Jano van Hemert, and Xin Yu Lim. Understanding tsp difficulty by learning from evolved instances. In *International Conference on Learning and Intelligent Optimization*, pages 266–280. Springer, 2010.

[194] Wanru Gao, Samadhi Nallaperuma, and Frank Neumann. Feature-based diversity optimization for problem instance classification. In *International Conference on Parallel Problem Solving from Nature*, pages 869–879. Springer, 2016.

[195] Aneta Neumann, Wanru Gao, Carola Doerr, Frank Neumann, and Markus Wagner. Discrepancy-based evolutionary diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 991–998. ACM, 2018.

[196] Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of heuristics*, 2(1):5–30, 1996.

[197] Emanuel Falkenauer. A new representation and operators for genetic algorithms applied to grouping problems. *Evolutionary computation*, 2(2):123–144, 1994.

[198] Peter Ross. Hyper-heuristics. In *Search methodologies*, pages 529–556. Springer, 2005.

[199] Armin Scholl, Robert Klein, and Christian Jurgens. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24(7):627 – 645, 1997. ISSN 0305-0548. doi: https://doi.org/10.1016/S0305-0548(96)00082-2.

[200] Agoston E Eiben and Jim E Smith. What is an evolutionary algorithm? In *Introduction to Evolutionary Computing*, pages 25–48. Springer, 2015.

[201] Jakob Bossek and Heike Trautmann. Evolving instances for maximizing performance differences of state-of-the-art inexact tsp solvers. In *International Conference on Learning and Intelligent Optimization*, pages 48–59. Springer, 2016.

[202] Jakob Bossek and Heike Trautmann. Understanding characteristics of evolved instances for state-of-the-art inexact tsp solvers with maximum performance difference. In *Conference of the Italian Association for Artificial Intelligence*, pages 3–12. Springer, 2016.

[203] Jakob Bossek, Pascal Kerschke, Aneta Neumann, Markus Wagner, Frank Neumann, and Heike Trautmann. Evolving diverse tsp instances by means of novel and creative mutation operators. In *Proceedings of the 15th ACM/SIGEVO Conference on Foundations of Genetic Algorithms*, pages 58–71, 2019.

[204] Michael R Garey and David S Johnson. Approximation algorithms for bin packing problems: A survey. In *Analysis and design of algorithms in combinatorial optimization*, pages 147–172. Springer, 1981.

[205] Ivan Amaya, José Carlos Ortiz-Bayliss, Santiago Enrique Conant-Pablos, Hugo Terashima-Marín, and Carlos A Coello Coello. Tailoring instances of the 1d bin packing problem for assessing strengths and weaknesses of its solvers. In *International Conference on Parallel Problem Solving from Nature*, pages 373–384. Springer, 2018.

[206] John H McDonald. *Handbook of biological statistics*, volume 2. sparky house publishing Baltimore, MD, 2009.

[207] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[208] Laurens Van Der Maaten. Accelerating t-sne using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245, 2014.

[209] Jarkko Venna and Samuel Kaski. Neighborhood preservation in nonlinear projection methods: An experimental study. In *International Conference on Artificial Neural Networks*, pages 485–491. Springer, 2001.

[210] Shahin Kamali and Alejandro López-Ortiz. An all-around near-optimal solution for the classic bin packing problem. *arXiv preprint arXiv:1404.4526*, 2014.

[211] Sally Floyd and Richard M Karp. Ffd bin packing for item sizes with uniform distributions on [0, 1/2]. *Algorithmica*, 6(1-6):222, 1991.

[212] Wolfgang Stille. *Solution techniques for specific bin packing problems with applications to assembly line optimization*. PhD thesis, Technische Universität, 2008.

[213] Norman Lloyd Johnson, Samuel Kotz, and Narayanaswamy Balakrishnan. *Continuous univariate distributions*, volume 1. Houghton Mifflin Boston, 1970.

[214] Ignacio Castiñeiras, Milan De Cauwer, and Barry O'Sullivan. Weibull-based benchmarks for bin packing. In *International Conference on Principles and Practice of Constraint Programming*, pages 207–222. Springer, 2012.

[215] Sevvandi Kandanaarachchi, Mario A Munoz, Rob J Hyndman, Kate Smith-Miles, et al. On normalization and algorithm selection for unsupervised outlier detection. Technical report, Monash University, Department of Econometrics and Business Statistics, 2018.

[216] Chan C Lee and Der-Tsai Lee. A simple on-line bin-packing algorithm. *Journal of the ACM (JACM)*, 32(3):562–572, 1985.

[217] Prakash Ramanan, Donna J Brown, Chung-Chieh Lee, and Der-Tsai Lee. On-line bin packing in linear time. *Journal of Algorithms*, 10(3):305–326, 1989.

[218] Ming Liu, Yinfeng Xu, Chengbin Chu, and Feifeng Zheng. Online scheduling on two uniform machines to minimize the makespan. *Theoretical Computer Science*, 410(21-23):2099–2109, 2009.

[219] Ji Young Lee and Franck Dernoncourt. Sequential short-text classification with recurrent and convolutional neural networks. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 515–520, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1062.

[220] Wonmin Byeon, Thomas M Breuel, Federico Raue, and Marcus Liwicki. Scene labeling with lstm recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3547–3555, 2015.

[221] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. Lstm fully convolutional networks for time series classification. *IEEE Access*, 6:1662–1669, 2017.

[222] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[223] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200): 675–701, 1937.

[224] Peter Bjorn Nemenyi. *Distribution-free multiple comparisons.* Princeton University, 1963.

[225] Frank Eibe, A. Hall Mark, and H. Witten Ian. *The WEKA Workbench. Online Appendix for Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, fourth edition edition, 2016.

[226] Dana H Ballard. Modular learning in neural networks. In *AAAI*, pages 279–284, 1987.

[227] Laurens Van Der Maaten. Learning a parametric embedding by preserving local structure. In *Artificial Intelligence and Statistics*, pages 384–391, 2009.

[228] Yong Kun Cho, James T Moore, Raymond R Hill, and Charles H Reilly. Exploiting empirical knowledge for bi-dimensional knapsack problem heuristics. *International Journal of Industrial and Systems Engineering*, 3(5):530–548, 2008.

[229] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[230] Oriol Vinyals and Quoc Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.

[231] Wojciech Zaremba and Ilya Sutskever. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.

[232] Félix G Harvey and Christopher Pal. Semi-supervised learning with encoder-decoder recurrent neural networks: Experiments with motion capture sequences. *arXiv preprint arXiv:1511.06653*, 2015.

[233] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[234] Jana Lehnfeld and Sigrid Knust. Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. *European Journal of Operational Research*, 239(2):297–312, 2014.

[235] Jatoth Mohan, Krishnanand Lanka, and A Neelakanteswara Rao. A review of dynamic job shop scheduling techniques. *Procedia Manufacturing*, 30:34–39, 2019.

[236] Fabien Tricoire, Judith Scagnetti, and Andreas Beham. New insights on the block relocation problem. *Computers & Operations Research*, 89:127–139, 2018.

[237] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[238] Gregory Gutin and Abraham P Punnen. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2006.

[239] Stephen Friess, Peter Tino, Zhao Xu, Stefan Menzel, Bernhard Sendhoff, and Xin Yao. Artificial neural networks as feature extractors in continuous evolutionary optimization. In *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021.

[240] Bernd Fritzke et al. A growing neural gas network learns topologies. *Advances in neural information processing systems*, 7:625–632, 1995.

[241] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

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

[243] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[244] Burr Settles. Active learning literature survey. 2009.

[245] Supriyo Chakraborty, Richard Tomsett, Ramya Raghavendra, Daniel Harborne, Moustafa Alzantot, Federico Cerutti, Mani Srivastava, Alun Preece, Simon Julier, Raghuveer M Rao, et al. Interpretability of deep learning models: A survey of results. In *2017 IEEE smartworld, ubiquitous intelligence & computing, advanced & trusted computed, scalable computing & communications, cloud & big data computing, Internet of people and smart city innovation (smartworld/SCALCOM/UIC/ATC/CBDcom/IOP/SCI)*, pages 1–6. IEEE, 2017.

[246] Fred Hohman, Haekyu Park, Caleb Robinson, and Duen Horng Polo Chau. S ummit: Scaling deep learning interpretability by visualizing activation and attribution summarizations. *IEEE transactions on visualization and computer graphics*, 26(1): 1096–1106, 2019.

[247] Joel Lehman and Kenneth O Stanley. Novelty search and the problem with objectives. In *Genetic programming theory and practice IX*, pages 37–56. Springer, 2011.

[248] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[249] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J Inman. 1d convolutional neural networks and applications: A survey. *Mechanical systems and signal processing*, 151:107398, 2021.

[250] Mitsuo Kawato, Kazunori Furukawa, and Ryoji Suzuki. A hierarchical neural-network model for control and learning of voluntary movement. *Biological cybernetics*, 57(3):169–185, 1987.

[251] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75, 2018.

[252] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 32(2):604–624, 2020.

[253] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[254] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[255] Alexander Tornede, Marcel Wever, and Eyke Hüllermeier. Towards meta-algorithm selection. *arXiv preprint arXiv:2011.08784*, 2020.

[256] Angelos Chatzimparmpas, Rafael M Martins, Ilir Jusufi, and Andreas Kerren. A survey of surveys on the use of visualization for interpreting machine learning models. *Information Visualization*, 19(3):207–233, 2020.

[257] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7): e0130140, 2015.

[258] Anupam Datta, Shayak Sen, and Yair Zick. Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In *2016 IEEE symposium on security and privacy (SP)*, pages 598–617. IEEE, 2016.

[259] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?": Explaining the predictions of any classifier. *arXiv e-prints*, pages arXiv–1602, 2016.

[260] Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016.

[261] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st international conference on neural information processing systems*, pages 4768–4777, 2017.

Appendix A

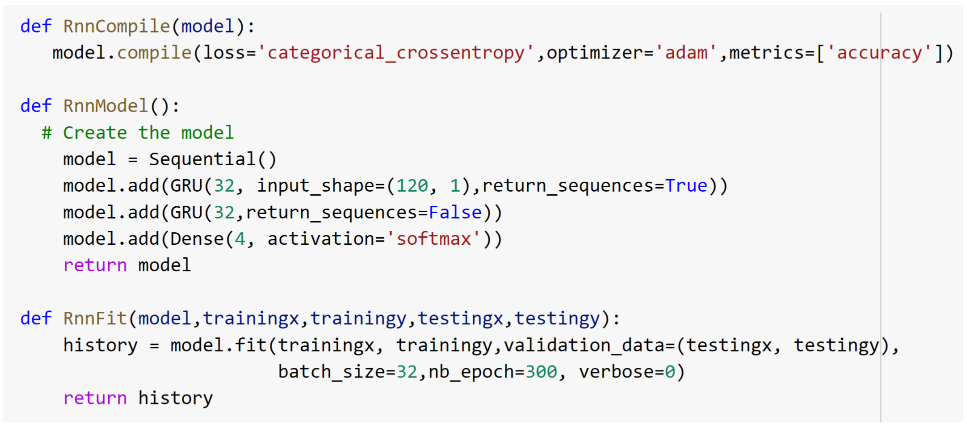
Chapter 4 Appendices

Table A.1 Range of values that are used in the LSTM and NN hyper-parameters tuning; the table also shows the final selected values. The rest of the hyper-parameters are the default values provided by Keras

LSTM/GRU	#Epoch	Batch size	#Layer	#Units
Range	[100-700]	[8-128]	[1-4]LSTM/GRU	-
Best DS(1,2)	300	32	2LSTM/GRU (tanh) + FC (softmax)	32,32,4
Best DS(3-5)	700	32	2LSTM/GRU (tanh)+ FC (softmax)	32,32,4
		NN Hyper-parameters		
Range	[50-3500]	-	[3-4]	[6-64]
Best	3000	32	2(relu) + 1(softmax)	10,15,4
All the models LSTM, GRU and NN use "adam" Optimizer and "CategoricalCrossentropy" as loss function				

Table A.2 Range of values over which grid-search was conducted to optimised the hyper-parameters for the ML experiments; the table also shows the final selected values.

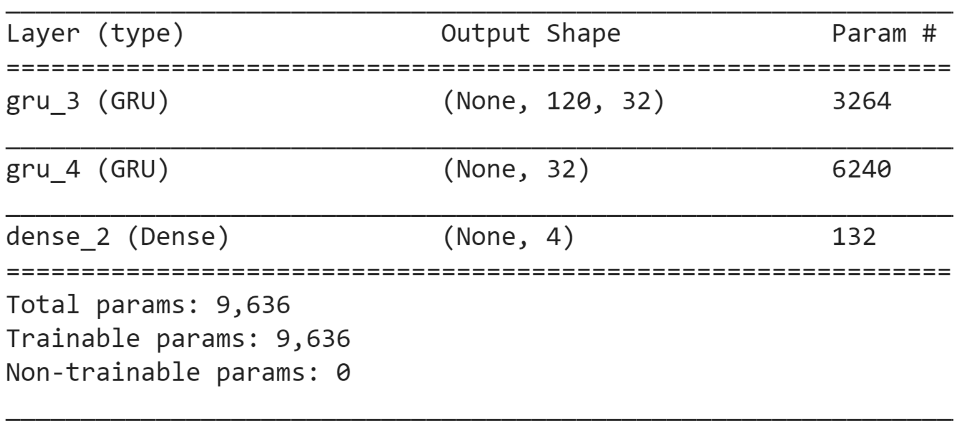
DT, RF	max_depth	max_features	min_leaf	min_split	n_estimators
Range	[5-100]	[1-10]	[2-100]	[2-100]	[32-200]
Best DT	30	5	10	100	-
Best RF	50	3	2	50	64
SVM	**kernel**	**gamma**	**C**	**degree**	**decision_function**
Sets	{'linear', 'rbf', 'poly'}	{'auto','scale'}	[0.1-1000]	[0-6]	{'ovo','ovr'}
Best	poly	scale	1000	6	ovo
KNN	**n_neighbors**	**weights**	**algorithm**	**leaf_size**	**P**
Sets	[1-30]	{'uniform', 'distance'}	auto	[30,50,100]	[1-5]
Best	26	distance	auto	30	1
NB	**var_smoothing**				
Set	[1e-09, 1e-01]				
Best	1e-09				



```
def RnnCompile(model):
   model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])

def RnnModel():
  # Create the model
    model = Sequential()
    model.add(GRU(32, input_shape=(120, 1),return_sequences=True))
    model.add(GRU(32,return_sequences=False))
    model.add(Dense(4, activation='softmax'))
    return model

def RnnFit(model,trainingx,trainingy,testingx,testingy):
    history = model.fit(trainingx, trainingy,validation_data=(testingx, testingy),
                        batch_size=32,nb_epoch=300, verbose=0)
    return history
```



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_3 (GRU)                  (None, 120, 32)           3264
_________________________________________________________________
gru_4 (GRU)                  (None, 32)                6240
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 132
=================================================================
Total params: 9,636
Trainable params: 9,636
Non-trainable params: 0
_________________________________________________________________
```

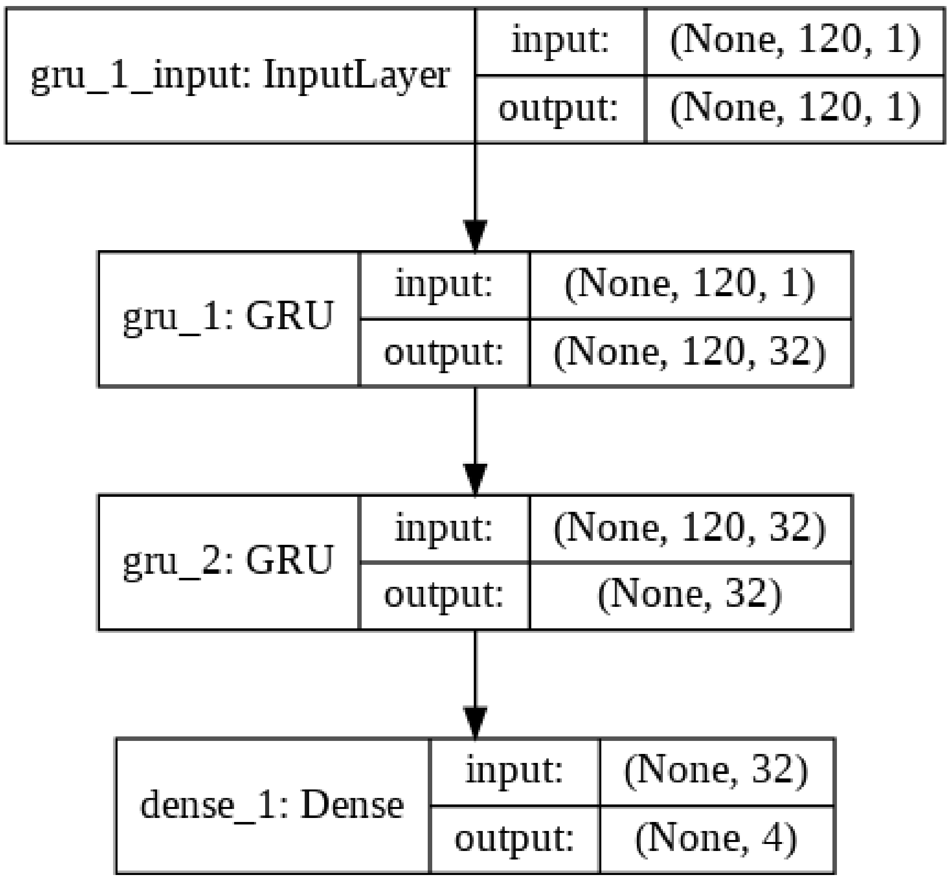


Fig. A.1 Schematic description of the GRU model (trained on DS2 as an example) shows a snippet of the code, the layers and number of trainable parameters (LSTM has a similar schematic).

Table A.3 Classification accuracy and the standard deviation (std) from the validation sets of each ML technique, the LSTM and GRU

	DL		ML					
Dataset	LSTM	GRU	NN	DT	RF	SVM	NB	KNN
DS1	78.62% (+/- 4.74%)	80.41% (+/- 2.04%)	65.41% (+/- 1.62%)	62.94% (+/- 1.06%)	65.78% (+/- 2.02%)	66.41% (+/- 1.31%)	61.34% (+/- 2.14%)	61.75% (+/- 1.13%)
DS2	89.91% (+/- 1.73%)	93.13% (+/- 1.03%)	58.66% (+/- 2.02%)	54.38% (+/- 2.40%)	58.16% (+/- 2.51%)	59.00% (+/- 2.69%)	55.66% (+/- 2.31%)	54.38% (+/- 2.21%)
DS3	80.94% (+/- 3.58%)	80.62% (+/- 1.87%)	67.12% (+/- 2.03%)	66.72% (+/- 2.13%)	67.50% (+/- 1.54%)	69.00% (+/- 2.53%)	51.75% (+/- 0.96%)	64.34% (+/- 1.88%)
DS4	95.34% (+/- 2.20%)	96.62% (+/- 1.22%)	73.16% (+/- 2.32%)	69.25% (+/- 2.33%)	71.94% (+/- 2.80%)	74.62% (+/- 2.60%)	66.06% (+/- 3.34%)	65.12% (+/- 3.20%)
DS5	82.06% (+/- 1.97%)	84.03% (+/- 2.94%)	62.06% (+/- 2.88%)	60.47% (+/- 3.25%)	63.94% (+/- 1.97%)	57.16% (+/- 2.83%)	43.00% (+/- 2.80%)	59.50% (+/- 2.18%)

Table A.4 P-values between LSTM, GRU, VBS, SBS and ML (best techniques) from the test set (800 instances) in terms of Falkenauer's performance and number of bins achieved by each of the methods mentioned for DS(1-5) using Friedman test followed by Nemenyi post-hoc test with 5% confidence level. For all the datasets, Friedman test obtains p-values = 0 $\ll$ 5%, i.e. reject the null hypothesis that the median performance is the same for all the methods mentioned and the tables show the p-values of the Nemenyi post-hoc test.

	Falkenauer's Performance					Bins				
DS1	VBS	BF	LSTM	GRU	ML	VBS	BF	LSTM	GRU	ML
VBS	-1.000	0.001	0.001	0.001	0.001	-1.000	0.001	0.001	0.001	0.001
BF	0.001	-1.000	0.001	0.001	0.001	0.001	-1.000	0.001	0.001	0.001
LSTM	0.001	0.001	-1.000	**0.900**	0.001	0.001	0.001	-1.000	**0.900**	0.001
GRU	0.001	0.001	**0.900**	-1.000	0.001	0.001	0.001	**0.900**	-1.000	0.001
ML	0.001	0.001	0.001	0.001	-1.000	0.001	0.001	0.001	0.001	-1.000

(a) DS1

	Falkenauer's Performance					Bins				
DS2	VBS	BF	LSTM	GRU	ML	VBS	BF	LSTM	GRU	ML
VBS	-1.000	0.001	0.029	**0.100**	0.001	-1.000	0.001	**0.254**	**0.354**	0.001
BF	0.001	-1.000	0.001	0.001	0.001	0.001	-1.000	0.001	0.001	0.001
LSTM	0.029	0.001	-1.000	**0.900**	0.001	**0.254**	0.001	-1.000	**0.900**	0.001
GRU	**0.100**	0.001	**0.900**	-1.000	0.001	**0.354**	0.001	**0.900**	-1.000	0.001
ML	0.001	0.001	0.001	0.001	-1.000	0.001	0.001	0.001	0.001	-1.000

(b) DS2

	Falkenauer's Performance					Bins				
DS3	VBS	BF	LSTM	GRU	ML	VBS	BF	LSTM	GRU	ML
VBS	-1.000	0.001	0.001	0.001	0.001	-1.000	0.001	0.001	0.001	0.001
BF	0.001	-1.000	0.001	0.001	0.001	0.001	-1.000	0.001	0.001	0.001
LSTM	0.001	0.001	-1.000	**0.900**	0.001	0.001	0.001	-1.000	**0.900**	0.001
GRU	0.001	0.001	**0.900**	-1.000	0.001	0.001	0.001	**0.900**	-1.000	0.001
ML	0.001	0.001	0.001	0.001	-1.000	0.001	0.001	0.001	0.001	-1.000

(c) DS3

	Falkenauer's Performance					Bins				
DS4	VBS	BF	LSTM	GRU	ML	VBS	BF	LSTM	GRU	ML
VBS	-1.000	0.001	**0.575**	**0.701**	0.001	-1.000	0.001	**0.710**	**0.696**	0.001
BF	0.001	-1.000	0.001	0.001	0.001	0.001	-1.000	0.001	0.001	**0.416**
LSTM	**0.575**	0.001	-1.000	**0.900**	0.001	**0.710**	0.001	-1.000	**0.900**	0.001
GRU	**0.701**	0.001	**0.900**	-1.000	0.001	**0.696**	0.001	**0.900**	-1.000	0.001
ML	0.001	0.001	0.001	0.001	-1.000	0.001	**0.416**	0.001	0.001	-1.000

(d) DS4

	Falkenauer's Performance					Bins				
DS5	VBS	BF	LSTM	GRU	ML	VBS	BF	LSTM	GRU	ML
VBS	-1.000	0.001	0.001	0.001	0.001	-1.000	0.001	0.001	0.001	0.001
BF	0.001	-1.000	0.001	0.001	0.001	0.001	-1.000	0.001	0.001	0.007
LSTM	0.001	0.001	-1.000	**0.900**	0.001	0.001	0.001	-1.000	**0.900**	0.001
GRU	0.001	0.001	**0.900**	-1.000	0.001	0.001	0.001	**0.900**	-1.000	0.001
ML	0.001	0.001	0.001	0.001	-1.000	0.001	0.007	0.001	0.001	-1.000

(e) DS5

Table A.5 P-values between LSTM, GRU, VBS, SBS and ML (best techniques) from the test set (800 instances) in terms of Falkenauer's performance and number of bins achieved by each of the methods mentioned for RDS(5-8) using Friedman test followed by Nemenyi post-hoc test with 5% confidence level. For all the datasets, Friedman test obtains p-values = 0 $\ll$ 5%, i.e. reject the null hypothesis that the median performance is the same for all the methods mentioned and the tables show the p-values of the Nemenyi post-hoc test.

	Falkenauer's Performance					Bins				
RDS5	VBS	BF	LSTM	GRU	ML	VBS	BF	LSTM	GRU	ML
VBS	-1.000	**0.900**	0.001	0.001	0.001	-1.000	**0.900**	0.001	0.001	0.001
BF	**0.900**	-1.000	0.001	0.001	0.001	**0.900**	-1.000	0.001	0.001	0.001
LSTM	0.001	0.001	-1.000	0.004	**0.900**	0.001	0.001	-1.000	0.010	0.038
GRU	0.001	0.001	0.004	-1.000	0.018	0.001	0.001	0.010	-1.000	0.001
ML	0.001	0.001	**0.900**	0.018	-1.000	0.001	0.001	0.038	0.001	-1.000

(a) RDS5

	Falkenauer's Performance					Bins				
RDS6	VBS	BF	LSTM	GRU	ML	VBS	BF	LSTM	GRU	ML
VBS	-1.000	**0.900**	0.001	0.001	0.001	-1.000	**0.900**	0.001	0.001	0.001
BF	**0.900**	-1.000	0.001	0.001	0.001	**0.900**	-1.000	0.001	0.001	0.001
LSTM	0.001	0.001	-1.000	0.001	**0.345**	0.001	0.001	-1.000	0.001	**0.373**
GRU	0.001	0.001	0.001	-1.000	0.001	0.001	0.001	0.001	-1.000	0.001
ML	0.001	0.001	**0.345**	0.001	-1.000	0.001	0.001	**0.373**	0.001	-1.000

(b) RDS6

	Falkenauer's Performance					Bins				
RDS7	VBS	BF	LSTM	GRU	ML	VBS	BF	LSTM	GRU	ML
VBS	-1.000	**0.900**	0.001	0.001	0.001	-1.000	**0.900**	0.001	0.001	0.001
BF	**0.900**	-1.000	0.001	0.001	0.001	**0.900**	-1.000	0.001	0.001	0.001
LSTM	0.001	0.001	-1.000	0.004	0.001	0.001	0.001	-1.000	0.001	0.001
GRU	0.001	0.001	0.004	-1.000	**0.485**	0.001	0.001	0.001	-1.000	**0.652**
ML	0.001	0.001	0.001	**0.485**	-1.000	0.001	0.001	0.001	**0.652**	-1.000

(c) RDS7

	Falkenauer's Performance					Bins				
RDS8	VBS	BF	LSTM	GRU	ML	VBS	BF	LSTM	GRU	ML
VBS	-1.000	**0.900**	0.001	0.001	0.001	-1.000	**0.900**	0.001	0.001	0.001
BF	**0.900**	-1.000	0.001	0.001	0.001	**0.900**	-1.000	0.001	0.001	0.001
LSTM	0.001	0.001	-1.000	**0.900**	0.001	0.001	0.001	-1.000	**0.900**	0.001
GRU	0.001	0.001	**0.900**	-1.000	0.001	0.001	0.001	**0.900**	-1.000	0.001
ML	0.001	0.001	0.001	0.001	-1.000	0.001	0.001	0.001	0.001	-1.000

(d) RDS8

Table A.6 The Confusion Matrix of the LSTM, GRU and Best ML models in experiments on DS(1-4) from the test set

	LSTM				GRU				RF			
Heuristic	BF	FF	NF	WF	BF	FF	NF	WF	BF	FF	NF	WF
BF	142	57	0	1	133	62	0	5	128	49	0	23
FF	67	125	0	8	66	131	0	3	63	92	0	45
NF	0	0	200	0	0	0	199	1	0	5	193	2
WF	1	4	3	192	4	6	2	188	19	50	4	127

(a) DS1

	LSTM				GRU				SVM			
Heuristic	BF	FF	NF	WF	BF	FF	NF	WF	BF	FF	NF	WF
BF	195	3	0	2	194	6	0	0	161	21	15	3
FF	10	179	1	10	9	183	1	7	20	117	20	43
NF	0	4	180	16	0	4	186	10	18	43	61	78
WF	3	13	6	178	1	12	8	179	4	35	31	130

(b) DS2

	LSTM				GRU				NN			
Heuristic	BF	FF	NF	WF	BF	FF	NF	WF	BF	FF	NF	WF
BF	119	81	0	0	132	66	1	1	128	46	0	26
FF	36	163	0	1	71	126	0	3	88	65	0	47
NF	2	1	196	1	0	0	200	0	0	0	197	3
WF	1	9	1	189	0	2	0	198	18	32	1	149

(c) DS3

	LSTM				GRU				SVM			
Heuristic	BF	FF	NF	WF	BF	FF	NF	WF	BF	FF	NF	WF
BF	194	4	0	2	191	6	0	3	155	34	0	11
FF	6	185	1	8	5	190	0	5	43	116	6	35
NF	0	0	196	4	0	0	200	0	1	10	169	20
WF	0	8	1	191	3	7	1	189	10	32	14	144

(d) DS4

Appendix B

Chapter 5 Appendices

Table B.1 The input and output length of the Encoder-Decoder after the padding process including item indexes, separators and end of sequences values

	DS1			DS2		
Output Length	9	18	27	12	24	34
Input Length	6	12	18	6	12	18

Table B.2 Range of values that used in the Encoder-Decoder LSTM hyper-parameters tuning; the table also shows the final selected values. The rest of the hyper-parameters are the default values provided by Keras

	#Epoch	Batch Size	#Layer	Memory Unites	Optimiser	LR	Loss Function
Range	[50,200]	[32, 2048]	[1_1 - 4_1, 2_2, 3_3]	[32, 2048]	adam	[0.0001, 0.001]	categ_crossentropy
Best	100	128	3_1 + Full-Connect Layer	1024	adam	0.001	categ_crossentropy

Table B.3 P-values between Encoder-Decoder LSTM and VBS for the test set from DS[1,2] in terms of Falkenauer's performance and number of bins using Wilcoxon Signed-Rank Test

	Falkenauer's performance	Number of bins used
DS1	0.0004	0.15
DS2	e-215	e-63

```
_________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
=================================================================================
 input_1 (InputLayer)           [(None, None, 61)]   0           []

 lstm (LSTM)                    (None, None, 1024)   4448256     ['input_1[0][0]']

 lstm_1 (LSTM)                  (None, None, 1024)   8392704     ['lstm[0][0]']

 input_2 (InputLayer)           [(None, None, 153)]  0           []

 lstm_2 (LSTM)                  [(None, 1024),       8392704     ['lstm_1[0][0]']
                                 (None, 1024),
                                 (None, 1024)]

 lstm_3 (LSTM)                  [(None, None, 1024)  4825088     ['input_2[0][0]',
                                , (None, 1024),                   'lstm_2[0][1]',
                                 (None, 1024)]                    'lstm_2[0][2]']

 dense (Dense)                  (None, None, 153)    156825      ['lstm_3[0][0]']

=================================================================================
Total params: 26,215,577
Trainable params: 26,215,577
Non-trainable params: 0
_________________________________________________________________________________
```

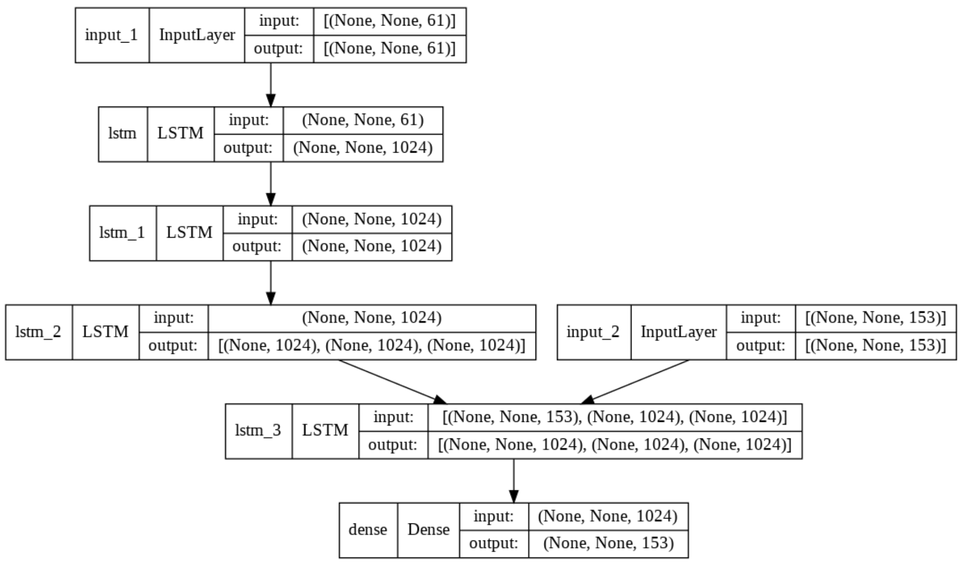


Fig. B.1 Schematic description of the Encoder-Decoder LSTM model (used in experiment #1 in Table 5.3 as an example) shows the layers and number of trainable parameters

Appendix C

Chapter 6 Appendices

Table C.1 The duplicate inputs with conflicted outputs percentages per dataset used

	DS1-3 bins	DS1-20 bins	DS2-3 bins	DS2-20 bins
Duplicate inputs with conflicted outputs	0.18%	0.07%	0.02%	0.07%

Table C.2 Range of values that used in the Encoder-Decoder LSTM hyper-parameters tuning; the table also shows the final selected values and the Neural Network hyper-parameters. The rest of the hyper-parameters are the default values provided by Keras

	#Epoch	Batch Size	#Layer	Memory Units/Neurons	Optimiser	Loss Function
Tuning Range	100	[16, 2048]	[1_1 - 4_1, 2_2]	[32, 2048]	adam	categorical_crossentropy
Best-LSTM	100	128	1_1 + Full-Connect Layer	1024	adam	categorical_crossentropy
NN	600	128	4	(16)-(32)-(16)-(6 or 40)	adam	categorical_crossentropy

Table C.3 Special Cases: corrections applied to handle cases with invalid decisions or multiple conflicting actions with using 3 bins as an example

Sequence of Actions	Description	Type	Corrected Decision	Description
0,0,0	No packing action is provided	Invalid	2,0,0	Close the first bin and open new one to pack the item in
1,0,0 / 0,1,0 / 0,0,1	The bin-capacity is broken	Invalid	2,0,0 / 0,2,0 / 0,0,2	Close the chosen bin and open new one to pack the item in
1,1,0 / 1,1,1 / 0,1,1	Multiple packing actions	Conflicting	1,0,0 / 1,0,0 / 0,1,0	Pack the item in the first chosen bin
2,2,0 / 2,2,2 / 0,2,2	Multiple "open new bin" actions	Conflicting	2,0,0 / 2,0,0 / 0,2,0	Close the first chosen bin and open new one to pack the item in
2,1,0 / 2,1,1 / 2,1,2	Multiple mixed actions	Conflicting	2,0,0	Close the first chosen bin and open new one to pack the item in

```
________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================
input_1 (InputLayer)            (None, None, 152)    0
________________________________________________________________________________
input_2 (InputLayer)            (None, None, 153)    0
________________________________________________________________________________
lstm_1 (LSTM)                   [(None, 1024), (None 4820992     input_1[0][0]
________________________________________________________________________________
lstm_2 (LSTM)                   [(None, None, 1024), 4825088     input_2[0][0]
                                                                 lstm_1[0][1]
                                                                 lstm_1[0][2]
________________________________________________________________________________
dense_1 (Dense)                 (None, None, 153)    156825      lstm_2[0][0]
================================================================================
Total params: 9,802,905
Trainable params: 9,802,905
Non-trainable params: 0
________________________________________________________________________________
```

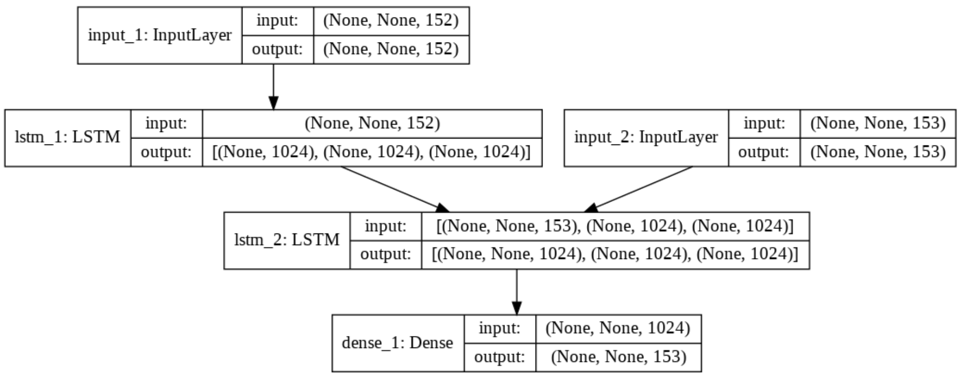

input_1: InputLayer
input: (None, None, 152)
output: (None, None, 152)
lstm_1: LSTM
input: (None, None, 152)
output: [(None, 1024), (None, 1024), (None, 1024)]
input_2: InputLayer
input: (None, None, 153)
output: (None, None, 153)
lstm_2: LSTM
input: [(None, None, 153), (None, 1024), (None, 1024)]
output: [(None, None, 1024), (None, 1024), (None, 1024)]
dense_1: Dense
input: (None, None, 1024)
output: (None, None, 153)

Fig. C.1 Schematic description of the Encoder-Decoder LSTM model (used in the proposed approach in Chapter 6) shows the layers and number of trainable parameters

Table C.4 P-values between the new VBSs and the baseline VBS over the differenttestsets (150 instances) from DS(1,2) in terms of Falkenauer's performance and numberof bins using Friedman test followed by Nemenyi post-hoc test with 5% confidence level. For all the cases, Friedman test obtains p-values = $0 \ll 5\%$, i.e. reject the null hypothesis that the median performance is the same for all the VBSs mentioned and the tables show the p-values of the Nemenyi post-hoc test.

	Falkenauer's Performance					Bins				
DS1- 3 Bins	VBS1	VBS2	VBS3	VBS4	VBS5	VBS1	VBS2	VBS3	VBS4	VBS5
VBS1	-1.000	0.005	0.019	0.001	0.001	-1.000	**0.680**	**0.900**	**0.471**	0.010
VBS2	0.005	-1.000	**0.900**	**0.139**	0.001	**0.680**	-1.000	**0.900**	**0.900**	**0.280**
VBS3	0.019	**0.900**	-1.000	**0.051**	0.001	**0.900**	**0.900**	-1.000	**0.804**	**0.059**
VBS4	0.001	**0.139**	**0.051**	-1.000	0.001	**0.471**	**0.900**	**0.804**	-1.000	**0.493**
VBS5	0.001	0.001	0.001	0.001	-1.000	0.010	**0.280**	**0.059**	**0.493**	-1.000

(a) DS1- 3 Bins

	Falkenauer's Performance					Bins				
DS1- 20 Bins	VBS1	VBS2	VBS3	VBS4	VBS5	VBS1	VBS2	VBS3	VBS4	VBS5
VBS1	-1.000	0.032	**0.900**	0.032	0.001	-1.000	**0.846**	**0.900**	**0.846**	**0.113**
VBS2	0.032	-1.000	0.032	**0.900**	0.011	**0.846**	-1.000	**0.846**	**0.900**	**0.597**
VBS3	**0.900**	0.032	-1.000	0.032	0.001	**0.900**	**0.846**	-1.000	**0.846**	**0.113**
VBS4	0.032	**0.900**	0.032	-1.000	0.011	**0.846**	**0.900**	**0.846**	-1.000	**0.597**
VBS5	0.001	0.011	0.001	0.011	-1.000	**0.113**	**0.597**	**0.113**	**0.597**	-1.000

(b) DS1- 20 Bins

	Falkenauer's Performance					Bins				
DS2- 3 Bins	VBS1	VBS2	VBS3	VBS4	VBS5	VBS1	VBS2	VBS3	VBS4	VBS5
VBS1	-1.000	0.023	**0.075**	0.001	0.001	-1.000	**0.846**	**0.900**	**0.690**	0.001
VBS2	0.023	-1.000	**0.900**	**0.437**	0.001	**0.846**	-1.000	**0.900**	**0.900**	0.006
VBS3	**0.075**	**0.900**	-1.000	**0.212**	0.001	**0.900**	**0.900**	-1.000	**0.900**	0.001
VBS4	0.001	**0.437**	**0.212**	-1.000	0.001	**0.690**	**0.900**	**0.900**	-1.000	0.015
VBS5	0.001	0.001	0.001	0.001	-1.000	0.001	0.006	0.001	0.015	-1.000

(c) DS2- 3 Bins

	Falkenauer's Performance					Bins				
DS2- 20 Bins	VBS1	VBS2	VBS3	VBS4	VBS5	VBS1	VBS2	VBS3	VBS4	VBS5
VBS1	-1.000	**0.701**	**0.900**	**0.701**	**0.271**	-1.000	**0.900**	**0.900**	**0.900**	**0.900**
VBS2	**0.701**	-1.000	**0.701**	**0.900**	**0.900**	**0.900**	-1.000	**0.900**	**0.900**	**0.900**
VBS3	**0.900**	**0.701**	-1.000	**0.701**	**0.271**	**0.900**	**0.900**	-1.000	**0.900**	**0.900**
VBS4	**0.701**	**0.900**	**0.701**	-1.000	**0.900**	**0.900**	**0.900**	**0.900**	-1.000	**0.900**
VBS5	**0.271**	**0.900**	**0.271**	**0.900**	-1.000	**0.900**	**0.900**	**0.900**	**0.900**	-1.000

(d) DS2- 20 Bins