Author:	Paul Kenneth Valdez Rigor
Supervisor (Main):	Assoc. Prof. Dr. Roliana Binti Ibrahim
Title:	Algorithm Selection for Subgraph Isomorphism Problems: A Reinforcement Learning Approach
Degree:	Masters of Science
Specialization:	Computer Science
Source:	main.tex
UTMThesis version:	v5.1
Date:	November 18, 2018

ALGORITHM SELECTION FOR SUBGRAPH ISOMORPHISM PROBLEMS: A REINFORCEMENT LEARNING APPROACH

PAUL KENNETH VALDEZ RIGOR

UNIVERSITI TEKNOLOGI MALAYSIA

"We hereby declare that we have read this dissertation and in our opinion this dissertation is sufficient in terms of scope and quality for the award of the degree of Masters of Science in Computer Science"

Signature	:	
Name	:	Assoc. Prof. Dr. Roliana Binti Ibrahim
Date	:	November 18, 2018

Signature	:	
Name	:	Dr. Haza Nuzly Abdull Hamed
Date	:	November 18, 2018

*Replace this page with the Cooperation Declaration form, which can be obtained from SPS or your faculty. This page is **OPTIONAL** when your research is done in collaboration with other institutions that requires their consent to publish the finding in this document.]*

ALGORITHM SELECTION FOR SUBGRAPH ISOMORPHISM PROBLEMS: A REINFORCEMENT LEARNING APPROACH

PAUL KENNETH VALDEZ RIGOR

A dissertation submitted in partial fulfilment of the
requirements for the award of the degree of
Masters of Science

Faculty of Computing
Universiti Teknologi Malaysia

OCTOBER 2018

I declare that this dissertation entitled *“Algorithm Selection for Subgraph Isomorphism Problems: A Reinforcement Learning Approach”* is the result of my own research except as cited in the references. The dissertation has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Signature	:	
Name	:	Paul Kenneth Valdez Rigor
Date	:	November 18, 2018

ABSTRACT

A set of hard computational problems can be solved more efficiently when using multiple algorithms instead of a single algorithm. This technique is called algorithm selection. An algorithm selection model captures the relationship between problem features and algorithm performance, using this knowledge to assign the best-performing algorithms to problems. Most approaches in literature use supervised learning techniques to learn the problem featurealgorithm performance relationship. This study presents a different perspective by training algorithm selection models using reinforcement learning—an experience-driven learning approach inspired by how humans and animals naturally learn from their environment. REINFORCE, the proposed reinforcement learning algorithm, is compared with pairwise random forest regression, a supervised learning algorithm from a previously related study. Experiment results show that an algorithm selection model trained using REINFORCE performs poorly as compared to one trained using pairwise random forest regression. However, zooming into the results reveal that REINFORCE did better than pairwise random forest regression on solving the easier problems, which constitutes the majority of the dataset. Still, REINFORCE fails because it is not able to perform well on solving the much harder problems on the dataset, which have a huge impact on the final calculation of algorithm selection performance scores. Nevertheless, this study maintains hope on the potential of reinforcement learning in producing algorithm selection models with better performance than models trained using supervised learning techniques.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AS	-	Algorithm Selection
ASlib	-	Algorithm Selection Library
COSEAL	-	Configuration and Selection of Algorithms
MCP	-	Misclassification Penalty
PRFR	-	Pairwise random forest regression
RL	-	Reinforcement Learning
SBS	-	Single Best Solver
SAT	-	Satisfiability
VBS	-	Virtual Best Solver
TSP	-	Travelling Salesman Problem
PORTSUB	-	Paper by Kotthoff *et al.* (2016)
REINFORCE	-	REward INcrement = Nonnegative Factor + Offset Reinforcement + Characteristic Eligibility
LLAMA	-	Leveraging Learning to Automatically Manage Algorithms

CHAPTER 1

INTRODUCTION

1.1 Introduction

The *algorithm selection* (AS) problem presents the question, ”Given a set of algorithms, how to choose one that best solves a given problem?” AS is best applied in solving intractable problems, whose solutions often require brute-force or heuristic approaches. An effective method of performing AS has broad implications on solving real-world problems which often are intractable in nature.

1.2 Problem Background

A survey of AS techniques (Kotthoff, 2016a) reported a majority of approaches that used supervised learning in training an AS model. Can reinforcement learning (RL) techniques be used? Recently, RL emerged as a superior approach to learning as demonstrated in applications such as automatic design of neural network architecture (Zoph and Le, 2016) and board game AI exhibiting superhuman performance (Silver *et al.*, 2016). This study explores the potential of RL in training AS models with superior performance. The AS models in this study focus on solving a set of hard computational tasks called subgraph isomorphism problems. This is an NP-complete problem under graph theory where the objective is to determine if a graph contains a pattern of a much smaller graph. A recent study tackled this problem by building and training an AS model using pairwise random forest regression (Kotthoff *et al.*, 2016). This study tries to improve upon the results of the previous study by proposing RL as an alternative to training AS models.

1.3 Problem Statement

This study addresses how reinforcement learning can be applied to algorithm selection for subgraph isomorphism problems. Specifically, it attempts to answer the following questions:

- How can the algorithm selection problem be viewed from the perspective of reinforcement learning?
- Can reinforcement learning be used to train an algorithm selection model in solving subgraph isomorphism problems?
- Does an algorithm selection model perform better when trained using reinforcement learning as compared to supervised learning?

1.4 Research Objectives

This study aims to use reinforcement learning in training an algorithm selection model for subgraph isomorphism problems. Specifically, it attempts to meet the following objectives:

- To prepare the subgraph isomorphism dataset for training the algorithm selection model.
- To implement a reinforcement learning algorithm for training an algorithm selection model.
- To evaluate whether reinforcement learning can improve the performance of the algorithm selection model as compared to supervised learning.

1.5 Scope of the Study

The constraints observed in this study are outlined below:

- The AS models are trained and tested using the subgraph isomorphism dataset generated from a previous related study by Kotthoff (Kotthoff *et al.*, 2016)
- The characteristics of the dataset impose restrictions on how training and testing can proceed.
  - The problems focus only on subgraph isomorphism problems.
  - The set of algorithms are predefined and fixed.
  - The model can only be trained to match one algorithm to one problem at a time.
- Only one RL algorithm is considered in this study.

1.6 Significance of the Study

This study contributes to the following:

- Investigates the feasibility of training algorithm selection models using reinforcement learning.
- Provides insights on how reinforcement learning algorithms can be applied to algorithm selection.
- Introduces the potential of reinforcement learning in training effective algorithm selection models.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter covers literature review on algorithm selection and reinforcement learning. The discussion focuses on background and methods surrounding these topics. The last section discusses the reinforcement learning approach to algorithm selection.

2.2 Algorithm Selection

The algorithm selection (AS) problem describes how a set of problems can be mapped to a set of algorithms such that the least amount of computing resources are spent to solve the whole set of problems (Rice, 1976). A model of the AS problem is depicted in Figure 2.1.

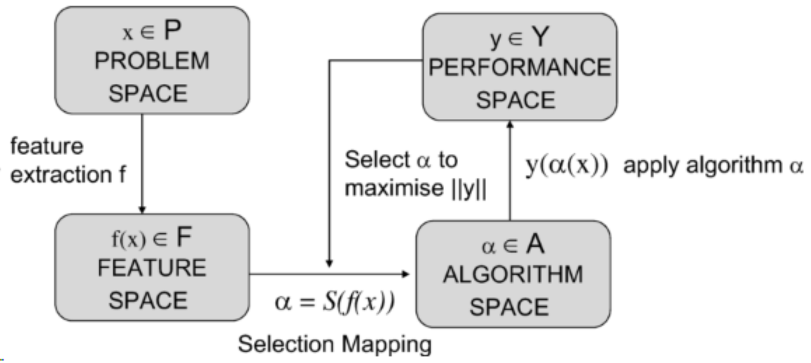


Figure 2.1: Model of the AS problem

An AS model is composed of four components: *problems, features, algorithms,* and *performance*.

1. Problem space ($\mathbf{P}$)
   The set of problem instances. Instances are often grouped into *problem classes*, which categorize their level of difficulty. The number of problems are usually vast and diverse.
2. Feature space ($\mathbf{F}$)
   The set of features describing the problem instances. Specifically, features define the measurable characteristics of problems, which provides an indication on the difficulty of problems.
3. Algorithm space ($\mathbf{A}$)
   The set of algorithms used to solving the problem instances. A large number of algorithms could be possibly used, but practically only a handful is considered.
4. Performance space ($\mathbf{Y}$)
   The set of metrics that defines how algorithm performance is measured.

Formally stating the AS problem,

Given problem instance $x \in \mathbf{P}$ with features $f(x) \in \mathbf{F}$, find a selection mapping $\alpha = S(f(x))$ in algorithm space $\mathbf{A}$, where the selected algorithm $\alpha \in \mathbf{A}$ provides the maximum performance $y(\alpha(x)) \in \mathbf{Y}$ in solving the problem instance $x \in \mathbf{P}$.

2.2.1 Motivation

AS is based on the observation that there is no single algorithm that can optimally solve all kinds of problems. The *No Free Lunch* (NFL) theorems assert that on a set of problems with varying degrees of difficulty, all algorithms perform the same on average (Wolpert and Macready, 1997). This implies that if an algorithm executes well on one class of problems, it pays off for decreased performance on other classes. NFL theorems have been demonstrated on problems such as search (Wolpert *et al.*, 1995), cross validation (Goutte, 1997), multiobjective optimization (Corne and Knowles, 2003), and sparse approximation (Xu *et al.*, 2012).

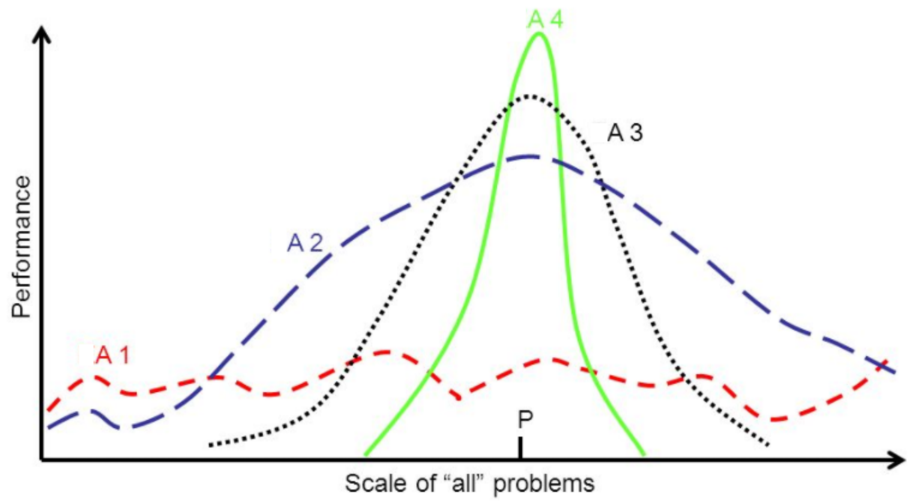


Figure 2.2: Typical distribution of algorithms with respect to performance

The NFL theorems suggest to shift the focus of solving problems to AS. Learning the problem characteristics can help in deciding which algorithm is most suitable in solving a problem. Also, a set of algorithms with complementary strengths is desirable. A collection of algorithms used for solving a set of problems is called an *algorithm portfolio*.

2.2.2 Algorithm Portfolios

Algorithm portfolios combine multiple algorithms with complementary strengths in order to solve problems more efficiently. This is inspired by *investment portfolios* concept in Economics where maximizing a utility with associated risks is handled by mixing strategies with desired risk and performance (Huberman *et al.*, 1997).

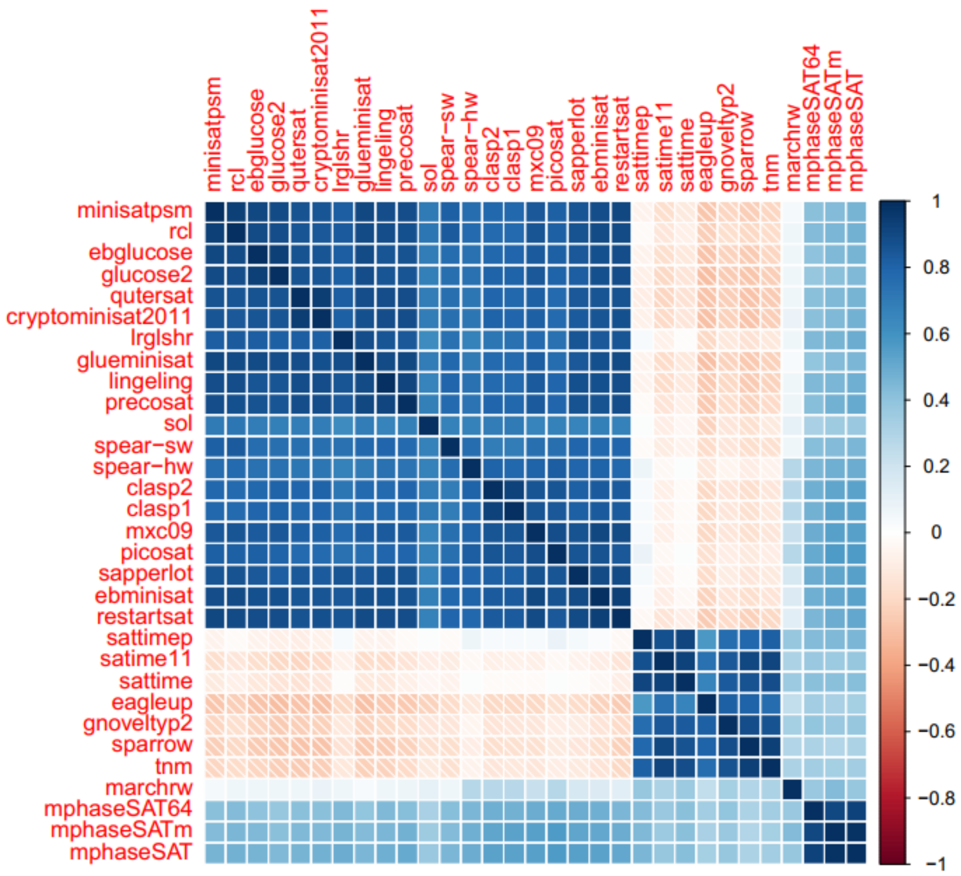
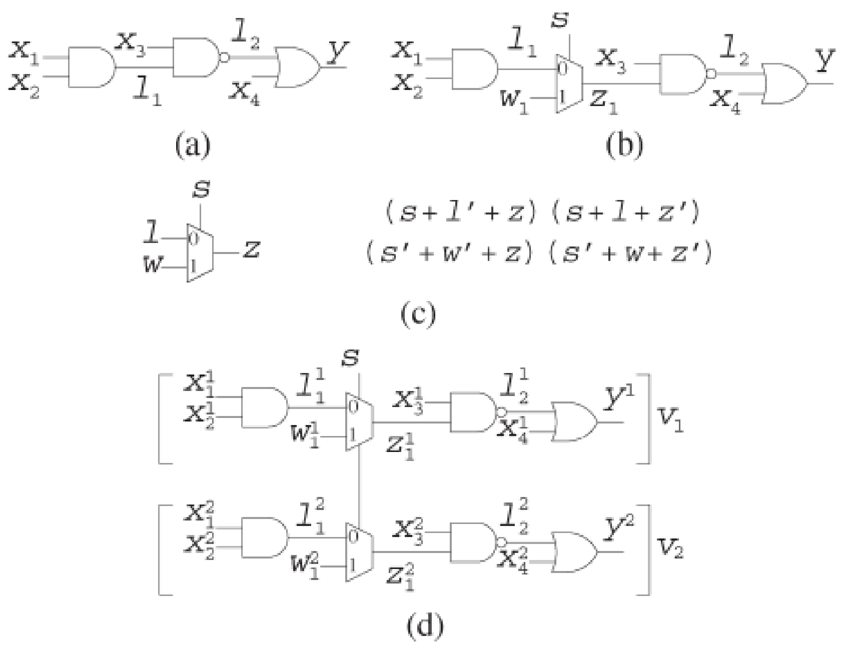


Figure 2.3: Correlation matrix of SAT solvers. Solvers with less correlation have more complementary performance. Red and blue boxes indicate negative and positive correlation respectively; shading represents the correlation strength (Bischl *et al.*, 2016)

Using algorithm portfolios have been proven effective on solving hard combinatorial problems (Gomes and Selman, 2001). Gomes and Selman concluded that algorithm portfolios can obtain superior performance by exploiting the variance in behavior among algorithms. One AS technique that takes advantage of this is called *presolving*. Presolving makes use of *presolvers*, which basically are algorithms that execute quickly and can solve a large number of problems (Xu *et al.*, 2008). Presolvers tend to eliminate the easier problems so AS can focus on solving the much harder problems.

2.2.3 Problem Domain

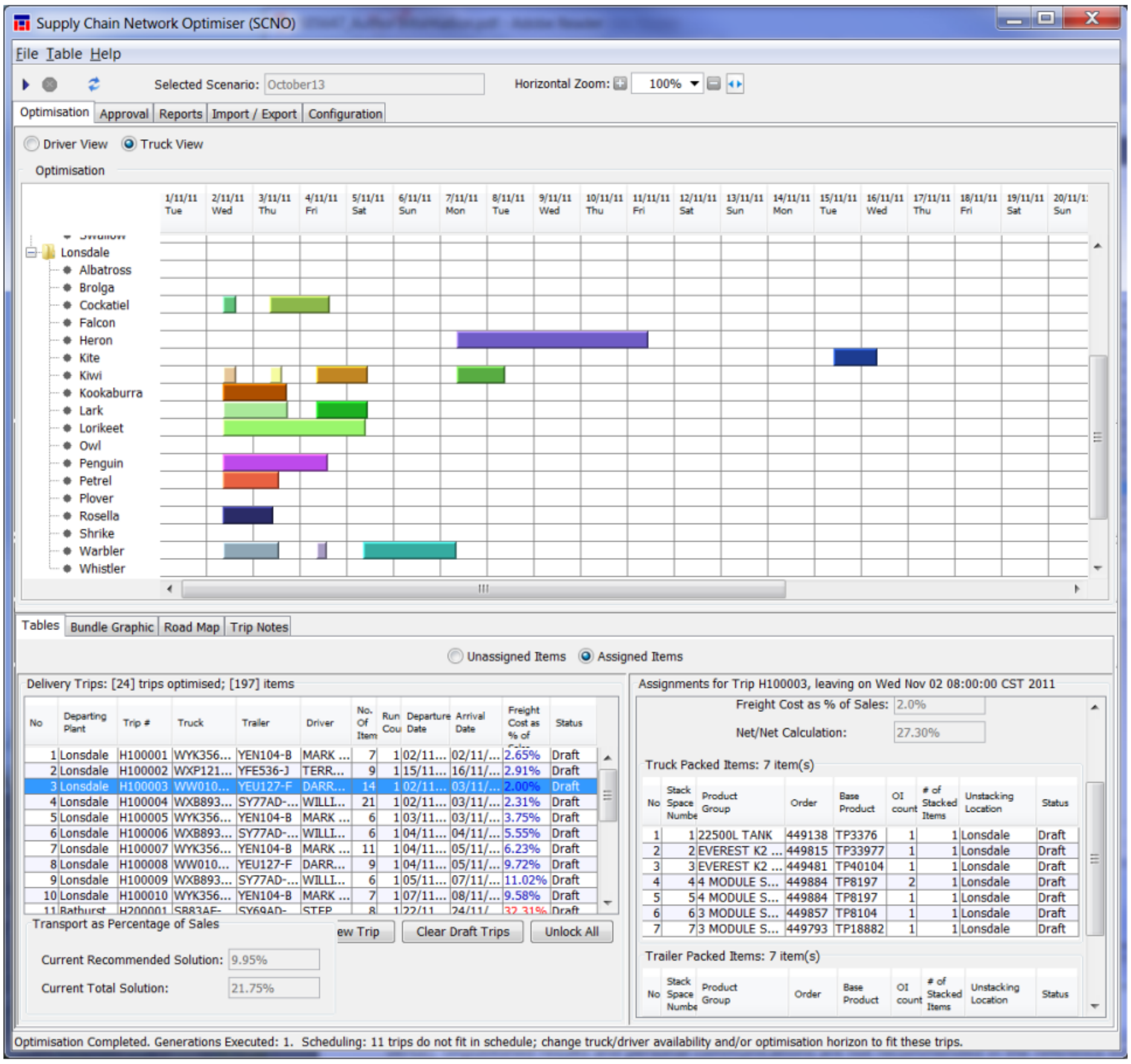
AS can be best applied on combinatorial search and optimization problems. One example is the Boolean satisfiability (SAT) problem, a classic NP-complete problem of finding assignments of variables to a Boolean formula such that it evaluates to TRUE. Successful AS implementations include SATzilla (Xu *et al.*, 2008), SATenstein (KhudaBukhsh *et al.*, 2009), and Hydra (Xu *et al.*, 2010). Efficient SAT solvers benefit applications such as IC design, computer-aided design and manufacturing, and automated theorem proving.



Modeling candidate fault locations. (a) Original circuit. (b) Modeling a potentially faulty line. (c) CNF representation of a multiplexer. (d) Circuit construction for two test vectors.

Figure 2.4: Fault-diagnosis of very-large-scale-integration (VLSI) circuits modelled as a SAT problem. (Smith *et al.*, 2005)

Travelling salesman problem (TSP), a famous example of an *NP*-hard problem, also benefits from AS. Given a list of points and the associated costs between pairs of points, the goal of TSP is to find a path with minimum total cost between two selected points. Points can represent destinations or, in a more abstract sense, nodes in a graph. Costs can refer to distance or any measure that represents the weight of relationship between a pair of nodes in a graph. An algorithm portfolio of inexact TSP solvers showed significant performance improvements after AS (Kotthoff *et al.*, 2015). The travelling thief problem (Bonyadi *et al.*, 2013), a combination of TSP and knapsack problem (Kellerer *et al.*, 2004), has also been tackled using AS techniques, resulting in substantial improvements over single solvers (Wagner *et al.*, 2018).


Supply Chain Network Optimiser (SCNO)
File Table Help
Horizontal Zoom: 100%
Optimisation Approval Reports Import / Export Configuration
Driver View Truck View
Optimisation
1/11/11 Tue 2/11/11 Wed 3/11/11 Thu 4/11/11 Fri 5/11/11 Sat 6/11/11 Sun 7/11/11 Mon 8/11/11 Tue 9/11/11 Wed 10/11/11 Thu 11/11/11 Fri 12/11/11 Sat 13/11/11 Sun 14/11/11 Mon 15/11/11 Tue 16/11/11 Wed 17/11/11 Thu 18/11/11 Fri 19/11/11 Sat 20/11/1
Lonsdale
Albatross
Brolga
Cockatiel
Falcon
Heron
Kite
Kiwi
Kookaburra
Lark
Lorikeet
Owl
Penguin
Petrel
Plover
Rosella
Shrike
Warbler
Whistler
Tables Bundle Graphic Road Map Trip Notes
Unassigned Items Assigned Items
ew Trip
Clear Draft Trips
Unlock All

Selected Scenario: October13

Delivery Trips: [24] trips optimised; [197] items

No	Departing Plant	Trip #	Truck	Trailer	Driver	No. Of Item	Run Cou	Departure Date	Arrival Date	Freight Cost as % of	Status
1	Lonsdale	H100001	WYK356...	YEN104-B	MARK ...	7	1	02/11...	02/11/...	2.65%	Draft
2	Lonsdale	H100002	WXP121...	YFE536-J	TERR...	9	1	15/11...	16/11/...	2.91%	Draft
3	Lonsdale	H100003	WW010...	YEU127-F	DARR...	14	1	02/11...	03/11/...	2.00%	Draft
4	Lonsdale	H100004	WXB893...	SY77AD-...	WILLI...	21	1	02/11...	03/11/...	2.31%	Draft
5	Lonsdale	H100005	WYK356...	YEN104-B	MARK ...	6	1	03/11...	03/11/...	3.75%	Draft
6	Lonsdale	H100006	WXB893...	SY77AD-...	WILLI...	6	1	04/11...	04/11/...	5.55%	Draft
7	Lonsdale	H100007	WYK356...	YEN104-B	MARK ...	11	1	04/11...	05/11/...	6.23%	Draft
8	Lonsdale	H100008	WW010...	YEU127-F	DARR...	9	1	04/11...	05/11/...	9.72%	Draft
9	Lonsdale	H100009	WXB893...	SY77AD-...	WILLI...	6	1	05/11...	07/11/...	11.02%	Draft
10	Lonsdale	H100010	WYK356...	YEN104-B	MARK ...	7	1	07/11...	08/11/...	9.58%	Draft
11	Bathurst	H200001	SB83AE-	SY69AD-	STEP	8	1	22/11	24/11/	32.31%	Draft

Transport as Percentage of Sales

Current Recommended Solution: 9.95%

Current Total Solution: 21.75%

Assignments for Trip H100003, leaving on Wed Nov 02 08:00:00 CST 2011

Freight Cost as % of Sales: 2.0%

Net/Net Calculation: 27.30%

Truck Packed Items: 7 item(s)

No	Stack Space Numbe	Product Group	Order	Base Product	OI count	# of Stacked Items	Unstacking Location	Status
1	1	22500L TANK	449138	TP3376	1	1	Lonsdale	Draft
2	2	EVEREST K2 ...	449815	TP33977	1	1	Lonsdale	Draft
3	3	EVEREST K2 ...	449481	TP40104	1	1	Lonsdale	Draft
4	4	4 MODULE S...	449884	TP8197	2	1	Lonsdale	Draft
5	5	4 MODULE S...	449884	TP8197	1	1	Lonsdale	Draft
6	6	3 MODULE S...	449857	TP8104	1	1	Lonsdale	Draft
7	7	3 MODULE S...	449793	TP18882	1	1	Lonsdale	Draft

Trailer Packed Items: 7 item(s)

No	Stack Space Numbe	Product Group	Order	Base Product	OI count	# of Stacked Items	Unstacking Location	Status

Optimisation Completed. Generations Executed: 1. Scheduling: 11 trips do not fit in schedule; change truck/driver availability and/or optimisation horizon to fit these trips.

Figure 2.5: Water tank delivery schedule generated by an optimizer software. This demonstrates the real-world application of the solution to travelling thief problem. (Stolk *et al.*, 2013)

In this study, AS is applied on solving subgraph isomorphism problems. This is an *NP*-complete problem whereby a smaller graph is located in a larger graph. As the number of graph nodes increases, the amount of effort expended in searching for subgraph isomorphisms exponentially increases. Using AS to search for subgraph isomorphisms have demonstrated significant performance improvements over standalone algorithms (Battiti and Mascia (2007); Kotthoff *et al.* (2016)). Subgraph isomorphism problems manifest in applications that use graphs to model problems. A few examples include chemistry, software engineering, and biocomputing.

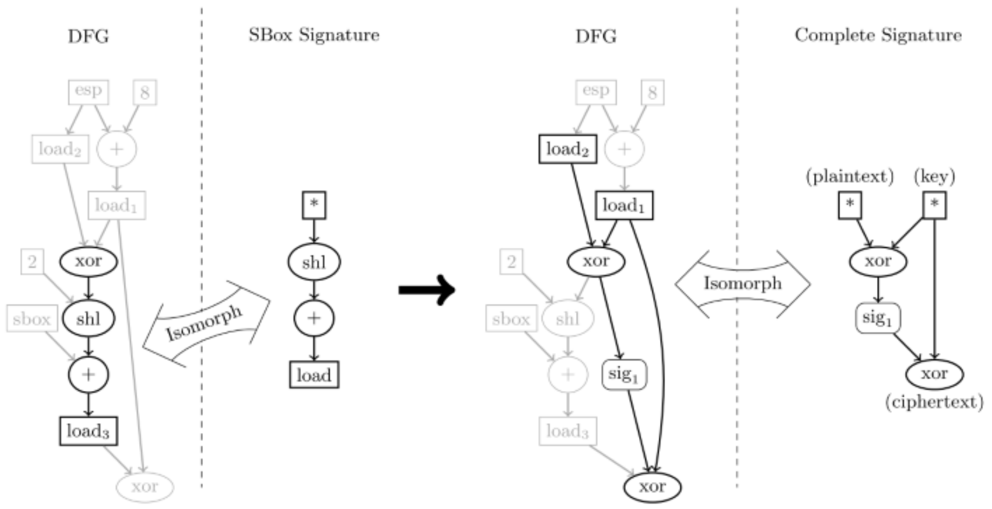


Figure 2.6: Identifying malware in executable files through recognition of cryptographic algorithm patterns. Searching for a cryptographic signature (pattern graph) from the program's data flow graph (DFG) is a subgraph isomorphism problem. (Lestringant *et al.*, 2015)

Combining the strengths of several algorithms through AS can lead to more efficient solutions to hard combinatorial problems. In retrospect, AS does not entirely discourage the formulation of new algorithms. Research in AS seeks not only to improve problem solving performance, but also to learn more about the nature of problems and the behaviors of algorithms used. It can also provide valuable insights in creating new algorithms or improving upon existing ones. Studies on new algorithms are just as important as AS research, since the effectiveness of AS still relies on how well individual algorithms perform.

2.3 Algorithm Selection Techniques

A comprehensive review of AS techniques can be referred from Smith-Miles (Smith-Miles, 2009) and Kotthoff (Kotthoff, 2016a). Typically, the following are considered when creating a new AS technique:

1. How can algorithm portfolios be constructed? (e.g. static portfolios vs. dynamic portfolios)
2. What algorithms to select and when? (e.g. single algorithm per problem vs. multiple algorithms with scheduling)
3. How to select algorithms? (modelling algorithm performance with respect to problems)

This study is limited to the use of a static portfolio and the selection of one algorithm per problem instance. This section mainly discusses techniques in (3).

On most approaches, machine learning is used to discover the relationship between algorithms and problems. This often includes a training phase, where algorithms are executed on a subset of problems to evaluate their performance. The general approach to AS (Bischl *et al.*, 2016) is depicted in Figure 2.7.

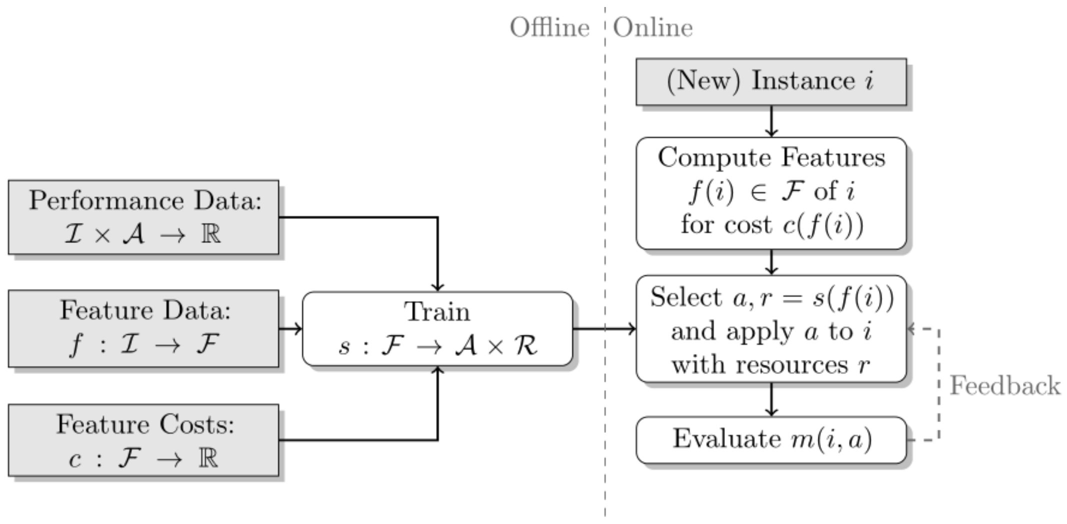

Offline
Online
(New) Instance i
Compute Features $f(i) \in \mathcal{F}$ of i for cost $c(f(i))$
Performance Data: $\mathcal{I} \times \mathcal{A} \to \mathbb{R}$
Feature Data: $f : \mathcal{I} \to \mathcal{F}$
Feature Costs: $c : \mathcal{F} \to \mathbb{R}$
Train $s : \mathcal{F} \to \mathcal{A} \times \mathcal{R}$
Select $a, r = s(f(i))$ and apply a to i with resources r
Feedback
Evaluate $m(i, a)$

Figure 2.7: AS workflow.

1. For each problem instance *i*, a vector of instance features $f(i) \in \mathbf{F}$ is computed.
2. A machine learning technique *s* selects an algorithm $a \in \mathbf{A}$ based on the feature vector from Step 1.
3. The selected algorithm *a* is applied to problem instance *i*.
4. Performance measure *m* is evaluated, taking into account feature computation costs and the performance of the selected algorithm.

Choosing a machine learning technique is often based from the nature of the problem to be solved. For instance, a classification model is viable for simple problem-algorithm mappings. Each problem can be labeled with the best-performing algorithm, then the classifier is trained to recognize problems and their corresponding optimal algorithms. One classifier implementation used a decision tree that assigned slower or faster algorithms to constraint satisfaction problems (Gent *et al.*, 2010).

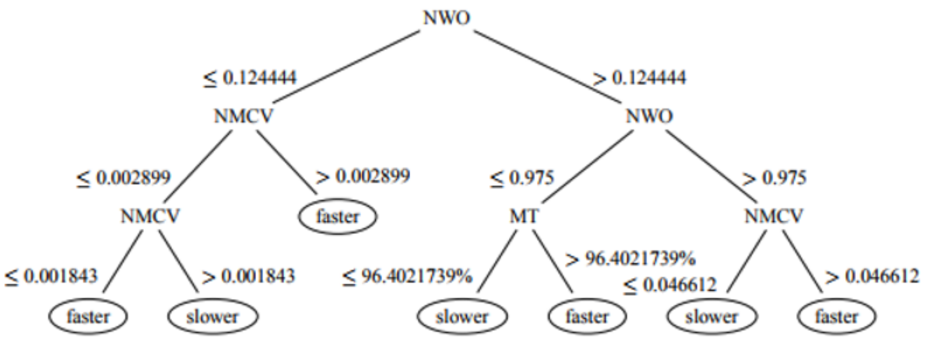


Figure 2.8: Decision tree classifier built using C4.5 algorithm.Nodes represent attribute rules the guide the decision of choosing a slower or a faster algorithm for a problem. (Gent *et al.*, 2010)

Alternatively, algorithm performance can be predicted using regression models. Training is slower since it is done for each algorithm per problem instance, as opposed to classification models where training iterates only for each problem instance. The advantage is that models are learned per individual algorithm instead of the whole algorithm portfolio, as with the case for classification models. SATzilla, a SAT solver that dominated SAT competitions, used ridge regression to learn an AS model of SAT problems (Xu *et al.*, 2008). For subgraph isomorphism problems, pairwise random forest regression stands as the current best approach Kotthoff *et al.* (2016).

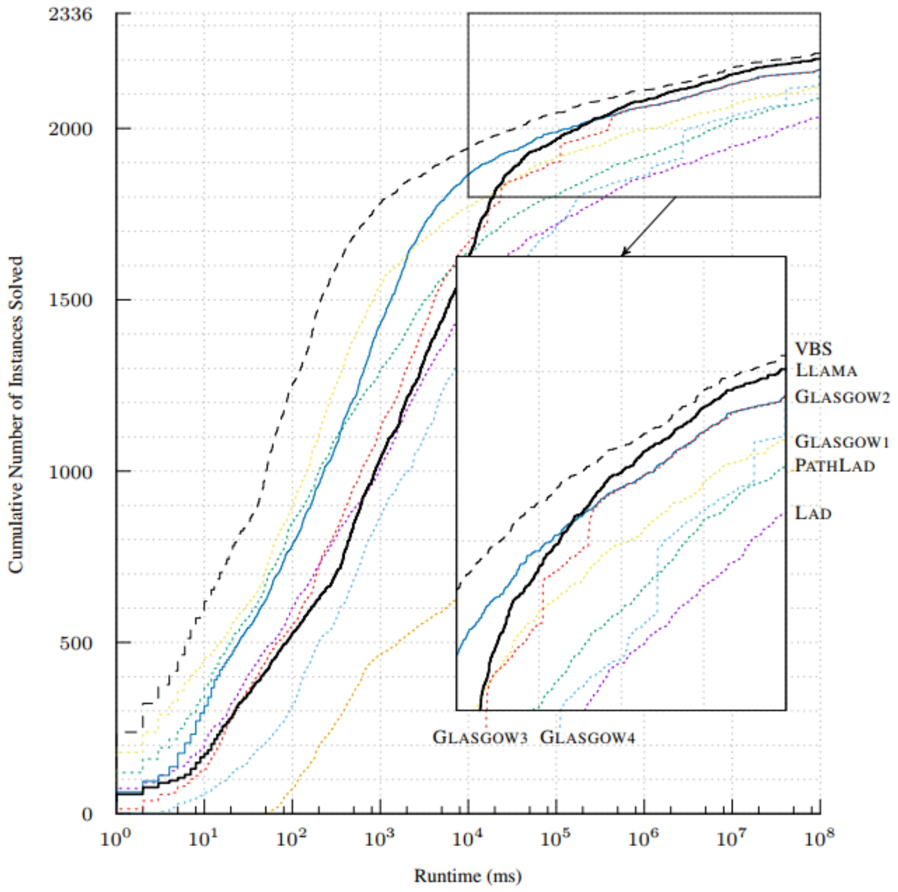


Figure 2.9: Performance comparison of subgraph isomorphism algorithms. *LLAMA* is the AS model trained using pairwise random forest regression. Its performance comes near the virtual best solver (VBS)the upper-bound of what an AS model can achieve in an algorithm portfolio. (Kotthoff *et al.*, 2016)

Another approach is to group problem instances into clusters and assign the cost-optimal solver for each cluster. This approach is useful when there is minimal knowledge about the problem. One example did not rely on problem features to do AS; instead a model of algorithm performance behavior is learned and then used to identify clusters of similar problems (Silverthorn and Miikkulainen, 2010). The AS model gathered roughly competitive results after comparing against SATzilla.

Method	**SAT Instances Solved**		
	random	crafted	indust.
Best Single	320.8 (3.7)	122.6 (3.8)	**153.5 (3.2)**
Random	261.6 (4.8)	89.3 (3.5)	54.5 (3.4)
SATzilla	407.5 (3.1)	125.6 (3.3)	137.6 (3.3)
Soft + Mult.	319.3 (8.2)	116.9 (4.9)	136.2 (4.3)
Soft + DCM	342.2 (8.2)	118.2 (4.3)	138.1 (5.0)
Hard + Mult.	340.3 (45.7)	104.8 (9.7)	129.4 (8.0)
Hard + DCM	**424.2 (12.9)**	**129.4 (6.5)**	146.0 (5.6)

Table 2.1: Comparison of the number of SAT problems solved between SATzilla vs. latent class clustering models. The last four methods in the table correspond to different latent class methods used. (Silverthorn and Miikkulainen, 2010)

Several machine-learning approaches to AS exist, such as hybrid regression-classification models (Kotthoff, 2012), ensembles (Kotthoff *et al.*, 2010), and reinforcement learning (Lagoudakis and Littman, 2000).

2.4 Reinforcement Learning Approach to Algorithm Selection

Training AS models typically use classification or regression methods (or a hybrid of these two) due to the complete availability of information on problems and algorithm performance. Reinforcement learning (RL) techniques are typically used when dynamic (a.k.a. online learning) AS is necessary, where the AS model continuously adapts and learns as it solves more problems. One example is an AS model trained using a modified Q-learning algorithm (Lagoudakis and Littman, 2000). The model continuously learns to select algorithms that minimize the overall time in solving all problems. A more sophisticated application used a multi-armed bandit approach to AS where the model progressively learns how to efficiently schedule and distribute problem solving across multiple CPUs (Gagliolo and Schmidhuber, 2009).

RL encompasses computational methods in learning from interactions with the environment to achieve long term goals (Sutton *et al.*, 1998). An *agent* perceives the *state* of its *environment* and receives *feedback* (instantly or time-delayed) for every *action* it takes. Feedback can be understood as rewards in which the agent seek to maximize over time. Learning is driven by the interaction between agent and environment: the environment provides stimulus to the agent, and then the agent makes decisions on how to respond. In RL terms this is called *policy* a set of rules that guides agent behavior.

Figure 2.10 summarizes the main idea behind RL. The RL model can be framed in the context of AS by mapping its elements to the model of AS problem (Figure 2.1). This is shown in Figure 2.11.

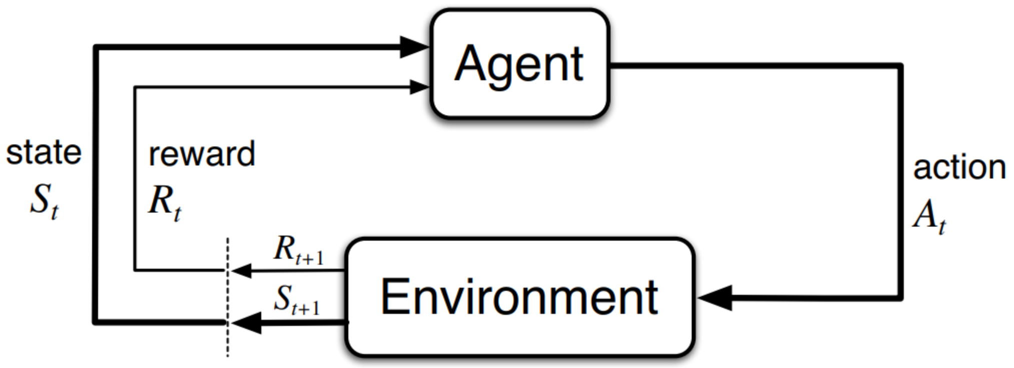


Figure 2.10: Reinforcement learning model

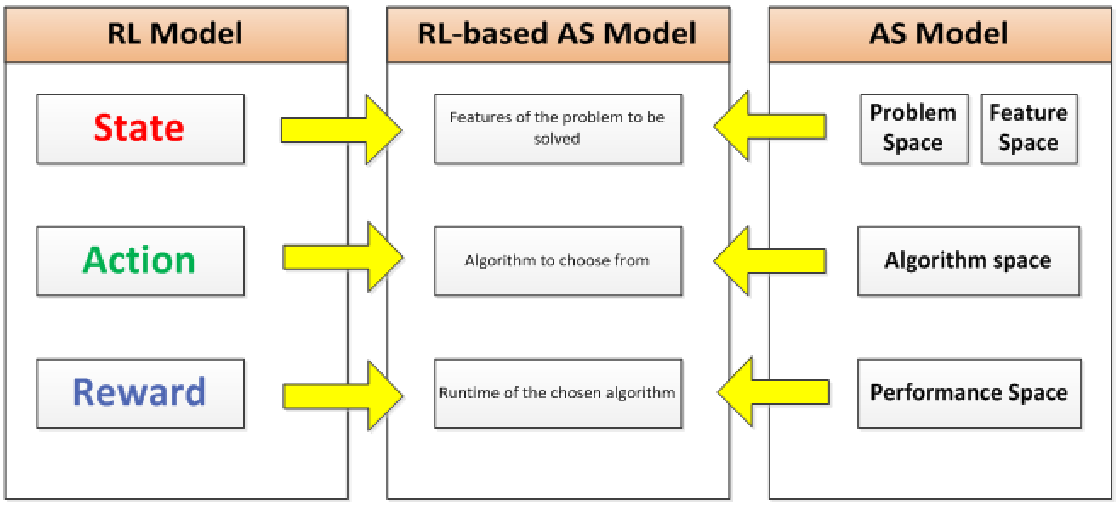


Figure 2.11: Reinforcement learning-based algorithm selection model

2.4.1 Policy Gradient Methods

The agent's *policy* is a function that outputs an action based from the input environment state. It can be learned by applying some performance measure and monitoring how it changes with respect to the environment state. *Policy gradient* methods represent a class of techniques under RL that learns a policy through change in performance measure. These methods seek to maximize long term rewards by optimizing performance.

$$\theta_{\mathbf{t+1}} = \theta_{\mathbf{t}} + \alpha \nabla J(\theta_t) \tag{2.1}$$

This study uses REINFORCE algorithm (Williams, 1992) to train an AS model. The acronym describes the policy parameter update equation used in its algorithm: **RE**ward **IN**crement *equals* **N**onnegative **F**actor *plus* **O**ffset **R**einforcement *plus* **C**haracteristic **E**ligibility.

$$\theta_{\mathbf{t+1}} = \theta_{\mathbf{t}} + \alpha G_t \frac{\nabla_\theta \pi(A_t|S_t, \theta_t)}{\pi(A_t|S_t, \theta_t)} \tag{2.2}$$

The intuition behind Equation 2.2 is as follows: each increment is proportional to the product of a reward G_t and a vector, the gradient of the probability of taking the action actually taken, $\nabla_\theta \pi(A_t|S_t, \theta_t)$, divided by the probability of taking that action $\pi(A_t|S_t, \theta_t)$. The vector is the direction in parameter space that most increases the probability of repeating the action A_t on future visits to state S_t. The update increases the parameter vector in this direction proportional to the reward, and inversely proportional to the action probability. The former makes sense because it causes the parameter to move most in the directions that favor actions that yield the highest reward. The latter makes sense because otherwise actions that are selected frequently are at an advantage (the updates will be more often in their direction) and might win out even if they do not yield the highest return. (Sutton *et al.*, 1998).

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

This chapter discusses how the research is executed. An overview of the research methodology is provided in the research framework. The dataset and software used in conducting the experiments are explained in detail. The general approach on how AS models are constructed and evaluated is also discussed.

3.2 Research Framework

The study underwent through four phases: *Identify Research Problem*, *Prepare Development Environment*, *Build and Evaluate AS Model*, and *Present Results*. The research flow is illustrated in Figure 3.1.

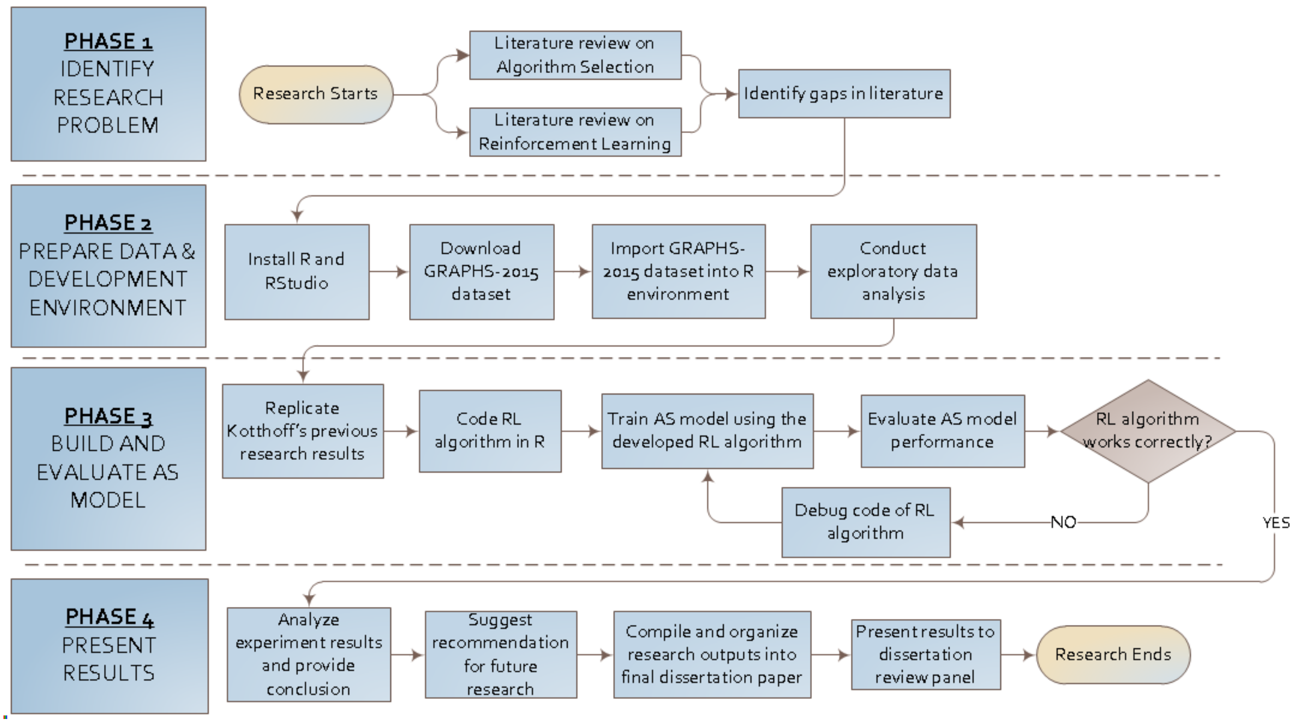

PHASE 1
IDENTIFY RESEARCH PROBLEM
Research Starts
Literature review on Algorithm Selection
Literature review on Reinforcement Learning
Identify gaps in literature
PHASE 2
PREPARE DATA & DEVELOPMENT ENVIRONMENT
Install R and RStudio
Download GRAPHS-2015 dataset
Import GRAPHS-2015 dataset into R environment
Conduct exploratory data analysis
PHASE 3
BUILD AND EVALUATE AS MODEL
Replicate Kotthoff's previous research results
Code RL algorithm in R
Train AS model using the developed RL algorithm
Evaluate AS model performance
RL algorithm works correctly?
NO
YES
Debug code of RL algorithm
PHASE 4
PRESENT RESULTS
Analyze experiment results and provide conclusion
Suggest recommendation for future research
Compile and organize research outputs into final dissertation paper
Present results to dissertation review panel
Research Ends

Figure 3.1: Research Framework

3.3 Phase 1: Identify Research Problem

This research is inspired by the Open Algorithm Selection Challenge 2017 (Lindauer *et al.*, 2017) which showcased the latest techniques in AS. The datasets and tools used in the challenge have been made publicly available, allowing anyone to easily replicate the results. This lowers the barrier to entry to AS research.

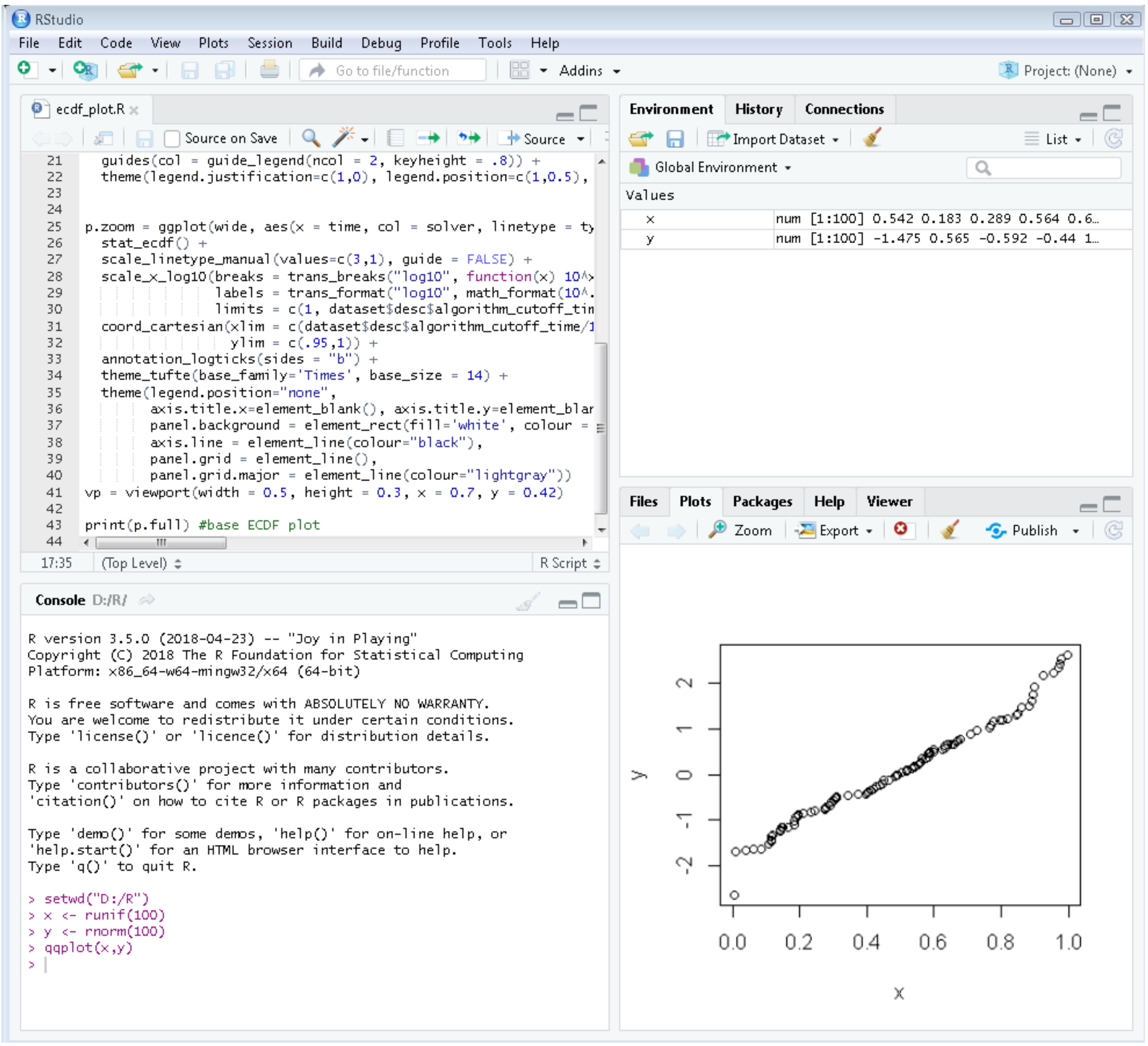
The AS survey papers by Kotthoff (Kotthoff *et al.*, 2016) and Smith-Miles (Smith-Miles, 2009) served as starting point in learning the background of AS. This study decided to focus on the application of AS to subgraph isomorphism problems. It has been chosen since a recent study of the same topic by Kotthoff and his colleagues (Kotthoff, 2016a) published data and results that can be easily reproduced. This study is referred to a number of times in this paper and will henceforth abbreviated to PORTSUB for convenience. This study follows up on PORTSUB by proposing RL as an alternative in training AS models.

3.4 Phase 2: Prepare Data and Development Environment

The experiments are executed on a server class desktop running Ubuntu Server 16.04 with 16 Gb RAM, Intel Xeon 4.0 GHz 12-core CPU, and an nVidia GeForce GTX1080Ti GPU. Experiments also ran well on a laptop installed with Windows 7 with 16 Gb RAM and Intel i7 3.0 GHz quad-core CPU. It is not really necessary to run the experiments on machines with specs as high as described since the computational requirements of training the AS models in this study are not very demanding. The experiments can be decently conducted on a Windows, MacOS, or GNU/Linux system with at least 4 Gb RAM and a 2.5 GHz quad-core CPU.

3.4.1 Install Development Software and Libraries

All experiments are executed in R. R is a programming language for statistical computing and graphics. Rs capabilities suit well the experimental needs of this research, which include mostly data handling, data analysis, and plotting results. To develop in R, using an IDE such as RStudio is recommended. RStudio is an integrated set of tools that helps one to be more productive with R.



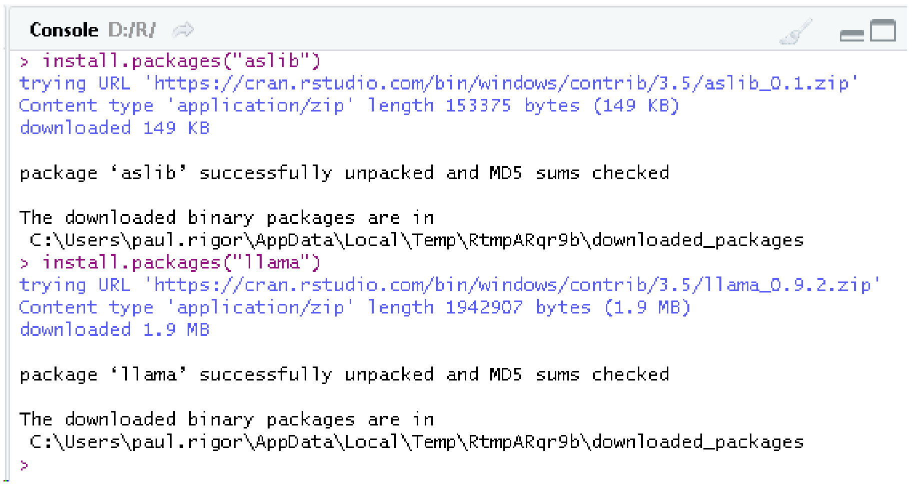
```
  guides(col = guide_legend(ncol = 2, keyheight = .8)) +
  theme(legend.justification=c(1,0), legend.position=c(1,0.5),


p.zoom = ggplot(wide, aes(x = time, col = solver, linetype = ty
  stat_ecdf() +
  scale_linetype_manual(values=c(3,1), guide = FALSE) +
  scale_x_log10(breaks = trans_breaks("log10", function(x) 10^>
                labels = trans_format("log10", math_format(10^.
                limits = c(1, dataset$desc$algorithm_cutoff_tin
  coord_cartesian(xlim = c(dataset$desc$algorithm_cutoff_time/1
                  ylim = c(.95,1)) +
  annotation_logticks(sides = "b") +
  theme_tufte(base_family='Times', base_size = 14) +
  theme(legend.position="none",
        axis.title.x=element_blank(), axis.title.y=element_blar
        panel.background = element_rect(fill='white', colour =
        axis.line = element_line(colour="black"),
        panel.grid = element_line(),
        panel.grid.major = element_line(colour="lightgray"))
vp = viewport(width = 0.5, height = 0.3, x = 0.7, y = 0.42)

print(p.full) #base ECDF plot
```

```
Values
  x    num [1:100] 0.542 0.183 0.289 0.564 0.6…
  y    num [1:100] -1.475 0.565 -0.592 -0.44 1…
```

```
R version 3.5.0 (2018-04-23) -- "Joy in Playing"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> setwd("D:/R")
> x <- runif(100)
> y <- rnorm(100)
> qqplot(x,y)
> 
```

Figure 3.2: RStudio IDE

Libraries in R are called *packages*. Packages bundle together code, data, documentation, and tests, and is easy to share with others (Wickham, 2015). Two R packages developed especially for AS research are available: these are ASlib and LLAMA.



```
> install.packages("aslib")
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/aslib_0.1.zip'
Content type 'application/zip' length 153375 bytes (149 KB)
downloaded 149 KB

package ‘aslib’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
 C:\Users\paul.rigor\AppData\Local\Temp\RtmpARqr9b\downloaded_packages
> install.packages("llama")
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.5/llama_0.9.2.zip'
Content type 'application/zip' length 1942907 bytes (1.9 MB)
downloaded 1.9 MB

package ‘llama’ successfully unpacked and MD5 sums checked

The downloaded binary packages are in
 C:\Users\paul.rigor\AppData\Local\Temp\RtmpARqr9b\downloaded_packages
>
```

Figure 3.3: Installing packages in R

3.4.1.1 Algorithm Selection Library (ASlib)

Algorithm Selection Benchmark Library (ASlib) is an R package for operating on and analyzing AS datasets. This package allows researchers to work on AS problems and perform reproducible comparisons of approaches (Bischl *et al.*, 2016). An ASlib dataset is called an *AS scenario*. This contains pre-computed results for an algorithm portfolio on a set of problem instances, i.e., the performance measure is known for all pairs of algorithms and instances. In addition, the set of pre-computed instance features and its associated costs (the time spent in computing a feature) is provided for each instance. Providing this information makes it easy to build AS models, which typically involves (1) generating problem instances, (2) constructing an algorithm portfolio, (3) coding the algorithms, and (4) building the AS model. ASlib enables researchers to focus on (4) by providing the necessary data on (1) - (3). Furthermore, ASlib allows a fair and convenient evaluation and comparison of algorithm selectors.

The basic structure of an ASlib scenario is shown on 3.1. The complete specification can be found online (Lindauer, 2016).

File Type	Filename	Description
Meta information file	description.txt	Global description file containing general information about the scenario, including the name of the scenario, performance measures, algorithms, features and limitations of computational resources.
Algorithm performance	algorithm_runs.arff	Contains performance measurements with possible repetitions and completion status of the algorithm runs. The performance metric can be arbitrary, e.g., runtime, solution quality, accuracy or loss.
Instance feature	feature_values.arff	Contains the feature vectors for all instances.
Feature costs	feature_costs.arff	Contains the costs of the feature groups, i.e., sets of features computed together.
Cross validation	cv.arff	Describes how to split the instance set into training and test sets to apply a standard machine learning approach to obtain an unbiased estimate of the performance of an algorithm selector.

Table 3.1: Basic structure of an ASlib scenario

3.4.1.2 Leveraging Learning to Automatically Manage Algorithms (LLAMA)

Several previous studies implement AS models using different software environments like MATLAB and C++. Moreover, the implementations are highly tuned and customized for a particular problem domain. The Leveraging Learning to Automatically Manage Algorithms (LLAMA) R package addresses this problem by providing a common infrastructure for researchers to build, evaluate, and apply AS models (Kotthoff, 2013).

At a high level, LLAMA takes problem features and algorithm performance data as input (imported to R using ASlib package), feeds it to the AS model, and then outputs the model performance results. LLAMA contains built-in AS model training algorithms; it also allows custom ones to be implemented. Lastly, built-in performance measures in LLAMA include number of solved problems, misclassification penalty (MCP), and penalized average runtime (PAR) score.

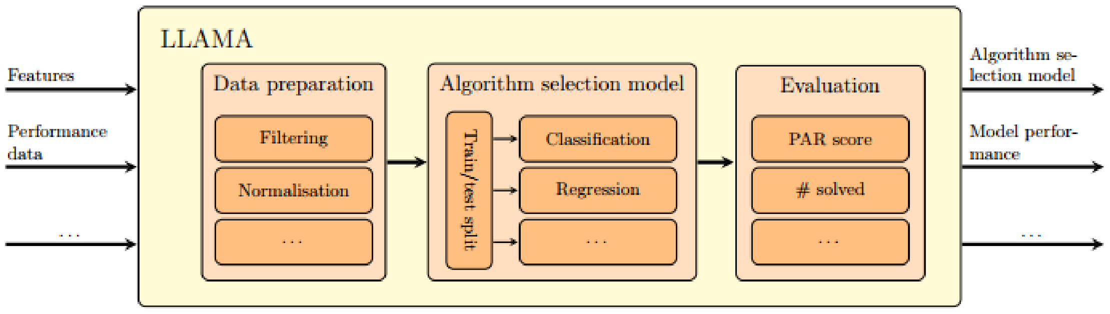
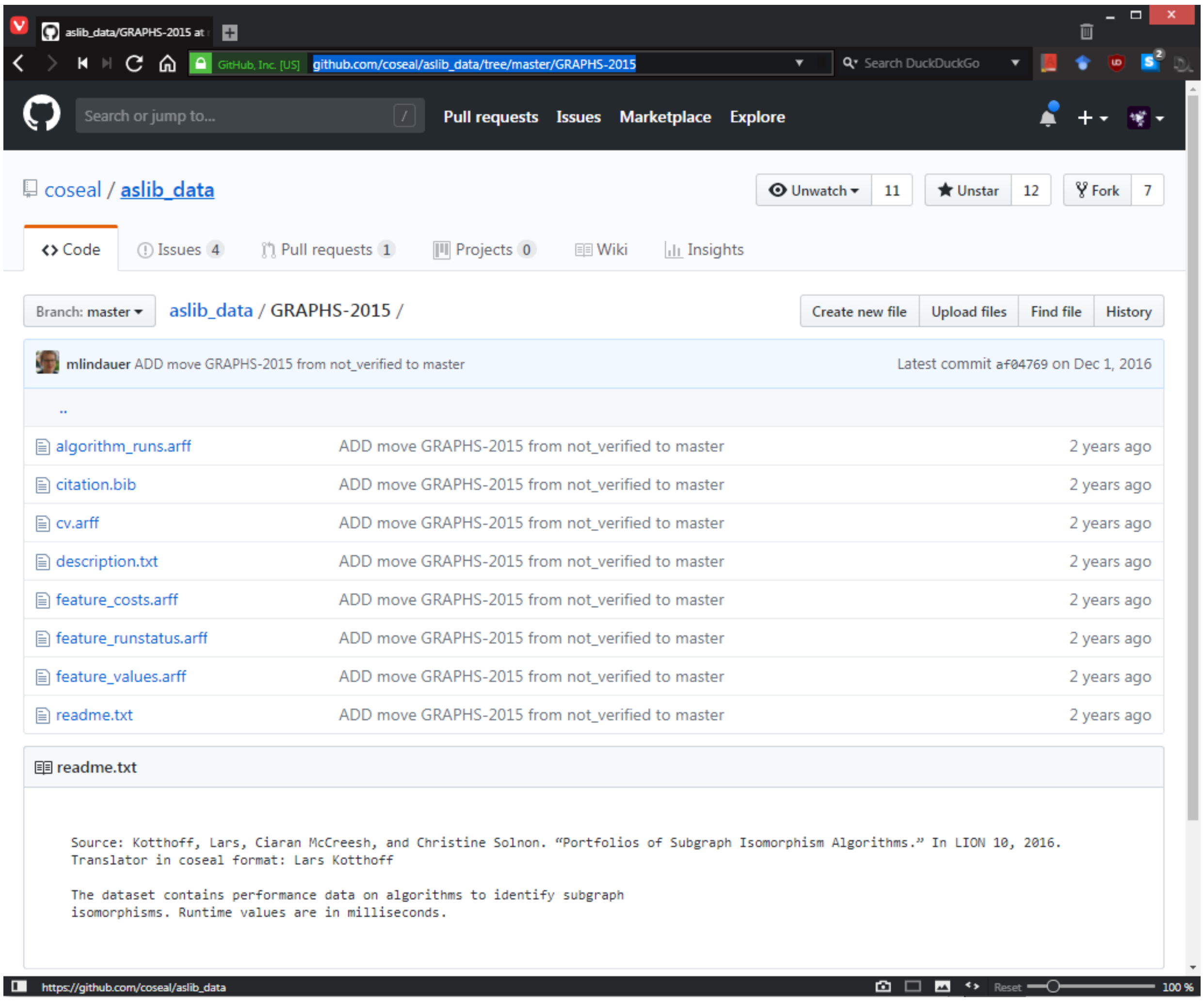


Figure 3.4: LLAMA framework

3.4.2 Prepare Dataset

PORTSUB published a dataset called GRAPHS-2015, which is used to train and test the AS models in this study. It is available online in GitHub (Kotthoff, 2016b) and is currently maintained by the Configuration and Selection of Algorithms (COSEAL) group.



coseal / aslib_data

aslib_data / GRAPHS-2015 /

mlindauer ADD move GRAPHS-2015 from not_verified to master — Latest commit af04769 on Dec 1, 2016

..		
algorithm_runs.arff	ADD move GRAPHS-2015 from not_verified to master	2 years ago
citation.bib	ADD move GRAPHS-2015 from not_verified to master	2 years ago
cv.arff	ADD move GRAPHS-2015 from not_verified to master	2 years ago
description.txt	ADD move GRAPHS-2015 from not_verified to master	2 years ago
feature_costs.arff	ADD move GRAPHS-2015 from not_verified to master	2 years ago
feature_runstatus.arff	ADD move GRAPHS-2015 from not_verified to master	2 years ago
feature_values.arff	ADD move GRAPHS-2015 from not_verified to master	2 years ago
readme.txt	ADD move GRAPHS-2015 from not_verified to master	2 years ago

readme.txt

```
Source: Kotthoff, Lars, Ciaran McCreesh, and Christine Solnon. “Portfolios of Subgraph Isomorphism Algorithms.” In LION 10, 2016.
Translator in coseal format: Lars Kotthoff

The dataset contains performance data on algorithms to identify subgraph
isomorphisms. Runtime values are in milliseconds.
```

Figure 3.5: GRAPHS-2015 GitHub page.

3.4.3 Replicate Previous Research Results

The results of PORTSUB are replicated to verify its validity and to later serve as reference for benchmarking the AS model trained using RL. PORTSUB used a supervised learning technique called pairwise random forest regression (PRFR) to train an AS model in solving subgraph isomorphism problems. An implementation of this algorithm is readily available in LLAMA.

3.5 Phase 3: Build and Evaluate AS Model

An AS model is constructed in R, using the ASlib and LLAMA libraries to load the dataset and perform evaluation. The RL algorithm REINFORCE is studied and applied to AS of subgraph isomorphism problems. Its implementation utilized several functions found from the TensorFlow R package. TensorFlow is an open-source machine learning library used for high performance numerical computation (Abadi *et al.*, 2016).

The effectiveness of AS is determined by comparing its performance against the virtual best solver (VBS) and the single best solver (SBS). VBS is the hypothetical model that does perfect AS, i.e., it always selects the best algorithm for each problem. SBS is the one algorithm from the algorithm portfolio that has the overall best performance across a set of problems. VBS defines the theoretical upper-bound performance that can be achieved by an AS model, while SBS defines the lower-bound performance. An AS model should at least get better performance than SBS to prove that AS is more effective than a single algorithm approach. An AS model becomes more effective as its performance gets closer to the VBS.

An AS model can be evaluated by its ability of how often it assigns the most optimal algorithms to a set of problems. In this study, the AS model performance is measured by calculating the mean misclassification penalty (MCP). This metric represents the additional time spent in solving problems when suboptimal algorithms were chosen. Individual algorithm performance is measured by its runtime, which is defined as the time taken by an algorithm to solve a problem instance.

VBS, by its definition, always have mean MCP equal to zero. Lower MCP values signify better AS model performance.

$$MCP = \frac{1}{N}\sum_{i=1}^{N}(t_{S_i} - t_{VBS_i}) \tag{3.1}$$

where:

$N =$	Total number of problem instances
$t_{S_i} =$	Runtime of the algorithm selected by the AS model
$t_{VBS_i} =$	Runtime of the algorithm selected by the VBS
$i =$	*ith* problem instance

3.6 Phase 4: Present Results

The last phase of this study compiles and organizes the findings into a full dissertation paper. The experiment results are discussed and conclusions are made. Lastly, recommendations for future research is presented.

CHAPTER 4

ALGORITHM SELECTION MODEL TRAINING ALGORITHMS

4.1 Introduction

This chapter explains the implementation details of the algorithms used in training the AS model, which are *pairwise random forest regression* (PRFR) and REINFORCE.

4.2 Data Preparation

Table 4.1 displays the summary on GRAPHS-2015 dataset.

Scenario ID	GRAPHS-2015
Source	Kotthoff (2016b)
Number of problem instances	5725
Number of features	35
Number of algorithms	7
Algorithm list	LAD
	SupplementalLAD
	VF2
	GLASGOW1
	GLASGOW2
	GLASGOW3
	GLASGOW4
Performance measure	Runtime (in milliseconds)

Table 4.1: GRAPHS-2015 dataset summary

Only a subset of GRAPHS-2015 was used for training AS models. A technique called *presolving* is used to eliminate the quickly solvable problem instances, leaving the harder problem instances for AS. The steps on removing presolved instances from GRAPHS-2015 were discussed in PORTSUB study. The flow and code is shown in Figures 4.1 and 4.2.

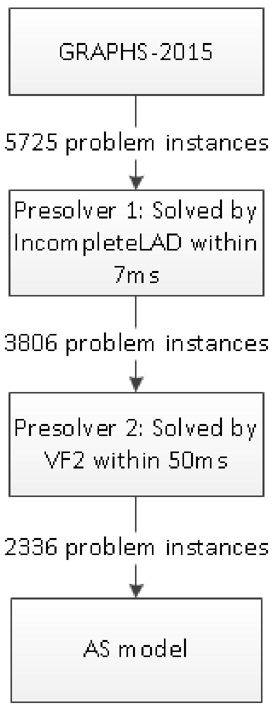

GRAPHS-2015
5725 problem instances
Presolver 1: Solved by IncompleteLAD within 7ms
3806 problem instances
Presolver 2: Solved by VF2 within 50ms
2336 problem instances
AS model

Figure 4.1: Filtering presolved instances from GRAPHS-2015. 2336 hard problems are left for AS.

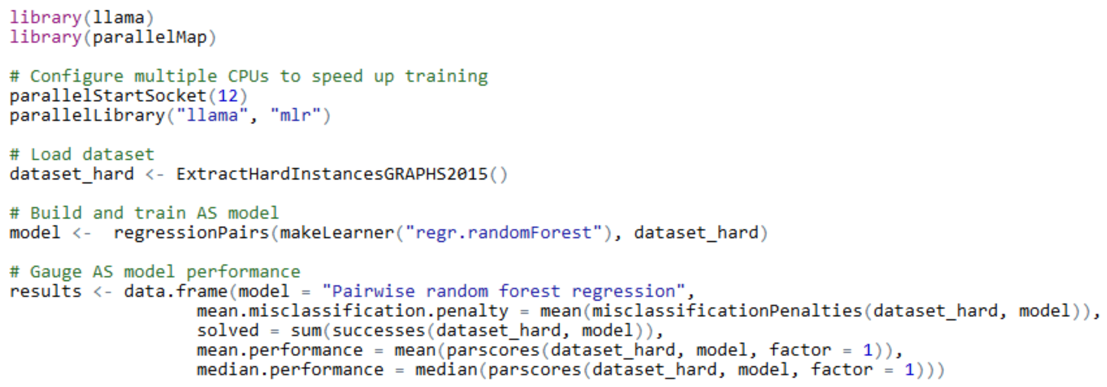
```r
library(aslib)
library (llama)

ExtractHardInstancesGRAPHS2015 <- function() {
  # Load dataset from file and convert to LLAMA object
  dataset_raw <- parseASScenario("GRAPHS-2015")
  dataset_structured <- convertToLlamaCVFolds(dataset_raw)

  #Presolver 1: Filter instances solved by IncompleteLAD
  dataset_notPresolved <- dataset_structured
  presolved_ids <- dataset_raw$feature.runstatus$instance_id[dataset_raw$feature.runstatus$lad_features == "presolved"]
  dataset_notPresolved$data <- subset(dataset_structured$data, !(dataset_structured$data$instance_id %in% presolved_ids))
  dataset_notPresolved$best <- subset(dataset_structured$best, !(dataset_structured$data$instance_id %in% presolved_ids))

  # Presolver 2: Filter instances solved by VF2 within 50 ms
  dataset_hard <- dataset_notPresolved
  dataset_hard$data <- subset(dataset_notPresolved$data, dataset_notPresolved$data$vf2 > 50)
  dataset_hard$best <- subset(dataset_notPresolved$best, dataset_notPresolved$data$vf2 > 50)

  # Regenerate train/test cross validation data splits
  dataset_hard <- cvFolds(dataset_hard, stratify = TRUE)

  # 2336 hard instances for algorithm selection
  return(dataset_hard)
}
```

Figure 4.2: R script for extracting hard instances from GRAPHS-2015

4.3 Pairwise Random Forest Regression

The AS model used in PORTSUB study was trained using a supervised learning approach called PRFR. It works as follows: for each pair of algorithms in the algorithm portfolio, a model is trained to predict the performance difference between them. If the first algorithm is better than the second, the difference is positive, otherwise negative. The algorithm with the highest cumulative performance difference, i.e. the most positive difference over all other algorithms, is chosen to be run.

The LLAMA R package contains an implementation of pairwise random forest regression written by Lars Kotthoff, one of the PORTSUB authors. Its usage in R is shown on Figure 4.3



```
library(llama)
library(parallelMap)

# Configure multiple CPUs to speed up training
parallelStartSocket(12)
parallelLibrary("llama", "mlr")

# Load dataset
dataset_hard <- ExtractHardInstancesGRAPHS2015()

# Build and train AS model
model <-  regressionPairs(makeLearner("regr.randomForest"), dataset_hard)

# Gauge AS model performance
results <- data.frame(model = "Pairwise random forest regression",
                  mean.misclassification.penalty = mean(misclassificationPenalties(dataset_hard, model)),
                  solved = sum(successes(dataset_hard, model)),
                  mean.performance = mean(parscores(dataset_hard, model, factor = 1)),
                  median.performance = median(parscores(dataset_hard, model, factor = 1)))
```

Figure 4.3: R script for building AS model trained using PRFR

4.4 REINFORCE

The adaptation of REINFORCE algorithm to AS is composed of three main components:

1. **Function approximator**

   An AS model can learn how algorithms match to problems with certain features. Patterns in problem features can be captured by function approximation methods, which translate feature values into parameters. For example, a simple function approximator such as linear regressor parameterizes inputs into slope and offset coefficients. Neural networks, a universal function approximator, represent parameters as network weights. Function approximation assists in generalizing AS model learning through discovery of feature patterns across problem instances.

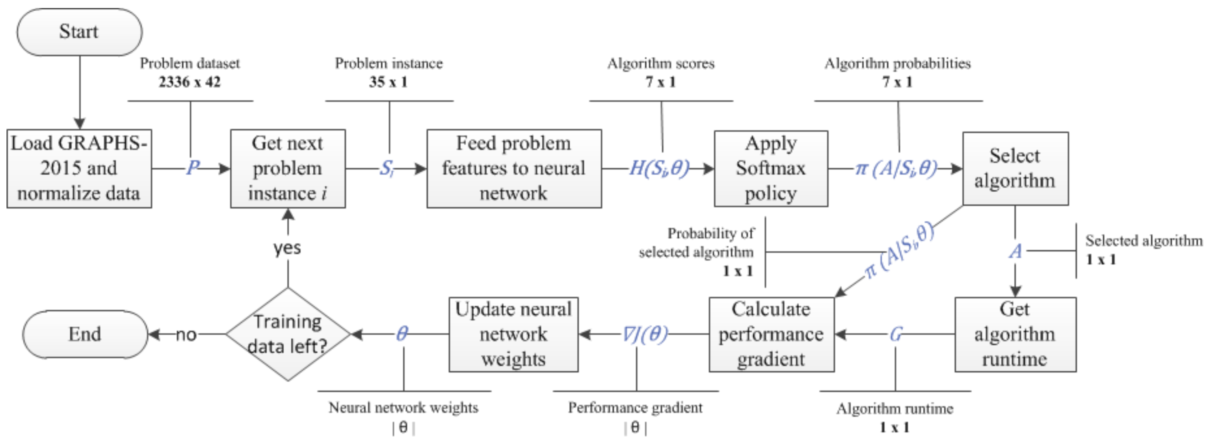
2. **Policy function**

A policy function takes function approximator parameters as input and computes a probability for each algorithm. Algorithm probabilities change with respect to problem features. The computed probabilities serve as the basis of selection among algorithms.

3. **Reward function**

The reward value is equivalent to the observed performance of the selected algorithm. It serves as feedback to the AS model to improve its capability in predicting optimal algorithms. Optionally, the reward value can be normalized or transformed to control how much the policy behavior should change with respect to feedback.

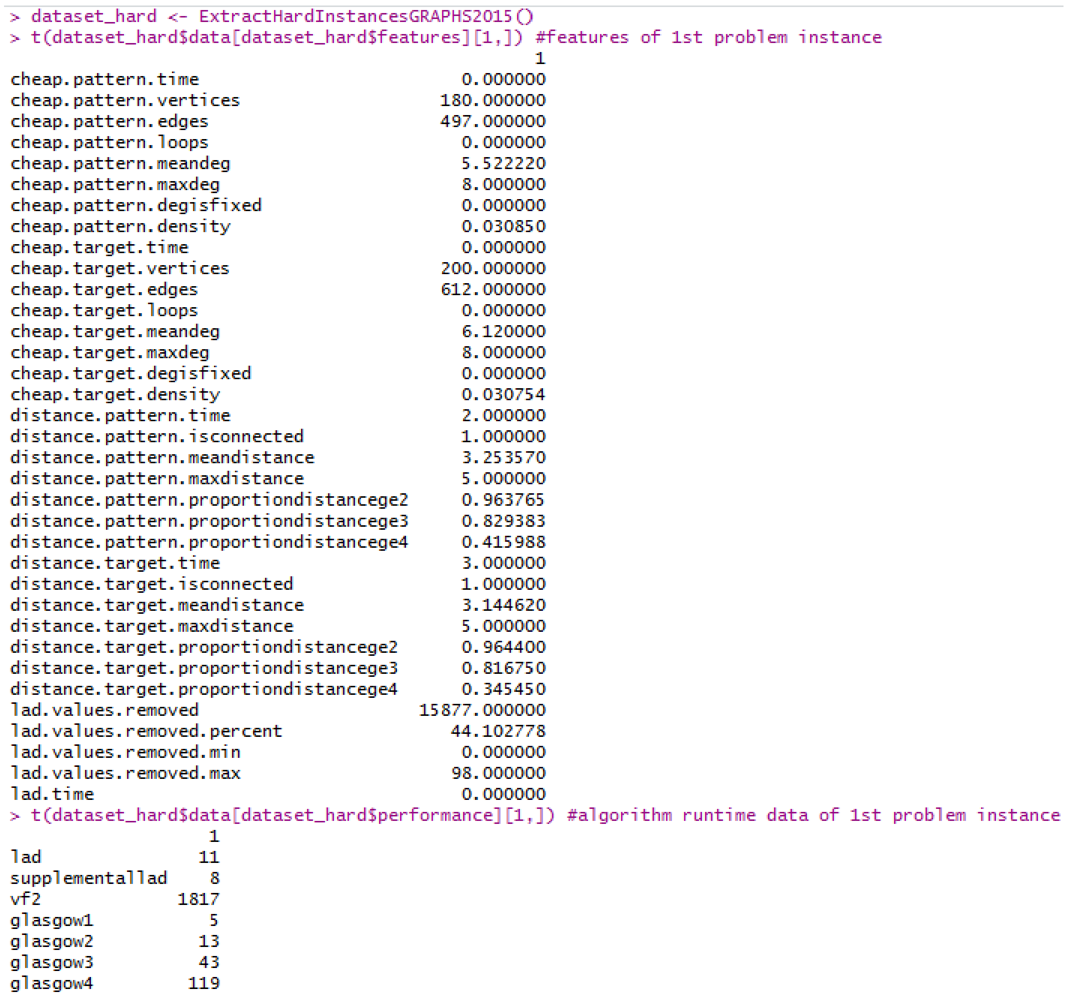
Figure 4.4 illustrates how an AS model is trained using *REINFORCE* algorithm.



Start
Problem dataset 2336 x 42
Problem instance 35 x 1
Algorithm scores 7 x 1
Algorithm probabilities 7 x 1
Load GRAPHS-2015 and normalize data
P
Get next problem instance i
S_i
Feed problem features to neural network
H(S_i,θ)
Apply Softmax policy
π (A|S_i,θ)
Select algorithm
Probability of selected algorithm 1 x 1
π (A|S_i,θ)
A
Selected algorithm 1 x 1
yes
End
no
Training data left?
θ
Update neural network weights
∇J(θ)
Calculate performance gradient
G
Get algorithm runtime
Neural network weights |θ|
Performance gradient |θ|
Algorithm runtime 1 x 1

Figure 4.4: Implementation of AS model trained using REINFORCE

1. The GRAPHS-2015 dataset is loaded into R workspace. Hard problem instances are extracted, which consists of 2336 problem instances with 42 attributes used for training: 35 problem features plus runtime data of 7 algorithms.



```
> dataset_hard <- ExtractHardInstancesGRAPHS2015()
> t(dataset_hard$data[dataset_hard$features][1,]) #features of 1st problem instance
                                                  1
cheap.pattern.time                         0.000000
cheap.pattern.vertices                   180.000000
cheap.pattern.edges                      497.000000
cheap.pattern.loops                        0.000000
cheap.pattern.meandeg                      5.522220
cheap.pattern.maxdeg                       8.000000
cheap.pattern.degisfixed                   0.000000
cheap.pattern.density                      0.030850
cheap.target.time                          0.000000
cheap.target.vertices                    200.000000
cheap.target.edges                       612.000000
cheap.target.loops                         0.000000
cheap.target.meandeg                       6.120000
cheap.target.maxdeg                        8.000000
cheap.target.degisfixed                    0.000000
cheap.target.density                       0.030754
distance.pattern.time                      2.000000
distance.pattern.isconnected               1.000000
distance.pattern.meandistance              3.253570
distance.pattern.maxdistance               5.000000
distance.pattern.proportiondistancege2     0.963765
distance.pattern.proportiondistancege3     0.829383
distance.pattern.proportiondistancege4     0.415988
distance.target.time                       3.000000
distance.target.isconnected                1.000000
distance.target.meandistance               3.144620
distance.target.maxdistance                5.000000
distance.target.proportiondistancege2      0.964400
distance.target.proportiondistancege3      0.816750
distance.target.proportiondistancege4      0.345450
lad.values.removed                     15877.000000
lad.values.removed.percent                44.102778
lad.values.removed.min                     0.000000
lad.values.removed.max                    98.000000
lad.time                                   0.000000
> t(dataset_hard$data[dataset_hard$performance][1,]) #algorithm runtime data of 1st problem instance
                  1
lad              11
supplementallad   8
vf2            1817
glasgow1          5
glasgow2         13
glasgow3         43
glasgow4        119
```

Figure 4.5: Sample problem instance from GRAPHS-2015 dataset

2. Problem features are normalized using *Z-score*. Normalization standardizes the range of data inputs which helps in resolving numerical issues when gradients are calculated during backpropagation. In effect, this speeds up neural network learning.

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j} \tag{4.1}$$

where:

$z_{ij} =$	normalized feature value
$x_{ij} =$	feature value
$\mu_j =$	mean of *jth* feature across **P** problem instances
$\sigma_j =$	standard deviation of *jth* feature across **P** problem instances
$i =$	*ith* instance from **P** problem instances
$j =$	*jth* feature out of 35 features

The algorithm runtime data is rescaled to simplify the reward value used for computing the performance gradient. This enables the AS model to learn the most important information during problem solving, which is the ranking of algorithm performance from best to worst. This information is more important than the knowledge of actual algorithm runtimes, as the objective of AS is mainly not to characterize individual algorithm performance, but to determine which algorithms work best on problems. Table 4.2 shows how this study mapped algorithm ranking (determined by sorting runtime from shortest to longest) to reward values.

Algorithm rank	**Assigned Reward Value**
1	16
2	4
3	1
4	-4
5	-16
6	-64
7	-256

Table 4.2: Algorithm ranking reward values

3. A neural network is trained to map problem features (input) to algorithm scores (output). It is composed of an input layer with 35 inputs, 3 hidden layers, and an output layer with 7 outputs. Algorithm scores represent selection preferences, i.e., the larger the score, the more often that algorithm is taken.

4. The algorithm scores are converted into probabilities using the softmax policy function. The probability assigned to an algorithm is proportional to its score, i.e., algorithms with the highest scores are given the highest probabilities of being selected. The softmax equation is given in the Equation below:

$$\pi(A|S_i, \theta) = \frac{\exp H_A(S_i, \theta)}{\sum_{b=1}^{7} \exp H_b(S_i, \theta)} \tag{4.2}$$

where:

$\pi(A\|S_i, \theta) =$	Probability of algorithm A
$H_A(S_i, \theta) =$	Algorithm score of algorithm A
$H_B(S_i, \theta) =$	Algorithm score of algorithm B
$S_i =$	features of problem instance *i*
$\theta =$	neural network weights

5. An algorithm is randomly selected based from the probabilities generated from the softmax policy function.

6. The rescaled reward value corresponding to the selected algorithm is read from the dataset. This value, along with the probability of the selected algorithm, is used to calculate the performance gradient:

$$\nabla J(\theta_t) = G_t \ln \pi(A|S_i, \theta_t) \tag{4.3}$$

where:

$\nabla J(\theta_t) =$	performance gradient
$G_t =$	reward
$\pi(A\|S_i, \theta_t) =$	Probability of selected algorithm

7. The performance gradient is used for backpropagation, a standard method of updating the weights of a neural network. Through this process, the neural

network learns to adjust the algorithm scores in proportion to the reward value associated with the selected algorithm.

$$\theta_{t+1} \leftarrow \theta_t + \alpha \nabla J(\theta_t) \tag{4.4}$$

where:

$\theta_{t+1} =$	Updated neural network weights
$\theta_t =$	current neural network weights
$\alpha =$	learning rate
$\nabla J(\theta_t) =$	performance gradient

4.5 Model Evaluation

The performance of PRFR and REINFORCE are evaluated using 10-fold cross validation. The 2336 hard instances from GRAPHS-2015 are randomly partitioned into 10 subsets of approximately equal size. Of the 10 subsets, 9 are combined to form the training set for the AS models, while the remaining subset is formed as a test set. Mean MCP (Equation 3.1) is calculated on the test set to evaluate the AS model performance. This is repeated 10 times for all possible combinations of training and test sets. At the end of this process, the final mean MCP is calculated by averaging the mean MCPs from all test sets.

4.6 R source code implementation

The implementation of AS models in this study can be retrieved from here: `https://github.com/kvrigor/algosel-rl`

CHAPTER 5

EXPERIMENT RESULTS

5.1 Introduction

This chapter presents the results from performing the experiments. The characteristics of the dataset are summarized on the exploratory data analysis section. The last section compares PRFR and REINFORCE in terms of their effectiveness in training an AS model for solving subgraph isomorphism problems.

5.2 Exploratory Data Analysis

5.2.1 Data Summary

The following statistical measures were taken for the results of each feature and algorithm:

- **min** - minimum
- **qu_-1st** - 25% quantile
- **med** - median
- **mean** - arithmetic mean
- **qu_3rd** - 75%-quantile
- **max** - maximum
- **sd** - standard deviation

- **coeff_var** - coefficient of variation (standard deviation / arithmetic mean)

Features	min	qu_1st	med	mean	qu_3rd	max	sd	coeff_var
cheap.pattern.time	0.0000	0.0000	0.0000	0.0035	0.0000	1.0000	0.0590	16.8908
cheap.pattern.vertices	4.0000	48.0000	80.0000	109.3787	128.0000	900.0000	114.9942	1.0513
cheap.pattern.edges	4.0000	112.0000	240.0000	598.8704	696.0000	12410.0000	1073.5139	1.7926
cheap.pattern.loops	0.0000	0.0000	0.0000	0.2451	0.0000	9.0000	1.3121	5.3543
cheap.pattern.meandeg	1.7867	3.6364	5.4286	10.4274	10.0000	99.0000	14.4378	1.3846
cheap.pattern.maxdeg	2.0000	8.0000	11.0000	20.6091	26.0000	269.0000	27.3313	1.3262
cheap.pattern.degisfixed	0.0000	0.0000	0.0000	0.1132	0.0000	1.0000	0.3168	2.7993
cheap.pattern.density	0.0045	0.0441	0.0978	0.1683	0.1830	1.0000	0.2194	1.3038
cheap.target.time	0.0000	0.0000	0.0000	3.9988	2.0000	55.0000	9.5821	2.3963
cheap.target.vertices	10.0000	216.0000	561.0000	1147.3385	1430.0000	6671.0000	1440.1846	1.2552
cheap.target.edges	15.0000	930.0000	2074.0000	7176.6114	5994.0000	209000.0000	21594.1676	3.0090
cheap.target.loops	0.0000	0.0000	0.0000	1.6306	0.0000	144.0000	13.3146	8.1656
cheap.target.meandeg	0.4357	4.1983	6.0000	19.4312	11.9172	270.6950	39.8146	2.0490
cheap.target.maxdeg	2.0000	10.0000	19.0000	102.2134	52.0000	4973.0000	384.0174	3.7570
cheap.target.degisfixed	0.0000	0.0000	0.0000	0.1492	0.0000	1.0000	0.3563	2.3885
cheap.target.density	0.0005	0.0030	0.0119	0.0729	0.0588	1.0000	0.1585	2.1736
distance.pattern.time	0.0000	0.0000	0.0000	2.0349	1.0000	93.0000	8.0467	3.9543
distance.pattern.isconnected	0.0000	1.0000	1.0000	0.7976	1.0000	1.0000	0.4019	0.5039
distance.pattern.meandistance	1.0000	2.0010	2.7778	3.9349	4.1227	80.6687	5.6418	1.4338
distance.pattern.maxdistance	1.0000	3.0000	5.0000	8.9328	9.0000	240.0000	16.9947	1.9025
distance.pattern.proportiondistancege2	0.0000	0.6000	0.8466	0.7406	0.9319	0.9929	0.2661	0.3593
distance.pattern.proportiondistancege3	0.0000	0.0974	0.5000	0.4605	0.8237	0.9834	0.3507	0.7615
distance.pattern.proportiondistancege4	0.0000	0.0000	0.1957	0.3091	0.6094	0.9752	0.3225	1.0432
distance.target.time	0.0000	5.0000	32.0000	553.0529	208.0000	9962.0000	1474.8373	2.6667
distance.target.isconnected	0.0000	0.0000	1.0000	0.7095	1.0000	1.0000	0.4540	0.6399
distance.target.meandistance	1.0000	2.4484	3.7010	7.4919	7.1508	100.6250	11.7237	1.5648
distance.target.maxdistance	1.0000	5.0000	8.0000	16.3768	16.0000	200.0000	24.8081	1.5148
distance.target.proportiondistancege2	0.0000	0.6082	0.9502	0.7944	0.9900	0.9993	0.2896	0.3645
distance.target.proportiondistancege3	0.0000	0.2472	0.7260	0.5990	0.9502	0.9985	0.3708	0.6190
distance.target.proportiondistancege4	0.0000	0.0288	0.3875	0.4514	0.8813	0.9973	0.3838	0.8502
lad.values.removed	0.0000	0.0000	0.0000	28733.8725	10673.0000	844304.0000	81827.2598	2.8478
lad.values.removed.percent	0.0000	2.3503	49.2371	51.7472	100.0000	100.0000	41.8261	0.8083
lad.values.removed.min	0.0000	0.0000	0.0000	6.7000	0.3190	99.9700	19.4023	2.8959
lad.values.removed.max	0.0000	0.0000	0.0000	33.1602	88.2143	99.9816	42.8695	1.2928
lad.time	0.0000	0.0000	0.0000	7.1210	3.0000	3964.0000	80.7295	11.3367

Table 5.1: Summary of features

Algorithm	min	qu_1st	med	mean	qu_3rd	max	sd	coeff_var
lad	0	3	33	5935035.15	1214	1.00E+08	22874350.97	3.8541
supplementallad	0	3	22	4841120.76	402	1.00E+08	20810140.52	4.2986
vf2	0	6	115	27565110.75	1.00E+08	1.00E+08	44243664.13	1.6051
glasgow1	0	4	25	4158094.27	199	1.00E+08	19437830.22	4.6747
glasgow2	0	12	61	3196878.81	414	1.00E+08	17126177.54	5.3572
glasgow3	0	33	168	3212027.69	1072	1.00E+08	17153623.91	5.3404
glasgow4	0	113	478	4333647.25	7178	1.00E+08	19233485.20	4.4382

Table 5.2: Summary of algorithms. All values represent runtime in milliseconds.

5.2.2 Distributions

Histograms for features and algorithm runtimes are displayed in the following plots. The red dashed lines denote the median of the distribution.

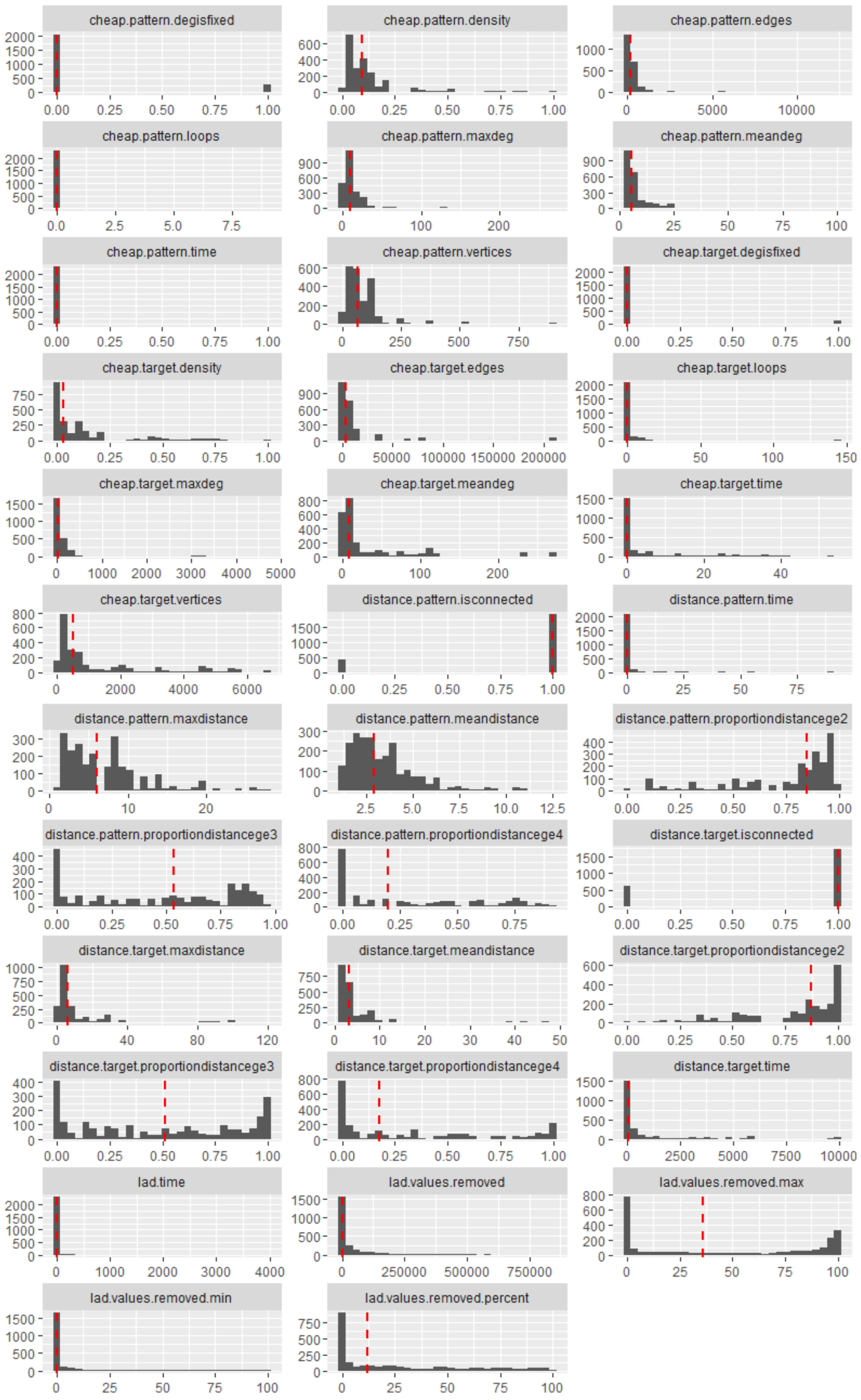


Figure 5.1: Feature distributions

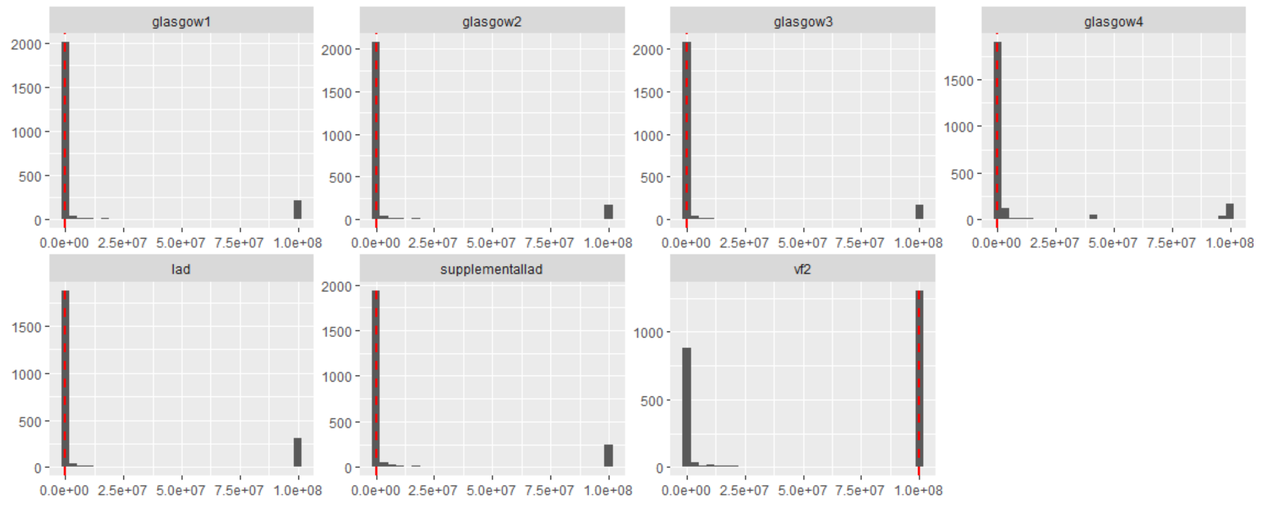


Figure 5.2: Algorithm distributions

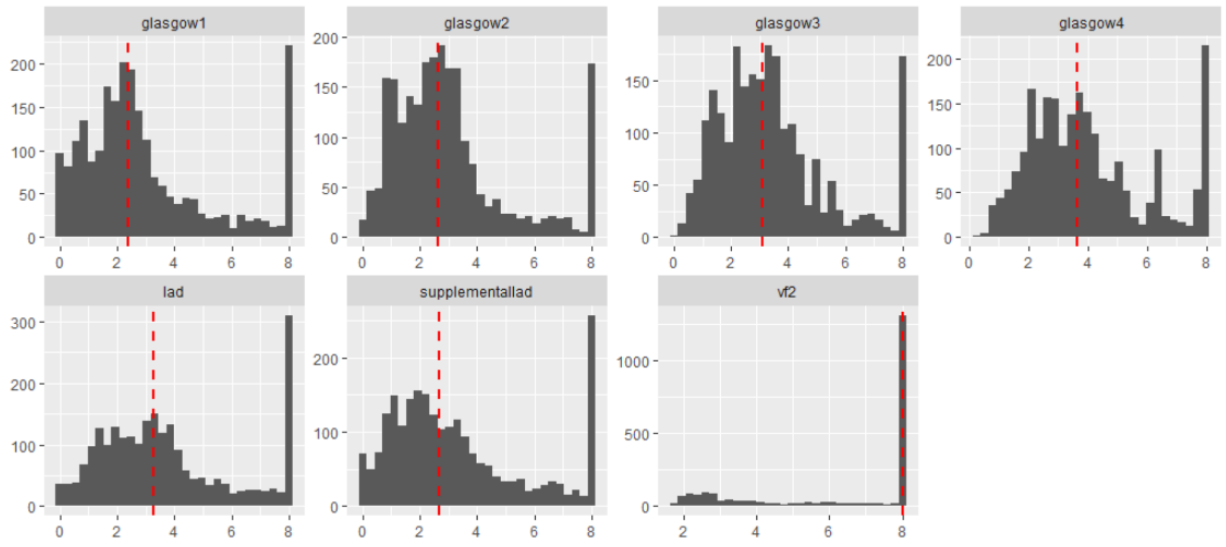


Figure 5.3: Log-scaled algorithm distributions

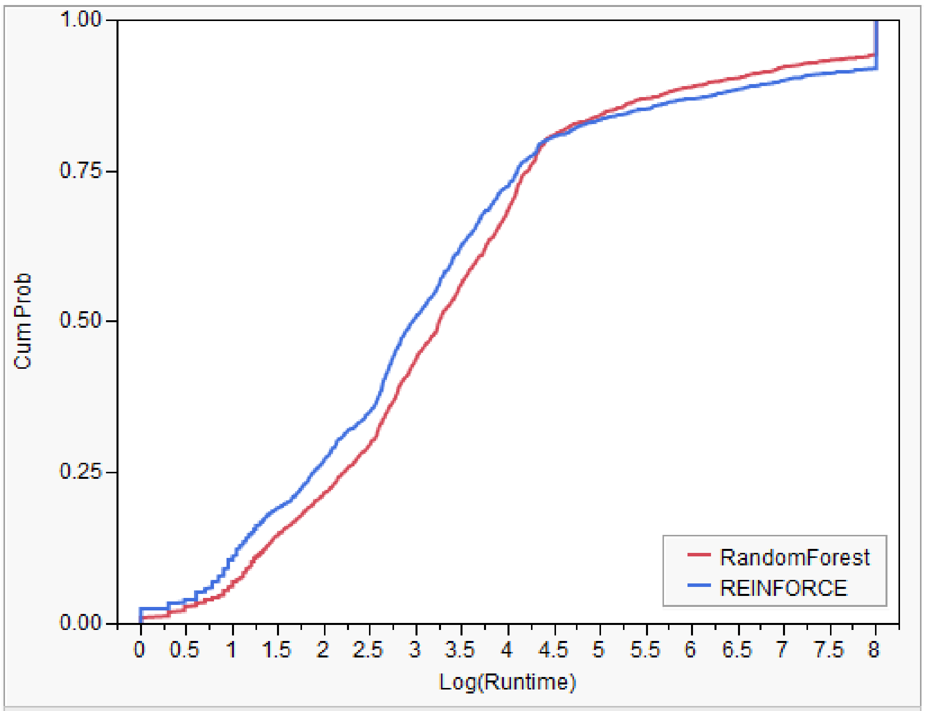
5.3 AS Model Performance Evaluation

The mean MCP of the AS model trained using REINFORCE is 4x worse than one trained using PRFR. REINFORCE is not able to surpass the performance standard set by PORTSUB, which was hoped to be surpassed in this study.

AS Model	**Mean MCP**	**# Solved problems**	**Mean Performance**	**Median Performance**
VBS	0.0	2,219	5,822,809.0	79.0
PRFR	621,662.5	2,208	6,446,129.0	1,748.5
SBS	1,963,498.6	2,173	7,786,308.0	539.0
REINFORCE	2,318,832.0	2,156	8,168,306.0	929.0

Table 5.3: AS model performance evaluation results

Despite REINFORCE failing to obtain better MCP than PRFR, it showed improvement on median performance, reporting almost 2x better than PRFR. Its significance becomes clearer by analyzing the cumulative density function (CDF) plot which visualizes the distribution of runtimes of the selected algorithms on all problems, as illustrated Figure 5.4. REINFORCE is able to select more optimal algorithms than PRFR on the easier problems (solvable within $10 - 10^4$ ms) on the dataset. For difficult problems which are solvable within at least 105-108 ms, REINFORCE selected worse-performing algorithms than PRFR. Although the easier problems constitute the majority (75%) of the dataset, its influence on the mean MCP is much lesser than the harder problems. REINFORCE performed better than PRFR in solving the easy problems, but underperformed when it came to solving the more difficult problems.



Quantiles

Level	Minimum	10%	25%	Median	75%	90%	Maximum
RandomForest	0	1.20412	2.191724	3.238924	4.201233	6.308288	8
REINFORCE	0	0.954243	1.863323	2.948412	4.095492	6.963527	8

Figure 5.4: Cumulative density plot of AS model performance

CHAPTER 6

CONCLUSION

6.1 Summary

Given problem features and algorithm performance, an AS model can learn how to map problems to algorithms through supervised learning techniques, such as classification or regression algorithms. This makes sense because AS involves modelling the relationship between two variables: problem features and algorithm performance. However, it is also possible to consider an AS model as a decision-making entity: it *senses* the problem features, *decides* which algorithm to choose, *observes* the performance of the chosen algorithm, and then *updates* its decision-making behavior based on the observed response. This portrays a type of learning based on *feedback* – the core concept behind RL. Its primary advantage is that learning is not dependent on right-wrong examples explicitly labeled by a ”teacher” as with supervised learning techniques; thus enabling an AS model to learn on its own based only on its observation of how algorithms behave with respect to problem features. It is more difficult to control the behavior of RL algorithms compared to supervised learning techniques, but when successful, an RL-trained AS model has far better learning generalization capability than one trained using supervised learning. Better generalization means better AS performance with previously unseen problems despite learning only from a small batch of problems.

Two AS models were implemented in this study: one trained using PRFR, a supervised learning technique used on a previous AS study by Kotthoff *et al.* (2016), and the other trained using REINFORCE, the RL algorithm proposed in this study. The AS models were compared using mean misclassification penalty (MCP), a performance metric which reflects the additional time required in solving problems where suboptimal algorithms were selected. A mean MCP of 0 indicates that there is zero penalty in choosing a suboptimal algorithm, signifying that the best algorithm is always selected for every problem. PRFR garnered an MCP of 6.22×10^5 milliseconds while REINFORCE measured 4x as worse, with mean MCP of 2.32×10^6 milliseconds. On the other hand, median performance results show that REINFORCE did better than PRFR on the easier problems on the dataset, representing 75% of the total. Still, PRFR outperformed REINFORCE because it specialized in solving the much harder problems (problems solvable within $10^5 - 10^8$ ms range) which have a huge impact on MCP computation, despite representing a minority on the dataset.

6.2 Recommendations for Future Work

A number of modifications on the REINFORCE algorithm can be explored:

- Reduce number of input features

  Only a handful of features might be necessary to perform effective AS. Dimensionality reduction methods (e.g. PCA, clustering) can help to reduce and simplify the problem feature space. This can assist a function approximator in focusing on few important features to learn algorithm scores.

- Redesign policy function and function approximator

  The behavior of the policy function depends on how the function approximator calibrates the algorithm scores for each problem instance. In turn, the function approximator learns the proper calibrations when the policy allows for balanced *exploration* and *exploitation* during AS: *explore* seldomly tried algorithms to learn their performance and *exploit* existing knowledge on algorithm performance to select the best algorithm.

  It is quite difficult to know which function approximation technique goes best with a given policy function (and vice versa). This can be determined through cycles of reviewing the literature to get ideas which methods might potentially work and heavy experimentation.

- Reward scaling

  How the algorithm runtime is scaled prior to performance gradient calculation has a huge impact on the resulting performance of the AS model. Unscaled runtime values can serve enough as feedback; however, learning becomes more erratic especially when there are plenty of outliers across runtime observations. Reward scaling helps to stabilize learning. The optimum scaling method can be empirically determined, taking into consideration the characteristics of runtime data distribution and the function approximation method used.

- Add regularization term to performance gradient

  A regularization term acts as an offset value to the performance gradient which can influence how the policy function conducts *exploration* and *exploitation* among available actions. Gradient regularization methods such as entropy regularization (Williams, 1992) and importance sampling (Nachum et. al., 2016) attempt to improve exploration, which helps the policy to avoid being stuck with known actions and incentivize behavior towards discovery of potentially better actions.

- Implement curriculum learning

  A training strategy can be implemented such that problem instances are presented to the AS model in some meaningful order instead of being random. Such strategy is called *curriculum learning* (Bengio *et al.*, 2009). This recommendation is based from the experiment results where REINFORCE performed well only on the easier problems but not on the harder ones. It might be possible to improve AS model performance by focusing the training on much harder problems.

Besides REINFORCE, there are plenty of other policy-gradient RL algorithms that can possibly be applied to AS. Another class of RL algorithms called *contextual bandits* can also be explored. Contextual bandits assume a simpler model of RL where reward values are fully received for every input instead of being delayed across a series of inputs. This model can also apply to AS since algorithm performance is based only from the algorithm currently selected and not on past nor future algorithm choices.

REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. *et al.* (2016). Tensorflow: a system for large-scale machine learning. In *OSDI*, vol. 16. 265–283.

Battiti, R. and Mascia, F. (2007). An algorithm portfolio for the sub-graph isomorphism problem. In *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics*. (pp. 106–120). Springer.

Bengio, Y., Louradour, J., Collobert, R. and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*. ACM, 41–48.

Bischl, B., Kerschke, P., Kotthoff, L., Lindauer, M., Malitsky, Y., Fréchette, A., Hoos, H., Hutter, F., Leyton-Brown, K., Tierney, K. *et al.* (2016). Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*. 237, 41–58.

Bonyadi, M. R., Michalewicz, Z. and Barone, L. (2013). The travelling thief problem: The first step in the transition from theoretical problems to realistic problems. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*. IEEE, 1037–1044.

Corne, D. W. and Knowles, J. D. (2003). No free lunch and free leftovers theorems for multiobjective optimisation problems. In *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 327–341.

Gagliolo, M. and Schmidhuber, J. (2009). Towards distributed algorithm portfolios. In *International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)*. Springer, 634–643.

Gent, I. P., Jefferson, C., Kotthoff, L., Miguel, I., Moore, N. C., Nightingale, P. and Petrie, K. E. (2010). Learning When to Use Lazy Learning in Constraint Solving. In *ECAI*. Citeseer, 873–878.

Gomes, C. P. and Selman, B. (2001). Algorithm portfolios. *Artificial Intelligence*. 126(1-2), 43–62.

Goutte, C. (1997). Note on free lunches and cross-validation. *Neural Computation*. 9(6), 1245–1249.

Huberman, B. A., Lukose, R. M. and Hogg, T. (1997). An economics approach to hard computational problems. *Science*. 275(5296), 51–54.

Kellerer, H., Pferschy, U. and Pisinger, D. (2004). Introduction to NP-Completeness of knapsack problems. In *Knapsack problems*. (pp. 483–493). Springer.

KhudaBukhsh, A. R., Xu, L., Hoos, H. H. and Leyton-Brown, K. (2009). *SATenstein: Automatically building local search SAT solvers from components*. Ph.D. Thesis. University of British Columbia.

Kotthoff, L. (2012). Hybrid Regression-Classification Models for Algorithm Selection. In *ECAI*. 480–485.

Kotthoff, L. (2013). LLAMA: leveraging learning to automatically manage algorithms. *arXiv preprint arXiv:1306.1031*.

Kotthoff, L. (2016a). Algorithm selection for combinatorial search problems: A survey. In *Data Mining and Constraint Programming*. (pp. 149–190). Springer.

Kotthoff, L. (2016b). *GRAPHS-2015*. Retrievable at `http://github.com/coseal/aslib_data/tree/master/GRAPHS-2015`.

Kotthoff, L., Kerschke, P., Hoos, H. and Trautmann, H. (2015). Improving the state of the art in inexact TSP solving using per-instance algorithm selection. In *International Conference on Learning and Intelligent Optimization*. Springer, 202–217.

Kotthoff, L., McCreesh, C. and Solnon, C. (2016). Portfolios of subgraph isomorphism algorithms. In *International Conference on Learning and Intelligent Optimization*. Springer, 107–122.

Kotthoff, L., Miguel, I. and Nightingale, P. (2010). Ensemble classification for constraint solver configuration. In *International Conference on Principles and Practice of Constraint Programming*. Springer, 321–329.

Lagoudakis, M. G. and Littman, M. L. (2000). Algorithm Selection using Reinforcement Learning. In *ICML*. Citeseer, 511–518.

Lestringant, P., Guihéry, F. and Fouque, P.-A. (2015). Automated identification of cryptographic primitives in binary code with data flow graph isomorphism. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*. ACM, 203–214.

Lindauer, M. (2016). *Algorithm Selection Library format specification*. Retrievable at `https://github.com/coseal/aslib-spec/blob/master/format.md`.

Lindauer, M., van Rijn, J. N. and Kotthoff, L. (2017). Open Algorithm Selection Challenge 2017: Setup and Scenarios. In *Open Algorithm Selection Challenge 2017*. 1–7.

Rice, J. R. (1976). The Algorithm Selection Problem. In *Advances in Computers*. (pp. 65–118). vol. 15. Elsevier.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. *et al.* (2016). Mastering the game of Go with deep neural networks and tree search. *nature*. 529(7587), 484.

Silverthorn, B. and Miikkulainen, R. (2010). Latent Class Models for Algorithm Portfolio Methods. In *AAAI*. 167–172.

Smith, A., Veneris, A., Ali, M. F. and Viglas, A. (2005). Fault diagnosis and logic debugging using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. 24(10), 1606–1621.

Smith-Miles, K. A. (2009). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*. 41(1), 6.

Stolk, J., Mann, I., Mohais, A. and Michalewicz, Z. (2013). Combining vehicle routing and packing for optimal delivery schedules of water tanks. *OR Insight*. 26(3), 167–190.

Sutton, R. S., Barto, A. G. *et al.* (1998). *Reinforcement learning: An introduction*. MIT press.

Wagner, M., Lindauer, M., Mısır, M., Nallaperuma, S. and Hutter, F. (2018). A case study of algorithm selection for the traveling thief problem. *Journal of Heuristics*. 24(3), 295–320.

Wickham, H. (2015). *R packages: organize, test, document, and share your code*. " O'Reilly Media, Inc.".

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*. 8(3-4), 229–256.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*. 1(1), 67–82.

Wolpert, D. H., Macready, W. G. *et al.* (1995). *No free lunch theorems for search*. Technical report. Technical Report SFI-TR-95-02-010, Santa Fe Institute.

Xu, H., Caramanis, C. and Mannor, S. (2012). Sparse algorithms are not stable: A no-free-lunch theorem. *IEEE transactions on pattern analysis and machine intelligence*. 34(1), 187–193.

Xu, L., Hoos, H. and Leyton-Brown, K. (2010). Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection. In *AAAI*, vol. 10. 210–216.

Xu, L., Hutter, F., Hoos, H. H. and Leyton-Brown, K. (2008). SATzilla: portfolio-based algorithm selection for SAT. *Journal of artificial intelligence research*. 32, 565–606.

Zoph, B. and Le, Q. V. (2016). Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.