

Evolutionary Search Strategies in Constraint Programming

by

Robert Andrew Bennetto



Dissertation presented for the degree of
Doctor of Philosophy in Industrial Engineering
in the Faculty of Engineering at Stellenbosch University

Promoter: JH van Vuuren

March 2020

Declaration

By submitting this dissertation electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: March 1, 2020

Abstract

Successful approaches towards solving complex combinatorial optimisation problems straddle a boundary between art and science. Discrete search spaces are typically notoriously large, contain high-dimensional dependencies and usually remain intractable for large instances. Generalised mathematical relaxations that allow discrete problems to be solved in a near-trivial manner are not known. As such, researchers have focussed on approximate solution techniques, heuristics and metaheuristics for solving classes of such complex problems. Many of these techniques have forgone mathematical guarantees in favour of bio-inspired mechanisms which can produce good solutions.

Constraint Programming (CP) is a discrete optimisation technique which, historically, has not received as much attention as its linear and mixed integer counterparts in the literature. CP has, however, gained popularity in recent years due to its direct modelling approach, its ability to solve a large range of medium-sized optimisation problems with complex constraints, and the maturity of the modelling language.

It is advocated in this dissertation that a best-of-breed approach to combinatorial optimisation be adopted which leverages the modelling flexibility of CP and evolutionary processes for search-policy generation as a function of the underlying abstract problem representation. Research into the algorithm selection problem which simultaneously solves the problem of generating new algorithms while allowing for selecting from an existing corpus of algorithms is limited.

A literature review of the relevant solution components is provided. An incremental approach to testing these components is employed where the merits of the various algorithmic components are demonstrated, highlighting novelties in the methodology applied. Commentary on the computational performance and quality of results achieved is also provided. It is concluded that evolutionary algorithms serve as a suitable metaheuristic paradigm, capable of finding high-quality branching schemes for resolving constraint satisfaction problems more effectively than those of a benchmark algorithm. A single-objective and a multi-objective optimisation approach are tested, and it is found that the multi-objective approach facilitates the formation of higher-quality strategies during the evolutionary search process.

It is further demonstrated that the search strategies uncovered not only extend their performance characteristics to unseen problem instances of the same class, but also that a deep learning model may be employed as a predictor of performance when selecting a strategy for an unseen problem instance. A commentary is provided as to likely areas of potential future work as a result of the methodology and findings of this dissertation.

Opsomming

Suksesvolle benaderings tot die oplossing van komplekse kombinatoriese optimeringsprobleme oorspan 'n grens tussen kuns en wetenskap. Diskrete soekruimtes is tipies problematies groot, bevat hoë-dimensionele afhanklikhede en groot gevalle bly normaalweg onoplosbaar. Veralgemene wiskundige verslappings waarvolgens diskrete probleme byna triviaal oplosbaar word, is nie bekend nie. As sulks, het navorsers gefokus op benaderde oplossingstegnieke, heuristieke en meta-heuristieke om klasse van sulke ingewikkelde probleme mee op te los. Baie van hierdie tegnieke het wiskundige waarborge ten gunste van bio-geïnspireerde meganismes wat goeie oplossings kan lewer, versaaak.

Beperkingsprogrammering (BP) is 'n diskrete optimeringstegniek wat histories nie soveel aandag soos lineêre en gemengde heelgetalige programmering, in die literatuur geniet het nie. BP het egter die afgelope paar jaar gewildheid verwerf vanweë die direkte modelleringsbenadering daarvan, die vermoë om 'n groot verskeidenheid medium-grootte optimeringsprobleme met ingewikkelde beperkings op te los, en die volwassenheid van die gepaardgaande modelleringstaal.

In hierdie proefskrif word daar voorgestel dat 'n beste-van-telingsbenadering tot kombinatoriese optimering in gebruik geneem word wat die modelleringsvryheid van BP en evolusionêre prosesse vir soekbeleidgenerering as 'n funksie van die onderliggende abstrakte probleemvoorstelling benut. Navorsing oor die algoritme-seleksieprobleem wat tegelykertyd die probleem van die ontwerp van nuwe algoritmes oplos en ook 'n keuse uit 'n bestaande groep algoritmes toelaat, is beperk.

'n Literatuuroorsig van die toepaslike oplossingskomponente word aangebied. Daar word gebruik gemaak van 'n inkrementele benadering tot die toetsing van hierdie komponente waartydens die meriete van die verskillende algoritmiese komponente gedemonstreer word, wat nuwe elemente in die toegepaste metodologie beklemtoon. Kommentaar oor die berekeningsprestasie en die kwaliteit van die resultate bereik, word ook gelewer. Daar word tot die gevolgtrekking gekom dat evolusionêre algoritmes as 'n geskikte meta-heuristiese paradigma dien en daartoe in staat is om hoë-kwaliteit vertakkingskemas vir die oplossing van beperkingvoldoeningsprobleme te vind wat meer doeltreffend is as dié van 'n maatstafalgoritme. 'n Enkeldoelige en 'n meerdoelige optimeringsbenadering word getoets, en daar word bevind dat die meerdoelige benadering die vorming van strategieë van hoër gehalte tydens die evolusionêre soekproses fasiliteer.

Daar word verder gedemonstreer dat die ontdekte soekstrategieë nie net hul toepaslikheid uitbrei na ongekende probleemgevallen van dieselfde klas nie, maar ook dat 'n diepleermodel, as 'n voorspeller van prestasie by die keuse van 'n strategie vir 'n ongesiene probleemgeval gebruik kan word. Kommentaar word gelewer oor waarskynlike gebiede van moontlike toekomstige werk wat uit die metodologie en bevindings in hierdie proefskrif voortvloei.

Acknowledgements

The author wishes to acknowledge the following people and institutions for their various contributions towards the completion of this work:

- Professor Jan H van Vuuren, my supervisor, for providing me the opportunity to participate in the Stellenbosch Unit for Operations Research and Engineering. It has been a privilege to observe and immerse myself (on occasion) in the culture you have cultivated in the unit. I've thoroughly enjoyed the conversations shared over the years and appreciate the way in which you've influenced this dissertation. I aspire to apply myself as thoroughly as you do.
- David Clark, for always being willing to take a walk and soundboard both conceptual and technical challenges.
- My employers, past and present, for their willingness support further study.
- My colleagues, past and present, for nurturing my desire to find an optimal incumbent.
- My mother and father, for encouraging me to study further.
- My wife, Jessica, who has sacrificed the most valuable commodity, time. I would have been unable to explore my curiosities in discrete optimisation without her support and I am truly fortunate to share my life with her.

Table of Contents

Abstract	iii
Opsomming	v
Acknowledgements	vii
List of Acronyms	xiii
List of Figures	xvii
List of Tables	xix
List of Algorithms	xxi
1 Introduction	1
1.1 Background	1
1.2 Problem description	2
1.3 Scope and objectives	3
1.4 Document structure	4
2 Literature Review	5
2.1 Mixed integer and constraint programming	6
2.2 Constraint propagation	10
2.3 Search algorithms	12
2.3.1 Backtracking search	12
2.4 Restart strategies	14
2.5 Variable and value ordering heuristics	15
2.6 CP grammar	17
2.7 Evolutionary programming	18
2.7.1 The genetic algorithm	19

2.7.2	Genetic programming representation	21
2.7.3	Fitness	22
2.7.4	Seeding operators	23
2.7.5	Selection operators	24
2.7.6	Crossover operators	24
2.7.7	Mutation operators	25
2.7.8	Memetic algorithms	26
2.7.9	Multi-objective genetic algorithms	27
2.8	Automatic algorithm selection	31
2.8.1	Evolutionary approaches to the ASP	34
2.9	Static CSP feature extraction	35
2.10	Deep learning	39
2.11	Cooperative applications	41
3	Research Hypotheses	43
3.1	Research motivation	43
3.1.1	Hypothesis 1: Evolutionary CSP augmentation	47
3.1.2	Hypothesis 2: Evolutionary meta-CSP augmentation	48
3.1.3	Hypothesis 3: Deep learning for algorithm selection	49
3.2	Hypothesis testing considerations	50
3.3	Test data	52
3.4	Benchmark strategy	55
3.5	Implementation notes	56
4	Evolutionary Search Strategies	63
4.1	The ranking function	66
4.2	GP objective functions	67
4.3	GP configuration parameters	70
4.4	MiniZinc Challenge results (2015)	71
4.5	GP run results (Hypothesis 3.1.1)	77
4.5.1	Classic CSP GP operators	77
4.5.2	Extendibility to unseen problem instances	86
4.5.3	Hypothesis 3.1.1 conclusions	89
4.5.4	Graph meta-data CSP GP operators	91
4.6	GP run results (Hypothesis 3.1.2)	92
4.6.1	Extendibility to unseen problem instances (Hypothesis 3.1.2)	100

4.6.2	Strategy comparison	101
4.6.3	Hypothesis 3.1.2 conclusions	106
5	Strategy Selection	109
5.1	Data preparation and representation	110
5.1.1	Representation	110
5.1.2	Data preparation	112
5.2	Linear regression for strategy selection	114
5.3	Deep learning for strategy selection	115
5.4	Portfolio selection	121
5.4.1	Extendibility to unseen problem instances	127
5.5	Hypothesis 3.1.3 conclusions	130
6	Conclusion	133
6.1	Dissertation summary	133
6.2	Appraisal of dissertation contributions	135
6.3	Suggestions for future work	136
6.3.1	Problem-specific GP grammars	136
6.3.2	Alternate levels of branching strategy	136
6.3.3	Constraint-specific GP grammars	136
6.3.4	Faceting applications	137
6.3.5	Evolutionary seeding	137
6.3.6	Enhancements to the deep learning model	138
6.3.7	The addition of CP solvers to the portfolio	139
6.4	A philosophical note	139
	References	141

List of Acronyms

- AC:** Arc Consistency
- ACE:** Adaptive Constraint Engine
- ACO:** Ant Colony Optimisation
- ADF:** Automatically Defined Function
- AMPL:** A Mathematical Programming Language
- ASP:** Algorithm Selection Problem
- BP:** Backpropagation
- CBJ:** Conflict-directed backjumping
- CLASS:** Composite Heuristic Learning Algorithm for SAT Search
- CNN:** Convolutional Neural Network
- CP:** Constraint Programming
- CPO:** Constraint Programming Optimiser (Ilog/IBM product)
- CPU:** Central Processing Unit
- CSP:** Constraint Satisfaction Problem
- CVRP:** Capacitated Vehicle Routing Problem
- DBT:** Dynamic backjumping
- DFJ:** Dantzig-Fulkerson-Johnson (formulation)
- DL:** Deep Learning
- EA:** Evolutionary Algorithm
- EAX:** Edge assembly crossover
- ECJ:** Evolutionary Computation in Java
- FD:** Finite Domain
- FDS:** Failure-directed search
- GA:** Genetic Algorithm

- GAC:** Generalised Arc Consistency
- GP:** Genetic Programming
- GPU:** Graphic Processing Unit
- GUID:** Globally Unique Identifier
- IBM:** International Business Machines
- IBS:** Impact-based search
- IP:** Integer Programming
- JSON:** Javascript Object Notation
- JSSP:** Job Shop Scheduling Problem
- KNN:** *K*-Nearest Neighbour
- LB:** Limited backjumping
- LNS:** Local neighbourhood search
- LP:** Linear Programming
- LPs:** Linear Programs
- LSTM:** Long Short-Term Memory
- MA:** Memetic Algorithm
- MAC:** Maintaining Arc Consistency
- MCTS:** Monte Carlo Tree Search
- MILP:** Mixed Integer Linear Programming
- MIP:** Mixed Integer Programming
- MIQLP:** Mixed Integer Quadratic Linear Programming
- ML:** Machine Learning
- MOMS:** Maximum Occurrences of clauses of the Minimum Size
- MPS:** Mathematical Programming System
- MSE:** Mean Squared Error
- MTZ:** Miller-Tucker-Zemlin (formulation)
- NSGAII:** Non-dominated Sorting Genetic Algorithm II
- OR:** Operations Research (or Operational Research)
- ORT:** OR-Tools (Google product)
- PAES:** Pareto-archived Evolution Strategy
- RL:** Reinforcement Learning

- RNN:** Recurrent Neural Network
- SA:** Simulated Annealing
- SAT:** Boolean Satisfaction
- SAW:** Simple Additive Weighting
- SGD:** Stochastic Gradient Descent
- SPEA:** Strength-Pareto Evolutionary Algorithm
- SQL:** Structured Query Language
- TD-Learning:** Temporal Difference Learning
- TS:** Tabu Search
- TSP:** Travelling Salesman Problem
- VM:** Virtual Machine
- VRP:** Vehicle Routing Problem

List of Figures

2.1	High-level CP pure CSP process flow	9
2.2	The conventional genetic algorithm flowchart [111]	20
2.3	Example of solution representation for an individual in GP: $x^2 + 2x - 1$	22
2.4	Convex and non-convex multi-objective Pareto frontiers	28
2.5	CSP plots for small subset of MiniZinc benchmark instances	36
3.1	Research hypothesis prototype result using CPO	45
3.2	Research hypotheses and interactions	47
3.3	Solution components	60
4.1	The GP CP sequential ranking process	64
4.2	Percentage of search time spent in the ranking function grouped by problem class	73
4.3	Default ORT branching speed aggregated by problem class	76
4.4	GP run results for Hypothesis 3.1.1 using SAW	78
4.5	GP run results for Hypothesis 3.1.1 using NSGA-II	79
4.5	GP run results for Hypothesis 3.1.1 using NSGA-II (continued)	80
4.5	GP run results for Hypothesis 3.1.1 using NSGA-II (continued)	81
4.6	Structural differences between training and zephyrus test sets	90
4.7	GP run results for Hypothesis 3.1.2 using SAW	93
4.8	GP run results for Hypothesis 3.1.2 using NSGA-II	94
4.8	GP run results for Hypothesis 3.1.2 using NSGA-II (continued)	95
4.8	GP run results for Hypothesis 3.1.2 using NSGA-II (continued)	96
5.1	M_1 — 136 parameter neural network training and validation MSE per epoch . .	116
5.2	M_2 — 28 577 parameter neural network training and validation MSE per epoch .	117
5.3	M_3 — 28 577 parameter neural network training and validation MSE per epoch .	118
5.4	M_4 performance contrast when employing 5% dropout rate per layer	119
5.5	M_5 performance contrast when employing 10% and 5% dropout rates per layer .	121

5.6	Guided local search convergence for best predicted strategy per problem instance	126
-----	--	-----

List of Tables

2.1	Description of graphs shown in Figure 2.5	37
2.2	Summary of vertex-level properties considered	38
3.1	Constraint satisfaction problem status codes	54
4.1	Description of the node and terminal set used in Tree 1, the variable selector	66
4.2	Description of the node and terminal set used in Tree 2, the value selector	66
4.3	GP parameters for single-objective optimisation	70
4.4	GP parameters modified for multi-objective optimisation	71
4.5	MiniZinc 2015 Challenge results	72
4.6	Percentage time spent in the ranking function for ranking Definition 4.0.1	72
4.7	ORT solve status code summary for the MiniZinc Challenge 2015 problems	73
4.8	Cplex, Gurobi and ORT performance on costas-array and cvrp	74
4.9	ORT Minizinc Challenge 2015 run results (base case calibration)	75
4.10	Base case, Hypothesis 3.1.1 single-objective and multi-objective GP	82
4.11	Summary of Pareto frontier selection scheme results for Hypothesis 3.1.1	83
4.12	Hypothesis 3.1.1 results summary by problem class (training set)	84
4.13	Wilcoxon rank test for significant differences	85
4.14	Base case, Hypothesis 3.1.1 SAW and multi-objective GP full search comparison	88
4.15	Hypothesis 3.1.1 SAW and multi-objective results summary by problem class	89
4.16	Description of the node and terminal set used in Tree 1 Hypothesis 3.1.2	91
4.17	Base case, Hypotheses 3.1.1 and 3.1.2 SAW and multi-objective GP full search	97
4.18	Hypotheses 3.1.1 and 3.1.2 SAW and multi-objective results summary	98
4.19	Hypotheses 3.1.1 (base) and 3.1.2 GP full search comparison (training set)	99
4.20	Hypothesis 3.1.1–Hypothesis 3.1.2 SAW and multi-objective results summary	100
4.21	Hypothesis 3.1.2 SAW and multi-objective strategies evaluated on the test set	101
4.22	Select GP trees evaluated for Hypotheses 3.1.1 and 3.1.2	102
4.23	Operator frequency of best strategies found for a subset of problem classes	104

4.24	Operator presence over best strategies found for a subset of problem classes . . .	106
5.1	Linear regression model performance summary	114
5.2	M_2 and M_3 — 28 577 parameter neural network layer architecture	118
5.3	M_5 — 65 841 parameter neural network layer architecture	120
5.4	Model train, validate and test MSE and architecture summary	120
5.5	Strategy origin summary for predicted best strategies by M_5 for \bar{P}	122
5.6	M_5 — predicted strategy compared to the default ORT search strategy.	124
5.7	Summary of predicted best strategy (M_5) against the default ORT search	125
5.8	M_5^R — predicted strategy compared to the default ORT search strategy	128
5.9	Performance of M_5 and M_5^R on training instances and the base ORT search . . .	129
5.10	Performance of M_5^R on unseen instances compared to the default ORT search . .	129

List of Algorithms

2.1	Conventional Genetic Algorithm	21
2.2	Memetic Algorithm	26
2.3	Fast-non-dominated-sort (P)	30

CHAPTER 1

Introduction

Contents

1.1 Background	1
1.2 Problem description	2
1.3 Scope and objectives	3
1.4 Document structure	4

This chapter serves to provide context to this dissertation. A rough outline of the central hypotheses and high-level intuition is provided. The structure of the remainder of the document is also elucidated.

1.1 Background

Linear programming and (by extension) integer programming solvers have been privy to significant improvements both from a hardware and algorithmic perspective since their commercialisation during the 1980s. During the late 1990s serious undertakings were made by commercial entities (such as IBM ILOG with CPLEX) to incorporate important theoretical and computational improvements made during the previous 30 years. The result is that many problems which were previously intractable in *Mixed Integer Linear Programming* (MILP) environments are now solvable [30].

The solvers for such mathematical frameworks have matured, but the complexity of problems being tackled and user expectations of the underlying frameworks have in the meantime also increased. Scheduling problems are (loosely speaking) among the hardest problems¹ to solve and have far reaching applicability for many practitioners. *Constraint Programming* (CP) is a particularly useful framework for modelling and solving scheduling problems due to the natural representation of such problems within the environment. There are, however, several shortcomings when working with specific problem classes in CP, one of which the author seeks to address in this dissertation.

This is not to say that the methodology applied in this dissertation is not applicable to *Mixed Integer Programming* (MIP)², *Boolean Satisfaction* (SAT) or any other search technique that

¹From the perspective of representation and the quality of relaxation bounds.

²The shorthand abbreviation MIP is used as a substitute for MILP with the implicit understanding that linear relaxations are typically present in an integer search process.

requires branching decisions. The core contribution of the work presented herein is framework agnostic and philosophical in nature, focussing on the grammatical representation of constraints and the underlying problem graph.

1.2 Problem description

Discrete optimisation is a rich field of study with far reaching applicability. Many real world problems, as well as the bulk of problems considered in the literature, are NP-hard. This means that no polynomial time algorithm is known which can guarantee an optimal solution for a problem of this type or arbitrary size. As a result, many researchers focus on approximation algorithms, heuristics, or metaheuristics in order to produce good (or near optimal) solutions to such problems.

A recurring theme throughout discrete optimisation research is that heuristics are hand-crafted for the domain in which they are applied, often taking advantage of either features of the search space, or features of the input data in order to solve the problem, or both. Academic effort is aimed at improving the state-of-the-art in Job Shop Scheduling, Vehicle Routing, Train Scheduling and Sports Scheduling through such heuristics, to name but a few areas of research. Experienced practitioners will translate techniques used on prior problems to new search spaces. This process is inherently experimental and complex, requiring a deep understanding of the search space, search state, search mechanics, problem structure, possible decompositions and reformulations. This is particularly true if the heuristics have not been understood in the context of the constraint structures for which they were hand-crafted.

CP is primarily a modelling paradigm used in combinatorial search which aims to standardise common constraint definitions used when modelling. CP differentiates between local constraints and global constraints. The benefit of this standardisation is that it provides a common interface to which arbitrary solvers may subscribe. Furthermore, the specification of global constraints allows for compact representations of constraints (discussed throughout this dissertation) allowing logical inferences to be made over the domains of variables in order to facilitate effective search in the general sense. The CP grammar provisions for the specification of search-specific strategies with reference to the search instance variables, facilitating the creation of bespoke heuristics and their application to larger problems instances. A *Constraint Satisfaction Problem* (CSP) is the term used to describe a problem instance which defines the variables, constraints and objective function³ to be used in a search for a feasible or optimal incumbent.

This dissertation seeks to investigate and derive an alternative approach to the determination of appropriate branching schemes. Branching schemes comprise variable selection and value selection in CP. The alternative approach is performed using evolutionary processes and meta-data tailored to the problem domain rather than to the problem instance. The intuition of the approach is that if algorithms are designed at the model level, and not at the instance level, there may be reusable approaches across models which can then be applied to unseen cases of the same type. Current literature does not suggest that this approach has been attempted in the manner demonstrated herein. A possible explanation for the absence of research in this particular domain is that it represents a departure from traditional CP methodologies which aim to maintain their generality and performance across all possible domains. The approach of learning within domains is adopted (a train and test approach) since it is common in practice to have multiple models of the same type with different inputs which can be used to learn an approximate strategy which leverages that particular domain information. In this regard, the

³The objective is an optional specification. When omitted the resulting CSP is a pure satisfaction problem.

methodology applied does not facilitate learning across domains and as such falls in line with the no free lunch theorem, but this does not limit the usefulness of the methodology within specific problem domains.

1.3 Scope and objectives

The primary aim of this study is to investigate to what degree, if any, effective search strategies for CP can be autonomously evolved to solve classes of combinatorial optimisation problems. The process chosen to derive these strategies is *Genetic Programming* (GP), a well known metaheuristic. The secondary aim is to determine to what degree additional graph meta-data extends the ability to build effective search strategies.

The two aims of this study can further be broken down into three hypotheses which will be tested:

1. Can an evolutionary process produce algorithms which are able to assist CP solvers resolve CSP problems given only graph meta-data?
2. Does the inclusion of problem-specific meta-data result in better algorithms being evolved?
3. Can one predict which tree structures will be effective given graph meta-data?

The research tasks to be performed in order to meet the aforementioned aims of the study are as follows:

- I Complete a literature review of CP and underlying search techniques used to solve CSPs.
- II Complete a literature review of evolutionary algorithms.
- III Complete a brief literature review of approaches to the Algorithm Selection Problem.
- IV Complete a brief literature review of machine learning techniques with a focus on neural networks and deep learning.
- V Implement the proposed integrations and computational processes for testing the hypotheses.
 - (a) Automating the compilation of codes required across multiple languages .
 - (b) Creating performant integrations between evolutionary branching strategies and CP search.
 - (c) Configuring distributed processing of optimisation runs and CP search.
 - (d) Standardising the data collection methodology and database to facilitate the storage of CP search statistics in a distributed manner.
- VI Test hypothesis 1
 - (a) Identify whether evolutionary processes can develop algorithmic approaches to solving CSPs that perform well.
 - (b) Determine whether the approaches developed extend to other unseen instances belonging to the same problem class.

VII Test hypothesis 2

- (a) Investigate to what degree problem-specific meta-data are able to assist in algorithmic development.
- (b) Determine whether the approaches developed extend to other unseen instances belonging to the same problem class.

VIII Test Hypothesis 3

- (a) Data preprocessing related to symmetry reduction of search strategies found.
- (b) Configuration and parameter tuning of a deep learning model.
- (c) Evaluation and analysis of the accuracy of the fitted model.

The methodology used to measure the quality of a candidate branching strategy is to consider the speed at which a feasible incumbent is found for pure satisfaction problems or the best objective value found for optimisation CSPs. If two strategies are able to find equivalent objective values they are then compared on the time taken to achieve the objective value.

Evolutionary runs are analysed less scientifically by inspecting the rate of convergence of the population to the best candidate solution in the population. It is typical to observe incremental improvements in the best candidate solution in the population as iterations progress.

Deep learning models are fitted using a training set, a test set and a validation set — all of which are disjoint data sets. Learning parameters are fitted using the training set where the model error is controlled using the validation set to ensure that overfitting to the training data is avoided. Model accuracy is reported against a test data set which does not participate in the training process.

1.4 Document structure

Chapter 2 contains a literature review of the relevant material to this dissertation. The purpose of Chapter 3 is to provide detailed research hypotheses to be tested. This chapter provides the descriptions and intuitions around the thrust of the dissertation. Chapters 4—5 provide the results of aforementioned hypotheses together with detailed analyses and commentary. Chapter 6 provides a summary of the work presented, key findings and ideas for future work.

CHAPTER 2

Literature Review

Contents

2.1	Mixed integer and constraint programming	6
2.2	Constraint propagation	10
2.3	Search algorithms	12
2.3.1	<i>Backtracking search</i>	12
2.4	Restart strategies	14
2.5	Variable and value ordering heuristics	15
2.6	CP grammar	17
2.7	Evolutionary programming	18
2.7.1	<i>The genetic algorithm</i>	19
2.7.2	<i>Genetic programming representation</i>	21
2.7.3	<i>Fitness</i>	22
2.7.4	<i>Seeding operators</i>	23
2.7.5	<i>Selection operators</i>	24
2.7.6	<i>Crossover operators</i>	24
2.7.7	<i>Mutation operators</i>	25
2.7.8	<i>Memetic algorithms</i>	26
2.7.9	<i>Multi-objective genetic algorithms</i>	27
2.8	Automatic algorithm selection	31
2.8.1	<i>Evolutionary approaches to the ASP</i>	34
2.9	Static CSP feature extraction	35
2.10	Deep learning	39
2.11	Cooperative applications	41

The literature review is organised in the following manner; an introduction to the general optimisation formulation and the positioning of optimisation frameworks is given. A detailed review of the search process and search mechanics employed by CP as well an overview of the core concepts in constraint propagation. Variable and value ordering heuristics are discussed in detail with respect to the literature as this is a central point in this dissertation. A note on the usefulness of the CP grammar in the context of expressing optimisation problems is elucidated. Genetic algorithms and the genetic programming derivative are discussed in the context of metaheuristics. Discussion pertaining to the algorithm selection problem and portfolio-based search as well as evolutionary applications thereof is provided. CSP features also form a core

discussion point in this dissertation and a brief review of common features is highlighted. A minimal guide to the machine learning environment and the journey to deep learning is explored. Finally, key cooperative applications between metaheuristics and CP are examined with a focus on extensibility (or the lack thereof).

2.1 Mixed integer and constraint programming

The typical mathematical optimisation problem considered by an *Operations Research* (OR) practitioner has three facets:

- Sufficient abstraction of the problem to a set of features which can be represented by decision variables, constraints and objectives.
- In tandem with the above, selecting an appropriate framework in which to solve the resulting formulation. Often the framework chosen determines which features can be modelled directly and at which level of abstraction.
- Solving the resulting model, typically sub-optimally (for larger problems) and in rare instances, to optimality. Depending on the framework used, optimality bounds may be available for the problem instance. If the resulting model cannot be solved to optimality, one reverts to the steps above to either simplify the representation, reduce the number of variables, solve the problem in stages or apply a decomposition technique.

Optimisation problems are abundant; the role of the OR practitioner in the abstraction of the problem remains crucial. Commercial solvers¹ provide rich mathematical frameworks for the problem specification and equally strong solution techniques for problems which can be modelled within these frameworks. Typically supported models include linear, integer and quadratic programs. Other problem formulations, such as non-linear least squares problems (for bundle adjustments or camera-pose related problems²), have active communities working on open source projects such as Google's *Ceres* solver. The focus in this dissertation is on MIP optimisation problems where appropriate.

The formal definition of *Linear Programming* (LP) or *Linear Programs* (LPs) is given as follows:

$$\min\{w^T \mathbf{x} : A\mathbf{x} \leq b, \mathbf{x} \geq 0\}. \quad (2.1)$$

where \mathbf{x} is a vector of decision variables, $A\mathbf{x} \leq b$ is a system of linear inequalities to be satisfied by \mathbf{x} and $w^T \mathbf{x}$ is the objective function to be minimised.

While LPs were not originally designed to solve integer problems, it was a natural extension for LPs to approximate solving integer problems [48]. The *Travelling Salesman Problem* (TSP) is considered to be among the best-studied combinatorial optimisation problems [30]. The Concorde TSP Solver [10] is the de facto standard used for solving large-scale TSP problems optimally. The algorithms employed in Concorde exploit fractional relaxations to generate cutting planes which, in turn, reduce the search space. Once no further cutting planes can be found, branch and bound search schemes (in the traditional MIP sense) are deployed to resolve the remaining sub-problems. In the case where the LP relaxation is integer, it is also the optimal solution using this approach. The cutting planes found by Concorde are incrementally

¹Such as *CPLEX*, *XPress* and *Gurobi*, to name a few.

²Minimising the squared residual terms of a model function, given a set of observed predictor and response variables.

added to the tableau (the Dantzig–Fulkerson–Johnson approach) and the new LP relaxation is repeatedly solved as new cutting planes are found. Concorde is a good example of leveraging problem domain information to assist in solving the resulting integer problem.

The problem of identifying *good* cutting planes for a given TSP remains non-trivial, in that, while all required cutting planes can be enumerated, this would result in 2^n constraints in order to specify all subtour elimination constraints. This is an intractably large number of constraints for any LP solver as each additional constraint results in an additional row in the linear system $A\mathbf{x}$ above (2.1). Not all 2^n cutting planes are however required, since many are dominated by other members of the set, making them redundant during the execution of the simplex algorithm. The process and heuristic algorithms used by Concorde have a strong mathematical basis allowing for the generation of a near-minimum number of cutting planes to assert optimality on the TSP presented.

The TSP has been fortunate to have received the attention it has in the literature. Tools exist for problems that can be cast into a TSP formulation, while they themselves may not be natural instances of the TSP, to enable practitioners to solve them effectively. Many problems are not as rigorously studied or as fundamental and reusable as the TSP. While the approach of the Concorde algorithm is elegant, the amount of time invested to rigorously decompose the problem is often not available in practice. Recent years have seen a rise in the popularity of metaheuristics and hyper-heuristics [28], a move away from leveraging top-tier applied mathematics as demonstrated in Concorde. This is largely due to the generality of metaheuristics to larger sets of problems which do not fit neatly into the LP framework [32].

Integer Programming (IP) or *Integer Programs* (IPs) are an extension of the standard form presented in (2.1) where

$$\mathbf{x} \in \mathbb{Z}^n. \quad (2.2)$$

The integrality requirement on \mathbf{x} means that solving the resulting set of inequalities is an NP-complete problem. The simplest case where $\mathbf{x} \in [0, 1]$ and no other constraints are present is one of Karp’s twenty one NP-complete problems [97] which is a subset of the general class of SAT.

MILP or *Mixed Integer Linear Programs* (MILPs) have the integrality constraint (2.2) over some subset of \mathbf{x} and as a result are also NP-complete. MILPs are often abbreviated as MIPs as the linear component is implied as a result of the solution framework. Branch and bound strategies are used to solve MIPs by partitioning the search space in two for each integer portion in \mathbf{x} [114]. When the LP relaxation value of x_i is fractional, branches $x_i = \lceil x_i \rceil$ and $x_i = \lfloor x_i \rfloor$ are created in the MIP search tree.

The branch and bound procedure generates two branches (referred to as a left and right branch) and the resulting estimated cost of each branch is then computed (the bound). In a MIP this can be done by computing the *relaxation* of the remaining set of variables in \mathbf{x} where x_i is fixed to its value in the relevant branch. This relaxation derives its name from dropping the integrality constraints on the remaining integer components in \mathbf{x} and solving the resulting pure LP in (2.1). This approach produces a bound on the minimum cost of a branch that can be no larger than its integer solution. Branches which exhibit cost bounds higher than other branches already shown to have feasible solutions (with respect to integer components) can be eliminated as sub-optimal.

Effective branch and bound procedures require sensible variable selection schemes and a formulation which produces integer solutions close to their linear relaxation solutions. In pathological cases where feasible integer solutions are not close to their relaxations, this results in an expo-

nential number of branches being explored in order to determine whether an optimal solution has been found [117]. A natural extension to combat poor branching decisions in MIP environments is the Branch and Price method. This method requires a computationally effective and consistent pricing function which can estimate and prioritise branching decisions. Many works are available on pricing methodologies applicable to specific problem domains such as TSPs [14], *Vehicle Routing Problems* (VRPs) [176], and *Job Shop Scheduling Problems* (JSSPs) [180].

Complex scheduling problems exhibit the property of having many integer components which do not admit feasible solutions close to their linear relaxations. The resulting MIP is difficult to solve and often cannot be represented in terms of convenient (linear) mathematical constraints. Other frameworks, such as CP, may be used in these instances — sometimes in conjunction with mathematical formulations or through dual decompositions to guide them to optimality [155].

CP was first developed commercially during the 1980s [30] as a technique for solving satisfiability problems or CSPs. CP is also referred to as *Finite Domain Programming* [92] which speaks to the fundamental construct in CP, namely that variable domains are purely integer and finite. By contrast, LP deals with the representation of constraints on the real domain. A discretisation of the reals will provide an approximation technique for the treatment of real numbers within CP but the underlying constructs remain integer and finite by design³.

CP lends itself more naturally to scheduling problems which have a large number of integer components and associated constraints. Rather than a mathematical notion⁴ driving the solution process, CP solvers use a host of constraint propagation algorithms (inference) combined with heuristic branching schemes in order to search for a feasible solution or set of solutions to the model provided. In some instances heuristics are used to quickly assert infeasibility rather than feasibility, analogous to the cutting plane method or separation routines used in MIP solvers. The search algorithm used in CP is key in determining the overall success of the approach.

CP centralises many filtering techniques developed in different problem domains by standardising the underlying problem as a graph. The analogy in LPs is that the simplex tableau is used as a standard representation so that aggregation techniques developed independently by researchers can be applied to any problem that can be written in standard form. The advantage of the standard form in CP is that it affords more flexibility in the types of problems that can be expressed over its linear counterpart. The disadvantage of this flexibility is that it becomes less clear how decisions should be ranked and acted on in the search space due to there being less predictability in the problem form. In order to compensate for this complexity, many of the strategies employed in CP are designed around constraints with good propagation properties.

Figure 2.1 provides an overview of the high-level mechanics of resolving a CSP in CP. In broad terms, each CSP consists of a set of variables with a well-defined domain for each variable. In addition, a set of constraints exists between sets of the variables that must be satisfied with respect to their domains. A formal definition of the constraint network is provided in the following section. The algorithmic task is to find a feasible assignment of values to all variables within their domains that satisfy all the constraints [178].

If a naive depth-first search algorithm is employed to perform the variable and value selection process, such an algorithm is guaranteed, given sufficient time, to terminate with either a feasible incumbent (*End*) or a state where no incumbent exists (*Infeasible*). A depth search process

³One could argue that any computational system that does not natively handle arbitrary precision is a finite-definite solver. LPs retain high enough precision ($\approx 10^{-12}$) so that for practical purposes they are considered to operate over the real numbers. Where this precision is insufficient, input data can typically be rescaled to keep within the tolerances.

⁴In LPs, the idea of convexity and traversing a high-dimensional polytope.

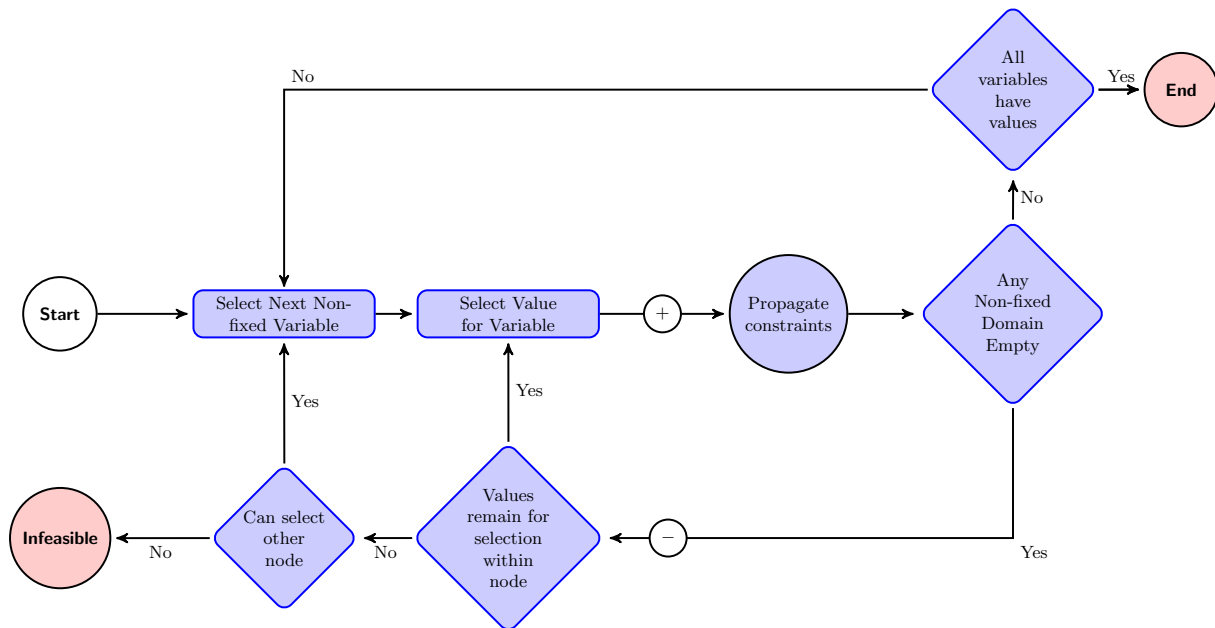


FIGURE 2.1: High-level CP pure CSP process flow.

creates a new tree node at each variable-value assignment, indicated by the (+) node, and retracts a tree node when rejecting a branch as being infeasible, indicated by the (-) node, in Figure 2.1. Superior search alternatives to this naive strategy are discussed later in this chapter.

Figure 2.1 illustrates that the propagation techniques provide the central backbone of the search process. The propagation, or inference, procedures are suitably placed to make a deduction, or infer as a consequence of a variable-value decision which other variable domains or arcs may no longer be acceptable in the CSP. Propagation techniques are discussed further in the next section. This is not to say that propagation algorithms operate in isolation from the search technique; hybrid techniques that use conflicts found during propagation procedures to inform future branching decisions are commonplace.

In a pure CSP one is primarily interested in finding at least one feasible incumbent. For the algorithmic process outlined in Figure 2.1 there are a few natural extensions which allow any optimisation problem to be framed in the context of a pure CSP. The first variation of the procedure would be to find all feasible solutions to a given CSP (albeit an uncommon request). At each point a feasible incumbent is found, a new constraint can be added to the problem prohibiting that solution from future visitation and the search simply restarted. This is not how enumeration approaches work in practice as the search sequence implicitly preserves portions of the search space already traversed during the enumeration.

One may pose the question that if an incumbent is successfully found during process 2.1 as to whether it is an optimal incumbent? This leads naturally to the second extension of the CSP process, which is to frame optimisation problems requiring maximisation or minimisation as CSP instances. This is achieved by monitoring the objective function value and at each point when a new incumbent is found, to place an additional constraint requiring that the following solution should have an objective value constrained by the current incumbent objective value. The direction of the constraint depends on whether the optimisation problem is a maximisation or minimisation problem. If no feasible solution exists before an incumbent is found, the problem is infeasible. If at least one feasible incumbent is found, and the process terminates in an *infeasible* state, an optimality proof for the last solution has been found.

2.2 Constraint propagation

A constraint within CP may be defined formally as follows [24]:

Definition 2.2.1 (Constraint) *A constraint c is a relation defined on a sequence of variables $X(c) = (x_{i_1}, \dots, x_{i_{|X(c)|}})$, called the scheme of c . Here c is the subset of $\mathbb{Z}^{|X(c)|}$ that contains the combinations of values (or tuples) $\tau \in \mathbb{Z}^{|X(c)|}$ that satisfy c . $|X(c)|$ is the arity of c . Testing whether a tuple τ satisfies a constraint c is called a constraint check.*

It is worth noting that in the general definition of a constraint in CP, the relation itself is not required to be explicitly defined, merely the set of variables over which the constraint applies. Global constraints enable special treatment of the above definition where a function of arbitrary arity, n , is applied over $\{x_1, \dots, x_n\}$, typically yielding enhanced performance in the handling of constraint checks and the propagation of values through the appropriate use of specific data-structures which often yield sub-linear query times.

A constraint network within CP is defined as follows [24]:

Definition 2.2.2 (Constraint Network) *A constraint network is composed of:*

- *a finite sequence of integer variables $X = (x_1, \dots, x_n)$,*
- *a domain for X , that is, a set $D = D(x_1) \times \dots \times D(x_n)$, where $D(x_i) \subset \mathbb{Z}$ is the finite set of values that variable x_i can take, and*
- *a set of constraints $C = \{c_1, \dots, c_e\}$, where variables in $X(c_j)$ are in X .*

A network N can be referred to in terms of its components pertaining to the variables, domains and constraints; that is, $N = (X, D, C)$. The process of search requires that values are assigned to the variables of the network in some manner — typically a backtracking algorithm, discussed further in the following section, that ensures constraint adherence in the search process. The process of assigning a value to a variable is referred to as an *instantiation* of that variable [24]:

Definition 2.2.3 (Instantiation) *Given a network $N = (X, D, C)$,*

- *An instantiation V on $Y = (x_1, \dots, x_k) \subset X$ is an assignment of values v_1, \dots, v_k to the variables x_1, \dots, x_k , that is, V is a tuple on Y . V is denoted by $((x_1, v_1), \dots, (x_k, v_k))$, where (x_i, v_i) denotes the value v_i for x_i .*
- *An instantiation V on Y is valid if, for all $x_i \in Y, V[x_i] \in D(x_i)$.*
- *An instantiation V on Y is locally consistent if and only if it is valid for all $c \in C$ with $X(c) \subset Y$ and $V[X(c)]$ satisfies c . If V is not locally consistent, it is locally inconsistent.*
- *A solution to a network N is an instantiation V on X which is locally consistent. The set of solutions of N is denoted by $\text{sol}(N)$.*
- *An instantiation V on Y is globally consistent (or consistent) if it can be extended to a solution (i.e. there exists an $s \in \text{sol}(N)$ with $V = s[Y]$).*

Constraint propagation is the form of inference used by CP to reduce the search space. Constraint propagation itself is not a search technique, but rather the mechanism used for ensuring consistency in the constraint network [60]. The definitions provided here centre around a sequence of variables rather than a set of variables. The reason for this definition is that it corresponds to the way in which the search procedure will be conducted.

A solution to the CSP is one where a valid instantiation is found for a given network N , for all variables X_N on domains D_N that satisfy all constraints C_N . CSPs are NP-complete problems in that no algorithm is known that can guarantee a polynomial time performance to find a solution or optimal solution. Constraint propagation transforms the network by reducing domains, adding additional constraints or tightening constraints. There are two ways of formalising constraint propagation, namely by rules iteration and local consistency.

Rules iteration [138] determines reduction rules based on the constraints — these reduction rules are then able to eliminate instantiations that cannot appear in the final solution. A *local consistency* is a property that must be satisfied, independently of constraints and domains, in order for the partial-instantiation to be part of the final solution.

The most common modification made to CSPs in order to satisfy them is through the domains of the variables. This is done through domain-based propagation and can be achieved through rules iteration or local consistency. Domain-based propagation will attempt to find a new value in the current partial solution that reduces the remaining search space (*i.e.* the domains of free variables) achieved through polynomial techniques that attempt to approximate a globally optimal allocation strategy. Domain-based rules iteratively process the available set of constraints and determine which values cannot appear in the final solution. Domain-based reduction rules are also referred to as propagators.

Arc consistency [128] is an intuitive constraint propagation technique which ensures that domains are consistent with their constraints. The analogy is that constraints are arcs between nodes (the domains) and that any affected node should ensure that all arcs associated with that node are locally consistent. A formal definition[24] is provided as follows:

Definition 2.2.4 ((Generalised) Arc Consistency ((G)AC)) *Given a network $N = (X, D, C)$, a constraint $c \in C$, and a variable $x_i \in X(c)$,*

- *A value $v_i \in D(x_i)$ is consistent with c in D if and only if there exists a valid tuple τ satisfying c such that $v_i = \tau[\{x_i\}]$. Such a tuple is called a support for (x_i, v_i) on c .*
- *The domain D is (generalised) arc consistent on c for x_i if and only if all the values in $D(x_i)$ are consistent with c in D (that is, $D(x_i) \subset \pi_{\{x_i\}}(c \cap \pi_{X(x)}(D))$).*
- *The network N is (generalised) arc consistent if and only if D is (generalised) arc consistent for all variables X on all constraints in C .*
- *The network N is arc consistent if and only if \emptyset is the only domain tighter than D which is (generalised) arc consistent for all variables on all constraints.*

The notion of arc consistency has roots in binary, normalised networks — also referred to as Boolean satisfaction problems (or SAT). The generalised notion of arc consistency extends to cover integer networks but is often referred to by other names such as *hyper arc consistency* or *domain consistency*.

Determining arc consistency can be computationally expensive (and potentially intractable) and choosing when to enforce GAC on a particular constraint is a central question in CP. CP solvers

hinge on multiple propagation algorithms such as AC3 [128], AC4 [137], AC6 [23] and AC2001 ([26, 27, 191]), culminating in algorithms which heuristically reorder the propagation list.

Modern CP solvers obfuscate the underlying propagation algorithms being employed and allow users to rather interact with the priorities of the search. This allows practitioners to experiment heuristically with variable and value priorities in order to improve the search performance for a given CSP. The natural extension is to use a scientific approach to heuristic selection which has resulted in two different approaches:

1. Applying an algorithm selection formulation to the problem by selecting from a portfolio of algorithms in order to solve the problem.
2. Optimising the variable priorities using metaheuristics.

The former of these approaches has a longer history in the research community and in many ways is an extendible approach over the latter since it can be applied across problems. The second approach, while demonstrating how far heuristic “black-box” solvers are from a problem-optimised strategy, does not yield dividends in the ability to solve problem instances beyond the instance being solved. These two approaches are discussed in detail later in this chapter.

2.3 Search algorithms

Search algorithms used to solve a resulting CSP fall into three main categories, namely; backtracking [74], local search [2] and dynamic programming algorithms [19]. Depth-First search briefly mentioned in §2.1, resides in the backtracking category and is the simplest example of a complete search. A complete search is one where the algorithm guarantees to find a solution if it exists or demonstrate that no solution exists. Dynamic programming algorithms are also complete algorithms, although they often have additional computational overheads in memory which result in these algorithms being less popular in practice. Local search algorithms, while often very effective at finding solutions, are unable to provide any guarantees as to the solution quality if a solution is found. The scope of discussion in this section is restricted to backtracking search algorithms.

2.3.1 Backtracking search

Backtracking algorithms do not explicitly define the search tree *a priori* but rather create the search tree throughout the search as feasible or infeasible portions of the search space are uncovered. One can consider such a search as being a sequence of decisions over the variable-value space in the CSP.

The search tree initialises with an empty root node, after which a variable is chosen in conjunction with a value to be assigned. This *extends* the root node to form a new node in the tree which represents the value assigned to the selected variable, say $\{x_1 = a_1\}$. This decision point of variable-value assignment is referred to as a *branching decision*. The approach followed to select the next variable and next value for a given variable is referred to as the *branching strategy*. Propagation is applied to the CSP as a result of the variable assignment and the process is repeated. The state of the search tree at any point can therefore be written as a sequence of decisions: $p = \{x_1 = a_1, \dots, x_n = a_n\}$.

Considering the search tree as a sequence of decisions allows for convenient manipulation of the data structure when rejecting a decision. Since each branching decision results in the union of

the existing decisions with the new decision, when refuting a decision, one can simply remove the last decision made. A branching decision can also take on multiple forms, such as branching on a variable-value assignment $\{x_i = a_i\}$, a domain partition $\{x_i \in P_{x_i}\}$, $P_{x_i} \subset D_x$ or partitions across multiple variables $\{x_i \in P_{x_i}, y_i \in P_{y_i}\}$, $P_{x_i} \subset D_x, P_{y_i} \subset D_y$. Branching decisions consisting of a variable assignment being true or not are referred to as binary choice points $\{x_i = a_i, x_i \neq a_i\}$.

It is required that at each branching decision a complete list is maintained (implicitly or explicitly) of the remaining branching decisions should backtracking be required in order to maintain a complete search. It is typical when diagramming trees to order these branching decisions from left to right where the priority of execution in the tree is left first, right last. This means that more favourable decisions are executed first, potentially resulting in a quicker search. Hooker [86] provides an excellent treatment of the required transparency when developing branching strategies in order to fully understand their relevance across problem domains.

If branching decisions are omitted opportunistically from a particular tree node in the search, then the search is no longer complete and may be considered heuristic in nature. Many local search routines can be framed in this manner by constructing a neighbourhood consisting of a subset of all possible branching decisions which are opportunistically explored in an attempt to find an improvement.

The naive backtracking algorithm maintains the chronological order of branching decisions in a last-in, first-out manner. Within this naive backtracking procedure there are additional variations as to how constraints and network consistency are enforced. There are schemes which require arc consistency with at least one uninstantiated variable (MAC [159, 66]) or require arc consistency with exactly one uninstantiated variable (Forward Checking [131, 81]) while determining possible branching decisions throughout the search.

Examples of algorithms which depart from the chronological ordering of search tree nodes are *Conflict-directed backjumping* (CBJ) [152], *limited backjumping* (LB) and *dynamic backjumping* (DBT) [69, 161] which utilise *nogood* reorderings. Unlike propagation schemes, which discover infeasible portions of the search space after the assignment, a *nogood* [168] can identify structures *a priori* [52] or during the search which are present in no solution, reducing the remaining search space, and providing a reference point for which the algorithm can revert back in the search tree thereby potentially bypassing multiple tree nodes in the reverting of branching conditions. Finding minimal *nogoods* is non-trivial and a function of the ordering of the variables and constraints checked, although there are schemes which provide generalised *nogoods* [98] which begin to approach the clause efficiency enjoyed by most modern SAT solvers. Clauses in SAT are analogous to *nogoods* in CP and cutting planes in MIP.

Many *nogoods* may be found during a non-trivial search. If too many *nogoods* are kept during the search, the performance of the branching may degrade as the reduction in search space is offset by the time taken to verify whether any *nogood* is active over a pending decision. As a result of this trade-off it is typical to restrict the number of *nogoods* stored according to some criterion [16] (relevance-bounded) or limit the recording of *nogoods* that are at least of a certain size [52] (size-bounded). The relevance criterion is determined by the number of actively assigned variable-value pairs in which a *nogood* participates. The power of *nogoods* came to the fore as a result of the *watch-literals* data structure [140] which, while developed for SAT, is a portable concept to CP. Empirical results show that combining *nogood* reorderings, variable reorderings (in line with the *nogoods*), relevance bounding, search restarts and watch-literals result in dramatic improvements in search performance [140].

2.4 Restart strategies

A significant vulnerability of the depth-first search approach is that a disproportionate amount of time may be spent searching variables between the top and bottom of the search tree. Nodes that are extended first (the “top”) are often left unchanged until the backtracking process returns to change initial decisions made. This leaves the search vulnerable to poor decisions that are made early on during the search [84]. One way to combat the disproportionate search effort over variables is to restart the search in order to return to making a decision on variables near the root node. It should be obvious, however, that unless the ranking of the variables is changed in some way, the search will return to the next leaf node in the search path before the restart was effected. For this reason, it is difficult to talk about restart strategies without considering strategies which also reorder the variables in some manner.

The addition of a controlled amount of randomisation to the search was demonstrated to achieve orders of magnitude speed-up in the search time with Gomes *et al.* [75] providing a statistical argument as to the heavy tailed distributions that deterministic algorithms exhibit. By limiting the number of backtrack moves permissible before a search is restarted and introducing randomness to break ties on equivalently ranked variables, a marked improvement was found across many problem domains and search schemes.

Several approaches have been developed for randomising either the variable order, the value order (within a variable selection) or the order within some tolerance for variables and values. Where heuristics are used to calculate the ordering of variables, one may select randomly according to some distribution, from a suite of heuristics at each tree node. Most of these approaches preserve the completeness of the search which is a desirable feature of the approaches. Some approaches, however, involve random backtrack jumps [151, 190] which, while effective, are unable to preserve the completeness property of the search algorithm [179].

There are multiple approaches to determining when to restart a search. The simplest of these approaches has already been mentioned and applies a fixed cut-off on number of branches explored before a restart of the search is performed. Another approach used [84] measures the distance of a backtrack from a terminal node and triggers a restart if the distance exceeds some fixed cut-off rather than counting the number of branches explored. A variation of the fixed cut-off strategy on the number branches is to rather count the number of nodes visited [99]. An alternative to the fixed cut-off strategy, which is based on the number of branches, is to enforce a variable cut-off based on some sequence of cut-offs provided where the next item in the sequence is used as the cut-off on each restart. Luby *et al.* [125] provide a universal restart strategy based on the survival function properties of the algorithm runtime when no runtime information is known about the distribution — this approach is aptly called *Luby restarts* when used in practice. Luby restarts have been criticised for an assumption of independence between successive runs in the derivation of the optimal restart policy, which has been shown to be suboptimal in such cases [99].

Fischetti and Monaci [58] considered a restart strategy applicable to MIP solvers which exhibit high variation in the solution process given different starting conditions. Such starting conditions are applicable to the approach employed by the solver in order to break ties on equivalently ranked tree nodes. The methodology developed by Fischetti and Monaci is termed a *bet-and-run* [58] approach whereby several short samples of induced random branching decisions are analysed and the most promising strategy is then employed to complete the exact search. The authors explored at most five random strategies to keep the computational overhead low for simpler problem instances and found that while the approach was not competitive for small problem instances, that more complex problem instances were, in general, solved more quickly

employing this approach. Fischetti and Monaci [58] successfully employ the exponential erraticism in search trajectories to their benefit through this naive restart policy in a MIP context and acknowledge that utilising techniques which are commonplace in CP and SAT, may lead to further improvements in the methodology.

2.5 Variable and value ordering heuristics

An optimal search is one where the minimum number of nodes is visited in order to solve the CSP [179]. One can imagine an oracle which is able to make a perfect decision at each decision point in the search to select the next variable and value assignment such that no backtracking is required and a feasible solution is obtained, assuming one exists, at the first leaf node of the search. In the case where no feasible solution exists, the oracle provides a perfect ordering for variables in such a way as to minimise the number of nodes required to assert that no feasible solution exists. This minimal order would be subject to the *nogoods* policies adopted, propagations enforced and backtracking scheme employed. Creating such an oracle that is able to make a decision for the first node in the tree has been shown to be as hard as solving the CSP itself [123]. Methods to approximate an optimal oracle [88] are difficult to derive in practice; the result is that the bulk of the research effort has concentrated on heuristic approximations which do not provide a guarantee as to the optimality of the branching decision [179].

The order in which variables are prioritised in the search have been shown to be a crucial determinant in the search efficacy [11, 68, 70, 81]. Two main ordering schemes are used in practice, namely static and dynamic orderings [11, 74]. A static ordering is one where the priorities of variables to be extended at each node in the search tree remain unchanged throughout the search. Dynamic orderings are commonplace in CP and several techniques may be used to evaluate how orderings should be re-prioritised based on the search experience. Some ordering schemes do not exploit the search experience (or history) but rather the state of the variables in the current state of the search at each node. This can provide different orderings of variables as a function of propagators and domain reductions. Designing heuristics as a function of the search state is a well explored topic, much of which hinges on which information to extract from the search state with respect to variables and value domains when building a heuristic. As a result of this new search problem (*i.e.* which features characterise a good heuristic), these approaches often reside under the topic of automated algorithm design or portfolio solvers and have received criticism for the lack of a theoretical foundation as to why certain approaches dominate others [86]. Portfolio solvers are discussed in more detail later in this chapter.

Examples of dynamic variable ordering heuristics which utilise the search experience are conflict-directed search [41, 119, 194] and failure-directed search [181]. Conflict-directed search identifies a variable which is the source of infeasibility, and when combined with backjumping, can determine the level to which the search should revert back in the tree and is referred to as *conflict-directed backjumping* (CBJ) [41]. The authors of [41] provide a detailed explanation of the arc-consistency maintained by the propagation operators used in the search and the conditions under which CBJ is able to benefit the search performance — illustrating that while CBJ was previously considered an unnecessary scheme [25], there are indeed merits to CBJ. Lecoutre *et al.* [119] provide a treatment of different combinations of backjumping, conflict detection and propagation schemes culminating in a process which prevents thrashing in the search. *Thrashing* occurs when a search process re-identifies infeasible sub-problems repeatedly while attempting to obtain a feasible solution. The intuitive interpretation for observed thrashing in a search procedure occurs when a large subtree is infeasible and the search procedure is required to complete the traversal before leaving the subtree [88].

A contrasting technique to CBJ is *failure-directed search* (FDS) which focusses on eliminating infeasible portions of the search space, rather than attempting to find feasible solutions directly [194]. The intuition behind FDS is rather elegant; it assumes that finding a feasible solution is harder than finding an infeasible solution, thus any reduction in the infeasible search space results in improved search performance over the remaining feasible search space. The ranking of variables is thus based on their likelihood to produce a feasible solution. A weakness of FDS arises when problems are less constrained as the search strategy is far less effective. In order to combat this, Zivan *et al.* [194] used a combination of *Local Neighbourhood Search* (LNS) and FDS in the Ilog CP Optimiser (at the time of publication).

As an aside, one can consider the FDS as being a close analogy to the cutting plane method used in MIP. The purpose of the cutting plane is to eliminate infeasible portions of the search space while the linear relaxation helps keep the next incumbent (after the introduction of a cut) restricted to a very small portion of the feasible space which is potentially large.

Another important ranking technique used by modern commercial CP solvers is *impact-based search* (IBS) [156]. IBS exploits the relative importance of each variable-value as computed by the search based on the effect of a variable-value decision on other variable domains. This allows the scheme to prioritise variables which have a higher impact on other variables which leads to typically smaller variable domains as the search tree grows in size. IBS is coupled with an appropriate restart strategy to allow the reprioritisation of variables periodically through the search. One may consider that there are two primary levels at which ranking can occur, namely every time a node is to be extended, a ranking can be computed based on the state and search history (local rankings) or, alternatively, a ranking of variables and values can be computed each time the search is restarted (global rankings). There is a trade-off between the additional computation of ranking at each tree node *versus* ranking once on a restart. The dominant of these approaches often being linked to the underlying problem and combination of backtracking approach, restart strategy and propagation scheme employed.

The final class of variable ordering heuristics utilise either the domain of the variables or the structure of the CSP, or potentially some combination of the two, to characterise the ranking of a variable or value. Using the size of the remaining variable domain to rank the variables was proposed by Haralick and Elliott [81] — where the size of the remaining domain changes throughout the search as a result of propagation algorithms used. This strategy of selecting the smallest remaining domain has been demonstrated to be an optimal strategy [143], under certain conditions, minimising the number of nodes required in the search tree. An extension to this method is applied in graph colouring [34] where the number of constraints associated with a variable, referred to as the degree of a variable, are used to break ties in the ranking of the variables where the remaining domains are equivalent. A further heuristic is proposed in [25] which divides the remaining domain size by the degree of the variable and it was demonstrated to work well on certain problems. This approach was further deployed by weighting the divisor in this scheme according to how many times the variable is involved in a dead-end state [33], once again demonstrating that this approach worked well on certain problems.

The value selection problem has not received as much attention as the variable selection problem, but still yields many possible approaches. Dechter *et al.* [53] provided a Bayesian approach to weighting the probability that a value exists in the final solution, computable *a priori* in polynomial time. A criticism of this method is that it does not consider the size of the subtree in which a solution value may exist, or the difficulty of obtaining such a subtree [179]. Simpler, less mathematically rigorous methods are available for selecting the value for a variable. Such methods include selecting a value that minimises the product of the remaining variable domains [67, 70]. This is also referred to as the “promise” heuristic. Selecting a value which maximises

the summation of the remaining domains has also been proposed [62] although this method is less theoretically attractive [179].

The second class of heuristics do not exploit the variable-value domains but rather leverage the structure of the CSP graph. This is not to say the methods are mutually exclusive and cannot co-exist in the implementation of a search algorithm. Examples range from finding cutsets in the graph [53, 160], analysing the bandwidth of the graph [189], applying recursive decompositions [61, 122], performing tree-based decompositions [140] to enforcing recently found *nogoods* [95].

One can imagine that in a situation where features of the variable domains, values or CSP graph may lead to more efficient heuristic orderings of the variables, the size of the feature space and the heuristics which can be formed as a result is an interesting aspect to investigate. Several authors have approached the problem of finding good heuristics based on these features [13, 57, 135, 144] while others have motivated for a strong theoretic basis for why heuristics perform well in different environments [86].

There are many possible features from both the variables and the CSP graph which can be considered when creating ordering heuristics in a search strategy. Wallace [182] performed a factor analysis of different ordering heuristics and found that the primary factors differentiating the search efficiency among the heuristics tested were immediate and future failure. This grouping of heuristics was determined by analysing the set of heuristics commonly used with the MAC and forward checking search algorithms.

The process of selecting among pre-defined heuristic approaches, or selecting between different solvers encapsulating more complex processes is discussed further in this chapter.

2.6 CP grammar

Without discounting or diminishing the significance of the propagation, filtering and search algorithms discussed in the prior sections, it is worth noting that a significant advantage of using CP lies beyond the implementations of such algorithms. CP provides the user a level of separation between these sophisticated algorithms and the problem domain through the modelling grammar. It can appear to be counter-intuitive to think of the grammar as decoupled from the solver, however, this is commonplace in CP as it allows problems to be framed independently of the solvers. The analogy in MIP environments would be the standard MPS or AMPL format for providing a tableau to a MIP solver. This separation makes testing and benchmarking of different solver implementations a somewhat trivial exercise. It has become increasingly common in the optimisation community to adopt the CP grammar for modelling and to utilise other non-CP techniques (such as SAT, LP-SAT or MIP) to solve the CSP. This allows the solver implementation to tailor a structural exploitation which is embedded in the grammar as to assist in solving the problem instance.

A simple example of leveraging multiple solution approaches would be using LP-SAT to solve a problem containing a *circuit* constraint. The circuit constraint asserts the well-known TSP over a set of nodes. The LP-SAT solver can then use the linear programming relaxation and cutting planes in addition to the standard SAT approach to solve the problem instance. If the *circuit* grammar was not used, the explicit structural requirements over the nodes would require a minimum of n^2 constraints using the *Miller-Tucker-Zemlin* (MTZ) formulation [134] which has very poor relaxation properties without performing integer variable lifting, or 2^n constraints employing the *Dantzig-Fulkerson-Johnson* (DFJ) formulation [48] which has good relaxation properties, but which cannot be practically enumerated for reasonably large problems. Herein

lies the advantage of having implicit constraints through the grammar which can be generated, or enforced, as required by the solver engine. The grammar, search strategy and solution technique can be considered independent components of an optimisation engine.

2.7 Evolutionary programming

Metaheuristic search is often employed by OR practitioners when standard exact solution techniques are intractable or cannot be represented in a sufficiently compact formulation⁵. *Genetic Algorithms* (GAs) have contributed significantly to the metaheuristic space though the implementation of the underlying principle of natural selection [73]. Conventional GAs assume a finite, chromosomal representation of the problem being modelled [73]. Much work has been done on different GA operators which can be applied for different problem types. Good examples of this applied to the TSP may be found in [78, 116, 141].

By contrast, GP does not deal with a fixed string representation of the problem, but rather evolves a program (a policy or decision tree) which is used to represent a solution to the problem [110]. The set of nodes and terminals permitted in solving the problem define the possible search space of solutions which can be generated, subject to a limitation on the depth of the decision tree. GPs have demonstrated their prowess over conventional GAs in certain problem domains in terms of time required to reach optimality, representation conciseness and efficiency⁶.

Koza [111] and Goldberg [73] would have struggled to estimate just how far-reaching the impact of their research would be. Thousands of researchers have built on their respective bodies of work resulting in refined, multi-objective, scalable derivatives of the original metaheuristics. Bio-inspired metaheuristics also gained popularity but not to the credibility of the scientific community [165]. The reason for this was driven mostly through seemingly novel parallels being drawn between nature and algorithmic approaches without a thorough treatment of the underlying processes [59].

A wide range of options exist outside evolutionary algorithms when discussing possible bio-or nature-inspired optimisation strategies. *Simulated Annealing* (SA) [104], *Tabu Search* (TS) [72], *Particle Swarm Optimisation* (PSO) [100] and *Ant Colony Optimisation* (ACO) [56] are all established and widely used metaheuristics which have demonstrated their efficacy over one another on some subset of test problems. For many practitioners, the choice of which metaheuristic to use is typically related to the familiarity to the user, the computational requirements and the representation of the problem. GP is the metaheuristic, from this aforementioned set, that fits the representation requirements in the context of this thesis.

In order to motivate the reasoning behind the selection of GP as the metaheuristic of choice, the representations of the various candidates against the requirements of the domain need only be considered. SA, GAs and PSO use a classic vector representation for the problem representation. This means that each control parameter and domain is known up-front and is then combined to produce a single parameter vector. The parameter vector has a corresponding mapping in the objective function where the interpretation and evaluation of the parameters is known. For problems where the number of parameters (or variables) are fixed, continuous, and domains known⁷, these metaheuristics are an appropriate fit. The applicability of the operators used by

⁵The memory footprint required by large problem instances (and their associated representations) is a significant factor when dealing with practical scheduling problems.

⁶The Santa Fe Ant problem provides a simple efficiency comparison of the two metaheuristics.

⁷Scaling parameters can be introduced to allow for larger domain searches but are themselves still well-defined.

the metaheuristics to the search-space and level of dependence between variables will determine which approach yields the greatest initial success.

ACO and TS are similar in that both map to a classic graph representation of the problem. This intuitive mapping is one of the reasons why ACO has received so much attention with respect to TSP-related problems. TS is grouped with ACO in the sense that an underlying representation of the problem as a graph is required but, in addition, improvement operators are then specified in terms of that graph. This means that TS does not generalise well in that improvement operators specific to the problem domain are required as input.

The problem being addressed in this dissertation is one which does not lend itself to a fixed number of variables or a fully specified graph. While TS is a potential candidate, in that strategies can be specified as operators in the search, the downfall is that it cannot actively evolve or test new strategies *i.e.* it performs a search given a fixed operator set, not a search over the operator set and search space. For this reason, GP is the prime candidate for use in this work.

2.7.1 The genetic algorithm

The conventional genetic algorithm is applied in its simplest form across most evolutionary algorithms. Exceptions are made when using models which run asynchronously or in a distributed fashion there the resulting algorithmic modifications are an intuitive compromise. A common nomenclature is used when discussing the algorithm. The term *individual* is used to describe a candidate solution. A collection of individuals, often being compared to one another, is referred to as a *population*. Operations are performed on the population, relating to individuals or collections of individuals: *Selection*, *reproduction*, *crossover* and *mutation*. Finally a *fitness function*, analogous to the objective function in classical mathematical optimisation, is used to determine the quality of individuals.

The GA commences with a seeding step during which an initial population is created. In simple problems, this can be done randomly. For more complex problems, sophisticated seeding schemes may be used to initialise the population at high quality, diverse locations within the solution space. Seeding operators with respect to GPs are discussed later in this section.

Figure 2.2 contains a flowchart description of the basic working of a GA, while Algorithm 2.1 contains a pseudo-code description of a GA without the details of which operator is being performed on individual i . The termination criterion is typically a prespecified maximum number of generations or a target objective value (whichever occurs first). The population is replaced at each iteration, or generation, by a new population of size M . Probabilities are assigned for the application rates of the different genetic operators of reproduction, crossover and mutation as P_r , P_c and P_m , respectively. The selection operator is typically also probabilistic but the interpretation differs based on the mechanism used. Each individual in the population is indicated as the i^{th} member in this representation.

The reproduction phase is not afforded attention in the proceeding sections as it does not modify an existing population individual during the process of transferring it to the new population. Reproduction is used to ensure that some information is preserved in its entirety in the new population. *Elitism* is a successful reproduction process which ensures that the best individual from each population is always carried forward to the new population with probability $P_r = 1$. This ensures a monotonic objective function when the objective function does not contain any stochastic elements.

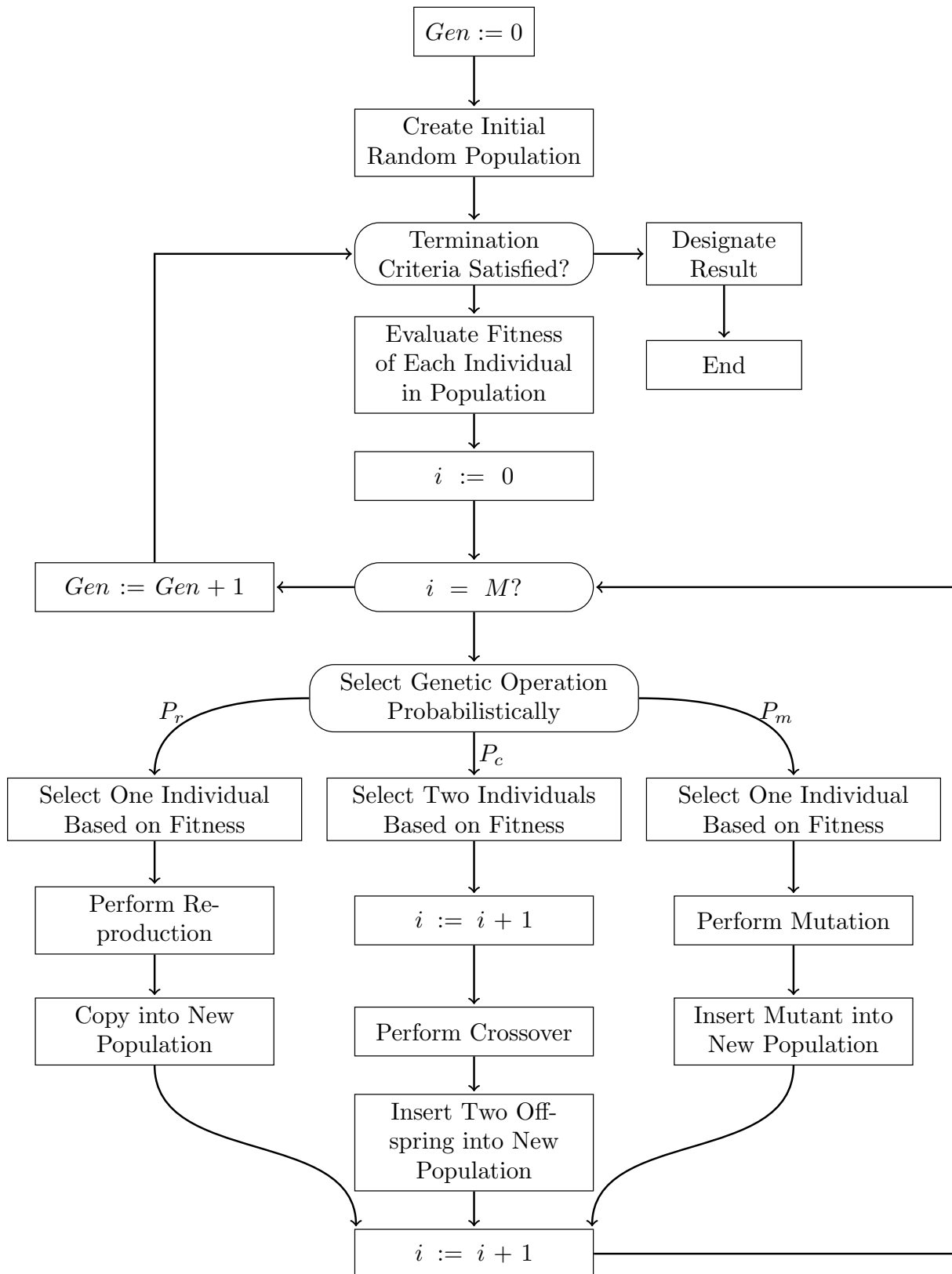


FIGURE 2.2: The conventional genetic algorithm flowchart [111].

Algorithm 2.1: Conventional Genetic Algorithm

Data: Population Parameters

```

1  $Gen := 0$ ;
2 Create initial population,  $P(Gen)$ ;
3 Evaluate population  $P(Gen)$ ;
4 while Termination criterion not met do
5   while  $|P'(Gen)| < |P(Gen)|$  do
6      $i \leftarrow Select(P(Gen))$ ;
7      $i \leftarrow PerformOperator(i)$ ;
8     Add  $i$  to population  $P'(Gen)$ ;
9   end
10  Evaluate population  $P'(Gen)$ ;
11   $P(Gen + 1) := P'(Gen)$ ;
12   $Gen := Gen + 1$ ;
13 end

```

It is worth noting that the flowchart in Figure 2.2 does not specify the problem representation or detail the operators used, and is applicable to both GAs and GPs.

2.7.2 Genetic programming representation

GPs inherit the mechanisms from GAs in terms of the population mechanics but distinguished by the solution representation. In conventional GAs, a fixed length representation is required for the problem where the number of parameters which participate in the solution are known *a priori*. In GPs, the solution is represented as a decision tree. Koza [111] frequently refers to the solution process as *program induction* indicating that the search space is not defined over the set of parameters (as with conventional GAs), but rather over the space of all programs that can be generated given a node and terminal set. This requires the user to describe the functional aspects of the problem space, rather than the variables directly, in order to represent the problem as a GP.

Modelling a hierarchical solution, or expression tree, provides great flexibility for certain problem types. As an example, consider determining the functional form of a regression model. In this example, one could consider providing the GP with a node set consisting of the arithmetic set $\{+, -, \div, \times\}$ and a terminal set consisting of $\{0, 1\}$. A more tailored function and terminal set may produce better solutions quicker. This remains a modelling consideration with all evolutionary programs and is referred to as the representation problem. Figure 2.3 provides an example of a tree-based representation for a candidate solution in GP.

The expression tree is generated, given a set of compatibility constraints between nodes and terminals. An unconstrained representation still requires that the arity of nodes is satisfied so that the tree will be completely filled with no missing terminals. Methods for generating initial population trees are described later in this section.

The example provided in Figure 2.3 uses a classic arithmetic node and terminal set which is a logical representation when modelling mathematical functions that often assume this precise functional form.

When modelling other types of problems, different function sets are typically used. The nodes in the sample tree shown in Figure 2.3 can be replaced by the logical operators $\{AND, OR, NOT\}$

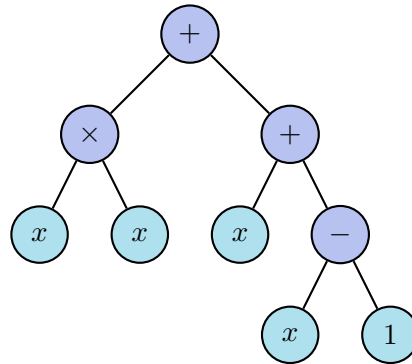


FIGURE 2.3: Example of solution representation for an individual in GP: $x^2 + 2x - 1$.

and terminals replaced with boolean variables $\{x_1, x_2\}$. This produces a representation which is no longer similar to the regression form and has nodes of different arity, unlike the regression tree. The arity of {AND, OR} are both 2 whereas the arity of the {NOT} is 1.

Another requirement on the GP tree is that of closure. This is to ensure that all functions within the tree are capable of returning a result, given any input. As an example, consider division in the regression tree example. A zero provided as the second input to the division operator yields an undefined result. In GP this is not an acceptable result as this renders the entire tree undefined. The solution is that methods should be protected against values which fall outside of their prescribed ranges. In the example of division, protected division where a zero is returned is employed when division by zero occurs. Similar rules are derived for logarithms, square roots, *etc.*

GP requires the function set to have been sufficiently specified in order to model the problem. A common disadvantage of GP is that while many functions can be derived from other functions, having to derive such functions for the problem at hand creates erroneous complexity for the evolutionary process. Making such derived functions available as primitives may significantly improve the performance. In line with the examples already provided, multiplication may be thought of as recursive addition with a constant. The OR function can be derived using combinations of the two remaining functions, but introduces considerable complexity to the tree structures. The natural extension made by Koza [109] in his second edition on GP is that of *Automatically Defined Functions* (ADFs). ADFs attempt to find common structures being derived in trees and attempt to reuse such components in other trees, or as new nodes that can be introduced where required, possibly multiple times within the same tree. The use of ADFs is not considered in this study but would make for an interesting extension or future work.

2.7.3 Fitness

Metaheuristics search in high-dimensional solution spaces for good solutions. The quality of a solution is typically measured by its objective function. The objective function provides a mathematical measure for the quality of the current candidate solution. The objective function is also referred to as the *fitness function* in GAs. There is often no difference in the treatment of these measures except that the fitness function conforms to the nomenclature of an evolutionary process such as survival of the *fittest*. Internally, the GA may transform the problem objective function to a new scaling (or normalisation in order to avoid sensitivity to the problem space), but the nature of this function transformation ensures that pair-wise comparisons are consistent with the original objective function.

The notion of *Raw fitness* (f_r) is used to measure candidate solution quality in terms of the original problem objective with no transformations. *Standardised fitness* (f_s) is used to convert the original problem to a minimisation problem (if it was a maximisation problem before) and sets the optimal result at zero. If the optimal result is unknown, a large constant can be added. The benefit of using standardised fitness is that the relative proximity to the true optimal is represented intuitively.

Adjusted fitness

$$f_a = \frac{1}{1 + f_s} \quad (2.3)$$

transforms the standardised fitness to the domain $(0, 1)$. Adjusted fitness has the benefit of treating small differences as being significant which are common near the end evolutionary run.

LP techniques achieve a close coupling between the optimisation process and the objective function. This provides the ability to inspect a portion of, or decompose, the objective function as part of the decision making process within the algorithm. By contrast, metaheuristics in general (including GAs) do not make any assumptions about the structure of the objective function, *e.g.* linear, non-linear, stochastic or continuous. Metaheuristics are commonly referred to as black-box solvers as a result of this decoupling between the search mechanics and the objective function.

The obfuscation of the objective function details to the conventional GA result in the search mechanics being relatively straight forward and computationally lightweight, as described later in this section. Evaluating the fitness function for an individual is the computationally dominant task in a GA and the number of evaluations made is a common measure of algorithmic cost.

2.7.4 Seeding operators

Seeding operators are the processes in GAs which build the initial population. The process in conventional GAs which employ the bit-wise representation is typically to sample from a uniform distribution and assign the values drawn as the initial values in the individual. GPs follow the same philosophy where the problem function set is the sample space and functions are drawn at random from this set.

The expression tree is initialised by selecting a function at random and then proceeding to browse the tree, sampling at each remaining node in the tree if required, until the tree forms a closed expression. For a large function set, the practical problem of generating a sufficiently compact expression trees arises. This is addressed by using either the *full* or *grow* method [111]. The depth of the expression tree, the shortest path from a root node to a terminal, is used to control the size of the tree by restricting the sample space of operators to the terminal set for any node at the specified maximum depth. The operators sample space can similarly be restricted by excluding terminal nodes to ensure that the expression tree is at its maximum depth to all leaf nodes.

The full method restricts the sample space to function nodes only until the maximum depth is reached, at which point only terminal nodes are provided for selection, ensuring expression closure. The grow method does not place any restrictions on the function sample space for nodes below the maximum depth but restricts sampling to the terminal set at the maximum depth. These two methods produce different types of expression trees and it is commonplace to consider a mixture of both when seeding the initial population, a technique called *ramped half-and-half*, to ensure diversity in the available structures.

It is sometimes possible to generate seeding individuals which are superior to those generated at random (in terms of their *fitness* as alluded to in §2.7.3). A criticism of this approach is that if solution-specific seeds are provided and mixed with random solutions, the random solutions are dominated during the first iteration of the algorithm and produce a population dominated by seeds in the following iteration. The requirement when working with a solution-specific seeded population is that the whole population should be seeded and diverse, *i.e.* all seeds should be able to compete on a similar fitness range so that one solution does not dominate the population.

2.7.5 Selection operators

Determining individuals available for the reproduction phase of a GA is typically performed through a probabilistic approach. The design of the GA supports being able to use different selection operators depending on the situation. Several selection operators are commonly used in practice.

Roulette wheel selection [73] assigns a probability to each individual in the population proportional to the fitness function quality of that individual. The larger the quality of the fitness function (which may mean a smaller value, depending on the context), the larger the probability of selection for reproduction. This method is not without criticism. Due to the close relationship between the fitness and the probability of selection, high-quality individuals can dominate all other solutions in the population if the raw fitness is used as a result of the scale of the function. One way to compensate for this phenomenon is to ensure that the standardised or normalised fitness is used, however; the general criticism that Roulette wheel selection introduces very high selection pressure on the population remains.

Rank selection is typically considered an analogous alternative to Roulette wheel selection where individuals are assigned a rank based on their fitness function quality, and not a probability of selection proportional to the fitness of an individual. Ranks are assigned weights which reduce the selective pressure of high-quality individuals that would be computed by Roulette wheel selection.

Tournament selection operates differently in that all members in the population are not compared with one another. Rather, n individuals (typically two, a larger tournament size increases the selective pressure) are selected without replacement to compete in a tournament at random. The winner of the tournament is the individual with the highest fitness and this individual is transferred to the new generation. Under this scheme, the worst individual in the population will never be carried forward to the new population since it can never win a tournament. Tournament selection is one of the most commonly used selection operators as it avoids comparing the numerical values of a fitness function which can result in premature convergence when there is one member in the population with a more favourable fitness value [150].

2.7.6 Crossover operators

Crossover refers to the process of sharing chromosome structures between two or more individuals to create two new individuals in a new population. The intuition is, if two solutions can combine their structural approaches in some way, that a better solution could be produced as a result of blending two solutions.

The chromosome representation of the conventional GA lends itself to easy-to-implement crossover operators such as a bit-wise swap (with some small probability) or a single-point crossover location [147].

The expression-tree representation used in GP results in the solution structures being less predictable in size and, as a result, other approaches are required when performing crossover [73].

When single-point crossover is performed between two individuals in GP, a node is selected from both trees at random according to the uniform distribution. The nodes selected in each tree are used as the crossover points. The subtrees related to the two nodes are then exchanged. The subtree could consist of a single terminal and would still be a legitimate exchange. It is worth noting that because of the flexible representation of the chromosome in GP, when two identical parents “mate” it is likely two new offspring will be produced. In contrast, generating offspring which have differing chromosomal information in the conventional GA is not possible in crossover. Some authors choose to forgo the classic crossover approach towards combining solution trees in favour of custom operators [63].

Uniform crossover [148] randomly picks the next node from two candidate trees where nodes share a common region. This typically increases the amount of mutation taking place at the root node as more of the two candidate trees will overlap where both are rooted. Other variations on uniform and single-point crossover include context-preserving crossover [46], size-fair crossover [115] and probabilistically restricted crossover [82].

The representation problem in GAs remains a complex first hurdle when modelling an optimisation problem. The degree to which the representation yields statistically independent modification of the solution representation is closely related to the ability for typical operators to function well⁸. The intuition is that if components of the representation are predominantly statistically independent, the number of conditional changes required to obtain a new, potentially better solution state, are small. The orthogonal example is where a single chromosomal or tree change requires specific and unlikely changes to occur in other portions of the chromosome or tree in order to obtain a better solution. The greater the dependency between variables in the representation, the less effective classic operators are as these rely on uniform, unconditional distributional sampling. Examples which support this intuition are numerous and effectively communicated by the bespoke crossover operators for the TSP when using typical list-based (or permutation) representations such as the EAX operator [141].

2.7.7 Mutation operators

Mutation operators in GP alter a tree in some random way in order to potentially find a superior solution. As the mutation operator is defined within the context of a tree, there are multiple types of operations which can be performed against the tree. A few of these operators are:

- subtree mutation [111],
- point mutation [132],
- hoist mutation [103],
- shrink mutation [9],
- permutation mutation [111],
- mutating constants at random [162], and
- mutating constants systematically [90].

⁸This is an observation made by the author based on anecdotal modelling observations with EAs.

The subtree mutation operator selects a node at random (representing some subtree) and replaces the subtree at random using the same process as the initial solution generating methods. Point mutation randomly changes a selected node in the tree to a node of the same arity. Hoist mutation, like subtree mutation, selects a node at random (together with some corresponding subtree) and promotes this subtree to be the candidate solution effectively “hoisting” the subtree to the root node. Shrink mutation replaces a randomly selected node with a terminal node, effectively shrinking the subtree to the selected node. Permutation mutation randomly reorders the arguments of a selected node. This operator makes sense in environments where the commutative property is not present over the arguments. Terminal nodes may be comprised of constants, and the approach of Schoenauer *et al.* [162] is to randomly mutate some of these constants in the tree. The systematic approach taken by Iba *et al.* [90] utilises numerical methods to determine a locally optimal allocation of values to the constants in the tree, however, this method may be less attractive if objective function evaluations are computationally expensive.

Some of these mutation operators have their motivations in attempting to make the solution trees smaller, and in other cases (such as subtree mutation), potentially making the tree larger. Classic crossover in GPs has a high likelihood of producing one larger and one smaller tree after crossover is performed [149], so having a mix of mutation operators which can alter the tree in either direction (smaller or larger) is intuitive. Methods that have used GP for the algorithm selection problem have criticised the degree to which mutation degrades the solution quality and decrease the mutation probability as a result [13].

2.7.8 Memetic algorithms

Memetic Algorithms (MAs) diverge from the conventional GA not in the representation but in the treatment of chromosome changes. MAs use local search heuristics [139] to assist in the global optimisation challenge. The local search heuristics can incorporate domain-specific knowledge or exploit a well-defined neighbourhood. An example of the small adjustment required to the conventional GA Algorithm 2.1 to obtain a MA is shown in Algorithm 2.2.

Algorithm 2.2: Memetic Algorithm

Data: Population Parameters

- 1 $Gen := 0;$
- 2 Create initial population, $P(Gen);$
- 3 Evaluate population $P(Gen);$
- 4 **while** *Termination criterion not met* **do**
- 5 **while** $|P'(Gen)| < |P(Gen)|$ **do**
- 6 $i \leftarrow Select(P(Gen));$
- 7 $i \leftarrow PerformOperator(i);$
- 8 Perform local search on $i;$
- 9 Add i to population $P'(Gen);$
- 10 **end**
- 11 Evaluate population $P'(Gen);$
- 12 $P(Gen + 1) := P'(Gen);$
- 13 $Gen := Gen + 1;$
- 14 **end**

The local search could be an additional metaheuristic (such as SA [36]) or some other local improvement operator applied incrementally throughout other operators being performed [154]. MAs are not strictly within the scope of this thesis, although, it is worth noting the strength

of coupling local search with a larger search process for effective optimisation. The hierarchy of the MA is inverted in this dissertation (Chapter 4) to achieve local search processes being guided by genetic search rather than genetic search being refined through local search.

2.7.9 Multi-objective genetic algorithms

The fitness function discussed in §2.7.3 is a single-objective fitness function in that only one metric is being optimised in the GA. Many real-world problems require measuring more than a single criterion when assessing the quality of a solution. One approximation technique is to use a *simple additive weighting* (SAW) of each of the objectives and condense this to a single, new, objective which can then be optimised in the existing paradigm. The method of SAW receives strong criticism amongst multi-criteria decision making proponents [169] for several good reasons. SAW implies a known explicit trade-off between conflicting objectives *i.e.* willing to forgo a units of dimension x for b units of dimension y . This requires knowledge of the domains of the dimensions being optimised and the users' preference between such dimensions. Secondly, there is an implied assumption of convexity between such objectives when adopting SAW, which may not hold in practice and thus important portions of the search space may be inaccessible.

This is not to say that SAW is not an appropriate method or useful — it is merely to emphasise that care should be taken in the treatment of the objectives, understanding the user utility function for each before a SAW is applied. In fact, there are many instances where multi-objective optimisation is applied but where SAW would have been sufficient to achieve the same outcomes with potentially lower degrees of complexity.

A question that naturally arises when working with multiple objectives is how one should compare one solution with M objectives to another with the same objectives. The Pareto method is typically used to compare two candidate solutions, which results in multiple solutions being considered optimal. Many optimal solutions can be an uncomfortable proposition for OR practitioners, although Deb [50] suggests that a reasonable approach is to treat the problem of generating the Pareto frontier as a separate problem from selecting an optimal solution from the frontier. With this two-step methodology in hand, one does not unnecessarily restrict the search space or bias the search process so as to lose potentially high-quality solutions.

In order to formally introduce the concept of Pareto-optimality, a definition of dominance with respect to a set of m objectives is first provided [50]:

Definition 2.7.1 (Multi-objective optimisation) For set W consisting of m objectives and where S is the feasible set of decision vectors:

$$\begin{aligned} & \text{minimise} && f_m(\mathbf{x}), m \in W \\ & \text{subject to} && \mathbf{x} \in S \end{aligned} \tag{2.4}$$

A dominance relation between two solutions $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ may be defined as follows:

Definition 2.7.2 (Domination) A solution $\mathbf{x}^{(1)}$ dominates a solution $\mathbf{x}^{(2)}$ if both of the following conditions are met:

1. The solution $\mathbf{x}^{(1)}$ is no worse than $\mathbf{x}^{(2)}$ in all objectives, and
2. The solution $\mathbf{x}^{(1)}$ is strictly better than $\mathbf{x}^{(2)}$ in at least one objective.

The notation $\mathbf{x}^{(1)} \preceq \mathbf{x}^{(2)}$ is adopted to indicate that solution 2 is dominated by solution 1 when both of these conditions are met. A situation arises where solutions are presented which do not dominate each other but, in turn, dominate other solutions. Solutions which are non-dominated can hence be separated from those which are dominated.

Definition 2.7.3 (Non-dominated set) Among a set of solutions P , the non-dominated set of solutions P' are those that are not dominated by any member of set P .

Definition 2.7.2 is referred to as the weak-domination criterion. Strong dominance can be defined follows:

Definition 2.7.4 (Strong Domination) A solution $\mathbf{x}^{(1)}$ strongly dominates solution $\mathbf{x}^{(2)}$ if a solution $\mathbf{x}^{(1)}$ is strictly better than $\mathbf{x}^{(2)}$ in all M objectives.

The shorthand notation for strong domination is $\mathbf{x}^{(1)} \prec \mathbf{x}^{(2)}$.

Figure 2.4 illustrates two Pareto frontiers given by the solid black line. The true Pareto frontier is often not known but is approximated by solutions found that form the non-dominated set which are illustrated by the solid black points in Figure 2.4. The property of transitivity holds for the calculation of dominance [50] which can be illustrated with reference to Figure 2.4(a). Both objectives in the figure are minimisation objectives. Considering Solution $\mathbf{x}^{(4)}$, one can see that both f_1 and f_2 are larger values than the values of $\mathbf{x}^{(2)}$, thus $\mathbf{x}^{(2)} \prec \mathbf{x}^{(4)}$. Furthermore, the same can be said for the relation between $\mathbf{x}^{(3)}$ and $\mathbf{x}^{(2)}$, where $\mathbf{x}^{(3)}$ has smaller objective function values in both dimensions than $\mathbf{x}^{(2)}$ where the first condition is met from Definition 2.7.2, thus $\mathbf{x}^{(3)} \prec \mathbf{x}^{(2)}$, but as a result of dominance relation between $\mathbf{x}^{(2)}$ and $\mathbf{x}^{(4)}$, it can also be concluded that $\mathbf{x}^{(3)} \prec \mathbf{x}^{(4)}$.

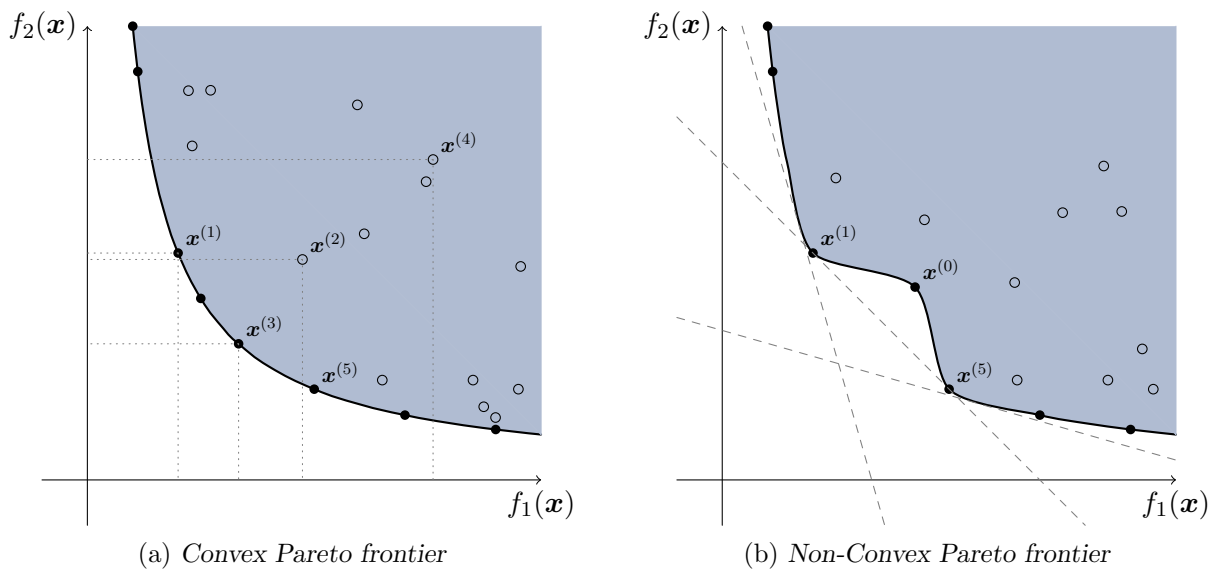


FIGURE 2.4: Convex and non-convex multi-objective Pareto frontiers for two minimisation objectives.

A comparison of $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$ shows that $\mathbf{x}^{(1)}$ does not strongly or weakly dominate $\mathbf{x}^{(2)}$ as $f_2(\mathbf{x}^{(2)}) \leq f_2(\mathbf{x}^{(1)})$ according to condition 1 in Definition 2.7.2, and similarly, $\mathbf{x}^{(2)}$ does not dominate $\mathbf{x}^{(1)}$. Since the non-dominated set may only consist of solutions which are not dominated by any other solution, $\mathbf{x}^{(2)}$ forms part of this as a result of the dominance by $\mathbf{x}^{(3)}$.

It should be clear that a naive implementation to compute the set P' requires at most $O(Mn^2)$ comparisons of all pairs of solutions (n) and objectives (M) to calculate. A more efficient method proposed by Kung *et al.* [113] is $O(n \log(n)^{M-2})$ for $M \geq 4$ and $O(n \log(n))$ otherwise.

Figure 2.4(b) illustrates a non-convex Pareto Frontier which highlights that should one perform simple additive weighting in order to explicitly condense $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ into a single objective function $f_{SAW}(\mathbf{x})$, there would be no possible way to express the non-convex portion of the Pareto frontier as a function of two weightings. A simple geometric analysis reveals that if tangents are drawn to the frontier that there is no weighted combination of $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ which would avoid rejecting solution $\mathbf{x}^{(0)}$ as being inferior to $\mathbf{x}^{(1)}$ or $\mathbf{x}^{(5)}$ in the SAW projection ($f_{SAW}(x)$).

The primary change required to modify the classic GA to cater for multiple objectives is in the selection of population members. The selection process requires comparing individuals in the population in order to determine which population members of high quality should be selected for breeding.

A highly successful approach was proposed by Deb *et al.* [51], called NSGAI (Non-dominated Sorting Genetic Algorithm II). The approach not only addresses the computational complexity of other multi-objective GA approaches, such as that in [166] by the same authors, but is also capable of retaining diversity in the frontier without explicit parametrisation, ensuring that the Pareto frontier is adequately explored.

Algorithm 2.3 outlines the process of determining a nondominated sort for a population of candidate solutions that have been evaluated. In order to ensure diversity in the selected population, Deb *et al.* [51] proposed a crowding distance calculation to bias selecting solutions which are lower in density around the estimated Pareto Frontier. More formally, if two solutions have the equivalent rank as a result of Algorithm 2.3, the solution with the lower crowding distance is selected. Deb *et al.* demonstrated that this algorithm significantly outperforms previous methods, SPEA [193] (strength-Pareto EA) and PAES [106] (Pareto-archived evolution strategy), when estimating the true Pareto frontier.

Algorithm 2.3: Fast-non-dominated-sort (P)

```

1  foreach  $p \in P$  do
2     $S_p = \emptyset$ 
3     $n_p = 0$ 
4    foreach  $q \in P$  do
5      if  $p \prec q$  then
6         $S_p = S_p \cup \{q\}$ 
7      else if  $q \prec p$  then
8         $n_p = n_p + 1$ 
9      end
10     if  $n_p = 0$  then
11        $p_{rank} = 1$ 
12        $\mathcal{F}_1 = \mathcal{F}_1 \cup \{p\}$ 
13     end
14  end
15   $i = 1$ 
16  while  $\mathcal{F}_i \neq \emptyset$  do
17     $Q = \emptyset$ 
18    foreach  $p \in \mathcal{F}_i$  do
19      foreach  $q \in S_p$  do
20         $n_q = n_q - 1$ 
21        if  $n_q = 0$  then
22           $q_{rank} = i + 1$ 
23           $Q = Q \cup \{q\}$ 
24        end
25      end
26     $i = i + 1$ 
27     $\mathcal{F}_i = Q$ 
28  end

```

2.8 Automatic algorithm selection

The *algorithm selection problem* (ASP) is one where a decision should be made as to which algorithm to use in order to solve a particular CSP. Two primary approaches emerge in ASP. The first approach is where low-level CSP algorithms such as propagators, *nogoods*, backtracking schemes and variable-value heuristics are selected to solve a particular CSP. The second approach is to rather select from a series of predefined algorithms which have been shown to work well on different classes of problems in that they exploit some structural feature of the underlying CSP which does not generalise across all CSPs. A third approach which could be considered is rather to select across different solvers for a given CSP. However, it is logical to simply classify these solvers as highly sophisticated algorithms, at which point they can be considered part of the latter categorisation, typically referred to portfolio search.

A common paradigm used to frame portfolio search is to consider the runtime of a search routine used as a random variable. The basis for this approach is that the majority of successful search methodologies incorporate an element of randomness at some level in the algorithm, as discussed in §2.4 and §2.5. As the runtime of the search algorithm is stochastic, one may characterise the mean and variance of the runtime of algorithms used. Rather than attempting to tailor algorithms directly to a problem, portfolio search utilises existing algorithmic approaches, but selects algorithms to run according to certain metrics which describe their potential success in solving the underlying CSP.

A simple introductory example of a portfolio search routine is one where there is only one algorithm in the portfolio [77]. This sole algorithm would then be selected by the portfolio search with some constraints on the runtime of the selected algorithm, attempting to solve the problem with the same algorithm multiple times. This is equivalent to defining a restart policy on the selected algorithm which has already been shown to be a favourable approach when the underlying search routine is stochastic, as discussed in §2.4.

Portfolio search [87] uses a collection of algorithms which may be copies of the same algorithm with different parameters or running on different processors [77]. Parallel search in portfolio-based optimisation is popular as the underlying algorithms used to perform the search are independent and require no communication of search metrics between one another. As the underlying search routine is stochastic, one may take an economic view by treating the variance of the process as a measure of the underlying risk [76] of the algorithm. This economic view allows one to model the *efficient frontier* when attempting to maximise the expected returns of the process. In this financial model, there can be no reward without some element of risk, and depending on the target returns, one will take on additional risks (at the cost of variability in the returns) to maximise the expected return. Gomes *et al.* [77] provide a treatment of the probabilistic distributions required for this type of portfolio selection and noted that the optimal portfolio design is sensitive to the estimates of the true underlying runtime distributions.

Other approaches to portfolio optimisation focus rather on “synthesizing” [135] an algorithm specialised to the task of selecting from a corpus of algorithms at the propagation and search-heuristic levels, called Multi-TAC. This approach falls more in the category of the algorithm selection problem, although there is a clear overlap between the designation of these two disciplines. Techniques which predominantly focus on metaheuristic-based approaches are discussed in §2.8.1. The philosophical approach of Minton *et al.* [135] is very much in line with the objectives of this dissertation as they train the model on a subset of typical examples that are representative of future scheduling tasks, however, the implementation and techniques used differ significantly, primarily in that Minton *et al.* develop a problem-specific grammar for the Minimal Maximal Matching problem to provide domain-specific information.

In contrast to the approach employed by Minton *et al.* [135], the objective in this dissertation is to use domain-specific information without modification of the underlying grammar.

Epstein *et al.* [57] present an adaptive learning process for solving CSPs, called ACE (adaptive constraint engine). The authors structure a hierarchical learning model based on the features of the CSP referred to as *concerns* in their work, as well as the runtime performance of the provided portfolio of algorithms. The control algorithm uses a weighted voting scheme to promote different algorithms to solve the CSP within the advisory scheme presented. One can draw an analogy between the learning process employed here and FDS where the failures of each heuristic are recorded and later used to recompute the voting state of new possible heuristics to present as the search strategy. Their empirical results back the theoretical findings that the value selection strategy of the minimum domain divided by the degree of a variable is typically a useful strategy for assigning values. Epstein *et al.* [57] found that when this operator was, however, removed from the available set of operators and the algorithm was permitted to learn its own advisors, it was found that there was no statistically significant difference between the performance of the algorithms. This suggests that there exist other near-equivalent strategies which can compensate for the lack of this operator as the product of domains appeared to be the dominant replacement strategy for the value ranking.

Three interesting portfolio-based search optimisers which have achieved success over the years are CPHYDRA [144], SATZILLA⁹ [187] and SUNNY-CP [7, 6]. These types of portfolio solvers require some kind of *machine-learning* (ML) technique in order to map new instances to previously seen instances in the prediction of algorithm performance across the available algorithms in the respective portfolios [108]. There are many permutations of this approach, including online and offline selection as well as dynamically varying heuristic selection during a search. It is common for these predictive techniques to use dynamic feature selection based on either problem specifics or domain-independent attributes [107]¹⁰.

Kotthoff *et al.* [108] performed a comprehensive analysis of many ML techniques and their performance in selecting a single algorithm from a portfolio of over 2000 data sets. The results of the study showed that most ML techniques were unable to outperform the majority classifier which is in line with expectations in view of the “no free lunch” theorem [185]. Another interesting finding is that most existing machine learning systems’ performance is not as good as previously thought. The authors were able to establish the latter finding by including tailored ML CSP classifiers in the analysis against a host of black-box ML techniques. It is worth noting that neural networks were not among the black-box classifiers used.

Xu *et al.* [187] used ridge regression [29] in SATzilla to predict the runtime of all algorithms in the corpus based on the CSP graph and historical runtime information for training cases. SATzilla performed well in the 2007 International SAT competition even though a single algorithm is used to solve each problem instance without changing the solver during the run. The authors noted that the significant advantage of SATzilla was that an algorithm was selected for each problem instance, rather than one algorithm per problem class as was the commonly accepted methodology at the time. Malitsky *et al.* [129] extended the approach used in SATzilla based on the observations in [130] by devising a cost-sensitive hierarchical clustering method for selecting an appropriate solver which addresses the computational workload of the training process used by the 2012 version of the SATzilla solver. SATzilla was later modified to use a weighted random forest approach in its prediction engine, yielding a win result in the sequential portfolio track in the 2012 SAT Challenge.

⁹While not technically a CP solver, it is possible to translate any problem expressed in CP into SAT using languages such as MiniZinc [93].

¹⁰See, for example, <http://4c.ucc.ie/~larsko/assurvey/>.

O’Mahony *et al.* [144] presented a successful application (CPHYDRA) of case-based reasoning for solving the algorithm selection problem at a higher level by modelling the selection of solvers rather than specific algorithms to solve a specified CSP. Case-based reasoning uses a database of previously analysed problems [1] and is an existing approach towards encapsulating expert experience in solving CSPs [121].

CPHYDRA uses feature extraction on the problem graph to derive 36 graph features which were used to determine the solver required for the problem such as node arity, the ratio of extensible or intentional to global constraints, *etc.*. The system developed is superior to other case-based reasoning systems in that it not only selects the solver, but also allocates the amount of time to be allocated to that solver such that the overall likelihood of success is maximised. This approach was shown to be successful in the 2008 CSP Solver Competition as CPHYDRA was able to outperform all individual solvers by selecting and scheduling solvers intelligently for all problems. Galiolo *et al.* [64, 65] framed the problem of selecting multiple algorithms and the time budget allocated to each in portfolio optimisation as a *time allocation* problem. The formulation presented samples different algorithms and updates the relative priority of other algorithms in the portfolio based on the runtime information gained during successive trials of an algorithm applied to a problem instance.

The solver sunny-cp uses a very simple algorithmic approach rather effectively. The problem of matching algorithms to problem instances is performed through a *k-nearest neighbour* (KNN) approach based on historical models solved. The algorithm creates a smallest sub-portfolio able to solve the maximum number of instances in the neighbourhood [8] with the time allocated to each item in the portfolio being proportionate to the number of instances solved. A large differentiating feature of sunny-cp and other portfolio solvers is that no explicit model (such as a regression model) is required to be built offline because of the KNN approach towards defining an appropriate portfolio. The authors of sunny-cp have also open-sourced¹¹ their implementation of the feature extraction algorithms used by the solver [5]. The same graph features are also used for consistency when testing Hypothesis 3.1.3 of this dissertation in Chapter 3.

It is worth noting that sunny-cp performed very well in the “open” solver category by taking Gold in the MiniZinc Challenge [171] during the period 2015–2017 and second place in 2018, losing to or-tools in the open category¹². Or-tools took Gold in four of the five competition categories in 2018. One of the primary reasons for this change in the performance of or-tools in 2018 was the move to a SAT-based solver over the CP solver. One could argue that should sunny-cp incorporate or-tools in its portfolio for 2019, it should win the open category again unless the or-tools algorithm takes yet another leap forward.

Other methodologies which extend the ASP to different domains include the work of Cauwet *et al.* [40] which consider the problem of modelling noisy optimisation processes using a minimum regret heuristic to rank which solvers and variable-selection strategies to use when allocating time budgets to each. Many portfolio approaches utilise a theoretical model characterising the expected risk of selecting an algorithm from the portfolio by maximising some return, minimising the expected runtime to completion, or minimising the expected regret.

A noteworthy alternate approach is that adopted by Peng and Tang *et al.* [145, 173] where a population-based method is used to explore the portfolios based on similar economic metrics to those already discussed. Peng *et al.* state that “*Unfortunately, no formal measurable definition of an algorithm’s risk on a set of different problems is available so far, not to mention any*

¹¹<https://github.com/CP-Unibo/mzn2feat>

¹²<https://www.minizinc.org/challenge.html>

effort on reducing the risk." [145] which, given the metrics discussed in this review of portfolio algorithms, was possibly an oversight. This is not an unreasonable oversight given that often research communities working on evolutionary algorithms are disconnected from their exact counterparts in MIP, CP and SAT. This should not detract from the approach employed which is further refined in [173], demonstrating that using evolutionary processes to solve the time budgeting problem is a viable approach. It is worth mentioning that the authors were considering the problem of determining which subset of evolutionary-based algorithms should be used and to what degree for a given problem which is conceptually the same problem being solved by modern portfolio solvers but applied to a different operator domain using a different rank-and-allocate routine.

2.8.1 Evolutionary approaches to the ASP

CSPs have proven to be a rich ground for the development and testing of new graph heuristics. The rigid specification of the framework provides a clear delegation of tasks in the process of searching for feasible solutions. There are many different ways of matching available heuristics to CSPs, each with their own benefits and shortcomings. One criticism of matching known heuristics to a CSP is that it presumes that an appropriate heuristic exists for the problem being solved, a "disingenuous" [12] approach.

The approaches adopted by [57] and [135], described in the previous section, are examples of developing new algorithms based on some portfolio of primitive structures. The work of Fukunaga [63] follows in the same vein where a system called CLASS (Composite heuristic Learning Algorithm for SAT Search) is developed to define new heuristics based on a simple population-based composition operator. Notably, s-expressions are used to define the heuristics developed. This is an interesting novelty as it opens up the search space to a complete grammar comprised of 'if' 'else' statements providing a high degree of control over the heuristics which can be developed. One could argue that this kind of fine-grained control may result in overfitting to specific problem characteristics of the underlying CSP, but the authors separated a test set from a training set which allowed them to verify some degree of generalisation to the unseen problems.

The authors found, when problem types from different classes were tested on the heuristics developed, that the performance degraded significantly for 50% of the heuristics. This result is often found when empirically evaluating search strategies in that the extensibility of strategies to problems with different fundamental characteristics in the CSP degrade the performance as, somewhat unsurprisingly, there is no free lunch [185]. Fukunaga [63] conducted an interesting analysis of the characteristics, described by Schuurmans *et al.* [163], of the search strategies developed in terms of their depth (number of remaining unassigned variables), mobility (how rapidly a search moves through the space) and coverage (how systematically a search is conducted). The analysis revealed that the search algorithms behaved well according to the metrics defined by Schuurmans *et al.* [163] in that they achieved a low depth (many variables assigned), high mobility (quick decision making) and high coverage (were systematic).

Bain *et al.* [13] attempted to address ASP by using GP to evolve expression trees which solve the specified CSPs. They compared the performance of GP to the beam search approach adopted in [135] and random search. The authors advocated the usage of the GP based on its flexibility to adapt to other unknown problem spaces and noted that previous conclusions drawn regarding the ability for GP to conclude meaningful search strategies were premature.

The study conducted by Bain *et al.* [13] was restricted to four classes of MAX-SAT problems which enabled the authors to focus on comparisons in backtracks performed and execution time thereby simplifying the question of whether one algorithmic approach dominated another.

The work of Bain *et al.* [13] bears the closest resemblance to the approach employed in Hypotheses 3.1.1 and 3.1.2 in that the GP is provided a grammar which is algebraic in nature, but with the exception of using the *Maximum Occurrences in clauses of the Minimum Size* (MOMS) [186] heuristic for variable selection in every tree. This decision was made as a result of up-front analysis of the effect of the MOMS heuristic in the solver being modified. Another significant difference is that the grammar used by the authors is based on information, such as clause weights or move rules, which are available in SAT and not necessarily CP solvers. For this reason, the grammar presented later in this dissertation is tailored to the CP structures. Other departures from the methodology adopted in [13] are discussed in the methodology chapter.

A more recent approach to the problem of automatic algorithm development has been to evolve both the constructive heuristic and the local search heuristic in tandem [38]. The methodology adopted is that of a grammar-based evolutionary model with a focus on designing good search spaces which lead to good heuristic definitions as opposed to the heuristics themselves. The grammatical evolution system presented can be defined as a hyper-heuristic as it searches the “spaces of local search heuristics” and not the solution space of the problem itself [38].

Burke *et al.* [38] present a thorough account of the work done in the bin packing arena with respect to evolutionary processes and heuristic design. The test results returned by the local search heuristics developed in respect of the multiple 1-D bin packing problem seem very promising in terms of quality of results. It should be noted, however, that the authors did not cast the problem as a CSP and as a result were able generate more natural expressions for local search operations tailored to the problem domain.

2.9 Static CSP feature extraction

Approaches used to solve the ASP which use features of the graph have been discussed in this review [6, 57, 108, 144, 187], however, it is pertinent to highlight some of general features extracted in these approaches and mention some of the additional features used in this particular study and their source.

The authors of sunny-cp provided a comprehensive set of features of the CSP used when predicting the performance of a candidate algorithm [5]. It is useful to cast the constraint network described in Definition 2.2.2 into a classic graph representation: $G = (V, E)$, where the graph G comprises undirected edges E between pairs of vertices in a set V . The application of Definition 2.2.2 in this context is straight forward in that variables and constraints are the vertices of the graph, the relationships between the variables and constraints are represented as edges.

Each node in the Figure 2.5 represents either a variable or constraint. An example of a constraint node is simply a condition which is required to be met between a pair of variables, such as $x_i \neq x_j$ would result in three nodes, two of which are variable nodes (x_i, x_j) and one constraint node (C_{\neq}), with two edges, (x_i, C_{\neq}) (C_{\neq}, x_j). All nodes represent either variable-value assignments to be performed by the solver, or required constraint checks between such assigned values. This is a convenient representation as some constraints may be conditioning on the outcome of other constraints, simply meaning that there will be edges between constraint vertices in the graph.

Using gravity-based layout models one can obtain an approximate sense of how constraints interact with one another in a handful of selected CSP instances in Figure 2.5. Aside from

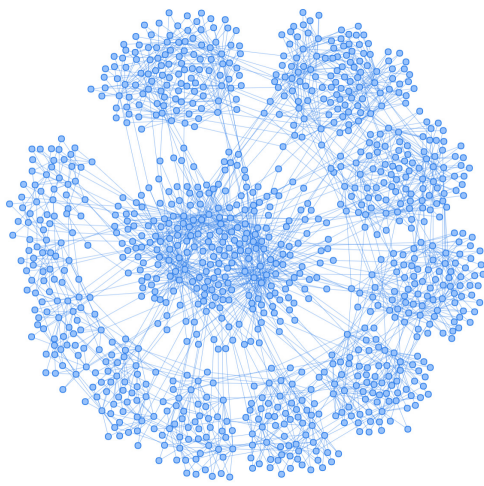
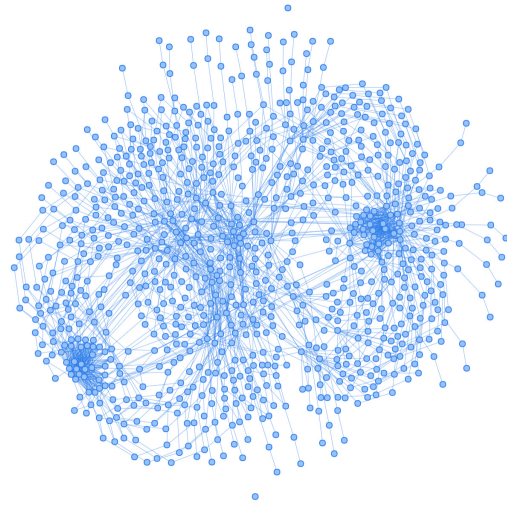
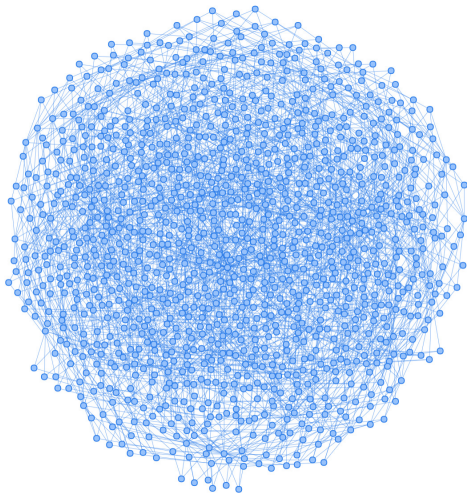
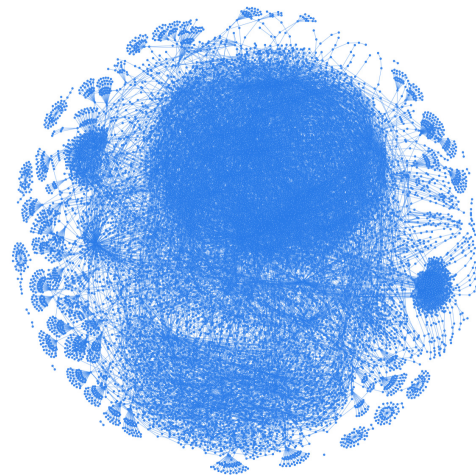
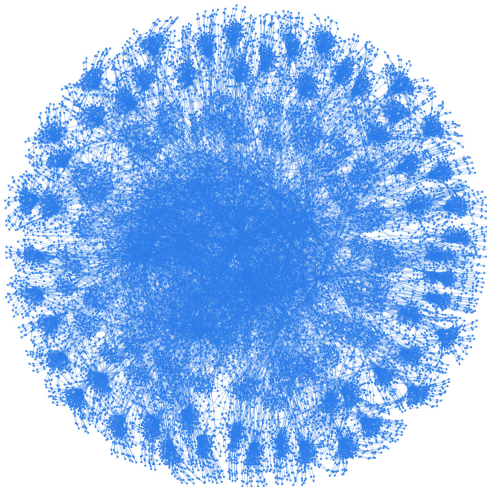
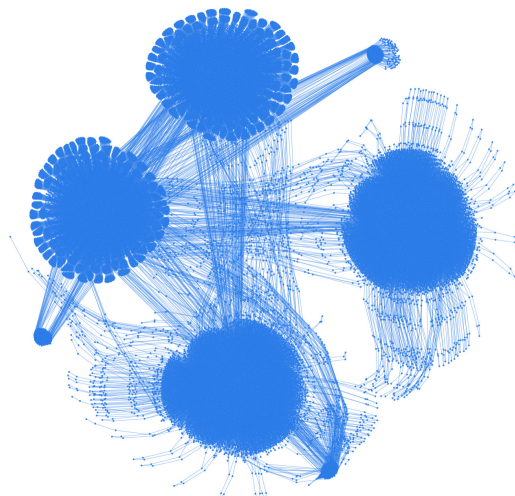
(a) *costas-array (16)*(b) *tdtsp (10,24,10)*(c) *grid coloring (10,5)*(d) *is (1YHXeG1xYs)*(e) *p1f (12)*(f) *cvrp (A-n37-k5.vrp)*

FIGURE 2.5: CSP plots for a small subset of MiniZinc benchmark instances.

these graphs being quite enchanting to look at in some cases, they also highlight the staggering complexity at which the underlying algorithms are operating and the level of reasoning required in order to make appropriate decisions in such complex and varied graphs. In the interest of completeness, a high-level graph description of Figure 2.5 is provided in Table 2.1. A distinction is made in the table between vertices which are represent variables (native and extracted) and vertices which represent pure constraint vertices. An extracted variable (or auxiliary variable) is a derived variable which is based on one or more other variables. As can be seen in the table, there are 136 variable vertices for the costas-array (16) problem, although, there are, in fact, 16 explicit variables in the MiniZinc formulation with the balance of the variables (120) being extracted variables in the translation to the FlatZinc format. The FlatZinc formulation remains equivalent to the MiniZinc formulation.

Problem class	Instance	# Variable Vertices	# Constraint vertices	# Edges
costas array	16	136	892	2,086
tdtsp	10,24,10	357	594	1,823
grid coloring	10,5	376	825	2,875
is	1YHXeG1xYs	5,997	7,875	19,615
p1f	12	6,259	14,793	39,181
cvrp	A-n37-k5.vrp	1,791	23,667	96,205

TABLE 2.1: Description of graphs shown in Figure 2.5.

One can consider that basic properties of the graph such as the number of variables, constraints and edges are top-level summary statistics of the graph that have had no aggregation performed over their underlying property. Where one considers summarising properties of a vertex in the graph, as opposed to the graph itself, but wishes to present the data at the graph level, some level of aggregation will need to be applied. The approach adopted by Amadini *et al.* [5] was to use summary statistics to report features at the highest level. An an example, one may consider the degree of each vertex in the graph, *i.e.* how many edges are associated with a particular vertex. Some vertices may have few edges, some potentially many, and so the motivation is to measure the mean and the variance in such an instance in order to understand both the average value as well as the variation of values. One could argue for higher moments of the distribution (skewness and kurtosis), but the merits of this has not yet been considered in the literature. Simpler measures, such as the minimum and maximum values have been used as well as more complex measures like entropy.

A complete description of the static features extracted by MZN2FEAT can be found in Amadini *et al.* [5]. The general principle is to measure the cardinality of objects in the context in which they exist. To illustrate: variables have a domain size, and so the min, max, average, variance and entropy of all variable domains are provided as features. For constraints, the degree is defined as the product of the two variable domains of a constraint. The min, max, average *etc.* are then also computable for the degree. Ratios have also been computed, presumably to normalise certain metrics to their domain context, such as the number of boolean variables in the model, and the ratio of the number of boolean variables to the total number of variables. The intuition behind several of these ratios is not clear as most ML processes perform a level of global normalisation across the data set *i.e.* normalised features across all problem instances, or rows, of the data set. It is also possible to re-create information contained in many of the ratio columns through other columns in the data set, which for logistic ridge regression (a method used by the authors in their first ML classifier) is useful, but not for more sophisticated methods such as neural networks. The authors extracted 144 static features from the CSP in total.

Feature	Typical complexity	Description
Closeness	$O(nm)$	How easily one vertex can be reached by other vertices
Betweenness	$O(nm)$	The number of geodesics passing through a vertex
Pagerank [35]	$O(m)^a$	The probabilistic weighting of the importance of vertices
Burt's constraint scores [39]	$O(n + nd^2)$	A score for each vertex, also known as structural holes
Strength	$O(n + m)$	In an unweighted network it is the degree of a vertex
Eigenvector Centrality [142]	$O(n + m)$	A measure of the importance of a vertex in the network
Kleinberg's hub scores [105]	$O(n)$	A measure of the importance of a vertex in the network (the principal eigenvector of $A \cdot A^T$)
Kleinberg's authority scores [105]	$O(n)$	Scores vertices based on how "authoritative" they are perceived to be (principal eigenvector of $A^T \cdot A$)
Transitivity score [15]	$O(nd^2)$	The transitivity is defined as the sum of the triangulation edge weight normalised by the strength and degree of the vertex
Eccentricity [80]	$O(n(n + m))$	The maximum shortest distance from all other vertices to the current vertex
K-Neighbourhood size	$O(ndk)$	The size of the neighbourhood of vertices within a distance of k of a given vertex ^b
Community infomap [158]	$O(n(n + m))$ [192]	Uses an information-flow approach to group vertices into approximate clusters

TABLE 2.2: Summary of vertex-level properties considered.

^aSome implementations cite $O(n + m)$ but the implementation used dominates with $O(m)$.^bThe order of the neighbourhood, k , is an input parameter.

The motivation for describing the graph at the CSP level is that this works well for portfolio search where control of the lower algorithms has already been predetermined by the member in the portfolio. It is proposed later in this dissertation that the feature set be extended to include properties of the graph at the vertex level for the purpose of deriving potentially useful variable and value selection strategies. The *igraph* package [45] may be used to compute the features of the graph at the vertex level. Vertex features used are summarised in Table 2.9, where n is the number of vertices, m is the number of edges, A is the adjacency matrix, A^T is the transpose of the adjacency matrix, d is average domain size of a vertex *i.e.* the number of edges in which the vertex participates, and k is the order of the neighbourhood.

No differentiation is made for the computational complexity of computing a feature for a subset of the vertices as they will always be required for all vertices. Some of the algorithms in Table 2.9 have a lower expected running time when computing over subsets of vertices, while others have the same running time, as all vertices need to be visited in order to complete the calculation of the metric.

2.10 Deep learning

ML is a broad topic in statistical learning which leverages multiple techniques in order to solve “learning” tasks. There are three main categories within ML, namely; supervised learning, unsupervised learning and reinforcement learning. In each of these three categories, the algorithm is said to *learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T , as measured by P , improves with experience E* [136]. In the context of ML, the experience refers to the data provided to the algorithm, or *training* data. The task refers to the concept being predicted and the performance refers to the quality of the prediction in the context of the algorithm being used. Supervised learning requires complete observations which demonstrate both the input (E) and output (T), known *a priori*. The most common example of this type of learning is linear regression where linear coefficients are fitted to minimise the *mean squared error* (MSE) of the expected output. Linear regression can be modified in a natural way using a sigmoid function to accommodate categorical data such as labelled output data.

Colloquially, one often refers to linear regression models simply as regression models, with the linearity being understood or with non-linear polynomials being introduced as variables in the regression, allowing it to remain linear in the solution process. Regression models have been a mainstay in the statistician’s toolbox for more than 200 years with the first recorded application in 1805 [188]. There are several practical benefits to using a statistical technique such as regression. The coefficients of variables in the fitted model are directly interpretable, making relationships or interactions relatively easy to uncover. Regression models can provide statistical significances of the variables and the overall model fit such that a consumer can often identify potential overfitting or key variables in a model. The assumption of normality in the error terms also allows for a mathematically closed form solution procedure to find optimal coefficients for a set of data, however, this can be a double-edged sword. The solution procedure to the closed form of a regression model requires computing the matrix inverse, currently an $O(n^{2.373})$ operation [184]. The rise of big-data, particularly within larger organisations, has led to an increase in the use of approximation methods to solve for the values of parameters in such regression models. The interpretability of the parameters remains, but statistical confidence intervals around parameter values fall away under such approximation techniques.

Classification tasks are supervised learning tasks where the response variable is non-continuous; such as a binary, categorical or ordinal variable. Regression techniques can be naturally extended to accommodate predicting such discrete outputs through a simple transformation to the response variable using functions such as the sigmoid function.

Unsupervised learning tasks are concerned with separating the experience (or data) with respect to some metric. A classic example of unsupervised learning is k -means clustering [83] where the algorithm attempts to find a set of k clusters which minimise the MSE to the centroid of each cluster, the mean point of a cluster, using a defined distance measurement. Once clusters have been found for training data, unseen points can then be assigned to clusters based on which cluster they are closest to in terms of the mean. This allows one to predict a cluster assignment for new points without having been trained explicitly on an existing clustering. There are many unsupervised learning techniques, each with a series of assumptions as to the structure of the domain and what defines a “good” cluster. The onus is on the practitioner to establish which assumptions and metrics are reasonable and preferred for the problem in question.

Reinforcement learning [172] (RL) is a more recent addition to the learning toolbox. A key difference in RL with respect to supervised and unsupervised learning is that this type of learning often occurs in an environment with a set of defined inputs (perception) and allowable out-

puts (action) [96] where an *agent* is required to determine an action based on the state (the agent behaviour) for which it will receive either a reward or a penalty. Posing an RL problem environment results in there being no pre-defined input and/or output data, which is a key differentiator requiring a different solution methodology to those tailored for supervised and unsupervised problems. A complete treatment of solution methodologies for RL environments is outside of the scope of this work. The key approaches, however, centre around Q-learning [183], *Temporal Difference Learning* (TD-Learning) [174] and *Monte Carlo Tree Search* (MCTS) [37].

RL is often applied to environments in which agents are required to play a game where the learning process requires developing a policy for decision making in the game. This analogy is similar to heuristic policies used to solve CSPs (discussed in §2.5). The CSP can be considered a game environment where the objective is to find a solution to the problem through some policy whereby variable and value selection decisions form the action space and the domain information forms the state space. As GPs generate decision trees, or a policy in an RL context, they have also been applied to solving RL-centric problems. The policy generation approach used by Q-learning, TD-learning and MCTS are statistical in nature as opposed to the metaheuristic approach employed by GP.

Deep learning is a term used to describe multi-layered neural networks where the emphasis lies in multiple layers of transformation of the input data, allowing for abstract concepts or hierarchical structures to be learned in the model. The word “deep” directly describes the layering scheme, or network architecture, employed by the neural network and encapsulates multiple network architectures including *Recurrence* (RNN) [133], *Convolution* (CNN)[112] and *Long Short-Term Memory* (LSTM) [85]. The primary application area for these techniques is in language modelling and image classification.

At the core of all types of machine learning lies an optimisation problem. The approach employed to solve the underlying optimisation problem differs depending on the formulation of the objective and the presentation of the variables. Optimisation techniques used in regression are not useful for solving problems such as *k*-means clustering, which require domain-specific heuristics. Similarly, techniques used in RL have little overlap with supervised and unsupervised learning. Highly successful optimisation approaches tend to utilise gradient-based descent procedures to ensure that search effort results in moving towards local minima. Neural networks are no exception in this regard, with *Backpropagation* (BP) [120] providing the computation of the gradient and *Stochastic Gradient Descent* (SGD) [31] as the algorithmic process being used to guide the descent process, increasing the probability of obtaining solutions at, or close to, the global optimum. These two processes form the backbone of the majority of advanced neural network implementations with research effort being focussed on how to deliver representative gradient updates over the variables in more complex network architectures.

There have been criticisms [118] of using SGD in convoluted neural network training due to the learning rate being known *a priori* in the SGD parameter tuning. The state-of-the-art learning over large data sets in parallel quickly demonstrated that SGD is indeed an acceptable procedure [43] where more than 1 billion variables were used to fit a model. This is a landmark achievement which demonstrates two key points: 1) the scalability of the descent methodology and 2) that tailoring the computational hardware to the algorithmic process can result in orders of magnitude speed-ups. Most would argue that performing training for neural networks on *Graphics Processing Units* (GPUs) is commonplace. Coates *et al.* [43] were able to demonstrate the power of this approach over traditional *Central Processing Unit* (CPU) methods at the limit of computability on equivalent model representations — requiring 2% of the hardware and a fraction of the training time required by Dean *et al.* [49]. This milestone result sparked renewed

interest in the use of neural networks for large ML tasks as well as increased sophistication in open-source software such as TensorFlow [3], Theano [175] and Torch [44].

Interesting applications of deep learning are abundant, with the network architecture often being highly tuned to the specific domain in which researchers are working. In the specific context of using deep learning techniques to solve CSPs, a specific line of research stands out, namely that of Loreggia *et al.* [124]. In this work, researchers cast the problem of describing the features of a CSP as an image and use a CNN to predict which solvers will perform well on the CSP. Loreggia *et al.* were able to design a portfolio approach to select an underlying solver for a particular CSP that performs well in general, although some CNN misclassification resulted in lower performance for some problems. The task in this context is to generalise the core features so as to ensure reduced misclassification *i.e.* selecting an inferior solver for certain CSPs, and it is unclear to what degree this methodology extends to unseen problem instances outside of the training sets. This however, does not discount the effectiveness of the approach used. Another novelty is that features need not be directly extracted from the CSP graph and are rather extracted from a two-dimensional projection of variable-clause participation. The projection used circumvents a graph analysis of the resulting CSP. A possible criticism of this approach is that one is now heavily reliant on the ML procedure to infer higher-dimensional relationships based on the image, whereas some relationships may already be known which could have been analysed and provided *a priori* to the learning process. Loreggia *et al.* avoided complicating the learning method and network architecture by keeping the input data as square images of the variable relations.

As an aside, Loreggia *et al.* [124] made a good argument for the use of more bespoke solvers to specific problem domains, as opposed to attempting to design universally good solvers as there is no free lunch [185]. The goal of Loreggia *et al.* was to derive an automated process for selecting the right solver (or algorithm) for the CSP at hand. This is an argument and approach thoroughly supported by the author, and forms part of the motivation for the line of research contained in this dissertation.

It is the authors personal view on statistical learning over recent years that there appears to have been a shift away from traditional statistical methods towards methods which do not focus on the ‘why’ but on the delivery of a model that simply works well. Vast models can be created by employing approximation techniques which are able to deliver accurate predictions but which obfuscate the underlying reasoning behind the model. This has also led to an increase of solution approaches in recent years which attempt to answer the question of ‘why’ retrospectively by interrogating the fitted model, referred to as explainer processes such as LIME [157] and SHAP [170]. This is not to discount the value of approximation techniques, but rather to highlight that there is often an incentive to create models that achieve great predictive accuracy but which may, in fact, be using information to achieve this which may be considered disingenuous. Several examples of models using information that does not generalise as desired are available in image recognition where the terrain is used to classify the type of animal in an image, rather than the animal itself, or where the focal point of the image is used to determine whether there is an animal present in the image.

2.11 Cooperative applications

The integration of CP and metaheuristics has been successfully implemented [101] using ACO. One may however question the usage of the word “integration” in this context since CP remains decoupled from the ACO procedure and is used to manage the search, propagation, and domain

filtering of variables efficiently, as opposed to having knowledge of the ACO algorithm. This does not limit the fast algorithmic benefits of CP which are being leveraged as a competent discrete optimisation engine in this context.

ACO has shown its prowess in providing good solutions to TSPs [55, 56] and hence are popular in domains where permutations of solutions are modelled. The integration with CP in [101] is very similar in that ants select a permutation of variables and value assignments. CP is used to validate the node selection and to constrain the remaining domains as the algorithm progresses. The objective of the ACO is to maximise the number of assignments to variables, which is also the goal in a normal CP branching scheme. The ACO run terminates when all variables have been assigned values or a pre-specified time limit is reached, with the problem remaining unsolved.

The author notes that this approach is very similar in principle to the work done in [20] as a custom graph validation engine was designed to handle the propagation of constraints with the metaheuristic, in this case a GA, handling the permutation of variables in the branching scheme.

Several applications and extensions to the ACO-CP pairing have followed subsequent to the work in [101], a recent example being the work done on the bicycle sharing problem in [54]. Di Gaspero *et al.* used the VRP formulation to model the bicycle sharing problem. Although one may be tempted to criticize this formulation of the problem as it can be cast more simply as a multi-commodity network flow problem with a minimum arc flow cost equation [4], yielding an LP with strong lower bounds on the objective, Di Gaspero *et al.* did show that the ACO algorithm guiding the CP search outperforms the standard CP solver.

A final criticism of this line of work is that each problem instance is dealt with independently of the next and the ACO is provided problem-specific variable and value information. In this regard the permutations generated are instance-specific and not class or problem-specific which renders any information gained in a single ACO run lost to other problem instances. Given the structural design of ACO, it seems unlikely that this metaheuristic can be extended to produce more generic reusable solutions. At this point, the problem could be viewed as tuning a search strategy and the CP engine could be replaced with an MIP engine to determine variable branching priorities. This once again highlights that universal algorithms will always have additional performance waiting to be unlocked, especially if strategies are being tuned to a problem instance and not to a problem domain.

CHAPTER 3

Research Hypotheses

Contents

3.1	Research motivation	43
	3.1.1 <i>Hypothesis 1: Evolutionary CSP augmentation</i>	47
	3.1.2 <i>Hypothesis 2: Evolutionary meta-CSP augmentation</i>	48
	3.1.3 <i>Hypothesis 3: Deep learning for algorithm selection</i>	49
3.2	Hypothesis testing considerations	50
3.3	Test data	52
3.4	Benchmark strategy	55
3.5	Implementation notes	56

The purpose of this chapter is provide context to the research hypotheses. The aim of the respective hypothesis and testing methodology are provided. Potential shortcomings of the methodological design of each hypothesis are discussed as well as the merits of alternative approaches. The intuition of each hypothesis forms a reference point should the anticipated outcomes materialise as counter-intuitive. A brief recount of the origin of the research motivation is elucidated.

3.1 Research motivation

CP is a framework for problem solving using the logical propagation of constraints through an underlying graph. There are several strong constraint propagation algorithms that can solve certain problems trivially; while these may at first glance appear quite difficult, the problem associated with finding feasible solutions remains NP-complete in the general sense.

CP, unlike its LP counterpart, does not have a strong mathematical grounding in its solution technique but rather has its roots in languages used during the 1980s, such as Prolog II. Its usefulness stems from a logical representation of constraints which support implicit representations as opposed to explicit representations of all constraints.

The current state-of-the-art is the unification of these frameworks, using the mathematical bounds and variable relaxations in a *master* model which guides the overall search for optimality of the model in an MILP framework and a *response* model which accommodates more of the incompatible constraints with respect to the MILP framework and the final feasibility of the

solutions found. This proves to be a particularly useful approach in complex scheduling problems [177] or where sensible decompositions can be applied [71].

IBM has a mature commercial CP offering in the marketplace. Machine learning algorithms are employed internally to select algorithms used to guide the solution process in terms of the selection of variables, values and branching strategies [107]. The author's experience with the IBM CP-Solver, however, highlighted several problems:

1. In certain larger scheduling problem instances, priorities for certain variables are not easily identified until it is too late in the search. The work in [54] addresses this directly through an ACO metaheuristic.
2. Due to a lack of data, arbitrary decision making at the start of the search hampers the solver's ability to attain feasibility later on. As stated above, this could also be dealt with using a metaheuristic with domain-dependent information [54].
3. Variable branching priorities are not aggressively adjusted for when the solver spends a large amount of time unsuccessfully attempting to resolve a set of feasible variable values. The inherent long-term dependence between variables is dealt with passively through constraint propagation as opposed to being directly modelled as a joint priority. This results in feasibility being erratically obtained for the same instance, given different random seeds. There are many ways to address this problem which involve online adjustments of the heuristic algorithms used as surveyed by Kotthoff *et al.* [107].

In the absence of direct user intervention, CP-Solver implementations run multiple independent instances in parallel or effect multiple restarts for the same problem instance with different random seeds to combat these shortcomings without direct augmentation. This is somewhat inefficient, both computationally and from a product development perspective, as it does not address the underlying inefficiency directly.

The personal experience of the author's use of competitor products to IBM (Artelys-Kalis¹ and OR-Tools²) has indicated that they require manual specification of the branching schemes for larger scheduling problems which may result in an inflexible solution approach. Changes in master data or problem features potentially require the specification of new branching schemes which take one out of the space of reusable solution codes and back to the inflexible solution techniques described above.

While working as an optimisation analyst on train scheduling problems, the author was fortunate to receive the opportunity to port a GA, using a local backtracking search for clean-up, to an exact framework utilising the Cplex Optimisation Suite. There were several advantages to this migration; knowing that optimality bounds could be attained for the problem instance, as well as being able to profile the solver without having to place development effort enhancing the solver to handle complex constraints, by leveraging existing advanced propagation procedures. The train-scheduling problem³ admitted a clear decomposition which would allow one portion of the problem to be solved in CPO (the Constraint Programming module) and the other in Cplex, using a MIP formulation with very strong bounds.

¹Interestingly, Gurobi have taken a very firm stance that integrated constraint programming in their MIQLP offering will not be supported. Gurobi pioneered the distributed cloud model for mathematical optimisation work-loads, being able to farm out processing time to their server on a pay per hour basis. IBM quickly followed suit.

²Google's CP Solver is one component in the open-source project OR-Tools, a tool kit of Operations Research-related functions, currently maintained and extended by a team of ex-ILOG (IBM) employees.

³The description of the problem is left intentionally vague as the final product is considered corporate intellectual property.

The CP objectives were complex and non-linear, but the CPO module was capable of handling the model. While profiling the results of the solver across a range of problems, it was noted that in some instances the solver was unable to find a feasible solution within the time limit provided, and so began the process of finding branching strategies which were able to improve the solution process of the model. Multiple branching strategies were manually specified which revealed possible large improvements in the search process. A prototype was constructed which used a GP to create possible branching strategies with different grammars, consisting of the arithmetic operator set, deterministic features and stochastic features.

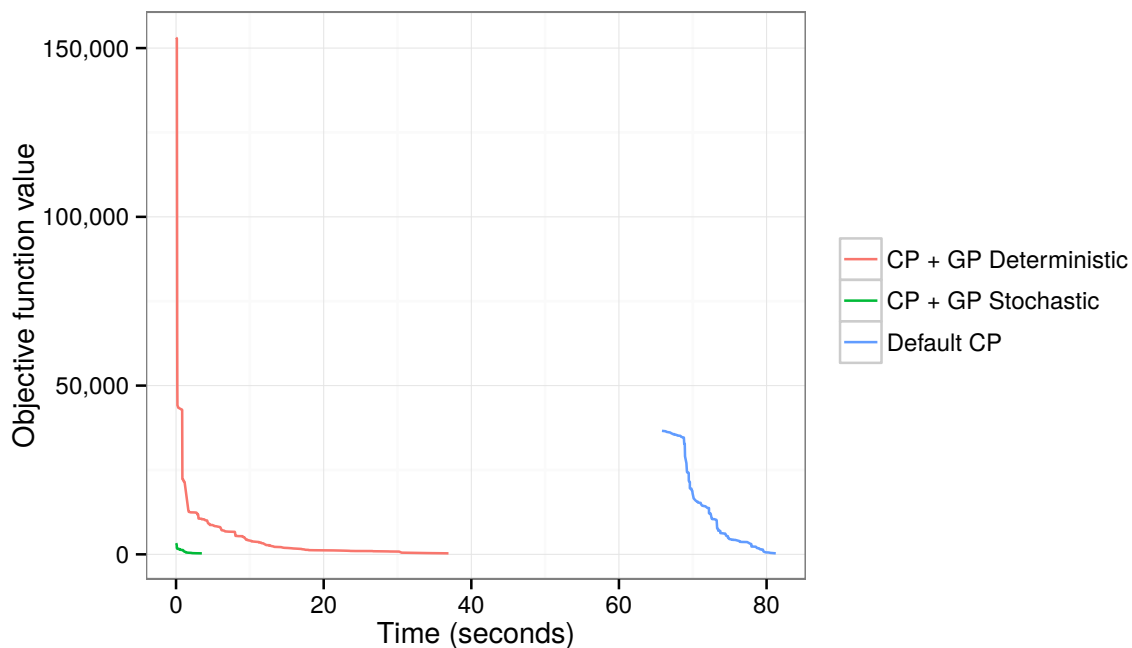


FIGURE 3.1: *Research hypothesis prototype result using CPO.*

The result of the research prototype is shown in Figure 3.1 where the blue line is the default CP search provided by CPO. It can be seen that the search took some time before a feasible incumbent was found, after which a relatively quick optimality proof followed. The red line is the same CP engine using an evolved GP search strategy limited to deterministic operators to guide the search. It is clear from the search characteristic that a feasible solution was found with relative ease and a longer optimality proof followed, which is arguably less efficient than the default search given a starting incumbent. The green line demonstrates the usefulness of using randomness in the search developed by the GP. This resulted in an optimality proof in approximately 4 seconds, which also outperformed all bespoke strategies that had been hand-crafted at the time.

Dozens of works have been published on hyper-parameter tuning of black-box solvers such as Cplex to improve performance [17, 21, 22, 89], typically referred to as the General Algorithm Configuration problem. The resulting improvements shown in Figure 3.1 are unsurprising in outcome, in that a better strategy exists; however, the quantum of the gap between the generic strategy and an optimised strategy being approximately a factor of $20\times$ was unexpected whereas a typical improvement would have been in the order of $2\times$, as found by [89]. The size of this potential gap is intriguing and led to several other possible questions being raised, such as:

- Was the result specific to this instance?

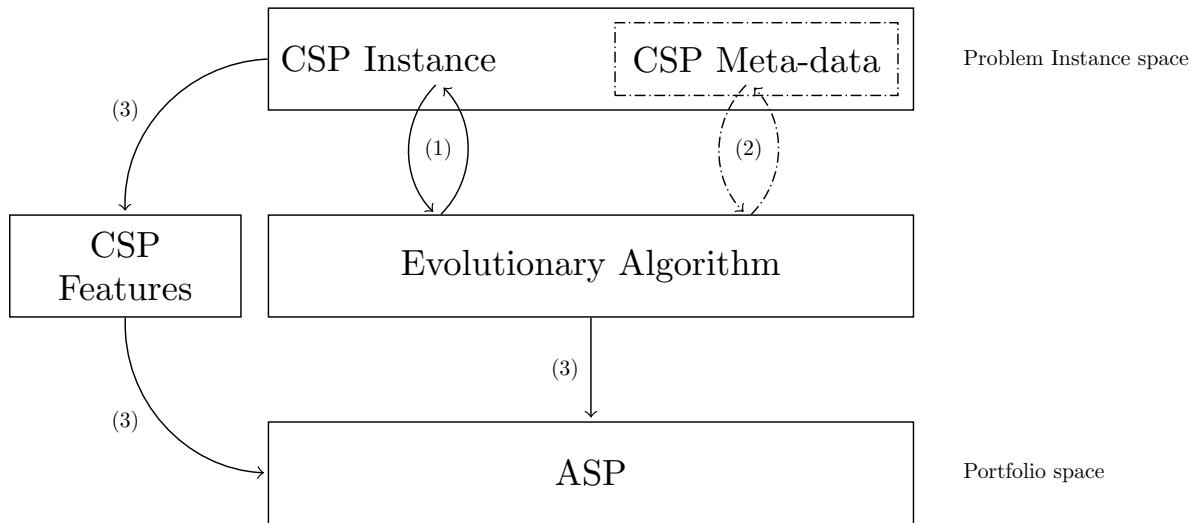
- To what degree would the strategies generalise across similar data for this particular model?
- How well does this approach work on pure feasibility problems?
- Are there certain dominant strategies which are typically more useful than others?
- How does one characterise the features of a model?
- Can one learn *a priori* which features should be paired with certain search strategies?

The characterisation of the hyper-parameter tuning of engines such as Cplex is a problem which resides in a more classic representation of the variable space such as binary, categorical and continuous variables. In contrast, CP search guidance adopts a tree-based representation which is less well-studied and admits multiple potential modelling approaches, such as GP, RL or MCTS, each with their own implementation advantages and disadvantages. The work in this dissertation can thus be viewed to fall within the hyper-parameter optimisation field of study but where the parametrisation of the problem space is, in fact, solving for potentially domain-extensible strategies.

The aforementioned shortcomings of CP in commercial environments is somewhat unsurprising. Emphasis in such commercial environments is often placed on reducing volatility in search times whilst maximising the average performance over a large number of benchmark problems, in line with the economical formulation of risk. This is a sensible requirement when distributing a product to a market which has a vast number of use-cases. The criticisms levelled against CP in this context could be considered unfounded, because according to the “no free lunch” theorem, one cannot reasonably expect such a generic framework to achieve unrivalled performance in respect of every type of problem. The motivation behind the research in this dissertation is driven by specific models for different problems often found in commercial environments, an attempt to close the performance gap between static and generic search routines and a move towards learning problem-specific search strategies with a view to re-use such strategies where problems demonstrate similar feature sets.

The problem of finding “good” branching strategies in CP has been studied in the literature by leveraging a range of methodologies, including evolutionary algorithms, evolutionary-inspired variants and using other local search methods such as random search or beam search, with varying success [12, 38, 57, 63, 135]. There have been several advances in online feature sets, such as IBS and FDS, provided by most CP engines which have been partially included in prior studies. Similarly, certain bespoke features and GP-specific tree configurations for SAT branching schemes are not considered in the methodology of this dissertation in an attempt to keep the reproducibility of results straight forward and solver agnostic as well as by using tooling that is publicly available. Many of the benchmark models and data sets used in the literature are not publicly available or are isolated to a single family or class of SAT problems. The intention in this dissertation is to test the methodology on a wider set of problem domains which are publicly available.

The research hypotheses for this dissertation are partitioned into three distinct tests. A summary of the structure of the overall interactions is provided in Figure 3.2. For each of the hypotheses a brief explanation of the statement, objectives, methodology and the author’s intuition is provided. A detailed treatment of each hypothesis result is provided in the Chapters that follow.

FIGURE 3.2: *Research hypotheses and interactions.*

3.1.1 Hypothesis 1: Evolutionary CSP augmentation

Statement: *An evolutionary process can produce strategies which are able to assist CP solvers resolve CSP problems based on primitive CSP domain information.*

Aim: Establish the performance of a default CSP search algorithm on a test set of problems, referred to as the base case. Allow an evolutionary process to develop algorithms for solving the same set of problems, provided only the primitive CSP graph information as variables in the GP trees developed. Using statistical methods, establish whether there is a significant performance difference between the evolutionary process and the base case.

Methodology: Several metrics can be used to measure the performance of different algorithms. It would be reasonable to consider the total computational resources consumed to achieve a particular result, which can be measured in a unit of time (such as seconds). The number of branches explored and fails encountered in the CP search will provide an indication of the workload and efficacy of different search strategies. It is difficult to use these as definitive metrics as some strategies may be to fail fast and often, while others may be to explore methodically.

While it may be interesting to measure the number of branches processed in CP, its usefulness as an indicator of overall performance in isolation is questionable. Some works in the literature measure branches, failures and coverage; the last metric being an indicator of how diversely the search space is being explored. Unfortunately this metric is unavailable in many non SAT-solvers and a bespoke methodology for efficiently computing this generically across solvers is outside the scope of this dissertation. The potential consequences of omitting the coverage metric will be reviewed. The focus of concern is on test data comprising multiple instances requiring a solution. It is thus possible to measure the aggregate solution status and limit the granularity of the metrics over which optimisation will take place. It may be possible that additional fine-grained metrics may guide a GP to a solution otherwise not considered. It is, however, assumed that solutions are predominantly represented by their overall outcome to a particular CSP search process.

The test data used will focus on a variety of problems and techniques required to solve them efficiently rather than on the pure number of problems. The 2015 MiniZinc Challenge problems will be used as the benchmark instances for testing and validation. The 2015 challenge comprised of 100 problem instances from 20 problem classes. One problem was removed from the set,

namely ‘large-scheduling’ as it required specific 64-bit binaries to have sufficient addressable memory for the MiniZinc to FlatZinc conversion. Fortunately, the entire 2015 challenge data set can be used for algorithm training as several new instances have subsequently become available which are mostly derivatives of the existing problem classes, and which exhibit similar underlying constraint structures. The problem classes also consist of a range of difficulties with a few being easy, but the majority of which are medium or hard.

CSPLIB is an open library of CSP benchmark instances which have widely been used. Instances are considered diverse and stem from key areas of interest in traditional CSP such as scheduling, bin packing, *etc.* The CSPLIB is used to extend the MiniZinc Challenge data set to include new instances of the same class, yielding a *validation* data set.

There are multiple approaches for determining possible grammars, with sets of allowable operators, to be used by a GP tasked with designing search strategies. Similar evolutionary modelling attempts in the literature have applied an arithmetic-based operator set which lends itself naturally to sorting-based algorithms for finalising search priorities at each search node. The same approach will be used to test this hypothesis. Initial GP populations are expected to perform poorly, potentially more so on harder problems, but the expectation is that the advantage of being able to customise a search strategy to a problem domain should result in strategies that are able to match a benchmark algorithm such as the default search strategy of a top-tier CP solver.

In order to avoid creating strategies which are tailored to specific problem instances, strategies will be evaluated over a set of problem instances and the quality of the overall performance achieved will be used in the objective function of the GP. The intuition here is that the GP will favour strategies which perform well on the class of problems rather than any one specific instance of a class.

Interesting observations to note while testing this hypothesis will be the time it takes the evolutionary process to match the performance of the default-search algorithm as well as the rate of convergence for problems of varying difficulty. The diversity of solutions generated and the efficacy of the metaheuristic is of general interest, but not the focus of this experiment. Parameter optimisation of the GP will be kept to a minimum and there are certainly potential improvements in the GP parameters chosen for the optimisation runs performed. Statistical tests and methods are used where applicable to demonstrate general trends and to highlight differences between the benchmark, test and training performance.

3.1.2 Hypothesis 2: Evolutionary meta-CSP augmentation

Statement: *The inclusion of instance-specific CSP graph meta-data will result in superior algorithmic strategies being evolved.*

Aim: Test for significant differences in the results of an evolutionary process with and without additional CSP graph meta-data.

Methodology: Problem-specific meta-data are extracted from the CSP graph definition and added to the meta-data provided to an evolutionary process. This allows for additional variables in the decision trees generated by the evolutionary process. The intention is that this will permit the metaheuristic to better tailor an algorithm for solving the CSP more efficiently if there is key information contained in the meta-data. The author’s intuition is that with additional unique information about the problem landscape which is not already expressed in the primitive features one may be able to produce better algorithms for solving the problem at hand.

Certain problem classes may not admit graph features which improve the search process. This in itself is an interesting result particularly if it is found as a trend across a class of problem instances. It may simply be that some problem constraint structures are largely one-dimensional and complex graph features yield little to no further insight in the solution process.

One would expect the GP to achieve performance comparable to the results achieved in Hypothesis 3.1.1 bearing in mind that the search space for the GP is much larger in this context as additional features have been introduced while keeping parameters such as the population size, number of iterations, *etc.* equal.

Interesting aspects to note here will be what the mean improvements are across different problem types and to what degree the additional meta-data are considered in the development of the new algorithms, if at all. Statistical tests will be conducted to establish whether any significant differences exist between these evolutionary runs and those executed to test Hypothesis 3.1.1. If no significant performance increase is found this would suggest that all required information for a search is already contained in the constraint graph specified.

Other metrics which can be explored at this point may involve the expression trees generated by the evolutionary process *i.e.* parsimony over complexity, the mean and variation in number of terms per tree, or marginal gains or losses in metaheuristic performance as a result.

3.1.3 Hypothesis 3: Deep learning for algorithm selection

Statement: *A deep learning process can predict which search strategies will perform well on a given CSP using the features of the CSP and the history of prior searches.*

Aim: Measure the predictive error across typical ML techniques, previously surveyed in [107], and test for significant differences compared to those associated with the deep learning technique.

Methodology: Deep learning is increasingly being used to model problems which have poorly understood domains involving many complicated, non-linear, high-dimensional interactions in their predictor variables. Comprehensive studies of ML techniques have been conducted [107], showing that in few instances ML techniques outperform one another on the Single Algorithm Selection Problem for CSPs.

A conventional methodology involving training, testing and validation will be adopted to fit a deep learning model to the data. In order to verify the robustness of the resulting model, a regression model will be used as the baseline for predictive accuracy.

The literature suggests that deep learning has not been applied to the ASP as it is a relatively new technique, only appearing during the last few years to be mature enough to handle complex problem domains and in line with the advent of *Big Data*. It would be prudent to confirm that deep learning does indeed solve the problem at least as well as its predecessors. There are complexities to consider related to the representation of the data in the context of branching strategies as these result in categorical variables numbering in the thousands. As with the previous hypothesis, one would at least expect deep learning to match the performance of simpler models, as it admits all the potential representations as smaller subsets. Linear regression may be considered a subset of this parent class of models, except that the increased search space may result in poor convergence on such parameter-rich models.

The predictions of the fitted deep learning model can be verified against unseen problem instances from the extended benchmark data set, and in respect of problems outside of the benchmark problem classes, making it possible to build a very simple portfolio solver selecting from thousands of experienced search strategies.

3.2 Hypothesis testing considerations

The architecture proposed in Figure 3.2 covers a range of techniques and disciplines. In order to test the hypotheses proposed in this dissertation, a number of architectural decisions need to be made. While algorithms at their purest level remain an abstract notion, efficient and clean integration of multiple frameworks in a cooperative manner in the same platform presents a complex task in its own right. Differences in the underlying platform and architecture should be avoided as a result.

Taking cognisance of the aforementioned complexities, the following implementation decisions need to be considered:

CP Solver. Two candidates stand out for consideration: ILOG’s CP Optimizer (CPO) [91] and the OR-Tools CP Solver (ORT) [146]. Both have interfaces in .Net⁴ and Java⁵ using SWIG [18] with the core solvers written in C++. It is no accident that both of these solvers use SWIG to provide access from managed languages given that the lead engineer and architect of ORT worked on the CPO product for a number of years. CPO served as a appropriate prototyping engine as it has well thought out interfaces to the solver which prohibit users from being able to dabble too deeply in the core solver. As the work for this dissertation progressed it became clear that using an open-source solver would allow the author to confirm which algorithms were being used for different processes as well as adding low-level data structures to the search in order to maintain reasonably high levels of performance.

The transparency of the ORT codes made it the solver of choice for this dissertation. The version of the ORT CP solver used was locked to the 2015 version that was used for the 2015 MiniZinc Challenge⁶. The choice of solver does not impact the results nor methodology discussed herein. If the intention was to simply quote the best-known performance on different problem instances, the 2018 version of the ORT SAT solver could be tweaked to achieve this result. The 2015 ORT CP solver was competitive but the team at Google felt that they had taken that particular CP methodology as far as they could⁷ which led to the engineering of their SAT solver which is indeed superior to the CP Solver. This makes the 2015 version of the ORT CP solver of particular interest as the perception at the time was that the performance could not be improved much further. This allows one to explore possible gains through tailoring the solver’s branching strategies to problem domains.

Evolutionary Algorithm. It would be possible, although not time-efficient, to code an evolutionary framework from the ground up that is consistent with many of the findings in the literature with respect to best practices, operators and population mechanics. There are a multitude of EA frameworks available for free usage under their license agreements. Unfortunately, however, the quality of the majority of these open-source products is mediocre at best with limited academic grounding.

⁴The Microsoft .Net Framework is a windows framework that handles the memory management of applications built within it.

⁵The Java VM is a runtime similar to .Net but is not bound to windows operating systems. It is arguable that with .Net core which is a docker-based distribution of .Net portable to Linux, that this has been less of a sticking point in recent years.

⁶Subject to a bug fix the author found in the FlatZinc parsing layer as well as a small issue in the model presolve for three of the benchmark problems. The presolve issue resulted in the ORT CP solver reporting an invalid optimal solution to one of the problem instances.

⁷Personal correspondence with L. Perron.

One library in particular does seem to be an exception to the norm. ECJ [127] is a well maintained evolutionary algorithms library that has been actively developed for several years. The primary contributor of ECJ is Sean Luke, author of *Essentials of metaheuristics* [126]. This library has a port to .Net and is natively written in Java. This poses a complication in that the language of ECJ is managed by the Java runtime whereas the CP solvers are unmanaged⁸. The approach adopted in this dissertation produces branching strategies which are native in a single language (C++) with the methodology of creating syntax-compliant codes being discussed as a separate item in this section.

Deep learning. In order to maximise the benefit from deep learning techniques, large networks are typically created and fitted. Classical methods do not support the calculation of this number of parameters in a reasonable amount of time when working on a single machine or in a cost-efficient manner when working with large CPU-based clusters. By using GPU processing power, large networks can be processed in reasonable time frames at reasonable costs.

Highly efficient GPU implementations of ML used to be closely guarded to protect the intellectual investments contained therein. The purpose of this proposal is not to produce a commercially competitive deep learning implementation; however, it does hinge on leveraging the state-of-the-art in ML in an efficient manner. The TensorFlow [3] engine is chosen as the ML layer of choice for several reasons: It supports GPU functionality, is well tested, well documented and widely used by a significant community. There are also several modelling layers, such as Keras [42], which provide higher-level modelling functionality while leveraging TensorFlow as the backing solver.

Distributed Computing. Another cornerstone of this proposal rests on a distributed architecture whereby multiple evolutionary runs can be conducted concurrently, reporting back their outcomes asynchronously to some repository. This housing of knowledge is the vessel for iterative, shared learning across runs. A possible extension at this point would be to support a decentralised architecture which has several benefits but which also introduces significant complexity. For the purposes of this dissertation, it would seem prudent to keep this as simple as possible in order to achieve the task at hand.

ECJ supports distributed computing as well as multi-threading using a classic master-slave architecture. ECJ is highly-configurable offering the ability to support a handful of slaves which, in turn, are able to batch-process multiple population individuals using multi-threading. The alternate model is also supported which incurs higher network overheads with there being as many single-threaded slaves as there are available machine threads. Using distributed processing, the experienced runtime of a particular GP search is reduced by utilising multiple identical machines. The total execution runtime remains unchanged for the GP.

Service Protocols. A repository layer that supports *JavaScript Object Notation* (JSON) will serve as a platform and language-independent data transport mechanism when accessing the repository. There are several architectures which will support this as an integration component with minimal effort.

The implementation of an additional distributed processing architecture is not required as ECJ already has performant-socket implementations for orchestrating distribution computation for the GA. System-wide cluster configuration and process management is handled

⁸The distinction between managed and unmanaged codes may at first glance seem pedantic; however directly integrating codes across this barrier requires moving through the operating system using processes which typically incur large overheads.

using Ansible⁹. If performant inter-process communication was required, the standard *Message Passing Interface* (MPI) would have been considered, but this is out of the scope of this dissertation.

Repository. There are two approaches to the storage, retrieval and processing of data. The *Structured Query Language* (SQL) and NoSQL¹⁰ database structures each have their own advantages and disadvantages. The type of data the author will be extracting from the CP graph, meta-data and evolutionary runs, each lend themselves to one of these structures. The CSP features, as well as run results, lend themselves to a SQL format since all fields and the shape thereof are known *a priori*.

The evolutionary algorithm, on the other hand, will produce complicated tree-based structures which are better suited to being stored as trees, as raw binary, in a NoSQL database. In addition, while one may know the current metrics for the CSP graph and meta-data, any additional features one wishes to measure will require a structural change on the back-end system if a SQL database is used. SQL databases are quite inflexible with respect to changes in the structure of the data being measured and, as a result, a NoSQL database REDIS [47] will be used as the backing repository. REDIS is a high-performance in-memory NoSQL database. Data are stored as key-value pairs but what makes REDIS particularly useful is the support of primitive data-structures such as lists, dictionaries, ordered sets and hash sets. Such data structures allow one to leverage underlying data structures if required or use the database for simple storage without needing to change the back-end layer.

One typical disadvantage of using NoSQL databases is that certain query operations are not performed as efficiently as would be the case in a well designed SQL database which would typically be as a result of fields not being indexed or a representative data structure being unavailable. Many SQL storage systems use ACID transactions [79] to ensure that data are persisted. REDIS runs in a single thread and queues transactions against the in-memory store, relying heavily on the in-memory response being sufficiently fast to avoid requiring more complex data-management procedures. The author's experience is that REDIS is a pleasure to work with and can store data in its native binary format, not only keeping the database size minimal in many instances, but also keeping translation overhead or interpretation errors to a minimum. Copying REDIS databases or sharing data between machines is equally simple and can be performed from the terminal in a single line.

3.3 Test data

Clearly defined test data allow for the understanding of algorithmic performance against a set of features which may have similarities within a class of problems or specific to a problem instance. Problem definitions and data descriptions can be considered synonyms in the preceding sections. CSP benchmark data sets exist for testing the performance of CP engines to solve discrete optimisation or feasibility problems. A frequently cited test data suite is CSPLib [94], consisting of 88 problem types grouped by application category at the time of writing. The benchmark sets have been steadily increasing in number each year.

The CSPLib is predominantly maintained as a collection of MiniZinc [93] model files. The MiniZinc language is designed to be a mid-tier constraint modelling language whose specification

⁹An open-source cluster configuration and process management tool built primarily for Unix-like platforms.

¹⁰A schemaless query language.

is independent of the solver. MiniZinc translates a model into FlatZinc which is then ready for consumption by a solver.

The reason for this additional layer of abstraction is to enable possible global constraint aggregation or decomposition by model inspection. Global constraints can provide strong propagation where solvers have specialised data structures for such constraints, often resulting in performant search. It is not always possible to convert a localised or ‘flattened’ set of constraints back to a global constraint which is why MiniZinc provides these two layers of interpretability.

Once a FlatZinc model has been generated, a solver can then parse and instantiate the model for solving. MiniZinc also allows for the optional specification of the strategy to be employed by the CP solver. It is the task of the solver to determine the strategy that would be best suited to solving the provided CSP when the strategy is not specified. The annual MiniZinc Challenge typically comprises a subset of the CSPLib problem sets; where new problems are used in the competition, they are added to the CSPLib shortly thereafter. Two competitive CP solvers, CPO and ORT (for which descriptions were provided in the previous section), are considered in this dissertation. The CPO solver has not been formally benchmarked to date in any MiniZinc challenge. The participation of the Ilog solver in these challenges is largely due to the positioning of MiniZinc as an intermediate modelling language which places it as a competitor language to OPL, the intermediate modelling language provided by IBM for integration between *CPLEX* and *CP Optimiser*.

The open-source nature of ORT allows for the exploration of the default strategies employed when solving the MiniZinc benchmark problems. In single-threaded operation, the solver makes use of heuristic dives to tackle the provided formulation, with IBS and heuristic separations over large domains being the primary variable-value selection process. In parallel mode, different strategies are deployed on available threads with random strategies being used for excess threads. The heuristics use basic domain properties to generate a ranking on both variables and values. ORT used in parallel execution employs a similar technique to the strategy used by CP-Hydra [144] where different strategies are given a time budget to spend solving the remaining sub-problem in the search.

Restarts are used by the ORT solver in conjunction with impact assessors to produce different search paths. Impact assessors measure the success or failure of variable or variable-value assignments as well as the reduction in search space as a result of assignments. Impact metrics are maintained incrementally throughout the search and are reset heuristically. A detailed description of the search process used by ORT is provided in §3.4.

A summary of the performance of the ORT solver on the MiniZinc 2015 benchmark data set is provided in the following chapter (Table 4.9). Pure CSPs have three possible status codes (S, C and UNK) with optimisation problems having an additional possible status (SC), shown in Table 3.1. Optimisation problems that are feasible but for which solutions have not been proven optimal receive the ‘S’ status code. If an optimality proof follows finding a solution, the status code is ‘SC’. Problems which are shown to be unsatisfiable are given the ‘C’ code *i.e.* completed a search with no feasible solution.

CPO is a proprietary solver and, as such, a clear description of how the algorithm determines its branching strategies is not public knowledge. That said, there have been two publications related to IBS [156] and FDS [181] by authors who were all affiliated with Ilog, the developers of CPO within IBM. Based on circumstantial observations of how the CPO solver behaves on harder problems, there does seem to be a preference for a particular basic strategy and after attempting some number restarts the strategy may change drastically in an attempt to provide a feasible incumbent. The author suspects that given the core team working at Ilog has published on the

Code	Description
S	A feasible S olution was found.
C	The search was C ompleted.
SC	A S olution was found and the search was C ompleted.
UNK	No solution was found within time limit, invalid solution or out-of-memory error.

TABLE 3.1: *Constraint satisfaction problem status codes.*

success of both IBS and FDS, that these techniques are the two core search techniques employed within CPO. This is speculative observation and certainly, as time goes on, the probability that a blended strategy which potentially employs multiple new techniques becomes increasingly likely.

The MiniZinc benchmark problems are a useful baseline in that it is expected that an evolutionary strategy with similar graph metrics would perform at least as well as the default search strategy in most instances. Exceptions to this expectation occur when the default search strategy used by the solver has complex sub-strategies which are particularly useful for a certain class of problem. Attempting to find one simple strategy which performs well on a class of problems is expected to fall short where the highly tailored heuristics are able to take advantage of the problem structure. The purpose of this research is to ascertain whether there is a strategy that can perform well within a tier of problems as well as be considered generally applicable to a class of problems that exhibit the same structure. As such, for each of the problems in Table 4.9, the GP will be evaluated with respect to its performance across the problem set as opposed to on a single instance of the problem.

The MiniZinc challenge allows for 20 minutes of search time per problem instance, or 1 hour 40 minutes for five data configurations within a problem class. A GP requires significantly more time than this to try different strategies. If 10 seconds of search time is provided per data configuration, referred to as the *sampling time*, a maximum sampling time of 50 seconds per individual in the population is expected when excluding the compiler time to construct optimised strategies which is approximately 1–2 seconds per GP individual. A population of 80 individuals and 15 generations would then require a maximum of 17 hours of search using this approach. A motivation for the decision to use 10 seconds of sampling time for each problem instance is based predominantly on total computational budget limits, but does introduce a bias in terms of which search strategies which perform well at the start of a search would typically be favoured. In an ideal world, a complete 20 minute search time for all problem instances would be desired to evaluate their quality and bring the expectation more in line with the final evaluation of a strategy. A resulting GP search would require a maximum of 84 days of computing time in order to complete a single run on a problem class where the sampling time was set to 20 minutes per problem instance. There is a trade-off in terms of additional information gained per unit of search time applied, but the additional search time comes at a computational cost as well. A study of the effect of varying the sampling time and the additional learning gained is perhaps a study in its own right as there are many paths one could take to determine an optimal sampling rate based on the problem size, structure, features *etc.*. As a result, further analysis of the sampling rate value chosen is considered out of the scope of this dissertation.

The MiniZinc modelling language allows for the specification of a search strategy with reference to the variables created. This is useful for testing the performance of different solvers with respect to their underlying propagation algorithms and search performance while holding the search path constant. The *free search* category in the MiniZinc challenge is of most interest in the context of this work. The results of the free search are compared to the results of the GP.

In order to test the extensibility of predictions made by the deep learning model in Hypothesis 3.1.3, the same set of test instances used in Hypotheses 3.1.1 and 3.1.2 are selected from the CSPLib. These problem instances are expected to demonstrate the degree to which features within the MiniZinc Challenge data set are common to features outside this set and should identify any vulnerabilities in the predictions made by the deep learning model.

3.4 Benchmark strategy

The free-search performance of the ORT CP solver is compared in this dissertation with strategies developed by an evolutionary process. As a result, it is worthwhile providing a description of the solution approaches taken by the ORT CP solver. Formal documentation on the free search strategy used for the MiniZinc challenge is not provided, but as the code is open-source, it can be inspected directly to deduce the strategic themes employed.

Some preliminary steps are taken by ORT to reduce the complexity of the problem instance. The first process is referred to as *presolve* and attempts to identify variables which have a reduced domain or variables which can be logically split into independent subsets. An initial probing step is also performed in the default search procedure to identify variables whose values can potentially be removed. The purpose of the probing is to reduce the total search space size. A criticism of this method is valid when the search space is particularly dense and complex as simply performing this first-order variable scan can be so computationally intensive that the entire time budget for the search is spent on this first step, with the solver timing out before the actual search commences.

Once the solver has commenced the search, several heuristic procedures may be employed, including the following:

- AssignMinValueToMinDomainSize,
- AssignMaxValueToMinDomainSize,
- AssignCenterValueToMinDomainSize,
- AssignRandomValueToFirstUnbound,
- AssignMinValueToRandomVariable,
- AssignMaxValueToRandomVariable,
- AssignRandomValueToRandomVariable,
- ImpactBased
 - AssignValueWithMinImpact (default),
 - AssignValueWithMaxImpact,
 - ChooseVariableWithMaxSumImpact (default),
 - ChooseVariableWithMaxAverageImpact, and
 - ChooseVariableWithMinImpact.

The duration over which a heuristic is run can be limited by either the run time, the number of branches, the number of failures, the number of solutions or a combination of these. IBS is

primarily used by ORT, which switches to using heuristics for short periods of time to sample the remaining search space. The internal process in code is referred to as using heuristics as *dives*, the intuition being that heuristics can dive into the problem and sample possible search paths. The result of these dives are updated impact scores on visited variables and potential solutions. Search restart rates are also determined heuristically as a function of the log search space, in line with the Luby restart expectations [125]. Heuristic dives are limited to a default of 30 fails per search heuristic and are checked at intervals by a default of 100 branches per heuristic throughout the search.

Conflict-directed search is used in conjunction with the restart mechanisms in ORT, where the last conflict point is used to guide the search to a different portion of the search space which is analogous to FDS with small variations in the actual implementation. *Nogoods* are also employed by ORT at the point at which search restarts. Both *nogoods* and last-conflict-directed search form part of the default strategy.

Variables with very large domains are handled by a heuristic split of the domain of the variable. This creates a decision point where the solution is either in, or not in, one of the domain partitions created. Domain partitions are created by choosing a bisecting value in the domain and potentially creating further partitions of the sub-partitions created. This is quite an attractive technique as it takes on a form similar to a binary-search tree. The total number of variable splits is limited to 100 in the default search. Unfortunately, this heuristic separation procedure for large domains was not adopted in the GP learning procedure, but could be included in future iterations of this work.

The search heuristics employed by ORT hinge on being able to inspect the domains of variables as a function of previous assignments. In the first hypothesis studied in this dissertation, the same decision points *i.e.* variable and value selection, are provided to the search procedure, whilst allowing an evolutionary process to determine the measure used for ranking decisions. The same primitive graph features available to ORT are used in the first hypothesis, considered in Chapter 4.

3.5 Implementation notes

Several systems are required to communicate and integrate with one another in order to build GPs for CSPs in a way that does not degrade performance whilst still supporting the automation of tasks as well as some degree of flexibility. Another consideration is that the high computational requirement, which entails several hours for a simple GP run, suggests that one should consider a distributed architecture that allows for a high degree of concurrency.

The components in the software stack for testing the first and second hypotheses in this dissertation are:

ECJ Evolutionary computational library, built in Java [127],

OR-Tools The underlying CSP solver, written in C++ (referred to as ORT) [146],

make A compilation tool used for specifying compiler arguments and managing build requirements¹¹,

MiniZinc A high-level modelling language that can produce a defined instance of a CSP (FlatZinc) based on a model specification [93], and

¹¹<https://www.gnu.org/software/make/>.

R A statistical computing language that can be used for data analysis or manipulation of data[153].

ORT is, at the time of writing, the highest ranking open-source CSP solver. This ranking makes using the source code an attractive proposition as one can modify it as required and gain key insight into a high-performance solver. ORT is written and compiled in C++, an *unmanaged* language. The specific terminology used here refers to how memory is managed within the application. An unmanaged process means that the memory is allocated and deallocated by the process itself *i.e.* the user or programmer. Examples of unmanaged languages are C, C++ and Fortran. These languages are designed to yield maximum performance and users are expected to understand their memory requirements and how best to manage the memory of the application.

Many modern languages, such as Java or .Net, are managed languages, meaning that the framework being used¹² handles the allocation and deallocation of memory by means of garbage collectors. These languages typically incur additional operating overheads, in the region of 2–5 times depending on the language and operations, and are not used for high-performance scientific computing. Managed languages are particularly useful in instances where complex, dynamic data structures are being used during runtime. Keeping track of or forming structures dynamically in unmanaged environments is possible, but cumbersome. As an example, the ECJ library pre-compiles Java classes but infers the required structures using reflection from parameter files during runtime. This means that the compiler cannot assert possible execution paths of the code *a priori* since structures are combined dynamically during runtime. In this instance, the structures being dynamically inferred are the decision trees.

Communication between managed and unmanaged code is possible, but requires marshalling at an operating system level. Pointers to memory must be stored explicitly in managed codes and measures have to be taken to prevent garbage collectors from freeing memory that is still being used by unmanaged processes. The reason for having to take these steps is that code outside of the framework scope is executed on objects that share memory with the current scope. Intelligent garbage collectors detect that the execution path for the current code ultimately disposes of a set of memory and pre-emptively deallocates the memory, causing an access violation in the managed child-process. Unmanaged processes can “call down” to managed processes invoking COM-Interop, which is an operating system level communication protocol.

The reason why a distinction between runtime environments is given a thorough treatment is as a result of the author’s experience using managed environment interfaces with an unmanaged code base. IBM CPO and ORT use SWIG to expose interfaces to their unmanaged code bases in managed environments. While this is a particularly attractive proposition to users of the software, there are some potential pitfalls when working with branching strategies in CP. The obvious benefit of using SWIG to handle the interface generation is that it can map a managed code base to numerous languages automatically. ORT and CPO both have a single C++ code base which can be accessed from Java or C-sharp in a seamless fashion which obfuscates the operating system interaction. This means that branching schemes can be specified in an unmanaged fashion during runtime and communicate branching decisions with the managed code base at the point of execution; herein lies the problem.

The additional overhead of communicating branching decisions between codes over COM, *i.e.* through the operating system, is so significant that it dominates the search time. Extensive testing estimated the slowdown due to COM to be in the order of 1000 times. To put that factor in perspective, one second of normal search would require almost an hour and a half of processing time through the SWIG interface. The fundamental problem is that the whole

¹²The Java Virtual Machine (VM) or the Microsoft .Net Framework respectively in this case.

search procedure is not coupled together in one unified environment, whether it be managed or unmanaged. Preliminary work quickly showed that additional time was required on the design of delivering such branching schemes to a solver in a way that could circumvent this overhead. In fact, identifying the swig interface design as the bottleneck was an exercise in itself.

In an ideal world, it would be desirable to link software together in such a way that optimises each component for maximum performance while keeping the design system-agnostic and without having to re-write or translate canonical libraries into other languages. ECJ is designed to run in a managed framework, taking advantage of the flexibility and reflection features offered by the framework. ORT, by contrast, runs in an unmanaged framework for performance reasons. Translating ORT into native Java for the sake of easier access to GP trees is not an exercise one would want to undertake and would also result in a performance reduction in the underlying CP search¹³. Similarly, while translating ECJ to C++ may be easier, it is also a deeply complex library and utilises the Java framework extensively for object construction and reflection, which is not supported to the same degree in native C++.

The solution settled upon in this dissertation is to define the syntax of the GP functions in such a way that they form valid C++ code which can be compiled into an optimised C++ application directly. Appropriate interfaces, synchronised definitions and checks are required in both languages in order to bind these two processes together — providing a single mechanism for controlling CP search in a highly optimised manner.

The process of creating a syntactically correct C++ decision tree is surprisingly straight forward in ECJ as all base-classes support recursive calls which traverse the tree in the correct order of execution. This means that nodes of the proposed decision tree need only print out their related C++ functions and recursively call down to any children that a node or function may have. The overall program string can then be included in an appropriate header file to be included in the final compilation of a program¹⁴. The first prototype of this compilation approach to embedding decision trees is shown in Figure 3.1 and was performed against the CPO solver which demonstrates that once a C++ interface has been defined, switching between underlying solvers is relatively straight forward.

MAKE is a language-agnostic utility used for defining build rules for difference applications. ECJ and ORT both have *makefiles* which define the process required to build the software for their respective environments¹⁵. In order to manage the injection of branching strategies into ORT, MAKE is used to define a build process which uses the specified search strategies generated by ECJ as input.

GP search strategies are placed in header files labelled by a *globally unique identifier* (GUID). The main search algorithm file is then copied and modified to reference the specified GUID as the branching strategy reference before compilation begins. The MAKE process creates an executable with the same GUID. When working in concurrent environments that are creating or manipulating files, it may occur that two process are attempting to create or delete files of the same name. Embedding GUIDs in the input and output file names ensures that environments in which multiple processes are running on the same machine instance, no overlap in the modified search processes being compiled will occur. Data generated from each search is tagged with the same GUID as the compilation process in order to make debugging or reproducing a result straight forward.

¹³There are pathological cases where poor programming in unmanaged languages is outperformed by managed languages which leverage more effective memory clean-up procedures, but this is an atypical case.

¹⁴This is quite appealing to the senses as the GP is quite literally writing compilable programs to solve a CSP.

¹⁵The recent changes to ECJ include a Maven build process for handling external dependencies more cleanly.

Executing a search with the correct parameters is also managed by MAKE. The result of this dependency is that ECJ invokes a MAKE command, triggering a new process as part of its objective function evaluation and never directly communicates with ORT. This decoupling ensures that other optimisation engines or compatible procedures can be tested without changing any interfaces in ECJ, except for the *makefile* definition within the current parametrisation.

A single evaluation of an individual in the population may require several evaluations of different sub-problems. Each sub-problem may, in itself, have different features or characteristics requiring reporting. This means that two levels of reporting are required; one at a population level and another at a problem instance level. The first FlatZinc problem considered in this study, *costas-array*, has five problem instances that contribute to the objective function evaluation. Discernment is desired between the performance on each instance as well as across the instances considered.

The result of any individual search is aggregated in ECJ by reading a JSON file describing the values of each objective being measured during a given search instance. Detailed search telemetry is saved to REDIS at the end of each run as well as the detailed GP data. This creates a link between multiple runs over problem instances that may be associated with the same GP objective evaluation.

A detailed discussion on the decisions to use an in-memory NoSQL data store are somewhat outside the scope of this dissertation. The high-level intuition, however, suggests that there are no machine memory constraints due to the relatively small size of population statistics, but are rather speed or throughput constrained. Traditional disk-based logging or writing can lead to a performance bottleneck in practice. The choice of a NoSQL database is to retain flexibility in the data structures (JSON) being stored without having to restructure database tables while data structures for different problems are developed. This facilitates reporting telemetry or the changing of objective functions to be performed without requiring a schema change to the database, which may not be backward compatible.

R is used to automate the process of generating FlatZinc models, model parameters and Java objective function definitions with respect to the FlatZinc files. MiniZinc models are fed with the appropriate data arguments in R to generate the FlatZinc formats. Templates which describe the outline structures required across the FlatZinc models are ingested by R and populated with the problem-specific parameters. Batch files which describe the final ECJ run parameters are produced by R to be executed in a shell script through the terminal. It is worth noting that the creation of Java classes, together with all FlatZinc files to be used in an objective function, the compilation of the classes, the creation of models and the analysis of final results, were all performed in R.

The degree of automation using R may seem excessive since FlatZinc models need only be created once per problem instance. When working in distributed computing however, it is simpler to replicate small model definitions (.mzn) and data parameters (.dzn) rather than complete FlatZinc definitions (.fzn) due to the increased size of the explicit definition file (.fzn). This automation allows one to run complex processes across a cluster in a distributed mode of working and be confident that there has been no finger slip or missed command.

TensorFlow models are also built and managed using R. It would have been typical to use Python, which has richer modelling *Application Programming Interfaces* (APIs) than those available in R for performing direct modelling in TensorFlow. It is rare that low-level modelling is required and as such, the abstract modelling layer, Keras, is used which has sufficient modelling flexibility and enables integration with multiple lower-level modelling languages, including TensorFlow. The author's experience with building and experimenting with TensorFlow models in Python

has not been positive, often requiring more than 30Gb of system memory as a result of the way that certain notebooks duplicate memory and manage data across sub-processes. The author's experience has also been that running models in R is quicker as a result of the tighter native data stream integration between Keras and R for sending data to TensorFlow. Data are manipulated using R and the Keras modelling interface to TensorFlow is used for ML modelling.

Figure 3.3 provides a high-level overview of the interactions between the components of the solution design. Process (a) refers to the creation of Java classes and the conversion from model files to FlatZinc files for solver ingestion. The GP Search creates a population for each class of models $P^{(m)}$ where m refers to the class of model. Multiple individuals are created within the population, indexed as j . In order to evaluate an individual, the decision trees are converted to C++ and compiled into an executable represented by process (b). The executable is then used in process (c) to conduct a CP search with the GP search strategy provided. The same decision tree is then evaluated over i problem instances which belong to class m . The results of both the decision trees used during the GP search and the search telemetry are fed to the REDIS database. Results are extracted to R for analysis, further learning, or strategy seeding during a subsequent GP search.

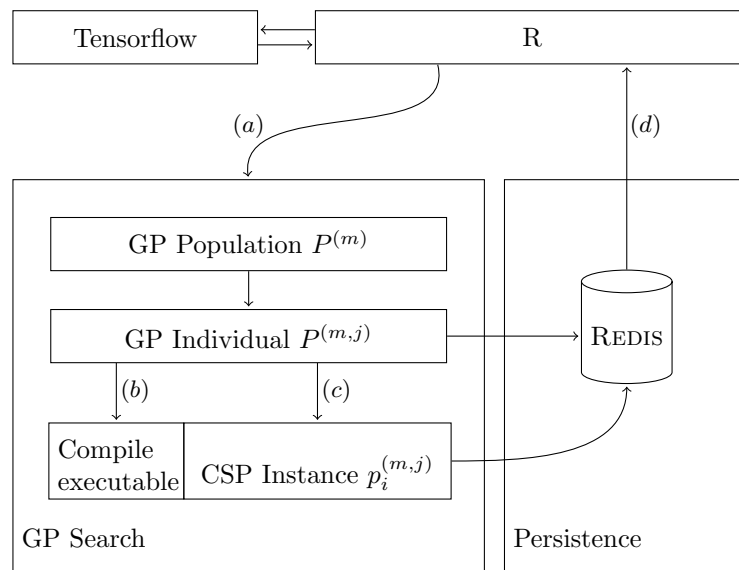


FIGURE 3.3: *Solution components.*

The standardisation of computational resources available to all experiments is important in order to make valid comparisons between runs. The hardware used to solve the 2015 MiniZinc benchmarks has quite a high specification (Intel i7 3770 chip running at 3.40GHz). Similar hardware was endorsed for the experiments conducted in this dissertation; namely, Intel i7 6770 chip running at 3.40Ghz using five identical 8-core instances running in a clustered configuration with the Centos¹⁶ operating system. The MiniZinc Challenge used the Ubuntu 14.04 distribution — a close enough match to the Centos operating system. Most importantly, all codes were compiled using the gnu C compiler with consistent compiler optimisation settings across all runs.

The thrust of the experiments performed related to Hypotheses 3.1.1 and 3.1.2 are not intended to replicate the results obtained in the MiniZinc Challenge exactly, but rather to create a relative baseline for changes in performance keeping the hardware constant. A historical

¹⁶A Linux derivative based on the Red Hat Enterprise Linux project.

reference is useful, but not absolutely required, otherwise a similar hardware specification and older versions of the code should also be used to draw a perfect comparison.

CHAPTER 4

Evolutionary Search Strategies

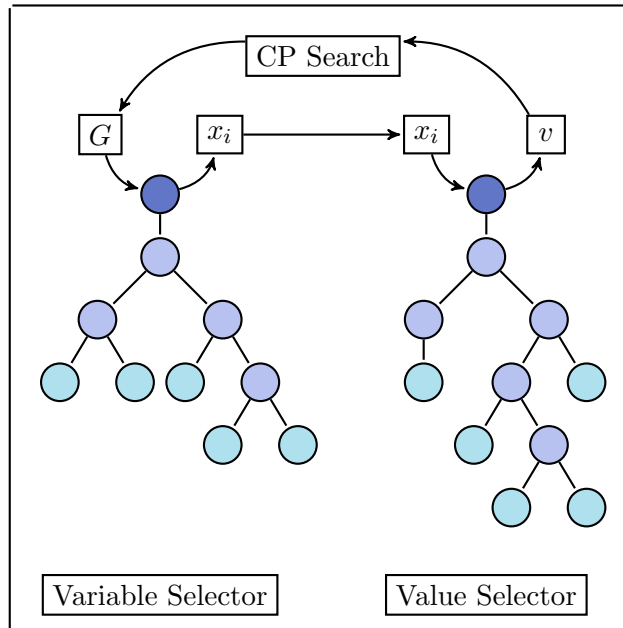
Contents

4.1	The ranking function	66
4.2	GP objective functions	67
4.3	GP configuration parameters	70
4.4	MiniZinc Challenge results (2015)	71
4.5	GP run results (Hypothesis 3.1.1)	77
4.5.1	<i>Classic CSP GP operators</i>	77
4.5.2	<i>Extendibility to unseen problem instances</i>	86
4.5.3	<i>Hypothesis 3.1.1 conclusions</i>	89
4.5.4	<i>Graph meta-data CSP GP operators</i>	91
4.6	GP run results (Hypothesis 3.1.2)	92
4.6.1	<i>Extendibility to unseen problem instances (Hypothesis 3.1.2)</i>	100
4.6.2	<i>Strategy comparison</i>	101
4.6.3	<i>Hypothesis 3.1.2 conclusions</i>	106

In order to utilise the Genetic Programming framework to develop a search strategy, it is required to specify the problem to be solved in terms of a set of nodes and terminals that form the decision tree. This representation is typical but requires that the nodes and terminals are an appropriate fit and reasonable parametrisation of the problem being solved.

In order to evolve a strategy to solve a CSP it is required that the high-level nodes and terminals are defined with respect to the available primitives in the decision tree. The fundamental search process in CP requires that two decisions are made independently of constraint propagation, or restart policies: Firstly, which variable (x_i) is selected to assign a value to in the search, and secondly, what value the selected variable should adopt (v). These two decision trees each require their own sets of nodes and terminals and are referred to as the variable selector and the value selector, respectively. Two parameters remain which need to be set, namely the restart policy and whether to use the last conflict in order to guide the search process. These two parameters can be viewed as decision trees which comprise a single terminal node representing the parameter selected and are excluded from Figure 4.1 which demonstrates the sequential dependency between the variable and value selectors. The term G in Figure 4.1 refers to the state of the CP Graph at the particular decision point being queried in the tree search.

CPO has a rich, out of the box, collection of variable and value selection features. As an example, the success rate of a variable and failure rate of a variable-value assignment forms part

FIGURE 4.1: *The GP CP sequential ranking process.*

of the standard API and are not natively defined in the ORT solver. In order to create a fair comparative test between these solvers, these measures are defined and implement online $O(1)$ storage and retrieval of these search metrics in ORT. The additional search statistics added to the ORT solver are the success rate of a variable, the number of failures on a variable value, instantiation on a variable value and the local impact of a variable value.

The tree representation used for the GP strategy in both Hypotheses 3.1.1 and 3.1.2 extend the standard arithmetic functions used in GP by introducing additional terminals and nodes which can be consumed arithmetically. The reason why this arithmetic set is well-suited to this problem is that a standard CP branching specification, from a user perspective, is to define a ranking function for variables or values. An example of this would be to say ‘select smallest’ or ‘select largest’ based on some evaluation criterion. The direction of the ranking function is arbitrary since a transformation of the underlying metric can always invert the ranking (such as negation). Selecting the smallest value in the ranking is used universally in this dissertation.

There are computational considerations to be made when using an uncached ranking function. There are special-instances where values in the ranking can be cached in a priority queue in order to maintain a low computational overhead ($O(\log n)$) but this is not true for an arbitrary function whose values are a function of the state of the search. Unfortunately, low level managed languages such as C++ do not natively support reflection. This means that understanding the terms participating in the ranking function during runtime are not easily discernible. Approximate independence in the search state space can be assumed in favour of this performance boost, but this will create a new type of ranking function which is disconnected from the underlying state in general. There are times when a ranking function approximation is acceptable, but it will not be used in this dissertation unless explicitly stated.

An alternate approach which could be used to emulate a cached ranking function while retaining some of the state information is to periodically update variable value rankings rather than updating this data structure at each node of the tree. This would, however, introduce an additional parameter which would need to be configured or tuned as a function of the instance and, as a result, is considered outside the scope of this dissertation.

Figure 4.1 intentionally obfuscates the details of the node and terminal sets used for each tree as one can consider the sequential procedure of variable value selection as being independent of the details of the decision tree structure. As an example, an arithmetic operator set has been employed in the following experiments, but could have included a grammar which supports loops, if-then clauses, *etc.* The tree constraints between operators in this case would become more complex but would still have been a reasonable alternative to create more complex functions if required.

The computational complexity of the decision points followed by the sequential algorithm illustrated in Figure 4.1 can be described as follows:

Definition 4.0.1 (Computational complexity of sequential ranking) *Given a network $N = (X, D, C)$, let n be the cardinality of the set X , that is $n = |X|$, and let m be the cardinality of the domain for a selected variable x_i , that is $m = D(x_i)$. Then each variable is required to be ranked based on the criteria provided in a network with no assigned variables, with n variables this ranking requires n calculations for each decision point ($O(1)$ complexity on evaluating the rank statistic). Once a variable has been selected, a rank value for each value is generated in domain $D(x_i)$ requiring, on average, m calculations (assuming a worst-case scenario where all m values still remain in the domain), resulting in m calculations in order to rank the value domain. The total average complexity at each decision point is thus $O(n + m)$ since this occurs at each decision point, assuming on average b branching decisions to be made in the tree, the average algorithm complexity $O(bn + bm)$.*

The computational complexity of decision points followed by a global ranking algorithm can be described as follows:

Definition 4.0.2 (Computational complexity of global ranking) *Given a network $N = (X, D, C)$, let n be the cardinality of the set X , that is $n = |X|$, and let m be the cardinality of the domain for a selected variable x_i , $m = D(x_i)$. Then, each variable value combination is required to be ranked based on the criteria provided in a network with no assigned variables, with n variables and an average of m values per domain variable, which produces nm variable value combinations. Finding the best ranked variable-value combinations at each decision point requires $n.m$ operations, and the total algorithm complexity is, on average, $O(bnm)$ where b is the number of branches explored.*

Both procedures described in Definitions 4.0.1 and 4.0.2 assume that no ranking information is carried forward from one branching decision to the next. This greatly simplifies the overall algorithm, but comes at a computational cost. As an example, if the ranking were to be produced at the initial root node and only variables affected by the propagation procedure are updated, a priority queue of variables could be maintained which would require $O(1)$ complexity to evaluate the next decision point and $O(\log n)$ to maintain the queue at each propagation point, where n is number of variables affected by propagation. In a worst-case scenario where all variables are affected by a single propagation, this would result in $O(n \log n)$ updates to the priority queue which results in a higher computational complexity than evaluating a simple rank function at each decision point. In an ideal world, choosing the smaller of these two complexities would be ideal, but as they are contingent on the problem instances, the simpler ranking procedure described in Definition 4.0.1 is employed.

4.1 The ranking function

Definition 4.0.1 provides an explanation of the ranking algorithm used to sequentially determine the next variable and value assignment in a CP search procedure. The possible components of the ranking function are described in the next section. The ranking functions are a reflection of the trees contained in a candidate individual in the GP population due for evaluation. The individual in the GP population is not a single tree, but rather four trees, each describing a different procedure or parameter in the CP search process.

The first tree describes the ranking process for variables and a description of its available functions is provided in Table 4.1. Similarly, the functions available for ranking the values of a given variable is provided in Table 4.2.

Function Name	Type	Arity	Description	Abbreviation
Add	Node	2	Simple addition	+
Mul	Node	2	Simple multiplication	*
Inv	Node	1	Protected inversion	Inv
Neg	Node	1	Negation	–
Rand	Terminal	0	Random uniform number	rand
MinX	Terminal	0	The minimum feasible value in the domain of variable x	Min(x)
MaxX	Terminal	0	The maximum feasible value in the domain of variable x	Max(x)
SizeX	Terminal	0	The feasible domain size of variable x	Size(x)
SuccessRateX	Terminal	0	The success rate of value assignments to variable x	SuccRate(x)
ImpactX	Terminal	0	The (sum of) impact of variable x	Impt(x)

TABLE 4.1: Description of the node and terminal set used in Tree 1, the variable selector.

The third and fourth tree functions, used to parametrise the CP search, contain exactly one terminal node. These singleton terminal trees operate in a similar fashion to the *Ephemeral Random Constant* (ERC) commonly used in GP, in that a value is generated at random and persisted for subsequent generations. The difference is that the domain of the random constant is defined along the set of integers where each integer maps to a parameter setting in the CP search process. Tree 3 specifies the boolean parameter for whether the last conflict observed in the search should be used as a hint during the next branching decision. Tree 4 specifies the

Function Name	Type	Arity	Description	Abbreviation
Add	Node	2	Simple addition	+
Mul	Node	2	Simple multiplication	*
Inv	Node	1	Protected inversion	Inv
Neg	Node	1	Negation	–
Rand	Terminal	0	Random number (uniform)	rand
LocalImpactValueX	Terminal	0	The local impact of the assignment of value v to variable x	Limpt(x,v)
NumberOfFailsValueX	Terminal	0	The number of fails of the assignment of value v to variable x	Fails(x,v)
NumberOfInstantiationsValX	Terminal	0	The number of instantiations of value v to variable x	Inst(x,v)

TABLE 4.2: Description of the node and terminal set used in Tree 2, the value selector.

log-restart rate of the search on the range $[-1, \dots, 17]$. Mutation operators on Trees 3 and 4 will generate new random values sampled from these domains.

The protected inversion operator produces similar results to the standard division operator, except that when the denominator is zero, this operator provides a value of one in order to avoid non-numeric values being produced by the tree. Recall that the arity of nodes refers to the number of children each node must have. Nodes with an arity of zero are referred to as terminals (or leaf nodes).

4.2 GP objective functions

This section contains a description of the objective function used by the GP to evaluate the quality of search strategies generated. The objective consists of three components which may be modelled as a single-objective problem using SAW to create a single term to be minimised. The alternate is to view the problem as a multi-objective optimisation problem in which each objective function is treated independently, without requiring an explicit trade-off to be specified between terms. The definition of the measures used in each term and normalisation procedures are provided in this section.

In order to build an informative objective function for the GP, maximum depth achieved in the search conducted is measured. If the depth reaches the total number of variables in the search, a feasible solution (and zero cost) has been achieved. The intuition is that the larger the number of variables that are feasibly allocated, the closer the search is to uncovering a feasible solution. A possible flaw to this measurement of success is that one could be introducing a thrashing bias in the solver, in that the solver favours continually pursuing a particular portion of the search space which is close to feasible, thereby avoiding restructuring the problem in such a way as to assert infeasibility or feasibility. An example of such restructuring may be to identify a clique of nodes which should be favoured earlier in the search tree. Although the potential of a thrashing search is a valid concern, the metric employed is an aggregate measurement of overall success and does favour searches which are perceived to attain higher levels of feasibility, without directly identifying the variables in the search. As such, the search depth metric does not encourage thrashing, but also does not discourage thrashing; rather, reliance on measuring the strategy against other search strategies is applied in order to determine which strategy is more effective, in which case a thrashing strategy would be removed from the population during later GP iterations.

The set of all problem classes is denoted by $P^{(\mathcal{M})}$, with individual problem classes being specified as $P^{(m)}$. A problem class will typically contain five problem instances used for training, but for generality, a problem instance within a class is denoted by $p_i^{(m)}$ where $i \in \{1, \dots, n\}$. A candidate branching strategy for a problem class is denoted by $P^{(m,b)}$ where b is candidate strategy. The total number of variables for problem instance i is denoted by V_i and the maximum depth achieved on problem instance i is denoted by d_i . The maximum search depth can thus easily be transformed and normalised to a measurement that describes the remaining infeasibility, which is to be minimised. If all variables are assigned during the search, a perfect score of zero is achieved and, conversely, if no feasible variable assignments are found (albeit highly unlikely) a score of 1 is given by

$$f_1 \left(P^{(m)} \right) = \sum_{i \in P^{(m)}} \frac{V_i - d_i}{V_i}. \quad (4.1)$$

The objective in (4.1) provides an equal weighting across small and large problem instances. Poor performance on larger problems will have a smaller influence on the objective function. It is possible to weight the problem instances according to their size so as to place a larger emphasis on the feasibility of large instances and reduce the cost of infeasibility on smaller instance, but it is unclear whether this is a preferable approach in general. Since an analysis will be conducted on an instance-by-instance basis, it would be reasonable to weight the success over such instances equally. Another potential motivation for the equal weighting scheme is that search success on smaller instances may lead to future success on larger instances not experienced within the sampling time, which would be reasonable if the strategy created was indeed generalisable to larger instances, although this may not be true in all cases.

A normalised objective result, in situations where a CSP instance has an objective function, may be obtained for a given instance based on the upper and lower values of the objective variable in the CSP. These upper and lower values are often bounded reasonably well as a result of initial propagation on the domains (*i.e.* they are not typically in the range $[-2^{63}, 2^{63}]$). CSPs conveniently express an objective function through a variable (defined as a normal integer variable in the CSP) which is referred to as o_i , the objective function value obtained for CSP instance i . The maximum and minimum value of the domain of o_i is denoted by o_i^{max} and o_i^{min} , respectively. The direction of the objective function, whether is to be minimised or maximised, is often referred to as the objective *sense*. Depending on the objective sense measured, the objective value obtained is relative to one of the two domain bounds o_i^{max} or o_i^{min} and written as

$$f_2(P^{(m)}) = \sum_{i \in P^{(m)}} \frac{f(o_i)}{o_i^{max} - o_i^{min}}$$

$$\text{where } f(o_i) = \begin{cases} o_i^{max} - o_i, & \text{if sense is max} \\ o_i - o_i^{min}, & \text{otherwise.} \end{cases} \quad (4.2)$$

If no feasible incumbent is found, a function value of 1 is returned simply by setting the value of o_i to o_i^{max} or o_i^{min} , depending on the sense. A zero result is returned for the function in (4.2) for instances where a CSP does not have an objective function as o_i does not exist in this context. The expression for f_2 in (4.2) ensures that all instance solution values are normalised to the domain $[0, 1]$ which is consistent with the normalisation range used for (4.1).

The last measurement desired is that related to how quickly a problem instance may be solved. As a result of imposing a short sampling time on each instance $p_i^{(m)}$, it would be typical that an attempt completes in neither an optimality nor infeasibility proof. In the case where an attempt to find a feasible solution does complete within the sampling time, however, this would be considered a very attractive solution approach¹. The sampling time is denoted by T , the maximum amount of time available to a search instance $P_i^{(m)}$, and the time used by the search process is denoted by t_i . The same normalisation procedure can be applied where the objective is to minimise the proportion of time used to achieve a result given a budget T , which can be written as

$$f_3(P^{(m)}) = \sum_{i \in P^{(m)}} \frac{t_i}{T}. \quad (4.3)$$

¹Or the problem may simply be too easy to solve, in which case most branching strategies will provide a solution within the sampling limit.

It would be trivial to extend T to have an allowable budget per problem instance but it will be assumed to remain constant unless otherwise specified. The definition of f_3 in (4.3) ensures that results for each instance $p_i^{(m)}$ are normalised on the domain $[0, 1]$.

There are some obvious correlations among these minimisation objectives. As an example, if (4.1) takes on a zero value, (4.3) will always take on a value less than one for pure CSP problems. Similarly, when (4.1) takes on a zero value, (4.2) will be guaranteed to be less than one. Lastly, when (4.3) has a value less than one, there is a guarantee to have a value of zero for (4.1) and a value less than one for (4.2). In an ideal situation, the functions in (4.1) and (4.2) both take on a value of zero, resulting in an optimality or infeasibility proof, and the remaining objective component is to minimise the time taken to achieve this result. The final objective function used for the single-objective GP is a SAW scheme with a unit weight on each term, that is

$$\text{minimise } f_1 \left(P^{(m)} \right) + f_2 \left(P^{(m)} \right) + f_3 \left(P^{(m)} \right). \quad (4.4)$$

It would be reasonable to question the use of a unit weight for each of (4.1), (4.2) and (4.3) in the formation of (4.4). Due to the correlation structure between these objectives already discussed, it would seem unnecessary to specify a more complex weighting scheme in this context since either extreme (very poor or very good solution quality) will result in arrival at the extremes of the domain for (4.4), namely, $[0, 3]$ within any one problem instance and $[0, 15]$ globally. A problem arises when considering the algorithmic progress between these extremes. The SAW scheme weights the importance of feasibility and time equally — in a pure CSP example where some subset of $P^{(m,a)}$ are feasible for strategy a (completing with small values for (4.1) and (4.3)) while others are infeasible (with poor values for (4.1)) and this branching solution is compared with a strategy b , $P^{(m,b)}$, which has no feasible solutions but is generally close to achieving a feasible outcome, where the sum over instances contained in $P^{(m,a)}$ may be greater than the sum over the instances contained in $P^{(m,b)}$. This means that $P^{(m,a)}$ would be preferred over $P^{(m,b)}$, which is perhaps not true. Given that if a sample of a longer search is performed, it may be found that with a sufficient time budget, the strategy of $P^{(m,b)}$ produces better results. The argument here can be made in either direction by simply tweaking the degree of feasibility achieved for (4.1). This example illustrates the potential bias towards different search strategies based on this weighting scheme.

It would be possible to mimic a tiered hierarchical objective function by applying weights and offsets in the construction of (4.4) such that the point at which a trade-off between objectives becomes negligible with the emphasis being placed on first minimising (4.1), then (4.2) and so forth. One may argue that this may decrease population diversity as it creates a myopic objective function where a large emphasis is then placed on each successive objective term. The risk is that some strategies which may be exploring certain features of the local search space are disregarded early on as they are aggressively dominated (in cost) by a single strategy that performs well in initial generations of the GP. There are several works which highlight the importance of having population diversity in initial populations that are similar in cost to avoid premature convergence in GAs.

The encapsulation of metrics for a given series of CSP solution attempts by objective (4.4) may be fallible, but it is assumed that it is sufficiently representative of the behaviour the GP is encouraged to find. There are situations where the metrics of (4.4) correctly identify superior strategies and situations where it is unclear that a particular series of solutions is better than another.

The number of times these more difficult comparisons are encountered during a GP run is unknown *a priori* and in the absence of a superior one-dimensional objective function, having a simplistic solution in hand provides, at minimum, a potential solution approach.

It is prudent to note that, without casting further doubt on the SAW objective proposed by (4.4), that there are no assurances as to the convexity of the search space when considering the sum of the objective terms. It is clear at this juncture that this problem lends itself to a multi-objective approach which can potentially produce a more favourable transition of solutions to an optimal policy. The multi-objective version of the problem is to

$$\text{minimize } f_k \left(P^{(m)} \right), k \in \{1, 2, 3\}. \quad (4.5)$$

To the author's best knowledge, a multi-objective approach has not been adopted in the literature for the development of search strategies using GP.

4.3 GP configuration parameters

The GP configuration predominantly conforms to the standard Koza parameter configurations for initial tree seeding, mutation and crossover. A population of 80 individuals and 15 generations was used for the experiments. The number of generations considered here is somewhat unconventional but, during initial testing it was found that with the small grammar used, convergent strategies evolved quickly for certain structures and additional generations did not result in significant gains. A summary of GP parameters employed is provided in Table 4.3.

Parameter	Value	Description
Generations	15	The number of population iterations performed by the GP.
Population size	80	The size of the population during each iteration.
Elitism	Yes	Ensures that the best individual from each population iteration is carried forward to the next generation.
Tournament size	2	The value used here deviates from the traditional value of 7 in GP. The higher the value, the higher the selection pressure to disregard poor individuals early on in the GP search.
Crossover rate	0.9	The probability that a node in the tree is selected for a crossover operation. Crossover exchanges the two resulting subtrees between two individuals at compatible nodes.
Maximum crossover depth	17	The limit on the selection of nodes for crossover.
Mutation rate	0.1	The rate at which subtree mutation is applied to tree nodes.
Mutation type	Subtree	Subtree mutation invokes a call to the <i>Grow</i> method which randomly creates a new subtree of depth 5.
Non-terminal Selection Probability	0.9	When generating new trees, the probability that a node with non-zero arity is selected at random.
Terminal Selection Rate	0.1	When generating new trees, the probability that a terminal is selected at random.
Seeding mechanism	50/50 Grow/Full	Also referred to as the half-builder. Half the population is seeded using the <i>Grow</i> method and the other half through the <i>Full</i> method.
<i>Full</i> depth range	[2,6]	The target range of minimum and maximum depths for a tree created using the <i>Full</i> method.
<i>Grow</i> depth range	[5,5]	The target range of minimum and maximum depths for a tree created using the <i>Grow</i> method.

TABLE 4.3: GP parameters for single-objective optimisation.

Important variations from the ‘‘Koza defaults’’ other than the number of generations considered is the tournament size being reduced from 7 to 2. A value of 2 is closer to the typical tournament size used when working with bit-string representations in traditional GAs in order to avoid

premature convergence. The crossover operator is very aggressive in a typical Koza-style GP, which means that even though a population may converge quickly on a particular derivative of an individual, the crossover operator is able to create two radically different individuals from the same individual. Producing two new individuals from the same individual employing crossover is not possible in the bit-string representation. The change in tournament size was found to bring about a small improvement in the context of the GP runs performed in the proceeding experiments. There are certainly far larger gains in the GP performance that could be leveraged through hyper-parameter optimisation or simply a grid search, but in keeping with the philosophy of this dissertation, the GP running in this configuration provides sufficient quality for the present, albeit almost certainly sub-optimal, purposes.

Each individual in the population takes at most 60 seconds to evaluate, comprising 10 seconds of sampling time per problem instance for training and five instances per problem, with a few additional seconds of initial compiler time to build the program, where the same compiled program is used consistently for all five problem instances, and then clean up any dangling files. The total computational time is thus far greater (at most 11 times more) for the GP search than the maximum duration permitted for a problem set under the ORT default search process (20 minutes per instance and 5 instances per set). This would thus not be an ideal methodology to apply if the intention were to solve a problem instance type that will only be seen once. If, however, one is expecting small variations from a particular class of problem, the additional offline training time incurred to improve future online search efficacy may be conceded.

The multi-objective variation of the GP run employs the same parameters for crossover and mutation but uses a different selection algorithm (2.3) which maintains a Pareto frontier of best solutions found. This ability of the NSGAI algorithm to maintain a frontier of best solutions requires that the population size is adjusted. This is as a result of frontier solution always being carried forward to the next iteration and it is desirable to have a similar number of individual evaluations when performing comparisons with the simpler single objective scheme. During experimentation, it was found that a population size of 150 individuals produced a very close match to the target number of individual evaluations in the SAW GP run. This is as a result of the fact that individuals being carried forward from one population to the next have already been evaluated on the frontier and are hence not re-evaluated. A summary of the changes to the parameters of the GP run for multi-objective optimisation are provided in Table 4.4.

Parameter	Value	Description
Population size	150	The size of the population at each iteration, including the Pareto frontier.
Elitism	No	Elitism is indirectly maintained through the frontier being persisted from one iteration to another and the idea of an explicit “best” individual in multi-objective optimisation is not possible.
Tournament size	None	Selection is managed through the non-dominated sorting algorithm.

TABLE 4.4: GP parameters modified for multi-objective optimisation.

4.4 MiniZinc Challenge results (2015)

In the 2015 MiniZinc benchmark challenge, the ORT solver achieved a bronze award in the free-search category. The solver was beaten by two proprietary solvers, *Opturion CPX* and *iZPlus*. The former was developed and is maintained by *National Information Communications Technology Australia* (NICTA), who are also the sponsors of the MiniZinc challenge. The *iZplus* solver is developed and maintained by NTT Data Sekisui Systems Corporation in Japan. The

iZplus solver only ranks highly on the free-search category. Another interesting observation is that the iZplus small code base is incredibly small ($< 50\text{Kb}$) when compared with the ORT code base which totals 9Mb in core source files which excludes all third party integration files.

The ORT solver and Opturion CPX solver were the only two solvers to successfully rank among the top three in all search categories, as shown in Table 4.5. Opturion CPX has academic licensing for testing but the code base is not open source. ORT is the best performing open source solver available that competed in the 2015 MiniZinc Challenge.

	Fixed search	Free search	Parallel search	Open class
Gold Medal	Opturion CPX	Opturion CPX	OR-Tools	sunny-cp
Silver Medal	OR-Tools	iZplus	Opturion CPX	OR-Tools
Bronze Medal	JaCoP	OR-Tools	Choco	Opturion CPX

TABLE 4.5: *MiniZinc 2015 Challenge results.*

A simple table describing the aggregate features for each class of problem in the challenge can be found in Table 4.6. The *large-scheduling* instance was removed from the test set as initial testing with the MiniZinc to FlatZinc conversion required specific x64 binaries to handle the memory requirements for executing the conversion, which were not available at the time. Table 4.6 also provides a summary of the time spent in a sample of evolved ranking functions, indicating that the ranking approach employed consumes approximately 20–30% of the search time on larger or more complex problems.

Problem Class	Total Time	Ranking Time	Time in Ranking Function (%)	Average Number of Variables	Average Number of Constraints	Average Constraints Per Variable
costas-array	2 410 379	138 859	5.76 %	18	1 143	63.5
cvrp	5 611 437	264 521	4.71 %	559	26 304	47.06
freepizza	4 933 955	85 012	1.72 %	144	25 927	180.05
gfd-schedule	1 478	319	21.55 %	6,098	15 590	2.56
grid-colouring	2 402 718	903 652	37.61 %	119	1 686	14.12
is	3 394 905	166 791	4.91 %	724	1 450	2
mapping	6 000 022	343 514	5.73 %	528	803	1.52
multi-knapsack	1 259 700	372 872	29.60 %	60	18	0.3
nmseq	1 155	14	1.21 %	226	679	3
opd	6 000 065	1 486 882	24.78 %	4 690	8 918	1.9
open_stacks	6 000 088	336 189	5.60 %	708	1 623	2.29
p1f	6 000 145	1 390 207	23.17 %	2 830	25 748	9.1
project-planning	6 000 157	1 511 800	25.20 %	28 205	1 004	0.04
radiation	6 000 029	144 298	2.40 %	1 186	1 016	0.86
roster	130	71	54.62 %	404	429	1.06
spot5	6 000 034	1 368 971	22.82 %	5 441	12 137	2.23
tdtsp	2 414 174	2 5928	1.07 %	95	662	6.97
triangular	6 000 030	1 283 092	21.38 %	310	2	0.01
zephyrus	5 272 551	391 702	7.43 %	371	800	2.15

TABLE 4.6: *Percentage time spent in the ranking function per search by problem class for ranking Definition 4.0.1.*

An experimental treatment of the average time spent evaluating the ranking functions using Definition 4.0.1 is provided in Table 4.6 and Figure 4.2, organised by benchmark problem. This statistic is provided to quantify the computational cost of the ranking function relative to the total computation time by problem instance. It is worth noting that some trivial instances overstate the relative cost of the ranking function.

In order to make comparisons between the default ORT search, and an alternate search process, the default ORT search has been re-run on hardware that will remain constant for all computational experiments performed. A summary of the results of the default ORT search on the MiniZinc Challenge data set is given in Table 4.7.

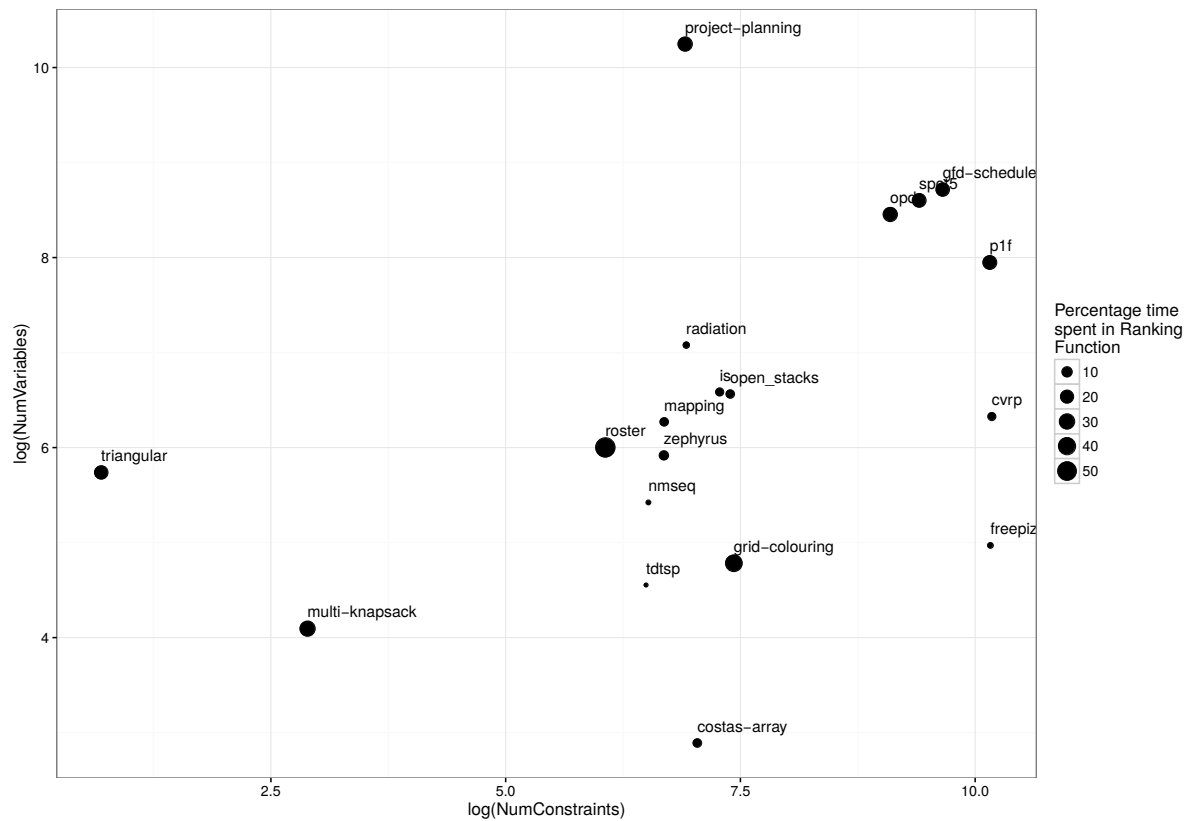


FIGURE 4.2: Percentage of search time spent in the ranking function grouped by problem class.

	C	S	SC	UNK
costas-array	0	3	0	2
cvrp	0	4	1	0
freepizza	0	4	1	0
gfd-schedule	0	1	1	3
grid-colouring	0	3	2	0
is	0	4	1	0
mapping	0	4	0	1
multi-knapsack	0	1	2	2
nmseq	0	5	0	0
opd	0	3	2	0
open_stacks	0	3	2	0
p1f	2	2	0	1
project-planning	0	5	0	0
radiation	0	0	5	0
roster	0	0	5	0
spot5	0	5	0	0
tdtsp	0	5	0	0
triangular	0	4	1	0
zephyrus	0	4	1	0

TABLE 4.7: ORT solve status code summary for the MiniZinc Challenge 2015 problem instances by class.

The summary provided in Table 4.7 demonstrates that the majority of problem instances have feasible solutions with several cases for which the search was able to admit an optimality proof. Several instances receive a UNK status code. In the cases of mapping, gfd-schedule, multi-knapsack and costas-array, the search is attempting to find a feasible incumbent, which does indeed exist. In the case of p1f, the search is attempting to provide an infeasibility proof which it is unable to complete within the time limit. As the curators of these data sets, additional insight to the search procedure is available; hence, this is not a criticism, merely an observation as to the search behaviour on these instances.

A reader who is unfamiliar with the details of these problem instances may conclude that they may be trivial to solve in commercial-grade solvers. Table 4.8 illustrates the performance of Cplex and Gurobi on the first two problems in the benchmark data which consist of a pure feasibility problem and a constrained optimisation problem. The cvrp is the classic *Capacitated Vehicle Routing Problem* (CVRP) and the data sets here are taken directly from the literature where optimal solutions to these CVRP problem instances have been found.

Problem Name	Problem Instance	Gurobi-free			CPLEX-free			ORT-free		
		Status	Time (s)	Objective	Status	Time (s)	Objective	Status	Time (s)	Objective
costas-array	16	UNK			UNK			S	19	
	17	UNK			UNK			S	240	
	18	UNK			UNK			S	17	
	19	UNK			UNK			UNK		
	20	UNK			UNK			UNK		
cvrp	A-n37-k5.vrp	S	1 200	1 802	UNK			S		1 642
	A-n64-k9.vrp	UNK			UNK			S	1 200	3 486
	B-n45-k5.vrp	S	1 200	2 425	UNK			S	1 200	2 728
	P-n16-k8.vrp	S	1 200	460	S	1 200	470	S	1 200	502
	simple2	SC	250	34	SC	423	34	SC	377	34

TABLE 4.8: *Cplex, Gurobi and ORT performance on costas-array and cvrp.*

Table 4.8 is somewhat disingenuous since MIP environments are well suited to problems with good linear relaxations to guide the search, which is not a property of costas-array and as such is perhaps a pathological worst-case for MIP in reality. While relaxations for the family of cut inequalities for the CVRP are well studied, being able to infer such families of inequalities from the FlatZinc format is a complex task for Cplex and Gurobi. The results achieved by these engines on the CVRP problems is rather impressive as the optimality proof in this context relies heavily on lift-and-project procedures within the MIP engines. It should be clear from the table that the ORT CP solver favours finding a feasible incumbent within the time limit.

Many of the instances used in the challenge data set are optimisation problems, so while the results in Table 4.7 look promising, one should consider comparing the best objective values found where an optimality proof is incomplete, *i.e.* in cases where the time limit was reached with a feasible solution but without an optimality proof. The costas-array and mnseq problem classes are the only pure satisfaction problems in the benchmark set, although there are a handful of optimisation problems which are infeasible by design, such as p1f.

The detailed results of the ORT run on the MiniZinc Challenge data are provided in Table 4.9. The objective sense of instances, number of variables and constraints, the normal and delayed propagations performed, as well as the number of branches explored in the search are provided. The times of the search, measured in seconds, correspond to reaching the 20 minute time limit where 1 200 seconds are displayed. At this time limit, if a feasible solution for optimisation problems is found, the best solution obtained during the search is provided in the last column, alternatively, if no solution is found the column is left blank. When a search completes within the time limit, either an optimality proof is provided, a feasible solution is found (such as costas-array 16, 17 and 18) or an infeasibility proof is provided (such as p1f 13 and 15). For an

4.4. MiniZinc Challenge results (2015)

Problem Class	Instance Name	Objective	Variables	Constraints	Solutions	Normal Propagations	Delayed Propagations	Branches	Status Code	Time (s)	Best Solution
costas-array	16	None	16	802	1	191 398 645	0	782 098	S	19	
	17		17	954	1	2 341 358 486	0	8 227 602	S	240	
	18		18	1 124	1	166 689 662	0	513 017	S	17	
	19		19	1 313	0	16 475 239 187	0	44 434 299	UNK	1 200	
	20		20	1 522	0	13 079 245 233	0	38 535 465	UNK	1 200	
cvrp	A-n37-k5.vrp	Min	610	23 002	3	30 452 120 000	1 406 766 973	17 665 973	S	1 200	1 642
	A-n64-k9.vrp		1 069	69 172	4	25 755 608 001	68 489 4 180	4 030 830	S	1 200	3 486
	B-n45-k5.vrp		746	34 098	2	33 294 150 000	1 513 800 832	12 002 541	S	1 200	2 728
	P-n16-k8.vrp		253	4 228	6	36 337 060 000	1 241 581 978	16 072 755	S	1 200	502
	simple2		117	1 020	7	7 522 686 703	372 915 238	11 579 374	SC	377	34
freepizza	pizza27	Min	180	32 600	197	6 557 856 443	0	6 421 748	S	1 200	882 425
	pizza39		190	36 890	440	6 125 497 876	0	4 708 241	S	1 200	939 352
	pizza45		140	19 759	113	6 420 602 070	0	4 424 374	S	1 200	656 489
	pizza6		10	159	26	4 920 134 527	0	72 097 340	SC	610	210
	pizza78		200	40 229	84	6 271 975 892	0	4 255 404	S	1 200	901 717
gfd-schedule	n120f5d50m50k20	Min	7 816	19 629	1	4 890 744 863	54 554 042	16 719 016	S	1 200	19 463
	n180f7d50m30k18		17 186	45 699	1	197 889	2 808	273	SC	1	1
	n30f3d30m7k4		616	1 374	0	7 385 581 939	434 271 695	62 330 829	UNK	1 200	
	n50f7d40m10k4		1 540	3 660	0	5 743 064 667	222 787 110	60 435 092	UNK	1 200	
	n75f5d30m20k20		3 129	6 782	0	9 800 257 631	535 814 081	130 804 014	UNK	1 200	
grid-colouring	10.5	Min	51	376	3	276 145 914	0	13 196 433	SC	31	3
	13.11		144	1 717	5	12 265 686 840	0	463 119 790	S	1 200	7
	19.17		324	5 815	6	12 172 310 000	0	316 800 403	S	1 200	12
	4.11		45	331	1	10 844 403 150	0	596 996 412	S	1 200	4
	4.8		33	193	2	11 844 953	0	752 041	SC	2	3
is	1YHXeG1xYs	Min	913	1 921	61	9 954 586 682	559 942 788	204 129 617	S	1 200	194 048
	A3PZaPjnUz		507	926	12	5 263 264 865	1 381 701 878	293 871 511	S	1 200	144 896
	HgSWGJHxY5		835	1 680	13	7 695 431 786	1 033 660 940	122 541 697	S	1 200	251 200
	jZ9pQqRxJ2		508	799	4	306 847 009	118 916 853	6 939 065	SC	82	210 944
	y21PnVA2Hj		860	1 840	3	10 970 336 136	516 250 798	260 530 636	S	1 200	236 544
mapping	full2x2	Min	172	235	1	8 029 451 396	2 411 746 111	194 166 666	S	1 200	1 103
	mesh2x2_mpeg		522	803	32	9 998 692 823	2 817 622 175	186 924 858	S	1 200	726
	mesh3x3_2		348	497	0	9 229 233 331	1 364 910 563	64 292 743	UNK	1 200	
	mesh3x3_mpeg_2		1 302	1 709	1	12 260 773 508	1 078 060 421	173 812 616	S	1 200	2 197
	ring_2		300	473	4	10 079 393 980	1 504 144 982	32 042 345	S	1 200	2 090
multi-knapsack	mknap1-6	Max	50	8	0	11 843 744 497	0	656 062 586	UNK	1 200	
	mknap2-1		60	33	1	2 674 396 943	0	16 659 025	SC	158	7 772
	mknap2-2		60	33	1	20 218 233 594	0	174 664 239	S	1 200	8 722
	mknap2-20		50	8	1	27 058 095	0	870 918	SC	3	6 339
	mknap2-32		80	8	0	20 034 771 977	0	854 633 646	UNK	1 200	
nmseq	176	None	177	530	1	47 959 761	0	1 756	S	6	
	207		208	623	1	45 174 034	0	2 769	S	7	
	269		270	809	1	172 372 041	0	2 947	S	45	
	393		394	1 181	1	539 624 105	0	4 513	S	244	
	83		84	251	1	6 886 412	0	2 202	S	1	
opd	fIener_et_al.10_350.100	Min	15 349	28 405	45	4 968 217 290	0	17 594 326	S	1 200	65
	medium.10.100.30		4 349	8 155	20	7 699 735 130	0	97 864 853	S	1 200	13
	small_bibd.10.30.09		1 269	2 485	8	12 099 573 239	0	181 111 979	SC	1 107	2
	small_bibd.11.22.10		1 021	2 161	6	10 163 769 222	0	155 917 747	S	1 200	5
	small_bibd.13.26.06		1 465	3 385	6	7 642 879 621	0	86 146 711	SC	1 078	1
open_stacks	problem.20.20.1	Min	744	1 668	8	5 790 248 031	1 569 596 577	16 865 690	S	1 200	11
	problem.30.15.1		783	1 689	8	71 958 186	13 645 126	246 705	SC	15	14
	wbo.10.20.1		379	949	4	1 528 228 251	542 228 901	9 358 559	SC	303	5
	wbp.15.30.1		899	2 205	7	3 439 315 800	2 281 187 446	16 015 458	S	1 200	7
	wbp.20.20.1		739	1 606	8	5 626 368 145	2 773 046 128	72 481 913	S	1 200	4
p1f	12	Min	1 617	12 309	13	10 863 703 783	1 177 500 003	10 399 042	S	1 200	602
	13		2 070	16 812	0	99 374 606	19 426 666	269 539	C	20	
	14		2 600	22 438	1	9 576 526 965	904 655 384	6 464 251	S	1 200	1 008
	15		3 213	29 358	0	404 590 035	67 476 687	1 353 028	C	68	
	17		4 712	47 824	0	5 344 794 617	1 159 396 380	5 639 655	UNK	1 200	
project-planning	ProjectPlannertest_12.7	Min	3 463	693	1	5 970 010 508	403 548 824	146 599 158	S	1 200	63
	ProjectPlannertest_14.7		12 801	920	2	4 824 746 889	178 456 224	110 895 911	S	1 200	78
	ProjectPlannertest_15.6		25 156	1 050	4	6 689 857 295	1 560 211 759	87 387 937	S	1 200	66
	ProjectPlannertest_16.6		49 803	1 178	22	9 559 125 712	3 810 697 453	84 382 648	S	1 200	39
	ProjectPlannertest_16.8		49 803	1 180	22	10 278 994 684	4 008 504 749	88 446 232	S	1 200	39
radiation	i14-9	Min	2 467	2 051	1	919 346 438	240 087 512	45 500 146	SC	252	6 513
	i6-11		544	495	1	1 064 509 866	249 124 246	35 338 011	SC	207	895
	i6-21		1 036	919	1	708 451 773	151 837 198	28 020 462	SC	143	1 413
	i7-9		619	548	1	367 416 685	8 939 838	1 262 819	SC	7	1 007
	i9-11		1 265	1 066	1	2 130 904 745	512 731 923	77 059 018	SC	427	2 141
roster	chicroster_dataset.11	Min	559	564	1	2 348	0	118	SC	1	17
	chicroster_dataset.17		671	676	1	3 069	0	142	SC	1	17
	chicroster_dataset.2		189	248	2	1 426	0	40	SC	0	0
	chicroster_dataset.5		279	294	1	1 102	0	51	SC	1	6
	chicroster_dataset.7		323	363	2	2 856	0	80	SC	1	0
spot5	1401	Min	10 964	24 078	1	1 669 407 350	1 648 595 524	1 728 768	S	1 200	521 097
	28		5 227	10 642	8	94 439 127	2 627 783 788	3 426 424	S	1 200	284 158
	414		10 109	24 373	473	16 124 328	1 791 590 000	554 344	S	1 200	42 564
	503		636	1 130	50	417 908 191	4 045 144 727	33 733 672	S	1 200	15 177
	54		272	462	24	468 973 786	3 569 984 882	33 339 298	S	1 200	81
tdtsp	inst.10.24.10	Min	67	358	13	17 139 770 000	3 206 315 367	209 431 603	S	1 200	13 917
	inst.10.34.00		67	358	7	16 549 663 734	3 097 985 077	188 378 833	S	1 200	8 353
	inst.10.42.10		67	358	1	18 543 048 516	3 510 784 587	179 242 916	S	1 200	15 329
	inst.20.14.10		137	1 118	15	29 050 320 000	2 380 920 989	43 493 411	S	1 200	17 449
	inst.20.25.00		137	1 118	63	14 197 690 000	1 210 287 736	5 845 232	S	1 200	19 898
triangular	n10	Max	55	2	4	310 340 311	0	73 598 326	SC	89	20
	n16		136	2	7	3 394 796 669	0	814 315 972	S	1 200	35
	n22		253	2	8	2 652 581 584	0	606 564 029	S	1 200	48
	n28		406	2	9	2 357 918 482	0	548 026 336	S	1 200	61
	n37		703	2	10	1 749 534 804	0	382 587 947	S	1 200	80
zephyrus	zephyrus.15.10	Min	462	995	1	9 456 200 000	159 295 917	45 330 224	S	1 200	36
	zephyrus.20.20		612	1 320	1	8 791 286 874	208 143 293	53 058 847	S	1 200	66
	zephyrus.5.20		162	345	1	7 818 222 641	593 583 292	133 994 952	S	1 200	66
	zephyrus.5.4		162	345	1	9 066 156 549	411 065 353	125 834 064	S	1 200	18
	zephyrus-FH-2-15		461	995	1	2 057 573 894	86 397 282	3 245 735	SC	237	12

TABLE 4.9: ORT Minizinc Challenge 2015 run results (base case calibration).

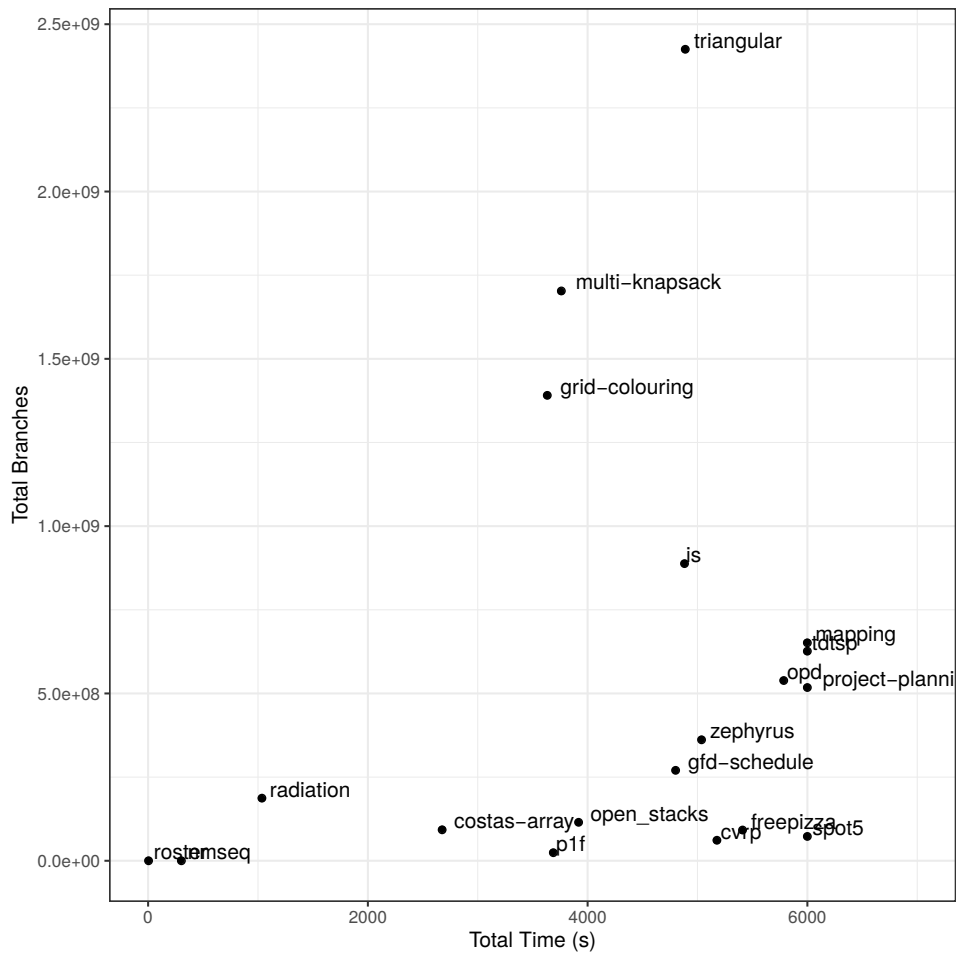


FIGURE 4.3: Default ORT branching speed aggregated by problem class.

indication of the relation of the number of constraints for each of the problem classes relative to the number of variables, Figure 4.3 contains a visualisation of the branching speed on each of the different problem classes, which has been aggregated over the instances to obtain a general feel for the problem class complexity.

4.5 GP run results (Hypothesis 3.1.1)

A description of the results of the GP runs are provided in this section. Two approaches are used to perform the GP optimisation with respect to the objective function configuration, namely single-objective and multi-objective optimisation. In addition, there are also two related hypotheses regarding the available function set provided to the GP. A high-level analysis of the run results of the single-objective and multi-objective approaches for Hypotheses 3.1.1 and 3.1.2 are provided separately, followed by a comparison of the results across both hypotheses thereafter.

4.5.1 Classic CSP GP operators

This section details the result of using GP to evolve search strategies in terms of the arithmetic operator set in tandem with the graph operators in Tables 4.1 and 4.2. The results of the run using SAW for each problem set are shown in Figure 4.4. The mean population performance is represented by a dotted line and the best individual in each generation is represented by a solid line. All problems are orientated to be minimisation problems for consistency where the lowest possible SAW objective score attainable by the GP is zero. It can be seen that the roster problem is far easier to solve than the other problems in the set with an initial random strategy achieving a near-perfect score.

One may observe that the intuitive trend of a decreasing mean objective value towards the best found solution is observed through all of the runs. Interesting things to note are that in some instances, the mean increases after the initial population and sometimes the best-known solution also increases during the run. Some of the evolved strategies include stochastic terms which result in a low reproducibility of unlikely, or extreme minima, searches achieved.

Another consistent trend observed in several of the runs is that the best starting strategy is improved upon towards the end of the run. Improvements observed as a result of the scaling in Figure 4.4 may be deceiving in optimisation problems as the objective contribution is normalised against the domain of the objective variable, resulting in very small changes in the best solution for larger optimisation problems for which it is unreasonable to expect an optimality proof within 10 seconds of sampling time.

The multi-objective GP runs for Hypothesis 3.1.1 are provided in Figure 4.5. The plots are provided in three dimensions where all strategies are able to produce variation in all components of the objective function (4.5). The latter plots are provided in two dimensions as there was no variation in the omitted component values. Lastly, problem classes p1f and triangular were omitted from the plots as no strategies were able to produce variation in more than a single objective function component within the sampling limit². A consistent convention followed between the two multi-objective plots for Hypothesis 3.1.1 is that points which are concluded to lie on the Pareto frontier are coloured in blue. Points not on the Pareto frontier are coloured based on a heat scale indicating at which generation during the GP search the individuals were created with red and yellow being used for individuals created near the start and the end of the GP run, respectively.

The best performing population individual from each of these GP runs is provided an opportunity to perform a search with a time limit of 20 minutes. In the case of the multi-objective GP

²In the case of p1f this raises the question as to whether the sampling limit should be increased for larger problems in order to evolve meaningful strategies. The triangular problem class has a tendency to always admit a feasible solution, and consume the full time limit, leaving only the instance objective as a facet.

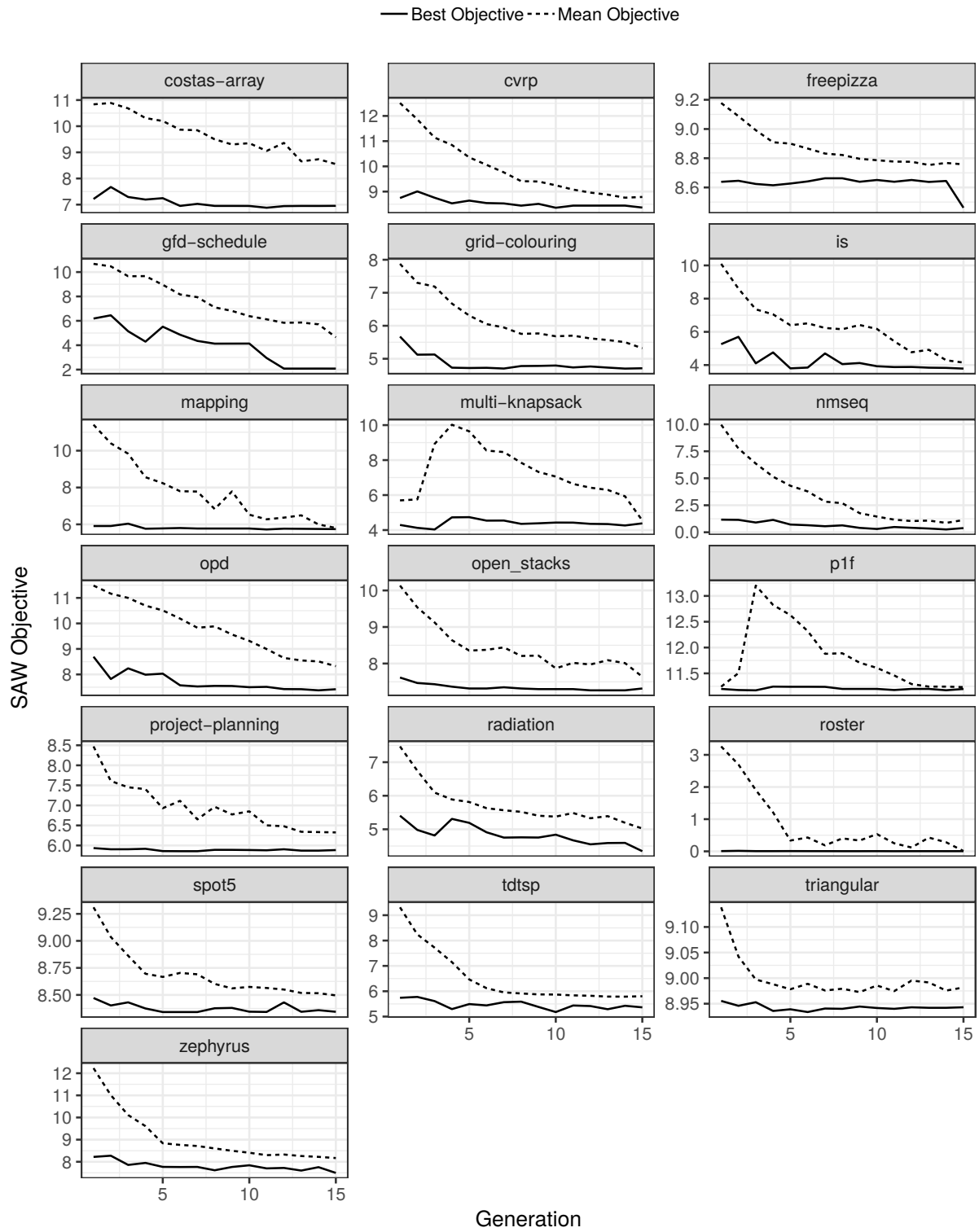
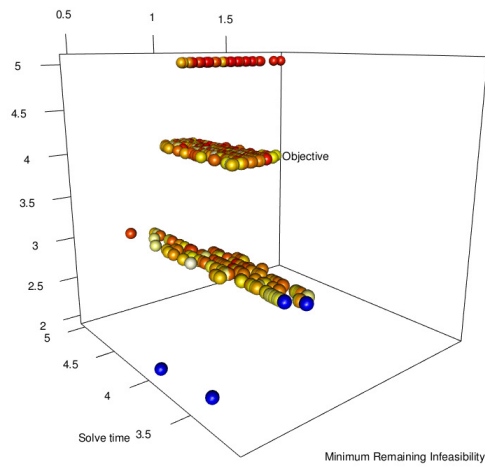
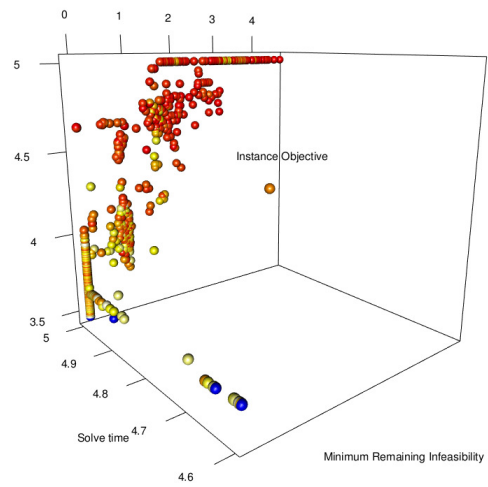


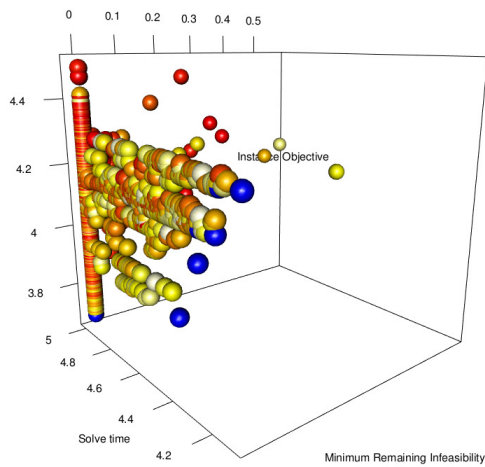
FIGURE 4.4: GP run results for Hypothesis 3.1.1 using SAW.



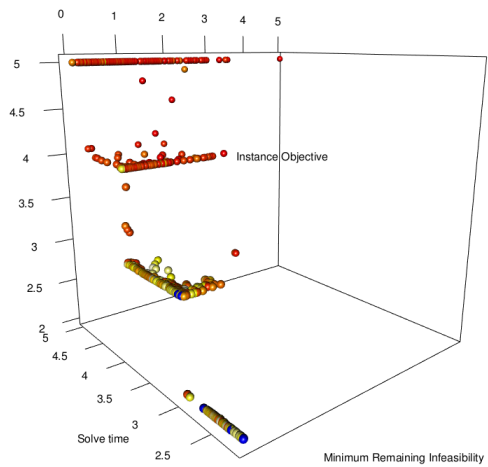
(a) *costas-array*



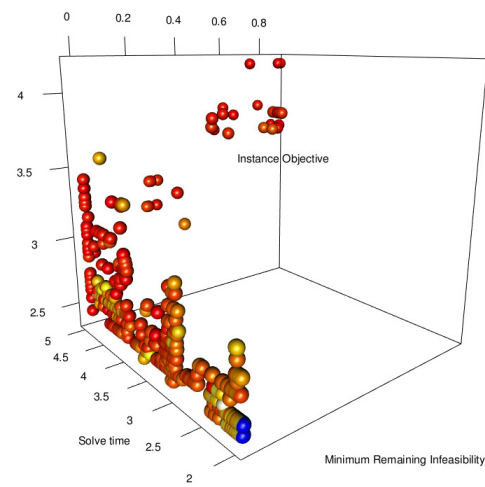
(b) *cvrp*



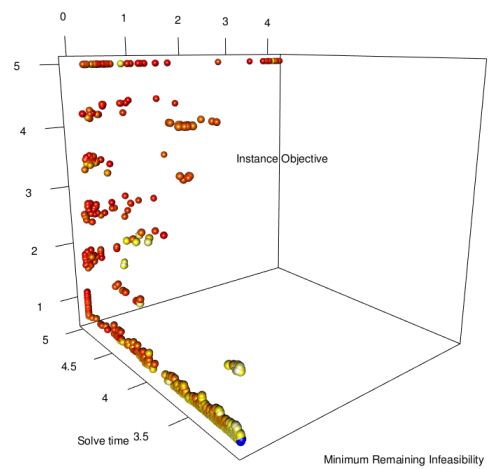
(c) *freepizza*



(d) *gfd-schedule*

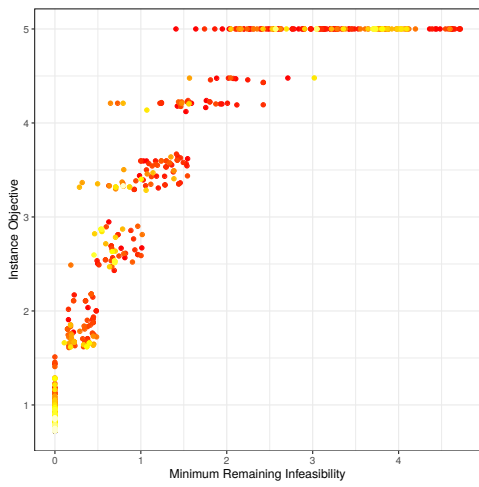


(e) *grid-colouring*

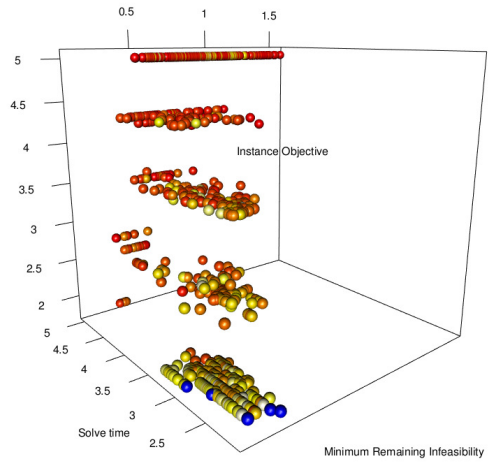


(f) *is*

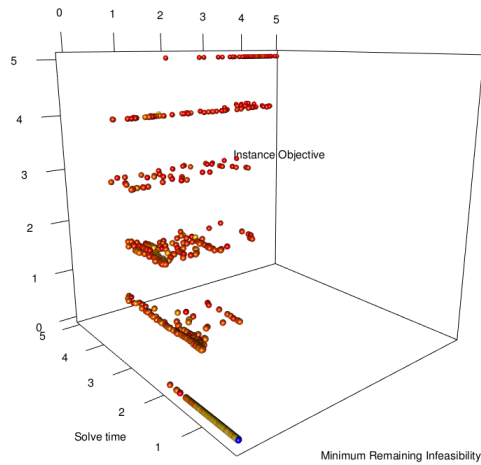
FIGURE 4.5: GP run results for Hypothesis 3.1.1 using NSGA-II.



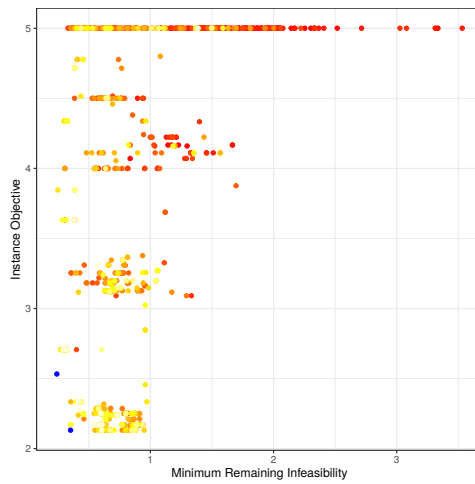
(g) *mapping*



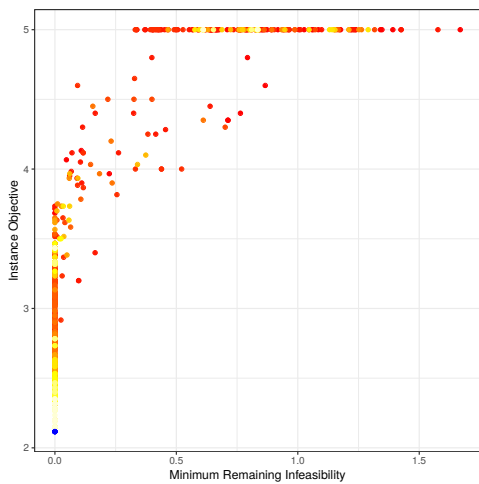
(h) *multi-knapsack*



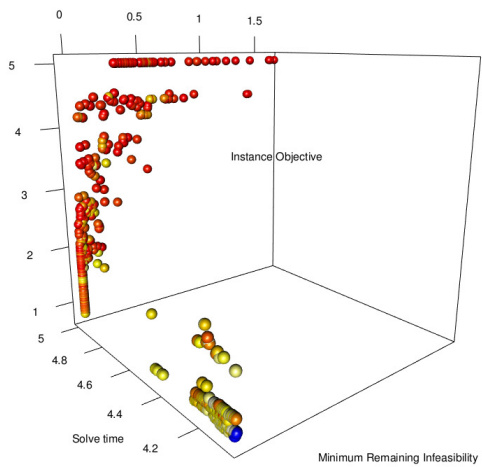
(i) *nmseq*



(j) *opd*

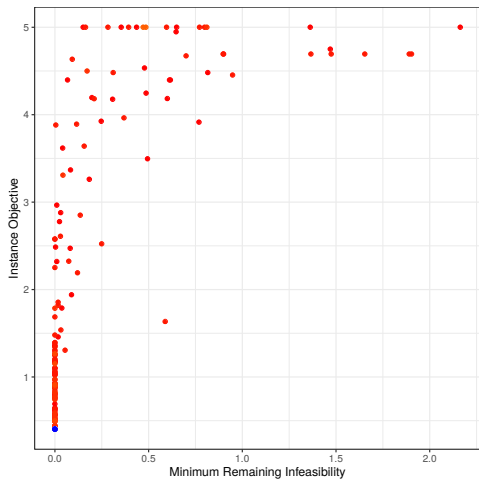


(k) *open_stacks*

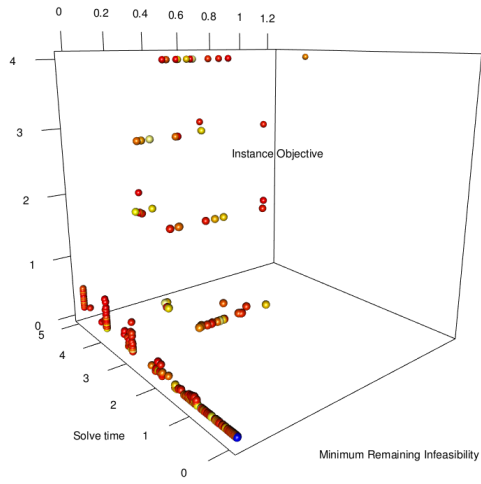


(l) *project-planning*

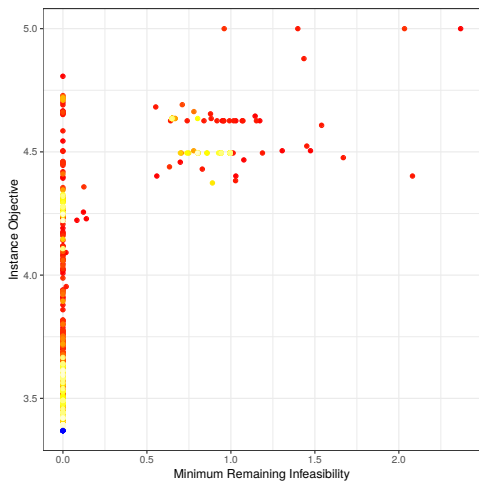
FIGURE 4.5: *GP run results for Hypothesis 3.1.1 using NSGA-II (continued).*



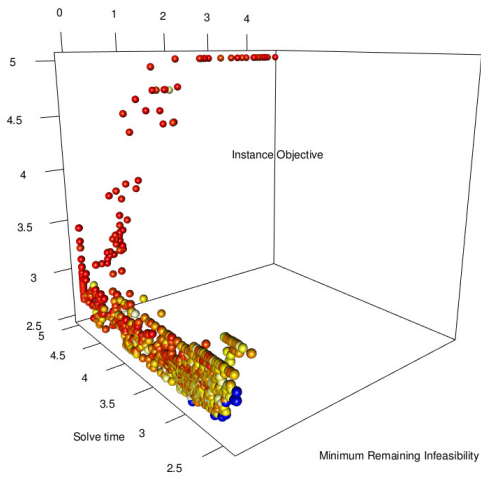
(m) radiation



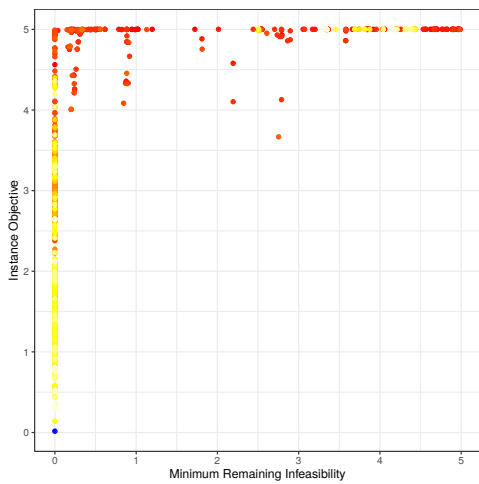
(n) roster



(o) spot5



(p) tdtsp



(q) zephyrus

FIGURE 4.5: GP run results for Hypothesis 3.1.1 using NSGA-II (continued).

Problem Class	Instance	ORT			Hypothesis 3.1.1 SAW					Hypothesis 3.1.1 multi-objective						
		Code	Time	Best	Code	Δ	Time	Δ	Best	Δ	Code	Δ	Time	Δ	Best	Δ
costas-array	16	S	19		S		1	+		S		6	+			
	17	S	240		S		1	+		S		7	+			
	18	S	17		S		364	-		S		8	+			
	19	UNK	1200		UNK		1200			S	+	105	+			
	20	UNK	1200		UNK		1200			UNK		1200				
cvrp	A-n37-k5.vrp	S	1200	1642	S	1200	1832	-		S	1200	2122	-			
	A-n64-k9.vrp	S	1200	3486	S	1200	3545	-		S	1200	4046	-			
	B-n45-k5.vrp	S	1200	2728	S	1200	2628	+		S	1200	2467	+			
	P-n16-k8.vrp	S	1200	502	S	1200	502			S	1200	450	+			
	simple2	SC	377	34	SC	30	+	34		SC	19	+	34			
freepizza	pizza27	S	1200	882425	S	1200	822299	+		S	1200	761294	+			
	pizza39	S	1200	939352	S	1200	987968	-		S	1200	837068	+			
	pizza45	S	1200	656489	S	1200	641397	+		S	1200	571934	+			
	pizza6	SC	610	210	SC	5	+	210		SC	113	+	210			
	pizza78	S	1200	901717	S	1200	896600	+		S	1200	714755	+			
gfd-schedule	n120f5d50m50k20	S	1200	19463	SC	+	1	+	1	+	SC	+	3	+	1	+
	n180f7d50m30k18	SC	1	1	UNK	-	1200	-		UNK	-	1200	-			
	n30f3d30m7k4	UNK	1200		SC	+	1	+	1		SC	+	1	+	1	
	n50f7d40m10k4	UNK	1200		SC	+	1	+	1		SC	+	2	+	1	
	n75f5d30m20k20	UNK	1200		SC	+	1	+	1		UNK	1200				
grid-colouring	10_5	SC	31	3	SC		1	+	3		SC		1	+	3	
	13_11	S	1200	7	S		1200		5	+	S		1200		4	+
	19_17	S	1200	12	S		1200		7	+	S		1200		5	+
	4_11	S	1200	4	SC	+	1	+	3	+	SC	+	1	+	3	+
	4_8	SC	2	3	SC		1	+	3		SC		1	+	3	
is	1YHXeG1xYs	S	1200	194048	S	1200	145440	+		S	1200	99328	+			
	A3PZaPjnUz	S	1200	144896	SC	+	1	+	103936	+	SC	+	1	+	103936	+
	HgSWGJHxY5	S	1200	251200	S	1200	276000	-		SC	+	1117	+	102176	+	
	jZ9pQqRxJ2	SC	82	210944	SC		2	+	210944		SC		1	+	210944	
	y21PnVA2Hj	S	1200	236544	S	1200	165776	+		S	1200	127088	+			
mapping	full2x2	S	1200	1103	S	1200	801	+		S	1200	795	+			
	mesh2x2_mpeg	S	1200	726	S	1200	1436	-		S	1200	1116	-			
	mesh3x3.2	UNK	1200		S	+	1200		1631		S	+	1200		1623	
	mesh3x3_mpeg_2	S	1200	2197	S	1200	1188	+		S	1200	1211	+			
	ring_2	S	1200	2090	S	1200	1940	+		S	1200	1940	+			
multi-knapsack	mknapp1-6	UNK	1200		SC	+	7	+	16537		SC	+	7	+	16537	
	mknapp2-1	SC	158	7772	SC		1	+	7772		SC		2	+	7772	
	mknapp2-2	S	1200	8722	SC	+	8	+	8722		SC	+	8	+	8722	
	mknapp2-20	SC	3	6339	SC		1	+	6339		SC		1	+	6339	
	mknapp2-32	UNK	1200		S	+	1200		8947		SC	+	547	+	8947	
nmseq	176	S	6		S		1	+		S		1	+			
	207	S	7		S		1	+		S		1	+			
	269	S	45		S		2	+		S		2	+			
	393	S	244		S		2	+		S		4	+			
	83	S	1		S		1			S		1				
opd	flener_et_al_10_350_100	S	1200	65	UNK	-	1200			UNK	-	1200				
	medium_10_100_30	S	1200	13	S		1200		21	-	S		1200		10	+
	small_bibd_10_30_09	SC	1107	2	S	-	1200	-	3	-	S	-	1200	-	3	-
	small_bibd_11_22_10	S	1200	5	S		1200		5		S		1200		5	
	small_bibd_13_26_06	SC	1078	1	S	-	1200	-	2	-	S	-	1200	-	2	-
open_stacks	problem_20_20_1	S	1200	11	S		1200		11		S		1200		11	
	problem_30_15_1	SC	15	14	S	-	1200	-	14		S	-	1200	-	14	
	wbo_10_20_1	SC	303	5	S	-	1200	-	5		S	-	1200	-	5	
	wbo_15_30_1	S	1200	7	S		1200		7		S		1200		6	+
	wbp_20_20_1	S	1200	4	S		1200		4		S		1200		4	
p1f	12	S	1200	602	UNK	-	1200			UNK	-	1200				
	13	C	20		UNK	-	1200	-		UNK	-	1200	-			
	14	S	1200	1008	UNK	-	1200			UNK	-	1200				
	15	C	68		UNK	-	1200	-		UNK	-	1200	-			
	17	UNK	1200		UNK	1200				UNK	1200					
project-planning	ProjectPlannertest_12_7	S	1200	63	S		1200		19	+	SC	+	1	+	17	+
	ProjectPlannertest_14_7	S	1200	78	S		1200		32	+	SC	+	347	+	27	+
	ProjectPlannertest_15_6	S	1200	66	S		1200		37	+	SC	+	994	+	31	+
	ProjectPlannertest_16_6	S	1200	39	S		1200		35	+	S		1200		31	+
	ProjectPlannertest_16_8	S	1200	39	S		1200		35	+	S		1200		31	+
radiation	i14-9	SC	252	6513	S	-	1200	-	6720	-	S	-	1200	-	6526	-
	i6-11	SC	207	895	SC		9	+	895		S	-	1200	-	896	-
	i6-21	SC	143	1413	S	-	1200	-	1718	-	S	-	1200	-	1417	-
	i7-9	SC	7	1007	SC		6		1007		S	-	1200	-	1009	-
	i9-11	SC	427	2141	SC		123	+	2141		S	-	1200	-	2151	-
roster	chicroster_dataset_11	SC	1	17	SC		1		17		SC		1		17	
	chicroster_dataset_17	SC	1	17	SC		1		17		SC		1		17	
	chicroster_dataset_2	SC	0	0	SC		1		0		SC		1		0	
	chicroster_dataset_5	SC	1	6	SC		1		6		SC		1		6	
	chicroster_dataset_7	SC	1	0	SC		1		0		SC		1		0	
spot5	1401	S	1200	521097	S		1200		496114	+	S		1200		500112	+
	28	S	1200	284158	S		1200		276105	+	S		1200		277105	+
	414	S	1200	42564	S		1200		44501	-	S		1200		49510	-
	503	S	1200	15177	S		1200		11125	+	S		1200		11134	+
	54	S	1200	81	S		1200		37	+	S		1200		37	+
tdtsp	inst_10_24_10	S	1200	13917	SC	+	5	+	9192	+	SC	+	3	+	9192	+
	inst_10_34_00	S	1200	8353	SC	+	3	+	6662	+	SC	+	2	+	6662	+
	inst_10_42_10	S	1200	15329	SC	+	4	+	8486	+	SC	+	2	+	8486	+
	inst_20_14_10	S	1200	17449	S		1200		14889	+	S		1200		15801	+
	inst_20_25_00	S	1200	19898	S		1200		16655	+	S		1200		15945	+
triangular	n10	SC	89	20	SC		159	-	20		S	-	1200	-	20	
	n16	S	1200	35	S		1200		34	-	S		1200		34	-
	n22	S	1200	48	S		1200		50	+	S		1200		50	+
	n28	S	1200	61	S		1200		64	+	S		1200		65	+
	n37	S	1200	80	S		1200		87	+	S		1200		88	+
zephyrus	zephyrus_15_10	S	1200	36	S		1200		36		S		1200		36	
	zephyrus_20_20	S	1200	66	S		1200		66		S		1200		66	
	zephyrus_5_20	S	1200	66	SC	+	830	+	66		SC	+	416	+	66	
	zephyrus_5_4	S	1200	18	S		1200		18		SC	+	390	+	18	
	zephyrus-FH-2-15	SC	237	12	S	-	1200	-	12		S	-	1200	-	12	

TABLE 4.10: Base case, Hypothesis 3.1.1 single-objective and multi-objective GP full search comparison (training set).

individuals, is it required to determine the criteria on which a single individual is selected from the Pareto frontier to use as the candidate search strategy for the full evaluation. Solutions acquired during the sampling time having the smallest value in any three of the individual objectives (in isolation) may be selected, based on the SAW objective by minimising the combined sum of objectives, some other weighting scheme, or even hierarchically.

Table 4.11 summarises the outcome of different selection schemes for solutions on the Pareto frontier. Each solution from the Pareto frontier was evaluated during a full 20 minute search and the result of hierarchically ranking according to different objectives, as well as the SAW scheme, is examined. The evaluation of the selection schemes are computed strictly on measurements available to the GP search based on the sampled search strategy. By restricting the selection criteria to measurements available at the end of the GP search, a single individual is selected from each frontier without having to evaluate the final quality of all individuals on the frontier.

Selection Scheme	# S	# SC	# C	# UNK	$\sum f_1$	$\sum f_2$	$\sum f_3$
{1, 2, 3}	56	29	0	10	1.35	35.03	60.51
{1, 3, 2}	58	27	0	10	1.35	36.10	60.40
{2, 1, 3}	57	29	0	9	1.35	33.66	60.41
{2, 3, 1}	56	29	0	10	1.62	34.66	61.10
{3, 1, 2}	56	27	0	12	1.92	38.10	61.97
{3, 2, 1}	57	27	0	11	1.90	36.73	61.97
SAW	56	29	0	10	1.62	35.16	61.11

TABLE 4.11: Summary of Pareto frontier selection scheme results for Hypothesis 3.1.1 multi-objective GP.

The difference in overall outcomes in Table 4.11 is determined by a subset of problems for which multiple solutions are available on the Pareto frontier as represented by the blue set of points in Figure 4.5. It is interesting to note that reprojecting the results of the multi-objective search into the SAW objective as a selection criterion produces a relatively average overall result in the context of other selection schemes.

The scheme {2, 1, 3} is considered the preferred scheme for comparing results as not only does it achieve a large number of feasibility proofs, but also the least number of unknown status codes. The overall sum of normalised objective values for this selection scheme is also the smallest among the set of comparisons. This scheme selects a point hierarchically from the Pareto frontier, first according to the smallest normalised objective sum (4.2), followed by the normalised remaining minimum infeasibility (4.1) and finally the normalised solution time (4.1).

The low priority of the normalised solution time in the preferred selection scheme can be considered a result of a measurement side-effect. When small problems are solved, refined strategies which attempt to exploit such smaller problems may be conceived. This increases the measurement quality of small instances that can be solved, but decreases the generality of the strategy to potentially solve bigger problems.

A discussion was previously provided as to the correlation of objective function components in the multi-objective function (4.5). This is highlighted by most GP multi-objective runs where a set of points concentrate at the minima of the three objective components with typically smaller trade-offs between the objectives in this region being observed.

Table 4.12 contains a summary of the instance-level results provided in Table 4.10. The shorthand used for column names are ‘Code’, ‘Time’ and ‘Best’ which are the status codes, search

Problem Class	Hypothesis 3.1.1 SAW			Hypothesis 3.1.1 multi-objective		
	–	+	Δ	–	+	Δ
costas-array	1	2	1	0	5	5
cvrp	2	2	0	2	3	1
freepizza	1	4	3	0	5	5
gfd-schedule	2	9	7	2	7	5
grid-colouring	0	6	6	0	6	6
is	1	6	5	0	9	9
mapping	1	4	3	1	4	3
multi-knapsack	0	7	7	0	8	8
nmseq	0	4	4	0	4	4
opd	8	0	–8	7	1	–6
open_stacks	4	0	–4	4	1	–3
p1f	6	0	–6	6	0	–6
project-planning	0	5	5	0	11	11
radiation	6	2	–4	15	0	–15
spot5	1	4	3	1	4	3
tdtsp	0	11	11	0	11	11
triangular	2	3	1	3	3	0
zephyrus	2	2	0	2	4	2
Total	37	71	34	43	86	43

TABLE 4.12: Hypothesis 3.1.1 SAW and multi-objective results summary by problem class (training set).

time and best solution found, respectively. The difference between the run in question and the base ORT search routine is provided in columns marked Δ , with differences being indicated as either improvements (+) or deteriorations (–). Double counting is incurred both positively and negatively. For example, if the ORT search is unable to find a solution, but Hypothesis 3.1.1 multi-objective is able to find a solution and prove it optimal, as demonstrated by the multi-knapsack (mknapsack2-32) instance, then there are two +’s associated with the search strategy produced. Similarly, if the converse occurs where a search was unable to outperform the base case, as demonstrated by p1f (15), then two negative scores for time and status code are associated with that search effort. Comparisons over the objective function results have not been made unless both objectives are present as the impact of a superior or inferior search has already been encapsulated in the status code or search time in such instances. Differences between search times of less than one second are not considered sufficiently significant to contribute to the scoring employed and are considered equivalent.

Strategy improvements found

The summary of results in Table 4.12 provides a reasonable overview of where significant improvements could be made from both an observational and statistical point of view. A Wilcoxon rank test is applied to the data to test whether the number of improvements found is significantly different from a mean improvement of zero³, the results of which are provided in Table 4.13. It can be seen that the changes in codes between the GP developed strategies and the base

³In this context, the non-parametric test is preferred over the t-test as the data consist of unit values which do not normalise well. The Wilcoxon rank test returns more prudent p -values as a result of the non-parametric assumptions.

ORT solver are not significant (a mean change of 1% and 2% for the SAW and multi-objective schemes, respectively). The change in objective values and time to solve are both significant at the $\alpha = 0.05$ level for both types of objective function used during the GP search. The overall model significance, when considering the total number of changes in all three measurement categories, is also provided and is unsurprisingly significant at most reasonable levels of significance.

Type	Δ Code		Δ Objective		Δ Time		Δ Total	
	μ	p -value	μ	p -value	μ	p -value	μ	p -value
SAW	0.0105	0.8582	0.2000	0.0038	0.1474	0.0236	0.1193	0.0011
Multi-objective	0.0211	0.7457	0.2526	0.0005	0.1789	0.0115	0.1509	0.0002

TABLE 4.13: Wilcoxon rank test for significant differences to a mean of zero in improvements and deteriorations by measurement category and overall changes.

It is notable that the multi-objective based GP search outperformed both the SAW GP search ($p = 0.0027$) as well as the base ORT search strategy ($p = 0.0002$). An observation on the treatment of individuals in the multi-objective GP search may shed some light on this better performance. Candidate solutions which achieve reasonable quality in one dimension are retained for a longer period during the search and combined with individuals throughout the search, leading to improved diversity, retention of properties which may be deemed useful during a search, and are passed on to future generations. The SAW objective scheme produces a myopic view of the quality of an individual in the population and, as such, search trees attributes which may prove useful in future generations may have been lost early on during the search.

Some of the most dramatic improvements seen in the GP search over the default ORT search are on the problems multi-knapsack, is, project-planning and tdtsp. In the case of is, two additional optimality proofs are observed, improvements in the time to provide the aforementioned proofs, as well as improvements in objective function values over the base search values where an optimality proof was not completed. The sizes of the improvements in the objective function values are also quite substantial on this particular problem set. This is a good example of where a simpler search using some ranking heuristics and control parameters, is able to outperform a far more complex series of search heuristics being employed in the ORT default search.

Another impressive search strategy found is for the multi-knapsack problem class where all problem instances admit an optimality proof for the multi-objective GP search with four of the five problem instances being proved optimal within 10 seconds and the last problem instance requiring just under 10 minutes. The multi-objective frontier for the multi-knapsack problem class in Figure 4.5 illustrates that the Pareto frontier contains multiple solutions which are all able to provide a large number of optimality proofs for this problem class with a small degree of variation in the aggregate solution time and remaining infeasibility. The tdtsp and project-planning problem classes admit a similar result in that three optimality proofs are provided within a few sections and the remaining two problem sets both outperform the solutions found during the default ORT search by a significant margin.

Other noteworthy improvements in search performance were found on the problems costas-array, freepizza, gfd-schedule, grid-colouring and mnseq. In particular, costas-array and mnseq are both pure feasibility problems and improvements in the time to find a feasible solution were observed in both problem classes with the multi-objective GP strategy being able to provide a feasible incumbent for the costas-array (19) instance.

Failure to improve

Not all GP strategies produced were able to outperform the default ORT search strategy. This is not an unacceptable result and should, in fact, have been expected to a certain degree. It is known that there is no free lunch and that the ORT default search strategy may perform very well on certain problem classes by exploiting its complex heuristic divide, variable partitioning and restart scheme. This is well illustrated by the four problem classes: radiation, p1f, opd and open_stacks. These are the only four problem classes for which a nett improvement in the default strategy, which is constrained by the available GP grammar for Hypothesis 3.1.1, was not obtained. The multi-objective GP fails in rather spectacular fashion on the radiation problem class, consistently producing objective values a fraction larger than those found by the default ORT search. The ORT default search is able to provide an optimality proof for all problem instances, which the SAW GP search was able to somewhat emulate, but was unable to match the performance of the default ORT search.

A visual example of a p1f instance is shown in Figure 2.5(e) and one can see that a large proportion of the CSP seems to form independent subsets of variables and constraints, clustering into many subtrees. The chart of the progress of the SAW GP search on the p1f problem set (Figure 4.4) shows that the search generated strategies struggled to find a solution to the problem instance. Two of the problem instances are infeasible by design and require infeasibility proofs which illustrates a problem with the objectives employed by the GP. In the task to demonstrate infeasibility, one would ideally desire the smallest infeasible clique of variables positioned close to the root node in order to perform the infeasibility assertion as quickly as possible. The metric of preferring searches with more feasible assigned variables is the anti-strategy to what is ideally required for this problem instance. A better metric to have used for such problems would have been a search space coverage metric⁴ or including the competing objective of maximising the minimum infeasibility obtained. In such a formulation, a multi-objective search may find strategies which work well at the top or bottom of the search tree. Moreover, computing the coverage if log-space is considered would make for an interesting piece of future work. An $O(n)$ algorithm can compute the coverage at each backtrack step, but would require solver-specific modifications.

It is a good result in that the test data comprise a diverse set of problem instances which highlight potential issues in the metrics used at a metaheuristic level. It is clear, however, that there is a bias in the data towards problems which admit feasible solutions.

4.5.2 Extendibility to unseen problem instances

A logical question that arises at this juncture is whether the GP strategies developed for a given problem class work well on unseen instances of the same class. The GP process used to derive the search strategies does not permit domain-specific information (directly identifiable variables or constraints) and a single strategy is applied across all training instances for a problem class. This introduces a bias towards a generalised strategy across such a class of problem.

In order to verify the degree to which this intuition is valid, a set of *hold-back* or *test* data are employed which are not used during the GP learning process. The detailed results of the performance of the Hypothesis 3.1.1 single-objective and multi-objective strategies, using the same {2, 1, 3} hierarchical selection criterion, are shown in Table 4.14. It is important to note

⁴This metric can be difficult to compute and standardise across different backtracking algorithms employed by solvers, but would be a suitable replacement objective.

that in instances where the search performed poorly during the training process, the expectation that the search would continue to perform poorly on the unseen instances is carried forward.

That the poor performance of p1f and radiation is indeed carried forward to the test set is highlighted in summary Table 4.15. The performance of the cvrp strategies retain their neutral performance; the flat results for costas-array is partly due to the data set consisting of smaller unseen instances. Unfortunately, an extended test set for problem classes is and freepizza is not available in the CSPLib repository and as such have been omitted from the extendibility test. These are two problem classes on which the GP search performed quite well. As a general observation, the positive performance over the default ORT search on problem classes such as grid-colouring, mapping, project-planning, spot5, tdtsp and triangular are all carried forward to the test set. This is a noteworthy observation as it suggests that there is common exploitable structure that dominates in these problem instances which the branching strategy has been able to leverage across the problem class.

A problem class which stands apart from the other trends observed in the test set is that of the zephyrus problem class. Reasonable improvements were found over the default ORT strategy on the training data, but the results on the test set are very poor, with it ranking as the overall worst problem set in terms of deterioration from the default ORT search strategy. This is in part due to the structure of the training data and test data having a different parametrisation and structure even though they are both decorated with the same problem class name. Illustrations of the problem structure for two training set and two test set problem instances are provided in Figure 4.6, showcasing the difference in the problem structure.

While this may seem at a first glance to be a specific failure of the algorithmic approach it, in fact, in a soft sense affirms the hypothesis. The aim is to design search strategies which exploit some structural feature of the underlying graph which is expected to be present in variations of the problem. If this structure is significantly modified, it does not follow that the algorithm would retain its ability to perform well on a problem that no longer exhibits the same structural exploits.

Figure 4.6 illustrates that the training data has two connected components with a mostly symmetric structure emanating from a core set of variables. The graphical representation of the test set examples have a single connected component with additional complexities and sub-structures within each primary region. It also appears that there are additional side constraints consisting of clusters of constraints in the north-west of Figures 4.6(c) and 4.6(d) which were not present in the original problem definition.

Problem Class	Instance	ORT			Hypothesis 3.1.1 SAW					Hypothesis 3.1.1 multi-objective						
		Code	Time	Best	Code	Δ	Time	Δ	Best	Δ	Code	Δ	Time	Δ	Best	Δ
costas-array	10	S	1		S		1			S		1				
	11	S	1		S		0			S		1				
	12	S	1		S		1			S		1				
	13	S	1		S		1			S		1				
	14	S	1		S		1			S		1				
	15	S	9		S		1	+		S		2	+			
cvrp	A-n32-k5.vrp	S	1200	1987	S		1200		2169	-	UNK	-	1200			
	A-n55-k9.vrp	S	1200	3083	S		1200		3012	+	S		1200		2996	+
	B-n38-k6.vrp	S	1200	2054	S		1200		2852	-	S		1200		2403	-
	B-n51-k7.vrp	S	1200	3337	UNK	-	1200				S		1200		3903	-
	P-n40-k5.vrp	S	1200	1309	UNK	-	1200				S		1200		1304	+
gfd-schedule	n10f2d10m10k3	SC	1	3	SC		1		3		SC		1		3	
	n25f5d20m10k3	S	1200	803	S		1200		205	+	S		1200		422	+
	n35f5d20m10k3	UNK	1200		S	+	1200		1107		S	+	1200		822	
	n55f2d50m30k3	S	1200	2704	S		1200		12507	-	S		1200		7155	-
	n60f7d50m30k10	S	1200	2215	S		1200		19115	-	S		1200		9658	-
grid-colouring	10.10	S	1200	6	S		1200		4	+	S		1200		4	+
	12.13	S	1200	7	S		1200		6	+	S		1200		4	+
	15.16	S	1200	11	S		1200		5	+	S		1200		5	+
	5.6	SC	1	3	SC		1		3		SC		1		3	
	7.8	S	1200	4	SC	+	1	+	3	+	SC	+	3	+	3	+
mapping	mesh2x2.1	S	1200	1060	SC	+	66	+	1000	+	SC	+	5	+	1000	+
	mesh2x2.mp3	S	1200	1254	SC	+	26	+	1102	+	SC	+	175	+	1102	+
	mesh3x3.mp3	S	1200	1314	S		1200		1436	-	S		1200		1262	+
	mesh4x4.1	UNK	1200		S	+	1200		2564		S	+	1200		2354	
	ring.1	UNK	1200		S	+	1200		1702		S	+	1200		1733	
multi-knapsack	mknapp2-10	UNK	1200		UNK		1200				SC	+	644	+	624319	
	mknapp2-31	UNK	1200		SC	+	25	+	9074		SC	+	37	+	9074	
nmseq	099	S	1		S		1				S		1			
	100	S	2		S		1				S		1			
	143	S	4		S		1	+			S		1	+		
	150	S	4		S		1	+			S		1	+		
	200	S	6		S		1	+			S		1	+		
opd	flener_et_al.15.350.100	S	1200	73	UNK	-	1200				UNK	-	1200			
	medium.13.250.80	S	1200	52	S		1200		186	-	UNK	-	1200			
	small.bibd.06.50.25	SC	289	10	S	-	1200	-	11	-	S	-	1200	-	11	-
	small.bibd.06.60.30	S	1200	13	S		1200		29	-	S	-	1200		15	-
	small.bibd.08.28.14	SC	443	6	S	-	1200	-	9	-	S	-	1200	-	7	-
open_stacks	problem.30.10.1	SC	1	12	S	-	1200	-	12		S	-	1200	-	12	
	wbo.10.30.1	S	1200	7	S		1200		6	+	S		1200		6	+
	wbo.15.15.1	SC	1	3	SC		1		3		SC		1		3	
	wbop.15.30.1	S	1200	6	S		1200		7	-	S		1200		6	
	wbop.30.10.1	SC	1	14	S	-	1200	-	14		S	-	1200	-	14	
plf	10	S	1200	300	UNK	-	1200				UNK	-	1200			
	11	C	1		UNK	-	1200	-			UNK	-	1200	-		
	7	C	1		UNK	-	1200	-			UNK	-	1200	-		
	8	SC	1	168	UNK	-	1200	-			UNK	-	1200	-		
	9	C	1		UNK	-	1200	-			UNK	-	1200	-		
project-planning	ProjectPlannertest.12.6	S	1200	68	S		1200		19	+	SC	+	2	+	17	+
	ProjectPlannertest.14.6	S	1200	73	S		1200		32	+	S		1200		27	+
	ProjectPlannertest.15.8	S	1200	66	S		1200		42	+	SC	+	682	+	31	+
	ProjectPlannertest.16.9	S	1200	39	S		1200		35	+	S		1200		35	+
	ProjectPlannertest.17.6	S	1200	95	S		1200		46	+	S		1200		48	+
radiation	09	SC	204	673	SC		1	+	673		S	-	1200	-	674	-
	i6-7	UNK	1200		SC	+	1	+	635		SC	+	332	+	635	
	i7-15	SC	61	1308	SC		288	-	1308		S	-	1200	-	1321	-
	i8-7	SC	10	1046	SC		1	+	1046		S	-	1200	-	1050	-
	i9-23	UNK	1200		S	+	1200		4373		S	+	1200		4388	
roster	chicroster_dataset.10	SC	1	18	SC		1		18		SC		1		18	
	chicroster_dataset.4	SC	0	2	SC		1		2		SC		1		2	
	chicroster_dataset.6	SC	1	8	SC		1		8		SC		1		8	
	chicroster_dataset.8	SC	1	7	SC		1		7		SC		1		7	
	chicroster_dataset.9	SC	1	13	SC		1		13		SC		1		13	
spot5	1502	S	1200	64056	S		1200		28043	+	S		1200		32050	+
	29	S	1200	14069	S		1200		8059	+	S		1200		8059	+
	412	S	1200	34457	S		1200		34397	+	S		1200		39407	-
	42	S	1200	191117	S		1200		164064	+	S		1200		168054	+
	5	S	1200	331	S		1200		275	+	S		1200		283	+
tdtsp	inst.10.35.20	SC	638	9055	SC		12	+	9055		SC		9	+	9055	
	inst.10.42.00	SC	318	8421	SC		3	+	8421		SC		2	+	8421	
	inst.10.45.00	SC	1	6819	SC		3	-	6819		SC		2		6819	
	inst.10.58.20	S	1200	13799	SC	+	11	+	10306	+	SC	+	12	+	10306	+
	inst.20.26.00	S	1200	18180	S		1200		14626	+	S		1200		14942	+
triangular	n18	S	1200	40	S		1200		40		S		1200		40	
	n26	S	1200	56	S		1200		61	+	S		1200		59	+
	n34	S	1200	74	S		1200		81	+	S		1200		79	+
	n40	S	1200	86	S		1200		97	+	S		1200		95	+
	n46	S	1200	98	S		1200		110	+	S		1200		112	+
zephyrus	12_6_8_3	SC	948	1300	S	-	1200	-	1300		S	-	1200	-	3055	-
	12_8_6_3	SC	294	1300	S	-	1200	-	1300		S	-	1200	-	6305	-
	14_10_8_3	UNK	1200		S	+	1200		9100		S	+	1200		10920	
	14_6_8_3	SC	247	1170	S	-	1200	-	8710	-	S	-	1200	-	5850	-
	14_8_6_3	SC	87	1170	S	-	1200	-	9230	-	S	-	1200	-	6760	-

TABLE 4.14: Base case, Hypothesis 3.1.1 SAW and multi-objective GP full search comparison (test set).

Problem Class	Hypothesis 3.1.1 SAW			Hypothesis 3.1.1 multi-objective		
	–	+	Δ	–	+	Δ
costas-array	0	1	1	0	1	1
cvrp	4	1	–3	3	2	–1
gfd-schedule	2	2	0	2	2	0
grid-colouring	0	6	6	0	6	6
mapping	1	8	7	0	9	9
multi-knapsack	0	2	2	0	4	4
nmseq	0	3	3	0	3	3
opd	9	0	–9	9	0	–9
open_stacks	5	1	–4	4	1	–3
p1f	9	0	–9	9	0	–9
project-planning	0	5	5	0	9	9
radiation	1	5	4	9	3	–6
spot5	0	5	5	1	4	3
tdtsp	1	6	5	0	6	6
triangular	0	4	4	0	4	4
zephyrus	10	1	–9	12	1	–11
Total	42	50	8	49	55	6

TABLE 4.15: Hypothesis 3.1.1 SAW and multi-objective results summary by problem class (test set).

4.5.3 Hypothesis 3.1.1 conclusions

The first hypothesis of this dissertation is to determine whether an evolutionary algorithm can be used to derive search strategies using relatively simplistic features of a problem domain. It transpires that the MiniZinc competition problem set provides a diverse set of problems which requires both asserting feasibility and infeasibility, the latter being a requirement which is neither encapsulated by the objectives of the SAW objective nor by the multi-objective objective GP. In spite of this shortcoming, one is still able to effectively evolve strategies for the majority of the problem classes using a short sampling period of 10 seconds on problem instances which result in an improvement through effective searches tailored to a problem class.

A statistical test of the differences between the default ORT search strategy and the derived strategies per problem class show that the results are significantly better in both the instance-level objective values found and the search time required on problem instances at the $\alpha = 0.01$ level. This is an unsurprising result as one would expect a bespoke approach to outperform a default strategy (albeit a highly complex strategy). What is initially counter-intuitive is that there are instances in which a bespoke strategy cannot be derived to outperform the default ORT search strategy. This is largely due to the strategy of ORT encapsulating multiple bespoke heuristics, which is very well suited for these particular problems. The grammar provided to the GP is intentionally limited to a handful of domain features which it is required to exploit in order to derive a strategy. The GP is thus, by design, unable to create strategies as complex as the default ORT strategy. This is sufficient for our analysis as one is attempting to find where simpler structural exploits may exist which have the potential to generalise well without excessive algorithmic complexity in the variable and value ranking functions.

The generalisability of strategies evolved translate well to unseen problem instances with the results highlighting a structural difference between the training and test set for one of the problem classes, which is not apparent when browsing the CSPLib. This concludes that the

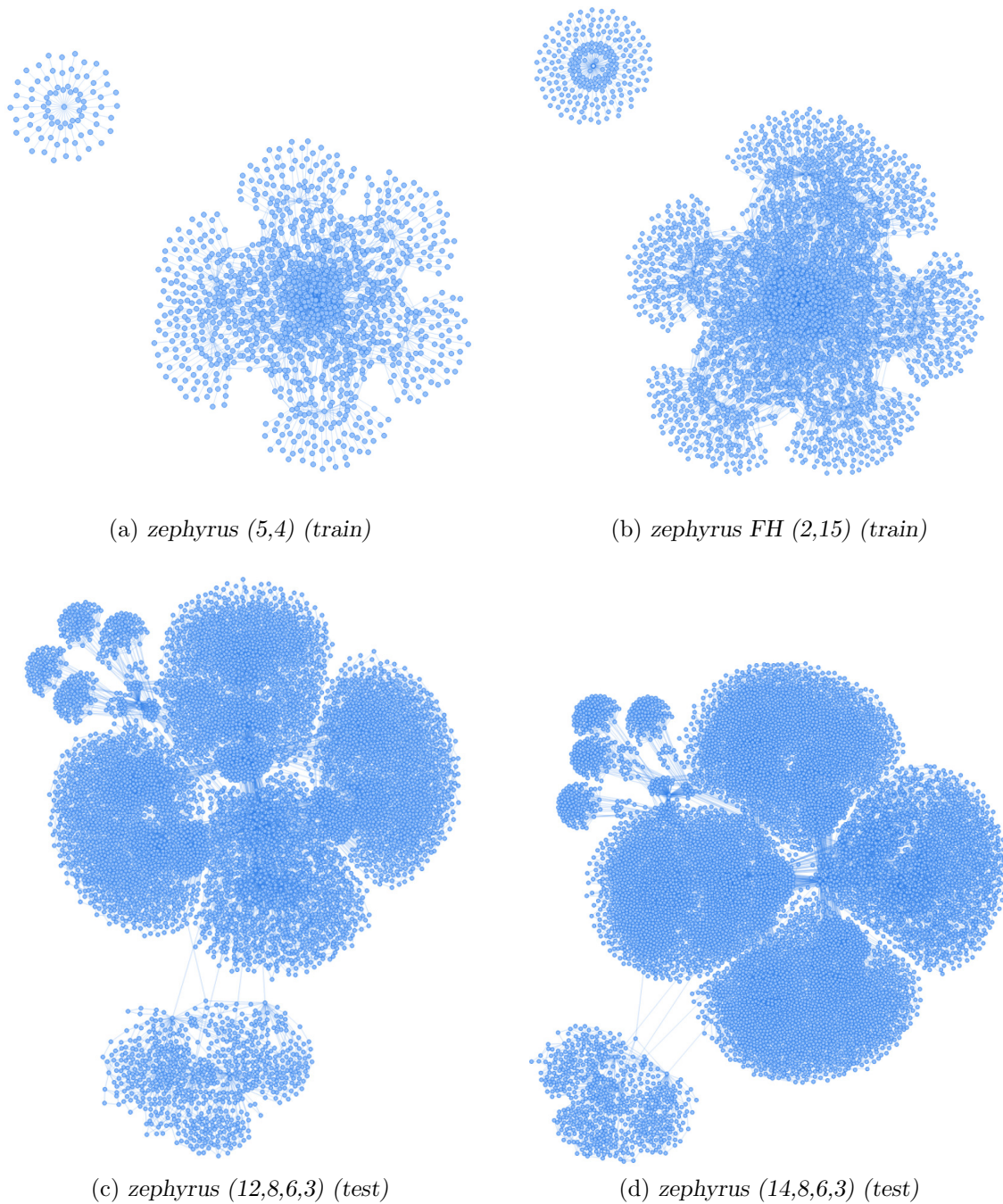


FIGURE 4.6: Structural differences between (a)–(b) training and (c)–(d) test sets for the *zephyrus* problem class.

results of Hypothesis 3.1.1 are quite promising in that an EA can optimise a search strategy for a class of problems, outperforming the default search strategy and generalising to problem instances of the same structure. Certain shortcomings of the objective function adopted were also highlighted for problem classes that admit infeasibility. One might imagine at this point that if a portfolio of search strategies were available which included the default search strategy, that this could result in significant improvements in both search status code, time and instance objective values obtained.

4.5.4 Graph meta-data CSP GP operators

The first Hypothesis 3.1.1 demonstrates that it is possible to use conventional GP processes based on simple function sets to learn search strategies for resolving a CSP. The second hypothesis is aimed at extending the function set and examining the behaviour of the GP and strategies developed in terms of this larger function set in hand, the intuition being that providing additional information to a ranking function may result in better rankings. A summary of the ranking functions provided to the grammar are detailed in Table 4.16.

It is worth noting that modifications to the GP are concerned primarily with the variable domain and not the value domain. Hence, the value selector operators used in Hypothesis 3.1.1 (Table 4.2) are unchanged for Hypothesis 3.1.2. A subset of the possible vertex scores were taken from Table 2.9 based on their computational runtimes on the benchmark CSP data, beyond which no additional selective procedures were applied.

Function Name	Type	Arity	Description	Abbreviation
Add	Node	2	Simple addition	+
Mul	Node	2	Simple multiplication	*
Inv	Node	1	Protected inversion	Inv
Neg	Node	1	Negation	-
Rand	Terminal	0	Random uniform number	rand
MinX	Terminal	0	The minimum feasible value in the domain of variable x	Min(x)
MaxX	Terminal	0	The maximum feasible value in the domain of variable x	Max(x)
SizeX	Terminal	0	The feasible domain size of variable x	Size(x)
SuccessRateX	Terminal	0	The success rate of value assignments to variable x	SuccRate(x)
ImpactX	Terminal	0	The (sum of) impact of variable x	Impt(x)
Closeness	Terminal	0	How easily variable x can be reached by other variables	C(x)
Betweenness	Terminal	0	The number of geodesics passing through a variable x	Btw(x)
Pagerank	Terminal	0	A probabilistic weighting of the importance of a variable x	P(x)
Burt constraint score	Terminal	0	A score for each variable x , also known as structural holes	Brt(x)
Strength	Terminal	0	In an unweighed network it is the degree of a variable x	S(x)
Eigenvector centrality	Terminal	0	A measure of the importance of a variable x in the network	E(x)
Kleinberg's Hubscore	Terminal	0	A measure of the importance of a variable x in the network	H(x)
Kleinberg's Authority score	Terminal	0	Score of a variable x based on how "authoritative" it is perceived to be	A(x)

TABLE 4.16: Description of the node and terminal set used in Tree 1 Hypothesis 3.1.2, the variable selector.

Holding the value selection grammar constant for this experiment is a result of the convenience of applying standard graph analysis to the constraint network. A value-domain analysis could be conducted if the network was reprojected into a binary normalised network, although this introduces complexity in terms of both the normalisation process (an NP-hard problem if the size normalised network is to be minimised) and in terms of the underlying graph analysis. The latter is as a result of the normalised network being guaranteed, if all variables are already binary, to be at least as large as the existing network, but in the typical case will be considerably larger. A larger normalised network will induce increased computational runtimes for resulting analyses. There may be benefit to a structured analysis of the value-domains, but the scope of this hypothesis has been restricted to focus on operators concerned with the variable relations.

A possible approximation of the effect of value domains between variables could be included in the graph analysis conducted herein by adjusting edge weights between nodes in the CSP based on the size of overlapping domains. The graph analysis herein assumes unit weights for all constraint relations (or edges) pertaining to the calculation of measures presented. An area of possible further research would be to investigate the use of such approximations of value domains between variables by means of different transformations to the edge weight.

4.6 GP run results (Hypothesis 3.1.2)

The GP used in conjunction with the operator sets in Tables 4.2 and 4.16 uses the same parameters and objectives already described in Tables 4.3 and 4.4. The intuition behind this decision is to hold the majority of parameters constant as variations of the grammar are examined. This allows for a treatment of the effect of a change in the grammar employed to be investigated independently of other factors. One would also intuitively expect the results of an extended grammar to perform at least as well as the simpler grammar, although there is another factor which changes as a result of the grammar, namely the size of the search space presented to the GP. The population size and number of generations permitted to the GP are held constant while the search space is increased. It is assumed that the search duration allocated to the GP and search efficacy of the GP is sufficient to converge on previously found strategies in a smaller search space. This may not hold in practice, but is interesting to pursue as it may demonstrate where the GP would require additional configuration changes to support searching effectively in such larger spaces.

Illustrations of the SAW and multi-objective GP run results are provided in Figures 4.7 and 4.8, respectively. As a point of reference, the GP results from the SAW Hypothesis 3.1.1 runs are overlaid as a baseline, shown in green in Figure 4.7. Providing an analogous visual comparison for Figure 4.8 is not as easily achieved and, as a result, multi-objective comparisons are restricted to empirical outcomes discussed later in this section. As before, run results with variation in a single dimension, triangular and p1f, are omitted from Figure 4.8 as the data are sufficiently described by Figure 4.7 in this context.

The SAW GP runs shown in Figure 4.7 illustrate that the Hypothesis 3.1.2 function set provides inconsistent results with respect to the best solutions obtained. In some instances, strategies are found which are able to outperform the Hypothesis 3.1.1 strategies within the sampling limit, such as freepizza, grid-coloring, radiation, tdtsp, triangular and zephyrus. Instances where the Hypothesis 3.1.1 strategies performed notably better than Hypothesis 3.1.2 within the sampling limit are: gfd-schedule, is, multi-knapsack and spot5. This is not an unexpected result due to the increased GP search space size induced by the Hypothesis 3.1.2 function set. A point of interest is that there are at least two problem classes where Hypothesis 3.1.2 significantly outperformed Hypothesis 3.1.1 within the sampling limit. The question now arises as to how these strategies perform when provided a full search time of 20 minutes. The results of the best individuals from the SAW GP run and multi-objective run are provided in Table 4.17. The scheme used to select multi-objective individuals from the Pareto frontier is the same as that previously used for Hypothesis 3.1.1, in order to maintain consistency.

The results in Table 4.17 are reported in the same manner as in Table 4.10 whereby improvements in the search strategy for problem classes are calculated with respect to the default ORT search procedure. It is interesting to note that the convergence of the GP searches between Hypothesis 3.1.1 and Hypothesis 3.1.2 result in similar outcomes when given a search budget of 20 minutes. There are a few instances where Hypothesis 3.1.2 finds solutions slightly slower, and in other instances, slightly quicker than the Hypothesis 3.1.1 multi-objective search strategies. Significant findings include a full suite of optimality proofs for the gfd-schedule problem class, which Hypothesis 3.1.1 was unable to achieve, and several improvements in the objective function values of triangular, a maximisation problem class. The tdtsp has improved objective function values for the larger problem instances which is also demonstrated by the freepizza class where the smallest problem instance forgoes optimality in order to obtain better objective function values for the remaining instances in the class.

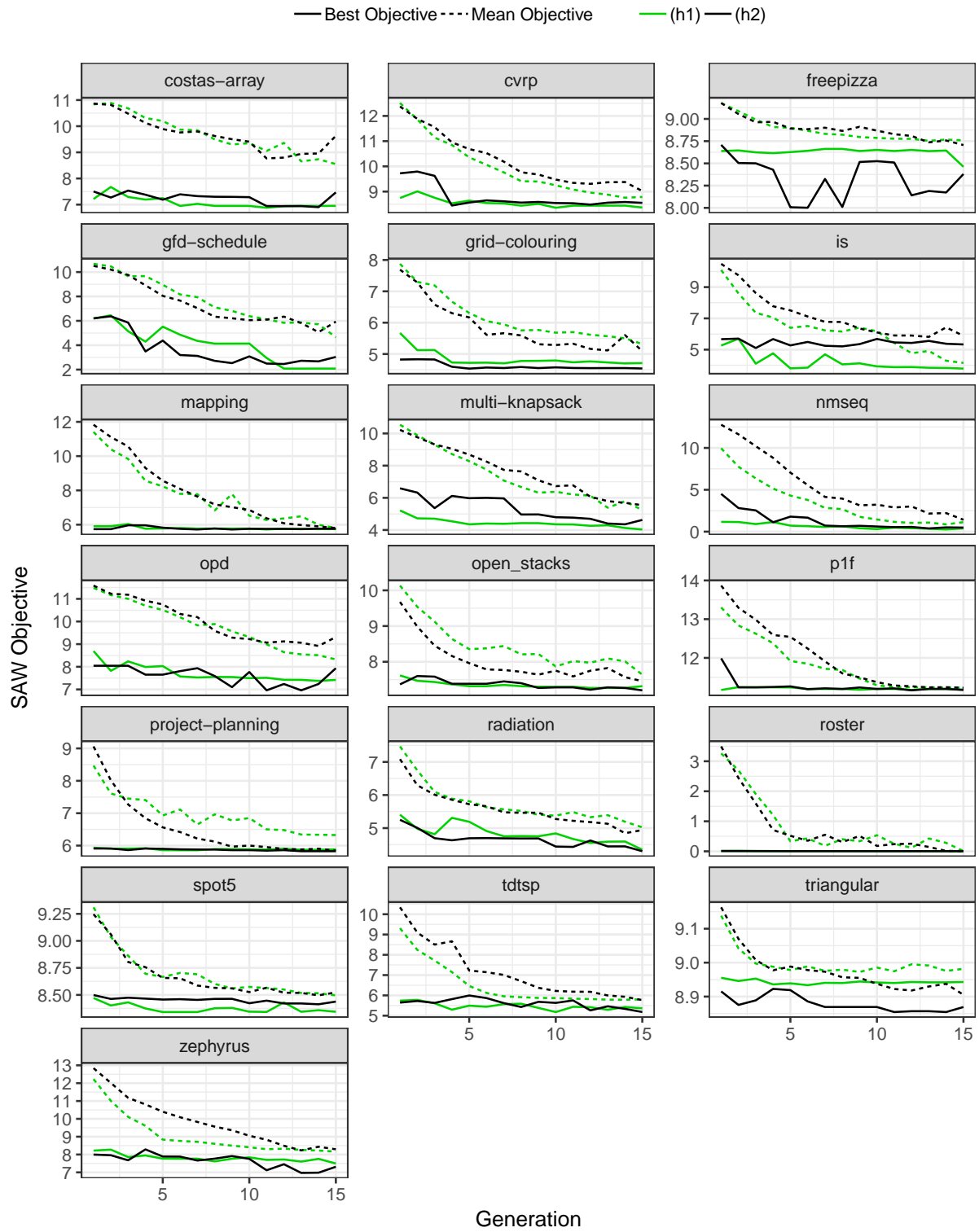
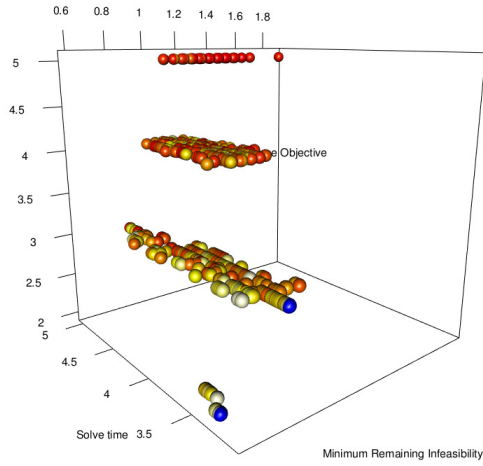
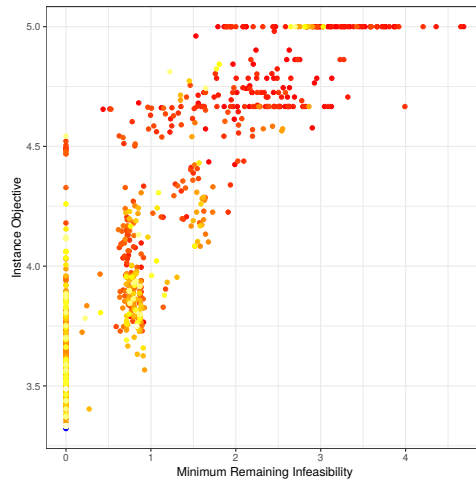


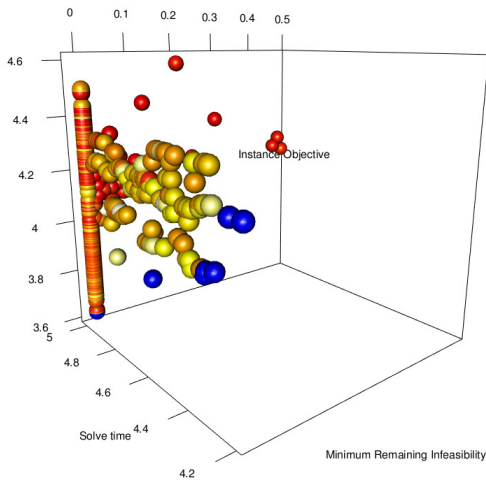
FIGURE 4.7: GP run results for Hypothesis 3.1.2 using SAW.



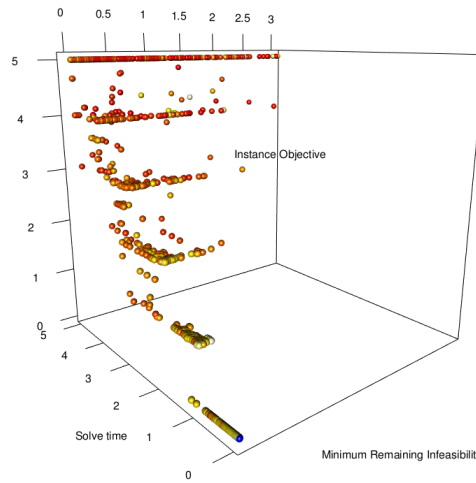
(a) *costas-array*



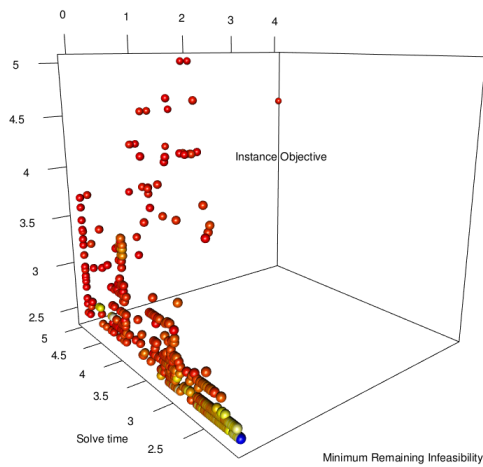
(b) *cvrp*



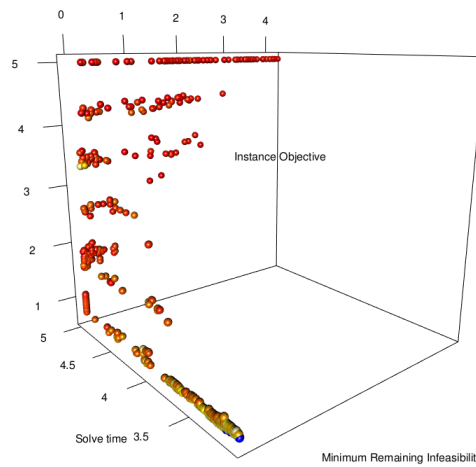
(c) *freepizza*



(d) *gfd-schedule*

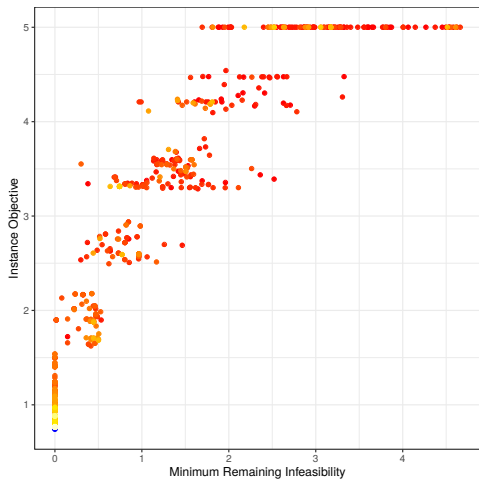


(e) *grid-colouring*

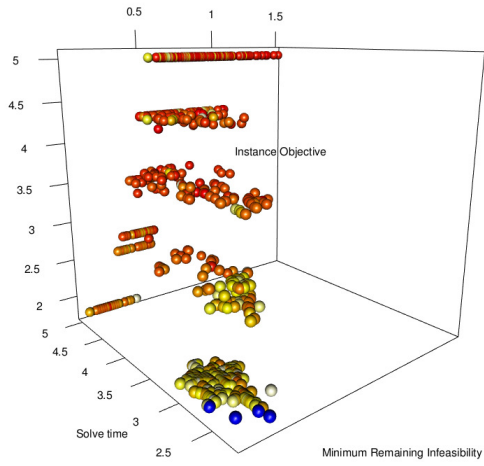


(f) *is*

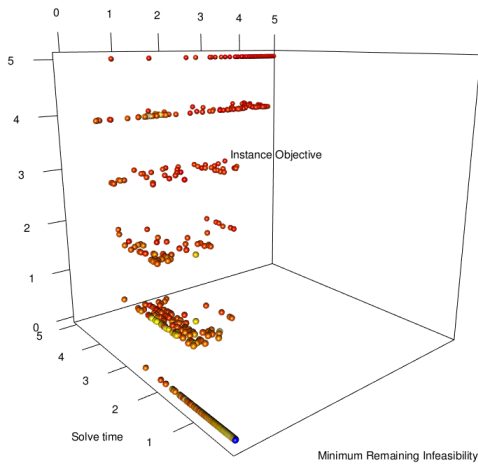
FIGURE 4.8: GP run results for Hypothesis 3.1.2 using NSGA-II.



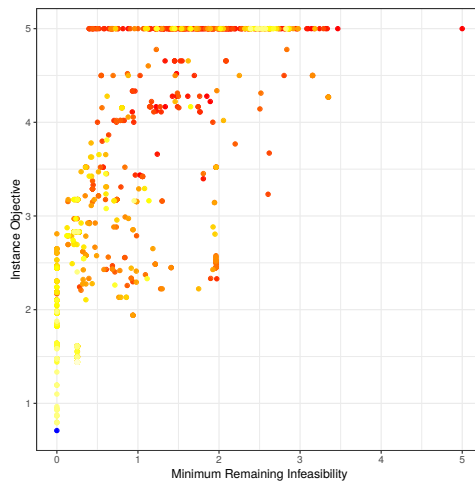
(g) *mapping*



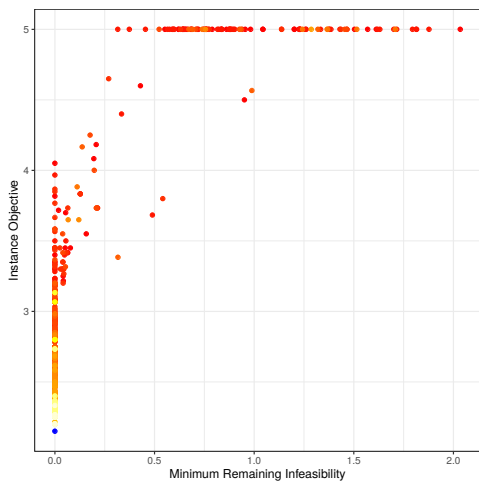
(h) *multi-knapsack*



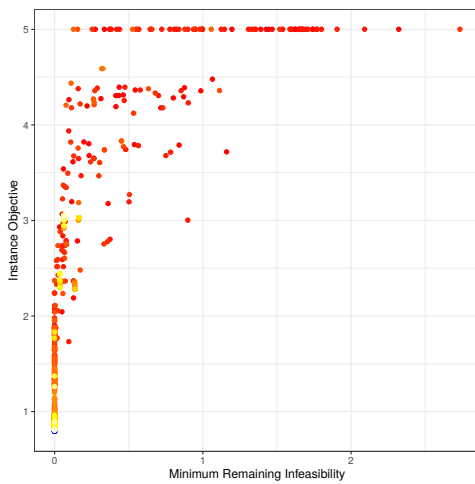
(i) *nmseq*



(j) *opd*



(k) *open_stacks*



(l) *project-planning*

FIGURE 4.8: GP run results for Hypothesis 3.1.2 using NSGA-II (continued).

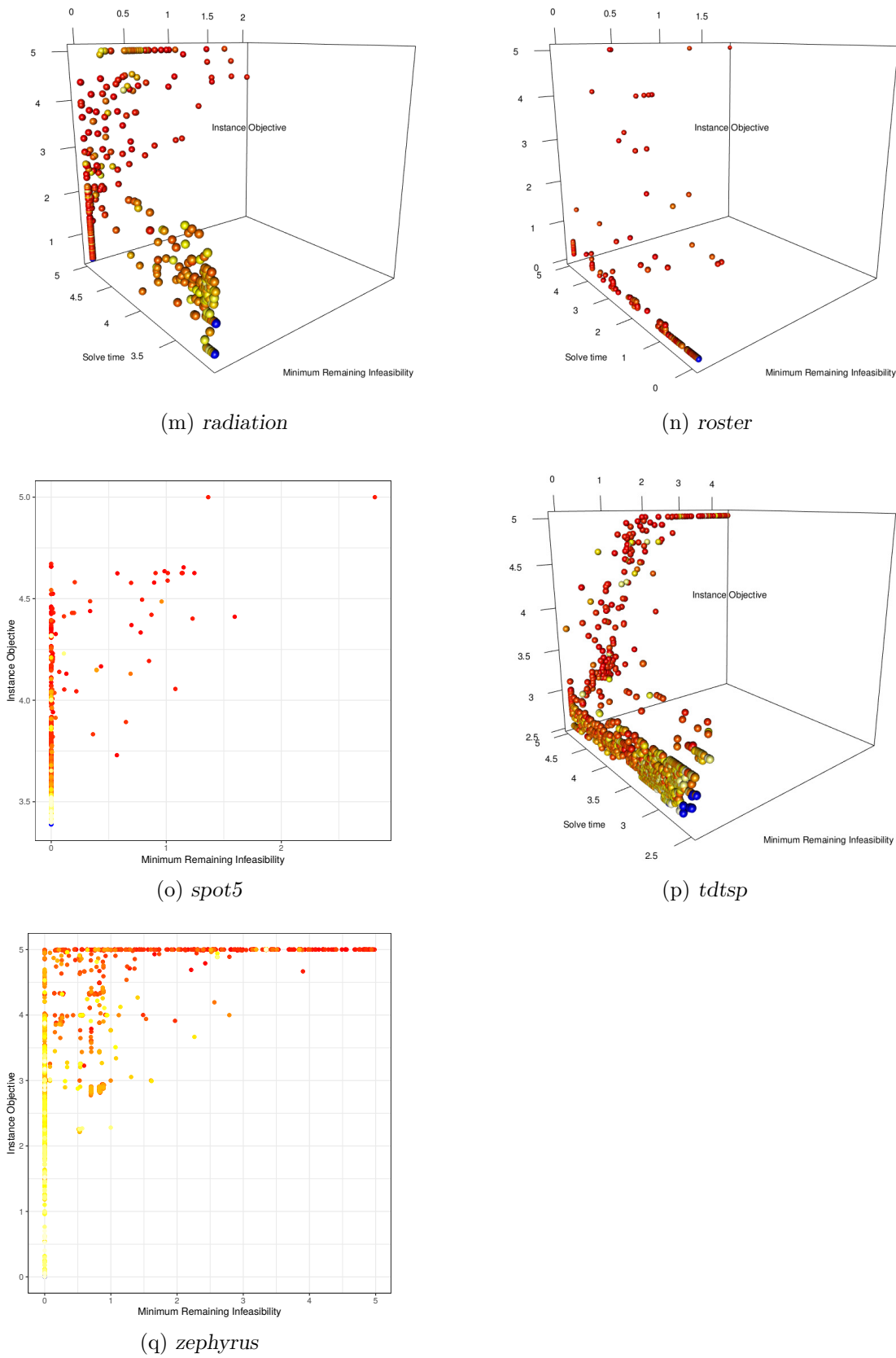


FIGURE 4.8: GP run results for Hypothesis 3.1.2 using NSGA-II (continued).

Problem Class	Instance	ORT			Hypothesis 3.1.1 SAW				Hypothesis 3.1.1 multi-objective				Hypothesis 3.1.2 SAW				Hypothesis 3.1.2 multi-objective														
		Code	Time	Best	Code	Δ	Time	Δ	Best	Δ	Code	Δ	Time	Δ	Best	Δ	Code	Δ	Time	Δ	Best	Δ									
costas-array	16	S	19		S		1	+		S		6	+		S		1	+		S		5	+								
	17	S	240		S		1	+		S		7	+		S		1	+		S		2	+								
	18	S	17		S		364	-		S		8	+		S		956	-		S		3	+								
	19	UNK	1200		UNK		1200	-		S		+	105	+		UNK		1200	-		S		+	1065	+						
	20	UNK	1200		UNK		1200	-		UNK		1200	-		UNK		1200	-		UNK		1200	-								
cvrpt	A-n37-k5.vrp	S	1200	1642	S		1200	1832	-	S		1200	2122	-	S		1200	2190	-	S		1200	1914	-							
	A-n64-k4.vrp	S	1200	3486	S		1200	3545	-	S		1200	4046	-	UNK		1200	-		S		1200	3554	-							
	B-n45-k5.vrp	S	1200	2728	S		1200	2628	+	S		1200	2467	+	S		1200	2586	+	S		1200	2654	-							
	P-n16-k8.vrp	S	1200	502	S		1200	502		S		1200	450	+	S		1200	506	-	S		1200	469	+							
	simple2	SC	377	34	SC		30	+	34		SC		19	+	34		SC		43	+	34		SC		17	+	34				
freepizza	pizza27	S	1200	882 425	S		1200	822 299	+	S		1200	761 294	+	S		1200	763 780	+	S		1200	724 896	+							
	pizza39	S	1200	939 352	S		1200	987 968	-	S		1200	837 068	+	S		1200	909 629	+	S		1200	839 000	+							
	pizza45	S	1200	656 489	S		1200	641 397	+	S		1200	571 934	+	S		1200	561 848	+	S		1200	569 138	+							
	pizza6	SC	610	210	SC		5	+	210		SC		113	+	210		SC		4	+	210		S		-	1200	-	230	-		
	pizza78	S	1200	901 717	S		1200	896 600	+	S		1200	714 755	+	S		1200	659 044	+	S		1200	638 240	+							
gfd-schedule	n1205f5d50m30k18	S	1200	19 463	SC		+	1	+	1		+	3	+	1		+	1	+	1		+	1	+	1		+	1			
	n180f7d50m30k18	UNK	1	1	UNK		-	1200	-			-	1200	-			-	1200	-			-	1200	-			-	1200	-		
	a30f3d30m7k4	UNK	1200		SC		+	1	+	1		+	1	+	1		+	1	+	1		+	1	+	1		+	1			
	a50f7d40m10k4	UNK	1200		SC		+	1	+	1		+	2	+	1		+	1200				+	1	+	1		+	1			
	n75f5d30m20k20	UNK	1200		SC		+	1	+	1		+	1200				+	1200				+	1	+	1		+	1			
grid-colouring	10_5	SC	31	3	SC		1	+	3		SC		1	+	3		SC		1	+	3		SC		2	+	3				
	13_11	S	1200	7	S		1200	5	+	S		1200	4	+	S		1200	4	+	S		1200	5	+							
	19_17	S	1200	12	S		1200	7	+	S		1200	5	+	S		1200	6	+	S		1200	5	+							
	4_11	S	1200	4	SC		+	1	+	3		+	1	+	3		+	1	+	3		+	1	+	3		+				
	4_8	SC	1	2	3	SC		1	+	3		SC		1	+	3		SC		1	+	3		SC		1	+	3			
is	1YHXG1yX5	S	1200	194 048	S		1200	145 440	+	S		1200	99 328	+	S		1200	196 384	-	S		1200	88 832	+							
	A3PZaPjnUz	S	1200	144 896	SC		+	1	+	103 936	+	SC		+	1	+	103 936	+	SC		+	2	+	103 936	+						
	HgSWGJHxY5	S	1200	251 200	S		1200	276 000	-	SC		+	1117	+	102 176	+	S		1200	397 120	-	S		1200	102 176	+					
	j29pQqRzJ2	SC	82	210 944	SC		2	+	210 944		SC		1	+	210 944		SC		129	-	210 944		SC		1	+	210 944				
	y21PaVA2Hj	S	1200	236 544	S		1200	165 776	+	S		1200	127 088	+	S		1200	169 312	+	S		1200	134 928	+							
mapping	full2k2	S	1200	1 103	S		1200	801	+	S		1200	795	+	S		1200	816	+	S		1200	809	+							
	mesh2x2.mpeg	S	1200	726	S		1200	1436	-	S		1200	1116	-	S		1200	1145	-	S		1200	1321	-							
	mesh3x3.2	UNK	1200		S		+	1200	1631		S		+	1200	1623		S		+	1200	1600		S		+	1200	1536				
	mesh3x3.mpeg.2	S	1200	2 197	S		1200	1188	+	S		1200	1 211	+	S		1200	1 310	+	S		1200	1 188	+							
	ring.2	S	1200	2 090	S		1200	1 940	+	S		1200	1 940	+	S		1200	2 020	+	S		1200	1 940	+							
multi-knapsack	mknnap-1	UNK	1200		SC		+	7	+	16 537		+	7	+	16 537		SC		+	8	+	16 537		SC		+	6	+	16 537		
	mknnap-2-1	SC	158	7 772	SC		1	+	7 772		SC		2	+	7 772		SC		2	+	7 772		SC		1	+	7 772				
	mknnap-2-2	S	1200	8 722	SC		+	8	+	8 722		SC		+	8	+	8 722		SC		+	10	+	8 722		SC		+	7	+	8 722
	mknnap-2-30	SC	3	6 339	SC		1	+	6 339		SC		1	+	6 339		SC		1	+	6 339		SC		1	+	6 339				
	mknnap-2-32	UNK	1200		S		+	1200	8 947		SC		+	547	+	8 947		UNK		1200		SC		+	730	+	8 947				
nmseq	176	S	6		S		1	+		S		1	+		S		1	+		S		1	+								
	207	S	7		S		1	+		S		1	+		S		2	+		S		1	+								
	269	S	45		S		2	+		S		2	+		S		2	+		S		1	+								
	393	S	244		S		2	+		S		4	+		S		2	+		S		2	+								
	83	S	1		S		1			S		1			S		1			S		1									
opd	flener_et_al_10_350_100	S	1200	65	UNK		-	1200	-	UNK		-	1200	-	S		1200	86	-	S		1200	97	-							
	mediam_10_100_30	S	1200	13	S		1200	21	-	S		1200	10	+	S		1200	19	-	S		1200	23	-							
	small_lbibd_10_30_09	SC	1107	2	S		-	1200	-	3		-	1200	-	S		-	1200	-	S		-	1200	-	3	-					
	small_lbibd_11_22_10	S	1200	5	S		1200	5	-	S		1200	5	-	S		1200	13	-	S		1200	5	-							
	small_lbibd_13_26_06	SC	1078	1	S		-	1200	-	2		-	1200	-	S		-	1200	-	S		-	1200	-	2	-					
open_stacks	problem_20_20_1	S	1200	11	S		1200	11		S		1200	11		S		1200	11		S		1200	11								
	problem_30_15_1	SC	15	14	S		-	1200	-	14		-	1200	-	S		-	1200	-	S		-	1200	-	14						
	wbo_10_20_1	SC	303	5	S		-	1200	-	5		-	1200	-	S		-	1200	-	S		-	1200	-	5						
	wbp_15_30_1	S	1200	7	S		1200	7		S		1200	6	+	S		1200	6	+	S		1200	6	+							
	wbp_20_20_1	S	1200	4	S		1200	4		S		1200	4		S		1200	4		S		1200	4								
pif	12	S	1200	602	UNK		-	1200	-	UNK		-	1200	-	UNK		-	1200	-	UNK		-	1200	-							
	13	C	20		UNK		-	1200	-	UNK		-	1200	-	UNK		-	1200	-	UNK		-	1200	-							
	14	S	1200	1 008	UNK		-	1200	-	UNK		-	1200	-	UNK		-	1200	-	UNK		-	1200	-							
	15	C	68		UNK		-	1200	-	UNK		-	1200	-	UNK		-	1200	-	UNK		-	1200	-							
	17	UNK	1200		UNK		1200			UNK		1200			UNK		1200			UNK		1200									
project-planning	ProjectPlannertest_12.7	S	1200	63	S		1200	19	+	SC		+	1	+	17	+	S		1200	19	+	S		1200	19	+					
	ProjectPlannertest_14.7	S	1200	78	S		1200	32	+	SC		+	347	+	27	+	S		1200	29	+	S		1200	27	+					
	ProjectPlannertest_15.6	S	1200	66	S		1200	37	+	SC		+	994																		

Several of the themes related to poor performance relative to the ORT search are unsurprisingly repeated in the Hypothesis 3.1.2 run results. This suggests that the poor performance of the objective metrics to model problem instances which require search effort near the root node of the tree cannot be overcome by the expanded grammar proposed and should be addressed directly through a revised measurement encapsulating the coverage of the search. The question of whether an expanded grammar is still able to converge to effective strategies is answered in Table 4.18 where the number of improvements and degradations in search performance are tabulated against the ORT and Hypothesis 3.1.1 run results. It is observed that a similar performance profile is obtained between the two grammar sets, maintaining the statistical significance of a difference in mean performance at the $\alpha = 0.01$ level. It is notable that the overall performance of Hypothesis 3.1.2 is lower for both the SAW and multi-objective objective schemes. This is in line with expectations as the search space for the GP is larger and the population parameters were specifically unmodified to avoid engineering a positive result.

Problem Class	Hypothesis 3.1.1 SAW			Hypothesis 3.1.1 multi-objective			Hypothesis 3.1.2 SAW			Hypothesis 3.1.2 multi-objective		
	-	+	Δ	-	+	Δ	-	+	Δ	-	+	Δ
costas-array	1	2	1	0	5	5	1	2	1	0	5	5
cvrp	2	2	0	2	3	1	3	2	-1	2	3	1
freepizza	1	4	3	0	5	5	0	5	5	3	4	1
gfd-schedule	2	9	7	2	7	5	0	5	5	1	6	5
grid-colouring	0	6	6	0	6	6	0	6	6	0	6	6
is	1	6	5	0	9	9	3	4	1	0	7	7
mapping	1	4	3	1	4	3	1	4	3	1	4	3
multi-knapsack	0	7	7	0	8	8	0	6	6	0	8	8
nmseq	0	4	4	0	4	4	0	4	4	0	4	4
opd	8	0	-8	7	1	-6	9	0	-9	8	0	-8
open_stacks	4	0	-4	4	1	-3	5	1	-4	4	1	-3
p1f	6	0	-6	6	0	-6	6	0	-6	6	0	-6
project-planning	0	5	5	0	11	11	0	5	5	0	5	5
radiation	6	2	-4	15	0	-15	7	2	-5	15	0	-15
spot5	1	4	3	1	4	3	2	3	1	1	4	3
tdtsp	0	11	11	0	11	11	0	11	11	0	11	11
triangular	2	3	1	3	3	0	2	4	2	1	4	3
zephyrus	2	2	0	2	4	2	3	4	1	2	4	2
Total	37	71	34	43	86	43	42	68	26	44	76	32

TABLE 4.18: *Hypotheses 3.1.1 and 3.1.2 SAW and multi-objective results summary by problem class (training set).*

The principal question surrounding Hypothesis 3.1.2 is whether there is a marked improvement on certain problem classes through the additional graph features provided to the GP. This requires a comparison of the search results of Hypotheses 3.1.1 and 3.1.2 with one another. A detailed description of the direct comparisons between Hypotheses 3.1.1 and 3.1.2 are provided in Table 4.19 with an aggregated summary by problem class provided in Table 4.20. The latter table highlights that there are no large discrepancies between the performance of the derived search strategies at an aggregate level in the multi-objective case, with the SAW strategies producing, on average, worse strategies than the Hypothesis 3.1.1 SAW GP Search. The table highlights several problem classes where a Hypothesis 3.1.2 feature set has facilitated a marked improvement over the Hypothesis 3.1.1 strategies developed, namely freepizza (SAW), gfd-schedule (multi-objective), radiation (SAW), triangular (multi-objective) and zephyrus (multi-objective). These improved strategies are found in spite of a larger GP search space, which raises a question as to whether more formidable strategies could be found with changes to the GP run configuration.

4.6. GP run results (Hypothesis 3.1.2)

99

Problem Class	Instance	Hypothesis 3.1.1 SAW			Hypothesis 3.1.1 multi-objective			Hypothesis 3.1.2 SAW					Hypothesis 3.1.2 multi-objective						
		Code	Time	Best	Code	Time	Best	Code	Δ	Time	Δ	Best	Δ	Code	Δ	Time	Δ	Best	Δ
costas-array	16	S	1		S	6		S	1				S	5					
	17	S	1		S	7		S	1				S	2	+				
	18	S	364		S	8		S	956	-			S	3	+				
	19	UNK	1200		S	105		UNK	1200				S	1065	-				
	20	UNK	1200		UNK	1200		UNK	1200				UNK	1200					
cvrp	A-n37-k5.vrp	S	1200	1832	S	1200	2122	S	1200		2190	-	S	1200		1914	+		
	A-n64-k9.vrp	S	1200	3545	S	1200	4046	UNK	-	1200			S	1200		3554	+		
	B-n45-k5.vrp	S	1200	2628	S	1200	2467	S	1200		2586	+	S	1200		2654	-		
	P-n16-k8.vrp	S	1200	502	S	1200	450	S	1200		506	-	S	1200		469	-		
	simple2	SC	30	34	SC	19	34	SC	43	-	34		SC	17	+	34			
freepizza	pizza27	S	1200	822299	S	1200	761294	S	1200		763780	+	S	1200		724896	+		
	pizza39	S	1200	987968	S	1200	837068	S	1200		909629	+	S	1200		839000	-		
	pizza45	S	1200	641397	S	1200	571934	S	1200		561848	+	S	1200		569138	+		
	pizza6	SC	5	210	SC	113	210	SC	4		210		S	-	1200	-	230	-	
	pizza78	S	1200	896600	S	1200	714755	S	1200		659044	+	S	1200		638240	+		
gfd-schedule	n120f5d50m50k20	SC	1	1	SC	3	1	SC	1		1		UNK	-	1200	-			
	n180f7d50m30k18	UNK	1200		UNK	1200		SC	+	2	+	1	SC	+	1	+	1		
	n30f3d30m7k4	SC	1	1	SC	1	1	SC	1		1		SC	1		1			
	n50f7d40m10k4	SC	1	1	SC	2	1	UNK	-	1200	-		SC	1		1			
	n75f5d30m20k20	SC	1	1	UNK	1200		UNK	-	1200	-		SC	+	1	+	1		
grid-colouring	10_5	SC	1	3	SC	1	3	SC	1		3		SC	2		3			
	13_11	S	1200	5	S	1200	4	S	1200		4	+	S	1200		4			
	19_17	S	1200	7	S	1200	5	S	1200		6	+	S	1200		5			
	4_11	SC	1	3	SC	1	3	SC	1		3		SC	1		3			
	4_8	SC	1	3	SC	1	3	SC	1		3		SC	1		3			
is	1YHXcG1xYs	S	1200	145440	S	1200	99328	S	1200		196384	-	S	1200		88832	+		
	A3PZaPjnUz	SC	1	103936	SC	1	103936	SC	5	-	103936		SC	2		103936			
	HgSWGJHxY5	S	1200	276000	SC	1117	102176	S	1200		397120	-	S	-	1200	-	102176		
	jZ9pQqRxJ2	SC	2	210944	SC	1	210944	SC	129	-	210944		SC	1		210944			
	y21PnVA2Hj	S	1200	165776	S	1200	127088	S	1200		169312	-	S	1200		134928	-		
mapping	full2x2	S	1200	801	S	1200	795	S	1200		816	-	S	1200		809	-		
	mesh2x2_mpeg	S	1200	1436	S	1200	1116	S	1200		1145	+	S	1200		1321	-		
	mesh3x3_2	S	1200	1631	S	1200	1623	S	1200		1600	+	S	1200		1536	+		
	mesh3x3_mpeg_2	S	1200	1188	S	1200	1211	S	1200		1310	-	S	1200		1188	+		
	ring_2	S	1200	1940	S	1200	1940	S	1200		2020	-	S	1200		1940			
multi-knapsack	mknap1-6	SC	7	16537	SC	7	16537	SC	8		16537		SC	6		16537			
	mknap2-1	SC	1	7772	SC	2	7772	SC	2		7772		SC	1		7772			
	mknap2-2	SC	8	8722	SC	8	8722	SC	10	-	8722		SC	7		8722			
	mknap2-20	SC	1	6339	SC	1	6339	SC	1		6339		SC	1		6339			
	mknap2-32	S	1200	8947	SC	547	8947	UNK	-	1200			SC	730	-	8947			
nmseq	176	S	1		S	1		S	1				S	1					
	207	S	1		S	1		S	2				S	1					
	269	S	2		S	2		S	2				S	1					
	393	S	2		S	4		S	2				S	2	+				
	83	S	1		S	1		S	1				S	1					
opd	flener_et_al.10.350.100	UNK	1200		UNK	1200		S	+	1200		86	S	+	1200		97		
	medium.10.100.30	S	1200	21	S	1200	10	S	1200		19	+	S	1200		23	-		
	small_bibd.10.30.09	S	1200	3	S	1200	3	S	1200		16	-	S	1200		3			
	small_bibd.11.22.10	S	1200	5	S	1200	5	S	1200		13	-	S	1200		5			
	small_bibd.13.26.06	S	1200	2	S	1200	2	S	1200		2		S	1200		2			
open_stacks	problem.20.20.1	S	1200	11	S	1200	11	S	1200		11		S	1200		11			
	problem.30.15.1	S	1200	14	S	1200	14	S	1200		15	-	S	1200		14			
	wbo.10.20.1	S	1200	5	S	1200	5	S	1200		5		S	1200		5			
	wbop.15.30.1	S	1200	7	S	1200	6	S	1200		6	+	S	1200		6			
	wbp.20.20.1	S	1200	4	S	1200	4	S	1200		4		S	1200		4			
plf	12	UNK	1200		UNK	1200		UNK	1200				UNK	1200					
	13	UNK	1200		UNK	1200		UNK	1200				UNK	1200					
	14	UNK	1200		UNK	1200		UNK	1200				UNK	1200					
	15	UNK	1200		UNK	1200		UNK	1200				UNK	1200					
	17	UNK	1200		UNK	1200		UNK	1200				UNK	1200					
project-planning	ProjectPlannertest_12.7	S	1200	19	SC	1	17	S	1200		19		S	-	1200	-	19	-	
	ProjectPlannertest_14.7	S	1200	32	SC	347	27	S	1200		29	+	S	-	1200	-	27		
	ProjectPlannertest_15.6	S	1200	37	SC	994	31	S	1200		40	-	S	-	1200	-	40	-	
	ProjectPlannertest_16.6	S	1200	35	S	1200	31	S	1200		33	+	S	1200		33	-		
	ProjectPlannertest_16.8	S	1200	35	S	1200	31	S	1200		38	-	S	1200		31			
radiation	i14-9	S	1200	6720	S	1200	6526	S	1200		6517	+	S	1200		6525	+		
	i6-11	SC	9	895	S	1200	896	SC	10		895		S	1200		898	-		
	i6-21	S	1200	1718	S	1200	1417	S	1200		1417	+	S	1200		1423	-		
	i7-9	SC	6	1007	S	1200	1009	SC	2	+	1007		S	1200		1008	+		
	i9-11	SC	123	2141	S	1200	2151	SC	793	-	2141		S	1200		2148	+		
roster	chicroster_dataset.11	SC	1	17	SC	1	17	SC	1		17		SC	1		17			
	chicroster_dataset.17	SC	1	17	SC	1	17	SC	1		17		SC	1		17			
	chicroster_dataset.2	SC	1	0	SC	1	0	SC	1		0		SC	1		0			
	chicroster_dataset.5	SC	1	6	SC	1	6	SC	1		6		SC	1		6			
	chicroster_dataset.7	SC	1	0	SC	1	0	SC	1		0		SC	1		0			
spot5	1401	S	1200	496114	S	1200	500112	S	1200		521129	-	S	1200		518116	-		
	28	S	1200	276105	S	1200	277105	S	1200		281115	-	S	1200		281115	-		
	414	S	1200	44501	S	1200	49510	S	1200		47499	-	S	1200		46507	+		
	503	S	1200	11125	S	1200	11134	S	1200		11137	-	S	1200		11130	+		
	54	S	1200	37	S	1200	37	S	1200		39	-	S	1200		39	-		
tdtsp	inst_10.24.10	SC	5	9192	SC	3	9192	SC	4		9192		SC	3		9192			
	inst_10.34.00	SC	3	6662	SC	2	6662	SC	2		6662		SC	2		6662			
	inst_10.42.10	SC	4	8486	SC	2	8486	SC	5		8486		SC	2		8486			
	inst_20.14.10	S	1200	14889	S	1200	15801	S	1200		15228	-	S	1200		14756	+		
	inst_20.25.00	S	1200	16655	S	1200	15945	S	1200		16810	-	S	1200		16799	-		
triangular	n10	SC	159	20	S	1200	20												

Problem Class	Hypothesis 3.1.2 SAW			Hypothesis 3.1.2 multi-objective		
	–	+	Δ H 3.1.1–H 3.1.2	–	+	Δ H 3.1.1–H3.1.2
costas-array	1	0	–1	1	2	1
cvrp	4	1	–3	2	3	1
freepizza	0	4	4	4	3	–1
gfd-schedule	4	2	–2	2	4	2
grid-colouring	0	2	2	0	0	0
is	5	0	–5	3	1	–2
mapping	3	2	–1	2	2	0
multi-knapsack	2	0	–2	1	0	–1
nmseq	0	0	0	0	1	1
opd	2	2	0	1	1	0
open_stacks	1	1	0	0	0	0
project-planning	2	2	0	9	0	–9
radiation	1	3	2	2	3	1
spot5	5	0	–5	3	2	–1
tdtsp	2	0	–2	1	1	0
triangular	2	4	2	0	6	6
zephyrus	1	3	2	0	2	2
Total	35	26	–9	31	31	0

TABLE 4.20: Hypothesis 3.1.1–Hypothesis 3.1.2 SAW and multi-objective results summary by problem class (training set).

4.6.1 Extendibility to unseen problem instances (Hypothesis 3.1.2)

It has already been demonstrated that branching strategies developed extend well to unseen problem instances where such instances exhibit similar structures to those in the training data. It is interesting to verify whether the more complex derived features of the underlying CSP variables also extend to unseen problem instances or have fallen victim to overfitting on the test data. The analysis is restricted in this context to those problems that have shown improvements over the Hypothesis 3.1.1 strategies developed, shown in Table 4.21.

The results in Table 4.21 affirm the susceptibility of a single search strategy being misaligned with the structure of the underlying problem. This is demonstrated by the zephyrus problem class highlighted in the previous section.

A contrast to the zephyrus example is the performance of the SAW scheme on the radiation and triangular problem classes. In this context, over this limited test set, the SAW scheme appears to have outperformed the multi-objective solutions selected from the frontier based on the calibration scheme used in the previous section. An approach to combat the sensitivity of the selection scheme is discussed in the next chapter.

The performance of the Hypothesis 3.1.2 strategies developed for the gfd-schedule problem class are somewhat of a disappointment. On the other hand, the performance of the Hypothesis 3.1.2 strategies on the test data was very impressive, especially when contrasted with the results obtained for the test set which are mediocre and only marginally better than the results of the default ORT search. It can be seen from the gfd-schedule instance description that there are five control parameters for generating instances with the last parameter primarily having a small value of k . The training data consisted of instances mostly with a large value of k , with the ORT search performing very well for small values of k . This introduces a bias, in that an algorithm

Problem Class	Instance	ORT			Hypothesis 3.1.2 SAW					Hypothesis 3.1.2 multi-objective						
		Code	Time	Best	Code	Δ	Time	Δ	Best	Δ	Code	Δ	Time	Δ	Best	Δ
gfd-schedule	n10f2d10m10k3	SC	1	3	SC		1		3		SC		1		3	
	n25f5d20m10k3	S	1200	803	S		1200		506	+	S		1200		204	+
	n35f5d20m10k3	UNK	1200		S	+	1200		808		S	+	1200		705	
	n55f2d50m30k3	S	1200	2704	S		1200		3514	-	S		1200		2704	
	n60f7d50m30k10	S	1200	2215	S		1200		9940	-	S		1200		9916	-
radiation	09	SC	204	673	SC		1	+	673		S	-	1200	-	673	
	i6-7	UNK	1200		SC	+	1	+	635		SC	+	1055	+	635	
	i7-15	SC	61	1308	SC		28	+	1308		S	-	1200	-	1316	-
	i8-7	SC	10	1046	SC		1	+	1046		S	-	1200	-	1049	-
	i9-23	UNK	1200		S	+	1200		4361		S	+	1200		4386	
triangular	n18	S	1200	40	S		1200		43	+	S		1200		43	+
	n26	S	1200	56	S		1200		68	+	S		1200		68	+
	n34	S	1200	74	S		1200		93	+	S		1200		93	+
	n40	S	1200	86	S		1200		110	+	S		1200		110	+
	n46	S	1200	98	S		1200		130	+	S		1200		114	+
zephyrus	12-6-8-3	SC	948	1300	S	-	1200	-	1300		S	-	1200	-	1755	-
	12-8-6-3	SC	294	1300	S	-	1200	-	1300		S	-	1200	-	1300	
	14-10-8-3	UNK	1200		S	+	1200		6240		S	+	1200		8840	
	14-6-8-3	SC	247	1170	S	-	1200	-	8190	-	S	-	1200	-	7865	-
	14-8-6-3	SC	87	1170	S	-	1200	-	9230	-	S	-	1200	-	10270	-

TABLE 4.21: Hypothesis 3.1.2 SAW and multi-objective strategies evaluated on the test set.

has been trained in this context to favour instances with large values of k , but is presented a test set with the opposite. The approach of deploying the same search strategy within a problem class between the training and test sets will be relaxed in the proceeding chapter as there are several instances where a dominant underlying structure may become more or less pervasive as the parametrisation of the problem changes.

4.6.2 Strategy comparison

The analysis of run results up to this point has been limited to the outcomes of the possible search strategies. The focus shifts in this section to consider the composition of search strategies which resulted in improvements in the overall search scheme. The discussion is limited to problem instances where considerable improvements were found over the default ORT search strategy. This is not done in an attempt avoid discussion of the inferior branching strategies found, but rather due to an interest in the poor performance of such strategies being lacking. Problem instances which exhibited very poor performance were concluded to be as a result of the objective term (4.1) which favours strategies that maximise the search depth, where strategies which prune effectively near the root node are preferred for such problem classes. The resulting search strategies in this context are not meaningful and are, as a result, omitted.

There are many ways in which the branching strategies developed can be compared. The evolved strategies form complex hierarchical, tree-based structures which make direct analysis difficult. For this reason, aggregate features of the trees are created to flatten the data structures and enable a high-level comparison. In total, several thousand unique trees for variable ranking tree and value ranking tree were created by GP runs in Hypotheses 3.1.1 and 3.1.2 which should be sufficient to tease out some high-level trends, should they exist.

The following chapter provides a detailed treatment of the reduction of trees which, while being unique instances, induce identical evaluation in the ranking functions. Such transformations are performed in order to improve the results of an ML process.

Table 4.22 details the search strategy trees found and evaluated against the ORT default search strategy. This table can be somewhat overwhelming as many of the trees exhibit a complex

hierarchical structure, which, when expressed as an evaluation string, can be rather daunting to parse as a human. The abbreviations used for tree functions can be found in Tables 4.2 and 4.16. In spite of this complexity, there are a few things one may note. A large spread of function sizes is found in the search process. In some instances, trees are compact, consisting of a single terminal node such as the variable ranking function for Hypothesis 3.1.1 SAW costas-array, shown in the first row. This wonderful simplicity, however, is offset by a very complex function for the value evaluation ranking. This particular strategy was faster than the default ORT strategy, but did not perform as well as the Hypothesis 3.1.1 multi-objective, Hypothesis 3.1.2 SAW and Hypothesis 3.1.2 multi-objective strategies found. It can be seen that these superior strategies are significantly more compact in their definition which result in slightly lower computational overheads.

There are several spurious operations which have been generated in certain trees, as shown in Table 4.22. This can be seen where double-negation has been applied in a function, resulting in additional operations which have no effect on the final function value. The GP has no direct incentive to create parsimonious trees, other than through hampered performance in the search performance objective.

The majority of Hypothesis 3.1.2 variable ranking functions contain a component that is provided by the extended grammar containing certain graph features. An illustration of this is the costas-array Hypothesis 3.1.2 multi-objective final solution which squares the pagerank values for nodes in order to determine the ranking of variables in the search. This is a good example of a function that could be converted to a once-off global ranking which would reduce the computational overhead of the ranking function significantly. Any strategy that contains the ‘rand’ function or any state information, such as impacts or failures, cannot simplistically be converted to a once-off global ranking scheme.

There are several similarities across the hypotheses and objectives per problem class that one may observe. In three of the four instances the preferred value selection strategy seems to be the rand function, with the one exception arguably approximating a rand function, given the presence of the rand function near the root of the tree. The mapping problem class also found the $\text{Max}(x)$ function to be the preferred choice for the value selection strategy for three of the four instances where the double inverse in the Hypothesis 3.1.1 SAW cancelled its effect.

The simplicity of the strategy developed for the triangular Hypothesis 3.1.2 multi-objective problem class sheds some light on the efficacy of that particular strategy over the strategies developed in Hypothesis 3.1.1 SAW and multi-objective and Hypothesis 3.1.2 SAW, where the variable selection strategy uses the negative inverse of the strength of a vertex (or variable) in the ranking. The number of instantiations or local impact are common value selection strategies towards which all four cases converged.

The Hypothesis 3.1.2 multi-objective triangular strategy was one of the most effective uses of the additional graph information provided in the extended grammar. It can be seen in the variable ranking column of the Hypothesis 3.1.2 tree descriptions that not all strategies utilised the additional tree nodes provided in the Hypothesis 3.1.2 grammar. This is again not a failure, since it is not a requirement that the additional features be used in the formulation of a branching strategy. If simpler features, in fact, provide a reasonable approximation, then this is an ideal outcome as one may avoid computing more complex graph features for a search.

It can be seen that the strategies found in Table 4.22 are considerably more variable and, in some cases, considerably larger than those found by other researchers using automatic algorithm generation or evolutionary inspired methods. This is in part due to the use of a conventional GP process which is not restricted to constructing the branching strategy through beam search

or bespoke operators, iteration by iteration, and can induce radical changes to the chromosome structure of the GP individual, resulting in larger trees.

Table 4.23 summarises the operator frequency for the solutions presented in Table 4.22. There are a few anecdotal trends one may observe in Table 4.23. There is a tendency during Hypothesis 3.1.1 strategies to create value selection schemes which are more complex. This is observed in the frequency of the basic operators $\{Inv, +, -, *\}$ and thus increased frequency in the instantiation, fails, random and local impact functions, since if additional operators with higher arity are selected, the tree must be filled by additional terminals.

The variable selection operator frequencies indicate that there is a somewhat even distribution of non-basic operators used, barring a few exceptions. Other interesting observations are the general trend of the maximum, as opposed to the minimum, of a variable being preferred as an indicator in a strategy. This may be due to most minimums being zero, which provides little information in a ranking function. The size of a variable domain in this regard is more informative, *i.e.* the range of the variable domain, and this is reflected as having a higher frequency than the max function. One may posit that the repeated presence of a function may be overstating the importance of a function in such an analysis; Table 4.24 attempts to address this.

It is also interesting that the random function is never used in isolation in the strategies formed in Table 4.23, suggesting that it is being used to make slight perturbations, or break ties, in rankings produced by other functions. This would be an intuitive use of the random function.

Table 4.24 summarises the presence of functions in each tree regardless of its frequency. There are two observations which stand out in this table. First, it can be seen that the totals of the value ranking functions used between Hypotheses 3.1.1 and 3.1.2 differ by a large margin. This is interesting as no additional value ranking functions were provided between Hypotheses 3.1.1 and 3.1.2, but additional variable ranking functions were provided. An argument can be made that the change in the structure of the search space for the variable ranking function has resulted in less exploitation by the GP search for the value function trees under the Hypothesis 3.1.2 tree composition. This is intuitive as the exploitation factor increases as convergence of the GP starts to take effect. By keeping the search effort constant between Hypotheses 3.1.1 and 3.1.2 runs, and changing only the size of the search space, one would expect there to be a lower degree of exploitation for non-trivial search spaces.

The second observation is that the general usage of all nodes for the variable ranking tree which bear domain information (such as size, max, success rate *etc.*) typically have a reduced frequency in Hypothesis 3.1.2 compared to Hypothesis 3.1.1. The intuition here is that the initialisation of the GP is going to produce a probabilistically even distribution of nodes of the same arity across individuals in the initial population. Less search effort is applied to any one node in Hypothesis 3.1.2 as is applied, on average, to Hypothesis 3.1.1, attempting to find substitute strategies which may, or may not, outperform the strategies produced in Hypothesis 3.1.1. It is interesting that certain tree nodes, such as the pagerank $P(x)$, do not carry as much value across multiple problem classes as other tree nodes, such as $S(x)$ (the strength of the vertex), which demonstrated a far improved strategy over Hypothesis 3.1.1.

		Hypothesis 3.1.1		Hypothesis 3.1.2	
Function		SAW	Multi-objective	SAW	Multi-objective
Value Ranking	–	6	6	1	7
	*	6	6	3	3
	+	6	8	4	3
	Fails(x,v)	6	7	4	2
	Inst(x,v)	5	6	5	3
	Inv	6	6	8	5
	Limpt(x,v)	6	2	4	2
	rand	6	9	5	7
	Total	47	50	34	32
Variable Ranking	–	3	6	3	3
	*	4	6	4	4
	+	6	7	6	4
	A(x)	0	0	1	1
	Brt(x)	0	0	4	0
	Btw(x)	0	0	1	1
	C(x)	0	0	3	1
	E(x)	0	0	2	0
	H(x)	0	0	1	2
	Impt(x)	2	2	3	2
	Inv	5	6	5	4
	Max(x)	5	3	3	2
	Min(x)	1	1	1	1
	P(x)	0	0	0	1
	rand	2	6	2	3
	Size(x)	5	4	2	2
	SuccRate(x)	3	5	3	2
	S(x)	0	0	1	3
	Total	36	46	45	36

TABLE 4.24: Operator presence over best strategies found for a subset of problem classes for Hypotheses 3.1.1 and 3.1.2 SAW and multi-objective.

The figures presented in Table 4.24 are not able to characterise the interactions between certain functions that are used in conjunction through some transformation. The table is also unable to determine whether a transformation between two functions reduces the effect of one of the functions to approximately zero. It is apparent that analysing such data structures is particularly complicated. In the chapter that follows, an attempt is made to indirectly model the attributes of the trees in a more general manner in order to determine to what degree, if any, a predictive model of performance can be obtained.

4.6.3 Hypothesis 3.1.2 conclusions

The question addressed in this hypothesis is whether providing additional graph features to a search strategy can result in improved search strategies. It was found that there were, in certain instances, equivalent or improved strategies for a limited number of problem classes. The effect of the extended GP grammar was not as considerable as anticipated, but there are reasonable explanations for this which are related to the design of the experiment. The GP configuration

was held constant between Hypotheses 3.1.1 and 3.1.2, although the size of the search space for the GP was increased significantly through the larger grammar. It was noted that the GP seems to have struggled to reach a state of exploitation through convergence for a grammar of this size. That said, rectifying this shortcoming can be addressed by performing configuration tuning on the GP parameters, allowing for more high-quality searches being conducted by the GP on this parameter set.

It is also interesting to observe the distribution change in the tree nodes used by the GP which achieve rather similar results to Hypothesis 3.1.1, but through different functions. This is intuitive in that there may be several projections of domain information, which, after undergoing certain transformations, produce similar ranking values. This may be as a result of symmetry in the graph or structural features which produce the same *relative* feature vectors because the ranking function is only concerned with the relative weighting of variables and values.

A direct analysis of the trees produced was conducted for a subset of problems which demonstrated improvements over the benchmark strategy. This yielded some insights, but by and large, served to highlight the complexity of such an analysis. The ability to interpret what some of the tree structures “mean” appears to be a near-intractable task for larger tree structures, although for smaller trees the strategies are trivial to parse. The observational parsing strategy does not scale, nor does it glean consistent insight into the inner workings of certain strategies. A rigorous treatment of the trees developed is provided in the following chapter.

It is interesting to note where the evolutionary optimisation process has worked well and where it has not. The process has a tendency to work well on problems which meet two criteria: A feasible solution and a representative set of objective components within the sampling limit. Having a feasible solution may seem like a difficult criterion, but for several problem classes, such as the TSP, it is trivial to construct a feasible (albeit a poor quality) solution to a CSP. Determining a policy towards an optimal sampling limit and the relationship between the parameters of the GP search and the grammar applied are both interesting areas of further study.

The ranking function applied also makes for a rich area of further study. The methodology applied within Hypotheses 3.1.1 and 3.1.2 is simplistic, yet effective, for many problem classes. That said, a strategy which works well in the initial phases of a CP search may be less effective near the end of a CSP search. The intuition here is rather simple to follow — it has been observed that the GP struggles with problems which require a non-trivial infeasibility assertion, which is not the initial requirement for many of the problem classes. However, as a series of improvements in the CSP objective function are found, the problem emphasis shifts as less feasible solutions (*i.e.* solutions which meet the constraint of being better quality than the current objective) exist in the updated CSP, until finally no solution exists, resulting in an optimality proof. If a TSP is cast as a CSP, it would be found that constructing an initial incumbent is trivial, but as improved solutions are found, the CSP transitions from a mostly unconstrained optimisation problem to an infeasibility proof. It has been observed that this end-state is not where the current formulation provided to the GP thrives, in that search strategies would be preferred which are able to prune closer to the root node in order to effect an optimality (or infeasibility) proof.

There are several ways in which the ranking function could be modified, not necessarily by altering the mechanics, but by rather calling the underlying tree structure to evaluate a rank value. The merits of cached global ranking values for deterministic ranking values has already been discussed and would make for a more performant implementation, but only to a maximum of a 20–30% speed improvement whereas far larger improvements in overall search performance are found through manipulation of the functions themselves. A GP could use additional trees in its chromosome structure to define solution strategies where a feasible solution already exists,

or where an optimality gap (albeit typically poor in CP) takes on a particular value, or where a certain amount of time has passed without admitting an improved solution. There are multiple possible ways in which a mixture of strategies could be combined to directly address the type of search required for a particular CSP search state. ADFs are also a candidate approach for encapsulating tree structures which are representative of some common functionality that may be typically required in multiple contexts and it would be interesting to test the effect of using ADFs on the existing benchmarks evaluated.

CHAPTER 5

Strategy Selection

Contents

5.1	Data preparation and representation	110
	5.1.1 <i>Representation</i>	110
	5.1.2 <i>Data preparation</i>	112
5.2	Linear regression for strategy selection	114
5.3	Deep learning for strategy selection	115
5.4	Portfolio selection	121
	5.4.1 <i>Extendibility to unseen problem instances</i>	127
5.5	Hypothesis 3.1.3 conclusions	130

In this chapter, the aim is to investigate Hypothesis 3.1.3. At the core of this chapter is the requirement to provision a prediction based on features of CSP in order to select an algorithm from a portfolio of existing search strategies. The input data are restricted to metrics which are collected within the sampling limit of a candidate search. The intuition behind this restriction is two-fold: First, collecting data within a small sampling limit is far quicker than requiring complete runs (potentially 20 minutes per observation) and, secondly, in order to standardise the duration of inspection, or measurement, of algorithm performance on a particular CSP. The latter requirement is to avoid modelling the expected duration of a run, or introducing additional parameters which adjust for variable durations of possible sampling limits. While this would be an interesting aspect of the data to model, it is considered outside the scope of this dissertation.

Two ML methods will be applied to the data in order to assess the accuracy of the underlying predictions over a training set, a testing set and a validation data set. The first model employed is a linear regression model, which is not expected to perform exceedingly well due to the limited parametrisation and inherent assumptions of the model which are linearity and the independence of variables. The linear model will, however, provide a reasonable benchmark of performance which is well understood by many practitioners.

The second method applied to the data is a deep learning model; more precisely, a multi-layered neural network. A description of the network architecture employed, the model fitting procedure and alternative representations is provided. The accuracy of the deep learning model is compared to that of the regression model before testing follows against CSP problem classes which do not form part of the training, testing or validation data sets. The problem sets considered in this chapter are consistent with those considered in the previous chapter for continuity and CSP run results are presented in the same format.

Analysis of the predictions produced by the regression and deep learning models indicate that both models have a tendency to match previously untested strategies to problem instances. A guided local search procedure is proposed to refine the deep learning model. The overall performance of the modelling approach is evaluated as a portfolio selection technique and compared to the benchmark default search strategy in ORT.

5.1 Data preparation and representation

The first hurdle presented in most prediction modelling exercises is that of how data may be represented. The modeller is tasked with casting the features measured into rows which form independent observations of the domain in question. The features of each row form the columns of the input data, which, as one would expect, takes on a tabular form when viewed in totality. Many features can be directly measured, or transformed through simple aggregation techniques, allowing the modeller to transform the data into a common data type, such as a numerical, categorical, or ordinal variable. Utilising these core data types permits the use of multiple frameworks as this representation is fundamental in traditional predictive modelling. This process of data-transformation is colloquially referred to as *data wrangling* and encapsulates the process of creating data-pipelines and transformations as a prelude to the modelling phase.

5.1.1 Representation

The desired process to model is that of a search strategy, which adopts a tree-based representation in its native form. Directly modelling a tree as an input to a predictive model such as a regression model does not fit the paradigm outlined by the common data types permitted. By contrast, many techniques may produce a decision tree as the output of a predictive model, which is of limited use in this context. A simple thought experiment of how to explicitly model a decision tree to a collection of input variables suggests that mapping all possible states an input tree can adopt is required in order to model the input data. This results in an exponential mapping between a set of input data and the variables in the predictive model thereby encapsulating the dependency between tree nodes through the structure of the variables. For a reasonably sized tree and grammar set, this may induce a prohibitively large number of variables ($O(n^{n-1})$) in the predictive model, far exceeding the number of observations of the data.

There are conditions under which such statistical models can be fitted using the appropriate Bayesian techniques where the number of columns exceeds the number of rows. In this context, however, it points to an incorrect encapsulation of the input representation. This particular level of granular modelling is unable to exploit symmetry reductions in the data which may improve the overall model fit.

An alternative approach to directly modelling all possible states and relations of the input data, is to model the tree as an image. This would allow a CNN to learn possible common relations between nodes as part of the convolutional layers, effectively compressing the explicit representation of states into a series of convolution layers.

While this may very well be a possible way of approaching the representation problem, there is a step in the procedure which feels decidedly imprecise. A compact, lossless representation is transformed to a digital image, which is then reinterpreted and ingested for training. There are examples in the literature where such methodologies to modelling are adopted, but it must be said that approaches such as these are inelegant.

A more elegant alternative approach to representing the tree as an image is to rather compute the maximum number of nodes present in a tree, independently of the type of node, and to provide an adjacency representation of the connections between such nodes. This is a compact, lossless representation, which would then require variables representing the interactions of node variables and adjacency variables for each tree. A representation such as this would be a poor fit for a linear regression model, as each estimated interaction term would reduce the number of *degrees of freedom* (DF) available, and very quickly there would be insufficient DF available to estimate the parameters of the model. This representation highlights the fact that the thought process of modelling a tree begins to approximate an LSTM model which would directly parse the tree statement as a sentence and model the interactions between terms of the sentence at a given interval. LSTM models perform well in language modelling and would be a suitable strategy for modelling the input grammars.

A complexity of the LSTM modelling method is that multiple grammars are formed independently, depicting the variable and value ranking trees as well as restart and conflict parameters, which steers slightly away from classical grammar parsing where a single input stream is considered. A possible modelling approach would be to create multiple LSTM layers, including the CSP instance features, concatenating such layers together into a master layer, thus consolidating the inputs before additional layers are applied. Another option would be to concatenate the grammars into a single sentence using an appropriate caret to separate the components of the tree, once again concatenating this layer with a CSP instance feature layer. The latter modelling option is simpler from a neural network construction perspective but loses the explicit structure which keeps the interpretation of the trees independent of one another.

As opposed to directly modelling the search strategies as trees or performing some digital image representation of the tree, a transformation is applied to the tree to reduce it to a set of aggregated statistics which induce an approximation of the properties of the tree. This process incurs a loss of information as it is not possible in all situations, apart from the trivial cases, to recover the structure and relations of a tree based solely on such aggregated data. The purpose is to discover properties which are potentially generalisable across problem classes, and moreover, allows that if a particular representation approximation is insufficient, it can be increased in granularity which tends towards a complete representation. This enables the control of the amount of information permitted to a learning algorithm and an understanding of the biases being leveraged in each stepwise increase in information. This simple aggregation process would also permit benchmarking a linear model (with an optimal fit) to a neural network (with a locally optimal fit) by increasing the number of parameters. Once a reasonable baseline of performance has been established, more complex models which provide fine-grained tree modelling may be considered.

The aggregated properties of the search strategy trees employed in the forthcoming learning models are provided in Table 4.23 whereby the frequency of operators in the proposed input tree are tabulated. This frequency-based representation incentivises a learning algorithm to derive a fit based on the approximate composition of operators, while withholding knowledge pertaining to the relationship of such operators. There is an added benefit that a probabilistic model can be derived once a learning model is fitted to determine which operators should be selected in a GA population initialisation procedure by conditioning on nodes already present in the tree. This model may also serve as an alternative mutation or crossover operator if embedded in a GP — an approach currently not attempted in the literature.

5.1.2 Data preparation

Before search strategy trees are provided as input to a learning model, some basic data pre-processing may be performed, such as a reduction of equivalent strategies in the inputs. There are obvious ways in which tree structures may be equivalent. A tree $T_x = a + b$ can be written as $T_y = b + a$ which will produce an identical output in a ranking function. As such, a lexicographical reordering of all commutative operators within a tree would allow for such equivalent strategies to be reduced to their lexicographical equivalents.

The GP run results of Hypotheses 3.1.1 and 3.1.2 utilising the sampling limit for the search strategy are used to derive a data set which provides possible input trees. In total, 86 858 GP individuals were evaluated across the four runs performed on the MiniZinc Challenge data, consisting of 8 688 unique variable ranking strategies, and 10 246 unique value ranking strategies. Combining the uniqueness of strategy combinations between the variable, value, restart and last-conflict trees produces 44 479 unique combinations of strategies sampled, slightly more than half the total number of individuals evaluated. This serves as an upper bound check on the uniqueness of search strategies evaluated. The uniqueness of the search trees discussed at this point have not undergone any lexicographical reductions.

A lexicographical reduction of the trees utilising the commutative property of the addition and multiplication operator, in conjunction with the removal of double negatives¹, was able to reduce the number of unique trees in the variable ranking tree by 9.2% and the value ranking tree by 11%. This result was initially surprising as it seemed probabilistically intuitive that the reduction rates would have been higher. Individual trees were inspected for possible further reductions and it was found that far more complex operations could be applied to further reduce evaluation-equivalent trees. To avoid programming an elaborate tree reduction procedure, a statistical approach was adopted, paying close attention to the sensitivity of particular functions to certain values.

A set of mock input data was created to emulate the values that would be returned by functions in Tables 4.2 and 4.16. Functions that exhibit sensitivity to particular values, such as the protected division operator, were provided specific samples which exercised the value sensitivity at the precise boundary at which special behaviour, such as zero in the case of protected division, is applicable. Hundreds of random samples of input values were generated and trees were considered evaluation-equivalent if the result returned by a tree across all samples is identical. The assertion of evaluation-equivalence is thus probabilistic, but for a sufficiently large number of samples, quickly becomes a guarantee for all intents and purposes.

The probabilistic approach to identifying equivalence between variable and value ranking trees was able to reduce the number of trees by 11.8% and 16.9% respectively which demonstrates that sometimes the simplest approach to a reduction problem is a probabilistic one. Although the statistical approach is computationally more intensive, the simplicity of the routine to reduce the trees makes this approach rather attractive. Once a collection of trees has been identified as being evaluation-equivalent, the tree with the smallest number of nodes is selected as a representative candidate for the equivalence set. The data are then updated to use the representative candidate trees for all equivalence sets.

The discussion thus far has focussed on the transformations of the search strategy trees to a format conducive to linear modelling techniques. The focus briefly shifts to the additional features which can be added to each search trial, namely, the features of the FlatZinc model.

¹Double division is not necessarily a redundant operator sequence as protected division of a zero will, in fact, not return the same result as normal division.

It is fortunate that the authors of the portfolio solver SUNNY-CP [7, 6] open-sourced the codes [5] in order to derive a set of 95 features, given a target FlatZinc model. In order to keep the results reproducible, only one feature was directly removed from the set of 95 features, namely, ‘gc_ratio_diff’, representing the global constraint ratio difference as a result of predominantly containing non-numeric values. The ‘o_dom_deg’ feature was adjusted to have a value of -1 instead of infinite values. 16 columns consisting of numeric values but with zero variance (*i.e.* the mean, minimum and maximum were equal) were also removed. The final set of features were thus reduced to 78 of the original 95 features.

Most ML models will produce superior results if the input data are normalised prior to training. This is a sensible requirement in the case of linear regression models as the error distribution assumed is a normal distribution. In the case of neural networks, this exact assumption is not present, but a normalisation of input data can improve the descent process as there are less extreme variations between the effect of parameters and their output. As such, a simple normalisation procedure was applied to the data. For a particular feature of the data \mathbf{x} , the normalisation

$$\mathbf{x}_n = \frac{\mathbf{x} - \bar{\mathbf{x}}}{sd(\mathbf{x})} \quad (5.1)$$

is applied. It should be noted that (5.1) transforms the data to typically fall in the range $[-1, 1]$. That said, a large degree of variation may still occur around the mean, especially if the source distribution is highly skewed. In order to address potential skewness, the additional transformation

$$\mathbf{x}_{nl} = \frac{\mathbf{y} - \bar{\mathbf{y}}}{sd(\mathbf{y})}, \text{ where } \mathbf{y} = \log(\mathbf{x}), \quad (5.2)$$

is applied. A final normalisation can then be selected between \mathbf{x}_n and \mathbf{x}_{nl} , whichever has higher symmetry about the mean, measured by selecting a normalisation that minimises the difference between absolute squared errors above and below the mean. This transformation produced 11 normal transformations and 98 log transformations for the input feature vectors. The bias towards the log transformation is as a result of the majority of the data being counts of certain attributes, which are bounded from below by zero and produce long tails. The appropriate offsets were applied to the input data (\mathbf{x}) where zeros are present in (5.2) to produce a strictly positive projection of the input vector. The total number of transformations is larger than the number of features described by the MZN2FEAT package. The reason for this increased number of features is that features of the input trees are appended to each observation in the data as a result of the aggregation transformations discussed. These total 23 additional features describing the frequency of the nodes used in each tree.

Once the dependent variables have been defined, the final component required to build a predictive model is the independent variable, or response variable. As discussed in Chapter 4, there are multiple attributes which can be measured from a given search strategy within the sampling limit. An aggregation of these metrics is performed to form an SAW objective function of the form (4.4). The observations in Chapter 4 outlined that a multi-objective approach to GP search is preferred, but that the performance of the SAW scheme was still able to outperform the default ORT search strategies. For simplicity, a single response variable is targeted that encapsulates multiple measurements which is the SAW objective. This is a convenient standardisation, which allows for comparisons between a regression and deep learning model. However, it is not a strict requirement in neural network modelling as most packages support modelling multiple output layers which are equivalent to multivariate linear regression response variables.

5.2 Linear regression for strategy selection

The data transformations described in the previous sections which relate to column normalisation and tree operator frequency aggregation are applied to the input data, consisting of 107 columns and 433 835 rows. There are three categorical columns of the 107 columns, namely the problem class, the last-conflict and log-restartsize variables. All detailed tree information for the variable and value selection strategies are condensed to a frequency matrix. The balance of the columns are numeric and are normalised according to the procedure described in the previous section.

The data are partitioned into training and testing data sets consisting of data randomly drawn at a rate of 90% and 10%, respectively. It is reasonable for the selected subset of data used for training to be large in the regression model as the number of parameters being fitted is relatively low compared to the number of observations. In addition, there are no bias terms to control in a regression. The test set will serve to determine whether the experienced MSE during training is carried forward to the test set in a similar quantity per observation. A preliminary linear regression model is fitted to the training data set, revealing a structural identity between several columns which are then unable to have parameters fitted. The eight offending MZN2FEAT feature columns were removed from the data set, namely s_bool_search, s_first_fail, s_goal, s_indomain_max, s_indomain_min, s_input_order, s_int_search, and s_other_val — all of which are search annotation columns and not applicable to free search.

A linear regression model is then fitted to the revised data set, described in Table 5.1. For conciseness, the details of the p -values for all fitted parameters are omitted. For the 133 parameters in the model, 119 variables had a significant p -value at the $\alpha = 0.01$ level, with 11 parameters being insignificant at the $\alpha = 0.05$ level.

Statistic	Value
Residual standard error	0.6495 on 433 702 DF
Multiple R-squared	0.5783
Adjusted R-squared	0.5781
MSE	0.4217
F-statistic	4 505 on 132 and 433 702 DF
Model p -value	$< 2.2 \times 10^{-16}$
Validation MSE	0.4193

TABLE 5.1: *Linear regression model performance summary.*

The mean square error is tricky to interpret directly as the response variable has been normalised. It can, however, be noted that a satisfactory validation MSE is obtained which is very close to the fitted model MSE which suggests that the estimates predicted by the linear model are, at a minimum, consistent on the validation data set. The overall p -value for the fitted model is significant, at any reasonable level of α , suggesting that while some variables may not be significant individually, the fitted model does explain a significant amount of the overall variability in the response variable.

A more complex model with higher-order interaction terms will certainly provide a better fit to the data. The computational complexity of the matrix inversion operations and the memory required to fit such a model, however, provide a barrier. A simple experiment was conducted to include all pair-wise interaction terms which caused approximately 13 000 parameters to be fitted in the regression model. Unfortunately there is insufficient memory available on most commodity computing devices in order to complete the closed-form inversion using the standard

lapack matrix operators available in R. That said, removing the categorical variables from the pair-wise interactions results in a sufficient decrease in the parameter space and, as such, the memory requirement to perform the matrix inversion decreases to approximately 35Gb of memory. Unfortunately, after 24 hours of computation, this attempt was abandoned. If providing a set of parameter estimates was a necessity, a SGD method could be applied to bypass the exact matrix inversion to fit parameter values. This is mentioned merely to provide a reference point for the size of the operations being performed in this context.

This simple linear regression model serves as a compact baseline model against which a deep learning model may now be compared.

5.3 Deep learning for strategy selection

Several candidate deep learning models are discussed in this section, the first of which employs an input scheme near-identical to that of the linear regression model provided as a baseline. This allows for the fitting of models with a higher number of interaction terms between input parameters and measuring the improvements in MSE through a parameter-rich representation. Variations of the simple neural network will be explored in an iterative manner.

It is customary when presenting deep learning models to provide an overview of the final architecture employed and the methodology applied to arrive at such an architecture. The initial deep learning model contains no *hidden layers*, instead it directly connects feature inputs to the output layer. This permits verifying that the inputs provided are consistent with the linear model and so the MSE of the model can be verified as being within the region of the regression model. *Dense* hidden layers may then be added iteratively to the neural network model. A dense set of connections between each layer means that each node in a given layer is fully connected to each node in the proceeding layer. For a given layer with n nodes, and a proceeding layer with m nodes, there are nm parameters to estimate between two fully connected layers.

The methodology applied to model a regression-style problem in a neural network requires that the output layer is aggregated to a single *neuron* which will provide the final estimated value, *i.e.* the dot product of the input values and weights, equivalent to the response variable in linear regression. It is typical to modify the network to incorporate *l1* and *l2 regularisation* terms with respect to the validation error. Regularisation terms assist in the reduction of potential overfitting of the data and allows for control of the degree to which overfitting is incurred. The determination of the weights of the regularisation terms is one of the parameters required as part of the model calibration procedure.

The input layer adopts the same shape as the input data, whereby categorical variables are encoded as a *one hot column*. A one hot column is a sparse representation, similar to that used by linear regression to encode a categorical variable, whereby a number of additional columns consisting of binary variables are created where the sum of the row across such columns equals exactly one, corresponding to the category represented. Modern neural network packages support a hashed, or compressed encodings, which was not adopted in this dissertation, although this is an attractive feature for categorical variables with a large number of possible states. A key difference between one hot encoding and the categorical encoding used by linear models is that there is typically no intercept in a neural network. The intercept in linear regression corresponds to the first categorical level in each category and parameters are estimated for the remaining categories which move the model to other states. A neural network excludes an intercept and models the additional initial categories independently from one another.

As the input layer is defined to take the shape of the input data, with the relevant one hot encodings, and the output layer defined as a single neuron, the number of hidden layers and the size of such layers are the last set of free variables to determine in the construction of a network architecture. There are many works available in the literature which focus solely on the methodology applied to determine a network architecture, but this is not the focus of this dissertation. The primary focus in this modelling exercise is to produce a network that performs sufficiently well, given a set of inputs and the baseline linear regression model as a point of comparison, without employing an overly-complex architecture, which is a subjective requirement.

The methodology applied to determining the network architecture is an iterative process of increasing the number of dense layers and measuring the MSE in respect of the training and testing data sets. When a layer is added that provides a relatively small reduction in the MSE, relative to the reduction of the previous layer, the addition of layers is halted and the final layer may be reduced in size to the point at which the small reduction in MSE is predominantly retained. At each step of this iterative process, a network consisting purely of *sigmoid* units is applied to the *hidden layers*, *i.e.* layers between the input layer and the output layer, and is trained over a number of *epochs*.

The term epoch in the context of training neural networks refers to the number of SGD iterations performed for a given set of parameters. Several of the mainstream activation functions available in the literature were tested and it was empirically established that the sigmoid activation function worked particularly well for the way in which the network was framed in this context. *Rectified linear units* (ReLU) were extensively experimented with, but it was found that the activation functions had a tendency to produce sporadic predictive errors in unseen samples. A potential reason for volatility is that the ReLU activation is unbounded in the contribution to the network as a function of a given input in the positive domain. A sigmoid function has the ability to bound the output on a $[0, 1]$ domain, which helps reduce the influence of input data, or relationships between neurons, which are larger than initially anticipated.

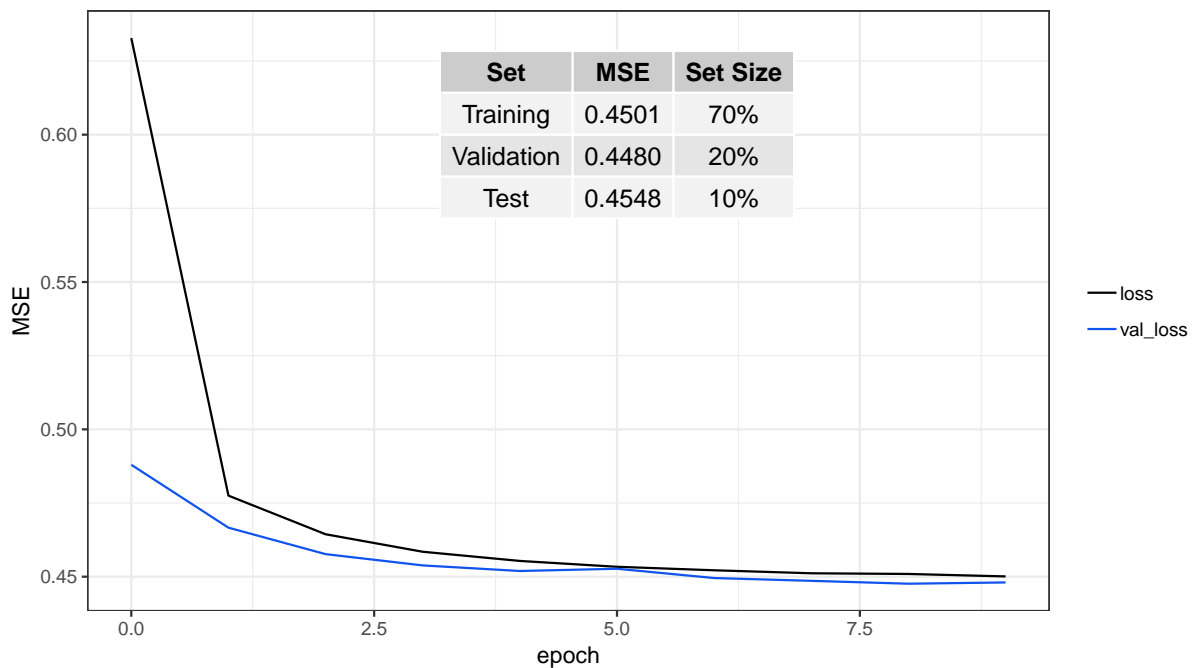


FIGURE 5.1: M_1 — 136 parameter neural network training and validation MSE per epoch.

The network design process is commenced by validating the results produced by the linear regression in a neural network context. The results² of a single input layer and output neuron are shown in Figure 5.1. The trajectory depicted in this figure is that followed by the SGD during training using the *Adam* optimisation technique [102]. The training error is depicted by the black line provided and validation error by the blue line (`val_loss`). No regularisation is applied to this model and a 70%, 20% split is made between the training and validation data sets.

Overfitting was not observed during the 10 epochs performed as the validation error does not climb above the training error and this is verified by a reasonable MSE on the test data set *i.e.* data which were withheld during the training procedure. The MSE produced by this simple neural network model is within 5% of the regression model, which is to be expected as the regression model determines an optimal parameter fit. That said, this step serves as a solid point of departure as it has been verified that the format of the data being provided to the neural network and the measurement of the MSE is consistent between these two learning models.

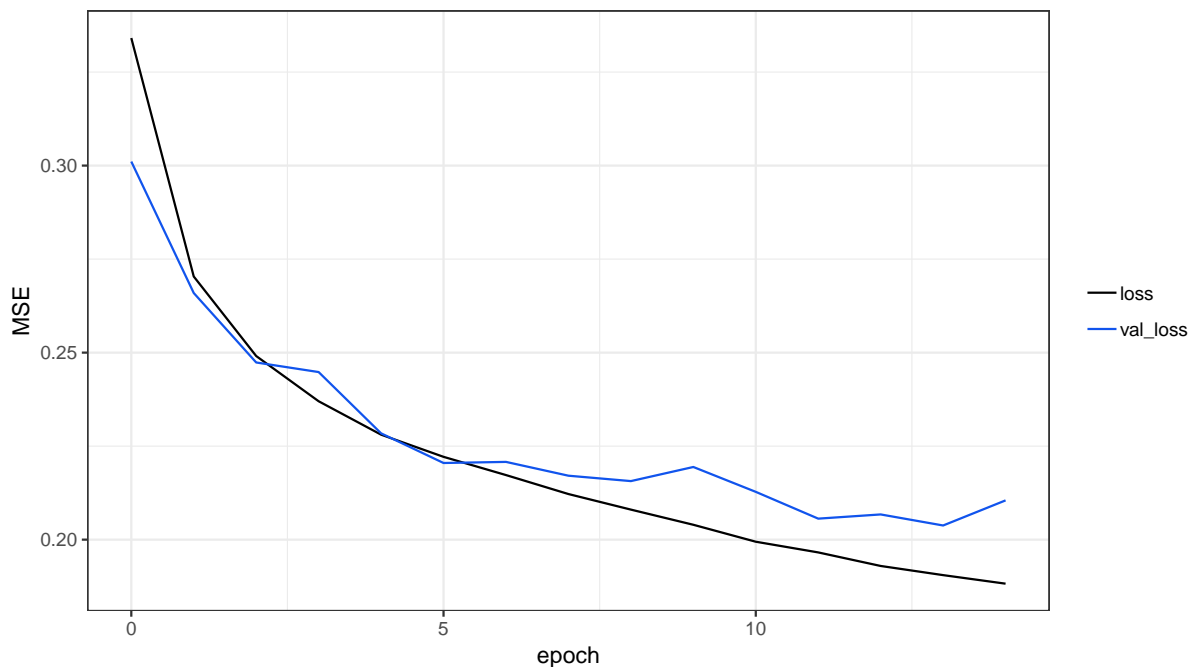


FIGURE 5.2: M_2 — 28 577 parameter neural network training and validation MSE per epoch.

The baseline established through M_1 may now be modified to include additional dense layers iteratively added to the neural network layers, decreasing in size, until a sufficiently small MSE is achieved. Attention is paid to retaining the competency of the network on both the validation and test data sets.

The intuition supporting the decrease in the size of each successive layer is that it is desirable to encourage the network to extract higher-level features from fine grained metrics. This is a commonplace technique in CNNs for image processing.

²The successive models are denoted M_1, M_2, M_3, \dots where M refers to model and i subscript the modelling iteration. The first model, which is approximately equivalent to the linear regression model, save for 3 additional parameters, is denoted M_1 . It makes sense in certain contexts to overlay the training runs of successive models on the same Figure to enable easy comparisons and in such cases, the associated runs will be adorned with the aforementioned notation.

The MSE produced by the architecture presented in Table 5.2 has significantly reduced the MSE by a factor of two, compared to the simple neural network with a single input layer and is denoted as M_2 .

The convergence profile of the training and validation errors presented in Figure 5.2 show that there appears to be overfitting during later epochs as the error on the validation data set begins to climb while the training error continues to decrease.

Layer	Dimension	Number of parameters	Activation Function
Input	136×1		
Dense 1	136×136	18 496	sigmoid
Dense 2	137×60	8 220	sigmoid
Dense 3	61×30	1 830	sigmoid
Dense 4	31×1	31	sigmoid
Output	1×1		
		28 577	

TABLE 5.2: M_2 and M_3 — 28 577 parameter neural network layer architecture.

In order to address the increase in validation error, l1 and l2 regularisation is employed with weights of 0.01 for both terms. The result of the regularised training run is shown in Figure 5.3. The number of epochs is increased to 20 for this run, and it can be seen in Figure 5.3 that the validation error does not diverge and that the gap between the training and validation error remains fairly constant near the end of the run. The batch size of training examples was fixed to 256 samples per batch and the training was performed on a GeForce GTX 1080 Ti GPU. Each epoch takes approximately 4 seconds to complete, with the total training time taking around a minute and a half, corresponding to the training run shown in Figure 5.3 according to the architecture in Table 5.2.

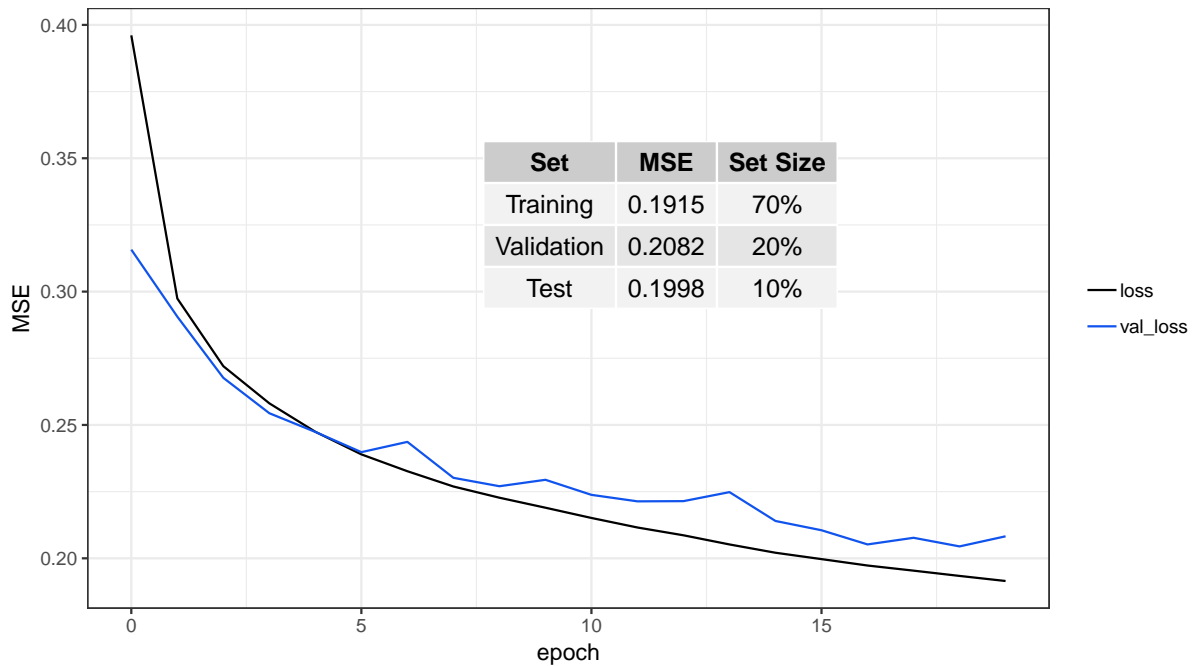


FIGURE 5.3: M_3 — 28 577 parameter neural network training and validation MSE per epoch using l1 and l2 regularisation.

Potential concerns when working with regularisation are firstly whether the parameter values of 0.01 for the l1 and l2 regularisation are appropriate, and secondly whether these values should remain constant throughout the training run. Approaches have been proposed in the literature in which the regularisation decays over the course of the training process. An alternative to using regularisation is to rather employ node *dropout* [167] at each dense layer in the network. The concept of dropout involves probabilistically removing activation nodes from the network during training. This not only encourages a sparse representation but also prevents downstream activations from attempting to correct for activation mistakes made earlier on in the network. The intended outcome of applying dropouts in a neural network is to encourage a robust model. It is expected that the network will be unable to match the performance of a previous run without any dropouts, but will result in a robust representation without sensitivities to specific states which may be overfitting the input data.

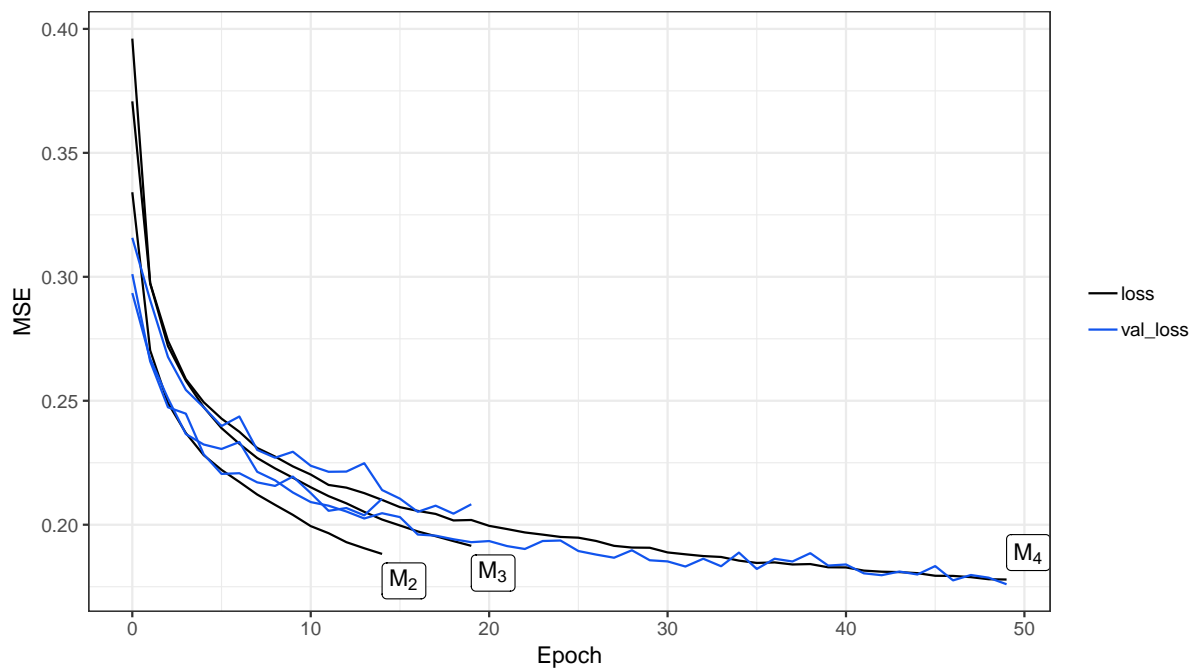


FIGURE 5.4: M_4 performance contrast when employing 5% dropout rate per layer.

As a result of the stochasticity of a model using node dropouts, training time is again increased to permit the optimisation procedure to reach a reasonable convergence point. The results of a model which removes the l1 and l2 regularisation terms, but introduces dropout at a rate of 5% per activation node for each layer of the network in Table 5.2 is shown in Figure 5.4. The recommendations of Srivastava *et al.* [167] suggest a dropout rate as high as 50% per node which works for larger (wide) networks. This was empirically trialled and found to have too great an impact on the resulting model fit for this small network and was reduced as a result. The epochs for this training run were again increased to a total of 50.

It is clear that the dropout methodology produces more erratic validation errors. This, in part, also illustrates the sensitivity of the response variable to specific portions of the network. This model is preferred to the l1 and l2 regularised network as it has been fitted in such a manner as to provide robust estimates, which, at progressively lower levels of MSE, makes it prudent to forgo accuracy in favour of generality.

It is worth noting how well the dropout methodology has worked in contrast to the regularisation. The validation error is kept close to, or below, the training error for the majority of the training run and results in a considerable improvement over M_3 achieving a validation error almost equivalent to the training error at the 20th epoch.

Now that it has been established that a stable methodology and appropriate parameters have been applied to fit a large neural network, an additional series of dense layers can be applied to the model to potentially learn increasingly higher-level interactions.

A large network architecture is proposed in Table 5.3 and, once again, the number of epochs is increased to 100, as shown in Figure 5.5. The diminishing returns of increasing the network size is visible in this figure as a marginal decrease in the MSE is achieved for slightly more than double the number of network parameters. This is, however, not a concern as the validation error is well controlled by the dropout nodes. It may be possible to further increase the size of the network, but it seems as though a reasonable compromise has been met between the generality and size of the network. The training, validation and test error for all models are provided in Table 5.4.

Layer	Dimension	Number of parameters	Dropout rate	Activation Function
Input	136×1			
Dense 1	136×136	18 496	0.1	sigmoid
Dense 2	137×136	18 496	0.1	sigmoid
Dense 3	137×136	18 496	0.1	sigmoid
Dense 4	137×60	8 220	0.1	sigmoid
Dense 5	61×30	1 830	0.05	sigmoid
Dense 6	31×1	31	0.05	sigmoid
Output	1×1			
		65 841		

TABLE 5.3: M_5 — 65 841 parameter neural network layer architecture.

The M_5 neural network utilising dropout is the model selected as candidate model to predict the performance of different search strategies using the operator frequency statistics representation. The training time for this model increased slightly as a result of the increased parameter space to 6 seconds per epoch, resulting in approximately 10 minutes of training time.

Set	Set Size	M_1	M_2	M_3	M_4	M_5
Test	10%	0.4548	0.2051	0.1998	0.1777	0.1673
Training	70%	0.4501	0.1882	0.1915	0.1779	0.1665
Validation	20%	0.4480	0.2105	0.2082	0.1760	0.1667
Parameters		136	28 577	28 577	28 577	65 841
l1 l2 regularisation		No	No	Yes	No	No
Dropout		No	No	No	Yes	Yes
Training iterations		10	15	20	50	100

TABLE 5.4: Model train, validate and test MSE and architecture summary.

It may seem disingenuous to compare a simple linear model to a heavily parametrised neural network. In an ideal world, a regression model with additional interaction terms would make for a superior benchmark, but this poses several new complexities. The determination of which interaction terms should be included, as well as a bespoke SGD method, are required in order to produce a linear regression model which can operate at this tier of complexity. Given that

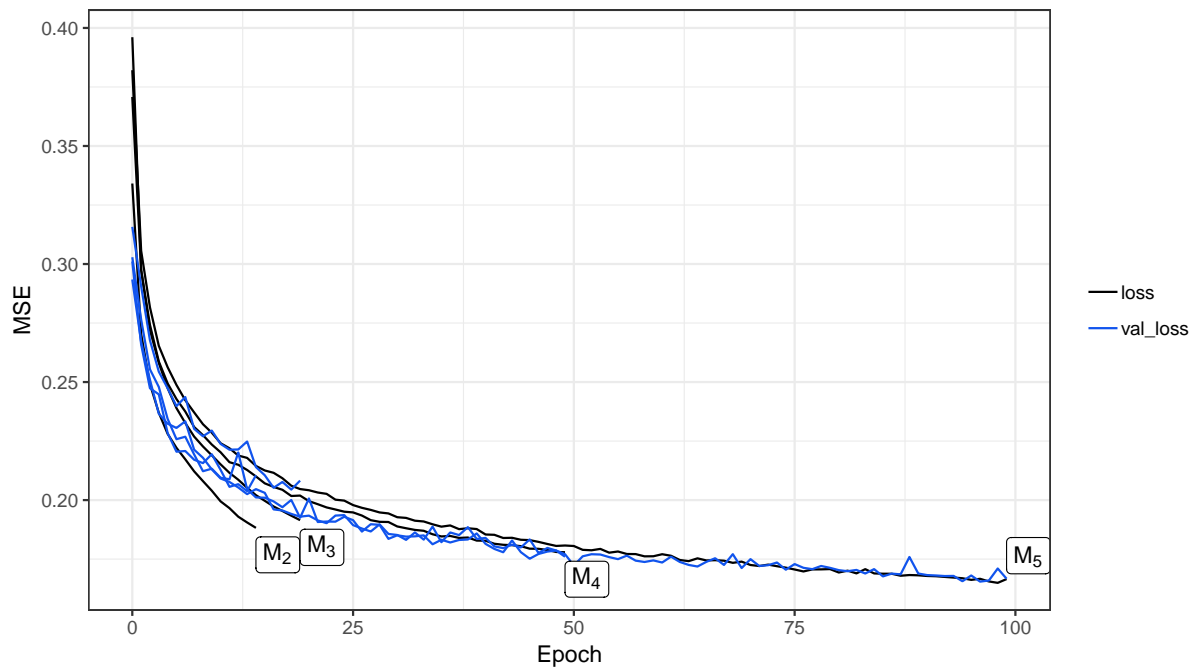


FIGURE 5.5: M_5 performance contrast when employing 10% and 5% dropout rates per layer.

the challenge to complete such a task is both methodological and technical in nature, it should be noted that the purpose of the linear model is to serve as a benchmark and to ensure that the results of more complex models are within some bounded tolerances.

It can be seen that the approach of providing aggregate features of a set of tree-based strategies, coupled with a deep neural network utilising dropouts, can produce robust estimates of the predicted objective function value for a CSP adopting the SAW scheme. An interesting extension to this approach would be to adopt a multiple input LSTM modelling approach to determine possible recurrences which would potentially identify useful subtrees in each of the strategy components.

As an aside with respect to the neural networks fitted in this section, it is quite remarkable how well the open source libraries function, both from an architectural consideration and from a computational efficacy perspective. The ubiquity of such top-tier tools to OR communities expands not only the scope for practitioners, but also enables practitioners to leverage such methods in different operational environments very easily.

5.4 Portfolio selection

In this section, the challenge of determining a suitable strategy for a problem instance is considered. The data used to fit the robust neural network model presented in the previous section is trained using data that are specific to the MiniZinc 2015 Challenge. The GP deployed to orchestrate the search for suitable strategies is restricted to employ the same strategy across all problem instances within a problem class. It is possible to relax this requirement and utilise the robust neural network model (M_5) to determine a suitable strategy for each problem instance.

The expectation would be, if the predictive model has not exhibited overfitting and if the task of determining a suitable strategy can be determined in some way based on the available problem

instance features, that the selection of search strategies should at least match the performance of the Hypothesis 3.1.2 SAW GP and potentially outperform the portfolio of algorithms derived by the GP. It will be interesting to observe the performance of the predictions of M_5 on problem instances which did not form part of the training instances.

All search strategies developed by the GP are assumed to form a portfolio of possible search strategies. The GP explored a total of 43 474 unique combinations of the variable (7 749), value (8 503), restart log size (18) and last-conflict (2) strategies. If the portfolio of algorithms is restricted to be explicitly within the subset of strategies already found, the total search space contains 43 474 discrete options. If, however, the specificity of the combinations is relaxed, allowing the model to predict the efficacy of options outside of the restricted search space, the resulting search space can be quantified as the product of the number of options per strategy, producing a search space of 2 372 030 892 discrete options.

The size of the relaxed search space, while still working within the strategies found, is an interesting number. A simple calculation reveals that the 2.3 billion search options would require approximately 2TB of memory in order to express all combinations simultaneously for a predictive algorithm given that the columns employ a minimal representation across the fields, *i.e.* 64 bits per double and 32 bits for integer fields. While it is possible to address this memory block on modern servers, it would be ideal to contain the analysis to operations which can be performed on commodity hardware. Another possibility would be to compute subsets of the option space, retaining the best prediction found, which would guarantee that the best prediction is attained. While this would be a reasonable approach, the computation time required to evaluate all discrete options remains a hindrance. By extrapolating based on the speed of predictions provided by M_5 , processing the complete option space would result in at least 22 hours of computation time. Given a CSP input, a set of predictions for the existing strategies may be concluded within a few seconds.

A possible approach to reduce this large space would be to consider neighbourhoods, or subsets of combinations which hold some seemingly good criterion fixed, thus iteratively converging on a best prediction, albeit an approximation, from this master set of options. This would make for an interesting extension to the approach employed herein, which focuses on the unique combinations of already seen strategies, referred to as \bar{P} , the portfolio of observed strategies.

For each problem instance, a prediction of which strategy in \bar{P} is expected to perform best on a particular problem instance is required. This requires forming a new input data set which consists of a repeated set of feature columns for the intended CSP, altering the operator frequency table for each candidate tree strategy. The normalisation procedures and quantities are held constant as M_5 has been fitted against the transformed data. If the model M_5 has not been fitted correctly to the data, the resulting predictions could result in strategies being preferred which have not been trialled on certain problems instances since the neural network is provided with the complete portfolio \bar{P} . Selecting a strategy which has never been trialled on a problem instance may reveal where strategies developed for one problem class can be successfully utilised in respect of another problem class. For certain instances where highly refined search strategies have already been established, it would seem unlikely that strategies from other problem classes may outperform such strategies.

Hypothesis 3.1.1		Hypothesis 3.1.2	
SAW	Multi-objective	SAW	Multi-objective
5	16	16	58

TABLE 5.5: Strategy origin summary for predicted best strategies by M_5 for \bar{P} .

The prediction of which strategies to use for the MiniZinc 2015 Challenge data set revealed that in many instances, M_5 selected a strategy which had been developed by the GP within a particular class of problems. There are multiple instances where strategies that performed exceedingly well in their own problem class, such as `gfd-schedule`, were predicted by the model to work in other problem classes. The relative frequency of an individual strategy showed that most strategies, for the 95 problem instances, were selected once or twice as the predicted best. The strategy with the highest frequency is a `gfd-schedule` strategy which was predicted as the best strategy for six of the problem instances. Each of the proposed strategies was run for a period of 20 minutes and compared to the base `or-tools` strategy. A summary of the origin of the selected strategies with respect to the GP run in which the strategy was found is provided in Table 5.5. It can be seen from the frequencies that there appears to be a bias to the Hypothesis 3.1.2 strategies, specifically from the multi-objective GP runs.

Table 5.7 provides a summary of the detailed search results in Table 5.6. It is interesting to note that since a single algorithm is no longer required for an entire problem class, the M_5 predictions frequently select an inferior search strategy, achieving slightly worse performance than the default ORT search strategy, and significantly worse than best strategies found in each category when simply using the best individual from the relevant GP search.

The results in Table 5.7 demonstrate that M_5 in its current form is a rather underwhelming predictor. This is, in part, due to the large number of possible search strategies and the probability of an overfitted parameter which would make the favourability of an arbitrary search strategy highly likely. There are, however, some promising decisions that were made by M_5 , specifically on the `tdtsp`, `costas-array` and `freepizza` problem classes, but which are overshadowed by the poor decisions made in almost all other classes.

There are several ways in which the poor prediction quality may be addressed. First, the candidate search trees could be restricted to trees which were directly sampled by the GP, introducing a bias in that the predicted tree should come from a set which has undergone some form of pre-convergence through the metaheuristic search. While this is a possible approach, it does not directly address the underlying poor fit of the model. A second approach would be to simplify the model by reducing the parameters in an attempt to introduce fewer higher-order parameters, and potentially more robust strategy prediction, favouring search strategies which have mostly been tried already on certain problem instances. A criticism of this approach is that the model has already been verified as having a small validation and test error which means that in a further training run, it will remain unclear whether the performance of the model is adequate until predictions are evaluated.

A simple test can be conducted to verify whether the second approach of reducing the parameter space has potential merit by reducing the complexity of the model. The linear regression model, which achieved a reasonable MSE and a high degree of statistical significance, may be employed to provide a prediction over \bar{P} . The results of the prediction provide a key insight, across all 43 474 strategies in \bar{P} : The same strategy is selected as the expected best strategy for all 95 problem instances. This ‘best’ strategy has, once again, its origin in the `gfd-schedule` instance and is producing significant bias in the fitted model. A potential remedy to correct for this leverage over the model would be to remove these training samples from the data. A drawback of this approach is that it requires manual intervention and a new data-adjustment procedure with an unknown termination criterion to determine when to stop removing training instances. This makes it an unattractive proposition. It remains that the issue is as a result of misleading training samples and not necessarily the high-dimensionality of the fitted model M_5 .

A third approach, which is explored in the remainder of this chapter, is to rather address the absence of information surrounding higher-order interaction terms where the model has

Problem Class	Instance	ORT			M_5					
		Code	Time	Best	Code	Δ	Time	Δ	Best	Δ
costas-array	16	S	19		S		20			
	17	S	240		S		1	+		
	18	S	17		S		14	+		
	19	UNK	1200		UNK		1200			
	20	UNK	1200		UNK		1200			
cvrp	A-n37-k5.vrp	S	1200	1642	S		1200		2207	-
	A-n64-k9.vrp	S	1200	3486	UNK	-	1200			
	B-n45-k5.vrp	S	1200	2728	S		1200		2775	-
	P-n16-k8.vrp	S	1200	502	S		1200		504	-
	simple2	SC	377	34	SC		19	+	34	
freepizza	pizza27	S	1200	882425	S		1200		762943	+
	pizza39	S	1200	939352	S		1200		839732	+
	pizza45	S	1200	656489	S		1200		573070	+
	pizza6	SC	610	210	SC		9	+	210	
	pizza78	S	1200	901717	S		1200		717244	+
gfd-schedule	n120f5d50m50k20	S	1200	19463	UNK	-	1200			
	n180f7d50m30k18	SC	1	1	UNK	-	1200			
	n30f3d30m7k4	UNK	1200		SC	+	1	+	1	
	n50f7d40m10k4	UNK	1200		SC	+	1	+	1	
	n75f5d30m20k20	UNK	1200		SC	+	1	+	1	
grid-colouring	10.5	SC	31	3	S	-	1200	-	4	-
	13.11	S	1200	7	S		1200		5	+
	19.17	S	1200	12	S		1200		9	+
	4.11	S	1200	4	SC	+	5	+	3	+
	4.8	SC	2	3	SC		1		3	
is	1YHXeG1xYs	S	1200	194048	S		1200		194432	-
	A3PZaPjnUz	S	1200	144896	SC	+	2	+	103936	+
	HgSWGJHxY5	S	1200	251200	S		1200		260000	-
	jZ9pQqRxJ2	SC	82	210944	SC		1109	-	210944	
	y21PnVA2Hj	S	1200	236544	S		1200		319312	-
mapping	full2x2	S	1200	1103	S		1200		794	+
	mesh2x2.mpeg	S	1200	726	UNK	-	1200			
	mesh3x3_2	UNK	1200		UNK		1200			
	mesh3x3.mpeg_2	S	1200	2197	S		1200		1318	+
	ring_2	S	1200	2090	UNK	-	1200			
multi-knapsack	mknap1-6	UNK	1200		SC	+	8	+	16537	
	mknap2-1	SC	158	7772	SC		3	+	7772	
	mknap2-2	S	1200	8722	SC	+	28	+	8722	
	mknap2-20	SC	3	6339	SC		1	+	6339	
	mknap2-32	UNK	1200		UNK		1200			
nmseq	176	S	6		UNK	-	1200	-		
	207	S	7		S		1	+		
	269	S	45		S		1	+		
	393	S	244		S		3	+		
	83	S	1		S		1			
opd	flener.et.al.10.350.100	S	1200	65	S		1200		124	-
	medium.10.100.30	S	1200	13	UNK	-	1200			
	small.bibd.10.30.09	SC	1107	2	S	-	1200	-	4	-
	small.bibd.11.22.10	S	1200	5	S		1200		8	-
	small.bibd.13.26.06	SC	1078	1	S	-	1200	-	2	-
open_stacks	problem_20.20.1	S	1200	11	S		1200		11	
	problem_30.15.1	SC	15	14	UNK	-	1200	-		
	wbo_10.20.1	SC	303	5	S	-	1200	-	6	-
	wbop_15.30.1	S	1200	7	S		1200		8	-
	wbp_20.20.1	S	1200	4	UNK	-	1200			
p1f	12	S	1200	602	UNK	-	1200			
	13	C	20		UNK	-	1200	-		
	14	S	1200	1008	UNK	-	1200			
	15	C	68		UNK	-	1200	-		
	17	UNK	1200		UNK		1200			
project-planning	ProjectPlannertest.12.7	S	1200	63	SC	+	6	+	17	+
	ProjectPlannertest.14.7	S	1200	78	UNK	-	1200			
	ProjectPlannertest.15.6	S	1200	66	UNK	-	1200			
	ProjectPlannertest.16.6	S	1200	39	UNK	-	1200			
	ProjectPlannertest.16.8	S	1200	39	S		1200		72	-
radiation	i14-9	SC	252	6513	S	-	1200	-	19139	-
	i6-11	SC	207	895	SC		2	+	895	
	i6-21	SC	143	1413	S	-	1200	-	1413	
	i7-9	SC	7	1007	SC		1	+	1007	
	i9-11	SC	427	2141	S	-	1200	-	2153	-
roster	chicroster_dataset_11	SC	1	17	SC		1		17	
	chicroster_dataset_17	SC	1	17	S	-	1200	-	31	-
	chicroster_dataset_2	SC	0	0	SC		1		0	
	chicroster_dataset_5	SC	1	6	SC		1		6	
	chicroster_dataset_7	SC	1	0	SC		1		0	
spot5	1401	S	1200	521097	S		1200		553110	-
	28	S	1200	284158	S		1200		313128	-
	414	S	1200	42564	S		1200		59573	-
	503	S	1200	15177	S		1200		15169	+
	54	S	1200	81	SC	+	806	+	37	+
tdtsp	inst.10.24.10	S	1200	13917	SC	+	585	+	9192	+
	inst.10.34.00	S	1200	8353	SC	+	3	+	6662	+
	inst.10.42.10	S	1200	15329	SC	+	8	+	8486	+
	inst.20.14.10	S	1200	17449	UNK	-	1200			
	inst.20.25.00	S	1200	19898	S		1200		17951	+
triangular	n10	SC	89	20	S	-	1200	-	20	
	n16	S	1200	35	S		1200		33	-
	n22	S	1200	48	S		1200		46	-
	n28	S	1200	61	S		1200		62	+
	n37	S	1200	80	S		1200		83	+
zephyrus	zephyrus_15.10	S	1200	36	S		1200		36	
	zephyrus_20.20	S	1200	66	S		1200		6370	-
	zephyrus_5.20	S	1200	66	SC	+	570	+	66	
	zephyrus_5.4	S	1200	18	SC	+	749	+	18	
	zephyrus-FH-2-15	SC	237	12	S	-	1200	-	12	

TABLE 5.6: M_5 — predicted strategy compared to the default ORT search strategy.

Problem Class	M_5		
	–	+	Δ
costas-array	0	2	2
cvrp	4	1	–3
freepizza	0	5	5
gfd-schedule	3	6	3
grid-colouring	3	5	2
is	4	3	–1
mapping	2	2	0
multi-knapsack	0	6	6
nmseq	2	3	1
opd	9	0	–9
open_stacks	7	0	–7
p1f	6	0	–6
project-planning	4	3	–1
radiation	8	2	–6
roster	3	0	–3
spot5	3	4	1
tdtsp	1	10	9
triangular	4	2	–2
zephyrus	3	4	1
Total	66	58	–8

TABLE 5.7: Summary of predicted best strategy (M_5) against the default ORT search by problem class.

estimated a favourable strategy for problem instances. For a given prediction, if a tree has not been sampled on a particular problem instance, it is evaluated within the sampling limit and the data from this evaluation added to corpus of training data which is then randomly segmented during further SGD iterations. This process is similar to that used by RL wherein predictions of preferred actions within the state space are updated over time through the experience of the last best perceived set of decisions.

This process of sampling higher dimensions attempts to directly address overfitting that may have occurred as a result of specific observations in the data. To illustrate this point, if the fitted linear model is considered as the predictive model of choice, given an ability to select any strategy from the corpus of \bar{P} , the linear model predicts that a single strategy will work best on all problem instances. This suggests that there are observations in the linear model with significant leverage over the fitted values, and resulting predictions as a result. The predicted best tree was evaluated given a complete search time of 20 minutes and the results were significantly worse than the default ORT search, suggesting that while this model has statistical significance, its practical use is limited.

The process of sampling higher dimensions requires three steps, namely, a prediction of the best strategy per problem instance, an evaluation of the predicted strategy within the sampling limit, provided that the strategy has not previously been evaluated against the problem instance, and finally, an update of the parameters of the neural network with the new data sampled. While testing this refinement process it was discovered that the majority of time was spent providing predictions as to the best tree for a given problem instance. As a result, the top ten best predictions per problem instance were selected to be sampled by the search strategy. This means that on the training data 950 strategies, at most, may be sampled during an iteration of the guided search procedure. Figure 5.6 provides a graphical intuition as to the convergence

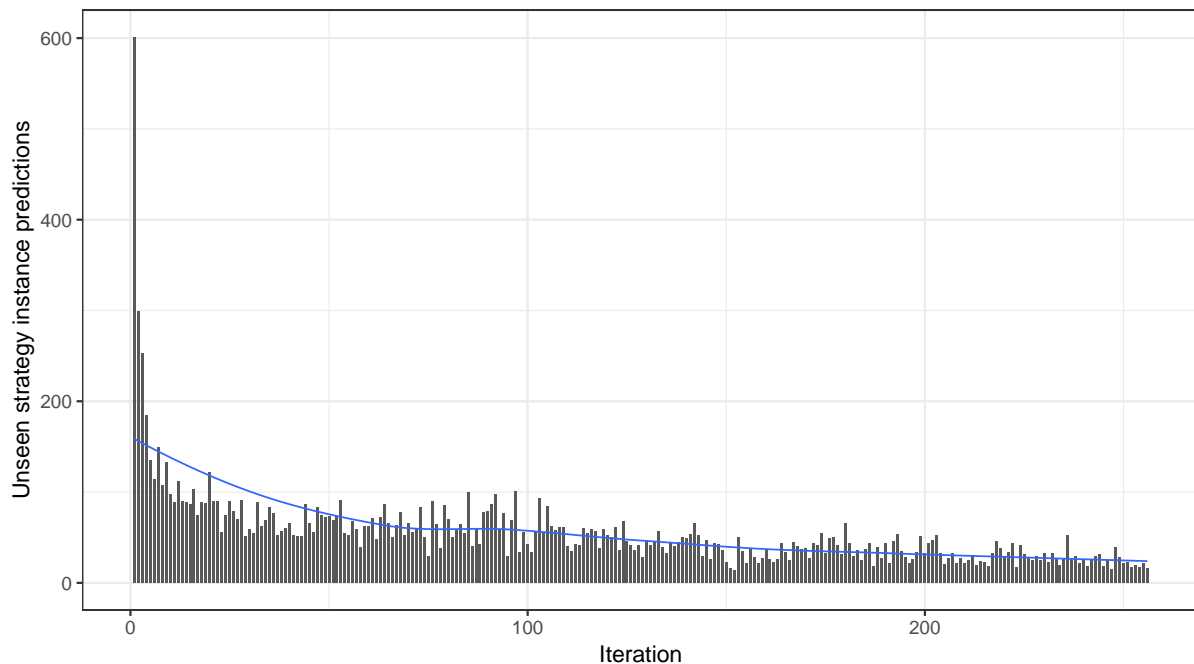


FIGURE 5.6: *Guided local search convergence for best predicted strategy per problem instance.*

of this process, demonstrating that in the first prediction the majority of the search strategies (almost two thirds) have never been trialled on the target problem instance.

The number of unseen strategy to problem instance pairings proposed across all problem instances reduces to 16 by the 256th iteration. This additional learning is computationally cost-effective and purposeful in the selection of which strategies to explore.

The process of guided local search in this context is similar to a RL process which is permitted a single action in the state space. This is not to compare the complexity of this process to that of an RL process, which is required to model the interdependency of repeated actions taken in the state space, it is merely noted that there is a point of common ground, given a simplification of the state space to a single temporal dimension. The algorithmic process of updating the predictive model for further predictions, given new information from the state space, is philosophically equivalent to RL in the process shown in Figure 5.6. The process could also be considered similar to semi-supervised learning, whereby a set of observations are marked as requiring labelling and are subsequently labelled by the search.

The iterative sampling of the best predicted strategies per problem instance assists to reduce the emphasis applied to search strategies which have not been attempted before on a particular problem class as a result of a particular search strategy working well within a particular problem class. The lion's share of the poor performance per problem class in Table 5.7 is attributed to strategies which have not been attempted before on such instances. A total of 14002 new observations are added to the data set which are again randomly partitioned into training, testing and validation data sets. The number of new observations produced is rather small (3%) when compared to the total number of samples gleaned from the GP runs, totalling approximately 430 000. This small degree of refinement in the neural network model is denoted by M_5^R .

This process of refinement of M_5 to produce M_5^R may be argued as an attempt to circumvent addressing a problem in the model, or modelling technique applied. Several ML techniques have

been applied in the literature to solve the ASP without requiring this additional refinement step. A key difference of the model presented herein lies in the size of the prediction space compared to other portfolio approaches. It would be typical to consider a handful of possible solvers, or permutations of parameters for a given set of problem instances, running the problem instances on such combinations to the full extent of the search. By contrast, a sampling approach with a large number of possible seen search strategies and an even larger number of unseen search strategies has been considered in this study. While the unseen search strategies are not considered part of the predictions for the best possible strategy for a problem instance, the prediction space remains far larger than other studies in this domain.

The approach employed by other researchers could have been emulated in this study by simply considering strategies which were in a top percentile, or an elite member of the population in the final generation of the GP, in order to bias the predictions to focus on search strategies which have already been determined to be of high quality. This is similar to the approach of adopting a solver in an ASP portfolio that is considered to perform well in order to reduce the number of solvers between which the ASP is required to select. This would introduce a very strong bias to selecting among a smaller set of strategies which have already been found to be good and it would be unsurprising to report a result which selects high-quality strategies in this context.

It is interesting to note the change in performance as a result of the refinement process, shown in Table 5.9, the precise details of which are provided in Table 5.8 for completeness. The results of M_5 are lacklustre at best, but the result of the small refinements performed to model M_5^R result in a rather dramatic change in the performance of the search strategies selected for full evaluation. Some of the more notable changes are those found to project-planning (+10), is (+8), gfd-schedule (+6) and grid-colouring (+4). Furthermore, the selection of strategies for gfd-schedule problem instances were able to outperform the class-level strategy selected by the GP. This is a trend that would be expected to occur more frequently as the model is refined further. The intuition is that the additional degrees of flexibility permitted by allowing an instance-specific strategy should be able to outperform a problem class level strategy as, in the extreme case where all strategies applied to problem instances are the same, this would mimic the behaviour of a class-level strategy. A counter argument to this intuition would be that it has been assumed that the data collected within the sampling limit are indicative of future performance. This has been shown to be false on larger problem instances. If the sampling limit were increased to match that of the final search runs, the training data would more closely resemble that of the final search, thereby removing this approximation of future search performance.

The performance of M_5^R is similar to that of Hypothesis 3.1.1 SAW with the exception that the variance of the losses and gains on problem instances are slightly larger for M_5^R as reflected by a comparison of the total losses (42 vs 37) and total gains (76 vs 71), with a resulting change of 34 in both M_5^R and Hypothesis 3.1.1 SAW. The losses in the context of M_5^R are expected as the objective has remained to minimise the remaining search depth, which, as previously discussed, results in a poor strategy for problem instances which require search near the root node.

5.4.1 Extendibility to unseen problem instances

The same test data set employed in Hypotheses 3.1.1 and 3.1.2 is used to measure performance of the predicted best strategies to employ, as determined by M_5^R . A summary of the performance on the unseen problem set is provided in Table 5.10.

The performance results of M_5^R on the test set are relatively neutral, demonstrating a similar number of losses and gains over the ORT default search strategy. That said, there are some interesting observations to be drawn from Table 5.10.

Problem Class	Instance	ORT			M_5^R					
		Code	Time	Best	Code	Δ	Time	Δ	Best	Δ
costas-array	16	S	19		S		1	+		
	17	S	240		S		1	+		
	18	S	17		S		4	+		
	19	UNK	1200		UNK		1200			
	20	UNK	1200		UNK		1200			
cvrp	A-n37-k5.vrp	S	1200	1642	S		1200		1914	-
	A-n64-k9.vrp	S	1200	3486	S		1200		3554	-
	B-n45-k5.vrp	S	1200	2728	S		1200		3007	-
	P-n16-k8.vrp	S	1200	502	S		1200		455	+
	simple2	SC	377	34	S	-	1200	-	34	
freepizza	pizza27	S	1200	882 425	S		1200		763 368	+
	pizza39	S	1200	939 352	S		1200		824 147	+
	pizza45	S	1200	656 489	S		1200		578 211	+
	pizza6	SC	610	210	SC		3	+	210	
	pizza78	S	1200	901 717	S		1200		678 458	+
gfd-schedule	n120f5d50m50k20	S	1200	19 463	SC	+	1	+	1	+
	n180f7d50m30k18	SC	1	1	SC		1		1	
	n30f3d30m7k4	UNK	1200		SC	+	1	+	1	
	n50f7d40m10k4	UNK	1200		SC	+	1	+	1	
	n75f5d30m20k20	UNK	1200		SC	+	1	+	1	
grid-colouring	10.5	SC	31	3	SC		1	+	3	
	13.11	S	1200	7	S		1200		5	+
	19.17	S	1200	12	S		1200		5	+
	4.11	S	1200	4	SC	+	1	+	3	+
	4.8	SC	2	3	SC		1		3	
is	1YHXeG1xYs	S	1200	194048	S		1200		71 296	+
	A3PZaPjnUz	S	1200	144 896	SC	+	1	+	103 936	+
	HgSWGJHxY5	S	1200	251 200	S		1200		124 960	+
	jZ9pQqRxJ2	SC	82	210 944	SC		1	+	210 944	
	y21PnVA2Hj	S	1200	236 544	S		1200		136 960	+
mapping	full2x2	S	1200	1103	S		1200		1948	-
	mesh2x2.mpeg	S	1200	726	S		1200		1 108	-
	mesh3x3_2	UNK	1200		S	+	1200		1 364	
	mesh3x3.mpeg_2	S	1200	2197	S		1200		1071	+
	ring_2	S	1200	2 090	S		1200		1 940	+
multi-knapsack	mknapsack1-6	UNK	1200		SC	+	6	+	16 537	
	mknapsack2-1	SC	158	7 772	SC		2	+	7 772	
	mknapsack2-2	S	1200	8 722	SC	+	11	+	8 722	
	mknapsack2-20	SC	3	6 339	SC		1	+	6 339	
	mknapsack2-32	UNK	1200		UNK		1200			
nmseq	176	S	6		S		1	+		
	207	S	7		S		1	+		
	269	S	45		S		2	+		
	393	S	244		S		2	+		
	83	S	1		S		1			
opd	flener.et.al.10.350.100	S	1200	65	S		1200		98	-
	medium.10.100.30	S	1200	13	S		1200		38	-
	small.bibd.10.30.09	SC	1 107	2	S	-	1200	-	4	-
	small.bibd.11.22.10	S	1200	5	S		1200		5	
	small.bibd.13.26.06	SC	1 078	1	S	-	1200	-	3	-
open_stacks	problem_20.20.1	S	1200	11	S		1200		11	
	problem_30.15.1	SC	15	14	S	-	1200	-	14	
	wbo_10.20.1	SC	303	5	S	-	1200	-	5	
	wbop_15.30.1	S	1200	7	S		1200		7	
	wbp_20.20.1	S	1200	4	S		1200		4	
p1f	12	S	1200	602	UNK	-	1200			
	13	C	20		UNK	-	1200	-		
	14	S	1200	1 008	UNK	-	1200			
	15	C	68		UNK	-	1200	-		
	17	UNK	1200		UNK		1200			
project-planning	ProjectPlannertest.12.7	S	1200	63	SC	+	2	+	17	+
	ProjectPlannertest.14.7	S	1200	78	SC	+	108	+	27	+
	ProjectPlannertest.15.6	S	1200	66	S		1200		40	+
	ProjectPlannertest.16.6	S	1200	39	S		1200		31	+
	ProjectPlannertest.16.8	S	1200	39	S		1200		31	+
radiation	i14-9	SC	252	6 513	S	-	1200	-	6 526	-
	i6-11	SC	207	895	SC		2	+	895	
	i6-21	SC	143	1 413	SC		567	-	1 413	
	i7-9	SC	7	1 007	SC		1	+	1 007	
	i9-11	SC	427	2 141	S	-	1200	-	2 153	-
roster	chicroster_dataset_11	SC	1	17	SC		1		17	
	chicroster_dataset_17	SC	1	17	SC		1		17	
	chicroster_dataset_2	SC	0	0	SC		1		0	
	chicroster_dataset_5	SC	1	6	SC		1		6	
	chicroster_dataset_7	SC	1	0	SC		1		0	
spot5	1401	S	1200	521 097	S		1200		522 117	-
	28	S	1200	284 158	S		1200		281 121	+
	414	S	1200	42 564	S		1200		46 514	-
	503	S	1200	15 177	UNK	-	1200			
	54	S	1200	81	SC	+	50	+	37	+
tdtsp	inst.10.24.10	S	1200	13 917	SC	+	3	+	9 192	+
	inst.10.34.00	S	1200	8 353	SC	+	3	+	6 662	+
	inst.10.42.10	S	1200	15 329	SC	+	3	+	8 486	+
	inst.20.14.10	S	1200	17 449	S		1200		16 034	+
	inst.20.25.00	S	1200	19 898	S		1200		17 704	+
triangular	n10	SC	89	20	S	-	1200	-	19	-
	n16	S	1200	35	S		1200		33	-
	n22	S	1200	48	S		1200		51	+
	n28	S	1200	61	S		1200		40	-
	n37	S	1200	80	S		1200		83	+
zephyrus	zephyrus_15.10	S	1200	36	S		1200		36	
	zephyrus_20.20	S	1200	66	S		1200		66	
	zephyrus_5.20	S	1200	66	SC	+	554	+	66	
	zephyrus_5.4	S	1200	18	SC	+	781	+	18	
	zephyrus-FH-2-15	SC	237	12	S	-	1200	-	12	

TABLE 5.8: M_5^R — predicted strategy compared to the default ORT search strategy.

Problem Class	M_5			M_5^R		
	-	+	Δ	-	+	Δ
costas-array	0	2	2	0	3	3
cvrp	4	1	-3	5	1	-4
freepizza	0	5	5	0	5	5
gfd-schedule	3	6	3	0	9	9
grid-colouring	3	5	2	0	6	6
is	4	3	-1	0	7	7
mapping	2	2	0	2	3	1
multi-knapsack	0	6	6	0	6	6
nmseq	2	3	1	0	4	4
opd	9	0	-9	8	0	-8
open_stacks	7	0	-7	4	0	-4
p1f	6	0	-6	6	0	-6
project-planning	4	3	-1	0	9	9
roster	3	0	-3	0	0	0
radiation	8	2	-6	7	2	-5
spot5	3	4	1	3	4	1
tdtsp	1	10	9	0	11	11
triangular	4	2	-2	5	2	-3
zephyrus	3	4	1	2	4	2
Total	66	58	-8	42	76	34

TABLE 5.9: Performance of M_5 and M_5^R on training instances compared to the base *ORT* search.

ProblemClass	M_5^R		
	-	+	Δ
costas-array	0	1	1
cvrp	9	0	-9
gfd-schedule	2	2	0
grid-colouring	0	6	6
mapping	2	3	1
multi-knapsack	0	2	2
nmseq	1	3	2
opd	8	0	-8
open_stacks	7	0	-7
p1f	9	0	-9
project-planning	0	9	9
radiation	0	6	6
spot5	3	2	-1
tdtsp	0	6	6
triangular	1	3	2
zephyrus	8	3	-5
Total	50	46	-4

TABLE 5.10: Performance of M_5^R on unseen problem instances compared to the default *ORT* search.

The performance on the radiation problem outperformed that of both Hypothesis 3.1.1 SAW and Hypothesis 3.1.1 multi-objective where a strategy was derived which did not generalise as well as anticipated.

The zephyrus problem, which was found to have a different underlying structure in the test data from than in the training data, while still not outperforming the default ORT search strategy, significantly mitigated the losses incurred on this problem class, by approximately halving the loss observed in Hypotheses 3.1.1 and 3.1.2.

A particularly poor result was obtained for the cvrp class of problems. One possibility for this poor result is that many strategies were found for the cvrp which admit a feasible solution (an f_1 score of zero), not completing within the time limit (an f_3 score of 1) and predominantly being scored on the f_2 objective. The domain of the objective f_2 , while normalised, produces relatively small errors as the objective margins are very small in this context with errors being pronounced in the third decimal. It is also possible that search strategies which perform well in terms of providing an initial incumbent do not retain the performance profile throughout a longer search *i.e.* during local search in progressively more constrained environments. This may be a result of the sampling approach not being completely representative of the actual search performed. This is perhaps one of the reasons why predictors for the ASP are traditionally fitted using complete search data. It can be noted that an increase in the sampling limit, tending towards the complete runtime, is not necessary in all problem classes, as there are several instances where improvements found in the training data are successfully carried forward to the test data.

5.5 Hypothesis 3.1.3 conclusions

The question was posed as to whether a deep learning model would be able to predict an appropriate strategy to employ when solving a CSP, based on the features of the CSP as well samples of previous searches conducted. The portfolio of search strategies was unconstrained and included all search strategies developed by the GP across all searches conducted (\bar{P}). A linear model was fitted to the GP search data and it was found that the resulting model was a poor predictor of a candidate search strategy in \bar{P} .

A series of neural networks were fitted to the data in an attempt to model higher-level interactions in the data. A network architecture was settled upon, M_5 , but it was demonstrated that this model also exhibited volatility when matching a high-quality search strategy from \bar{P} to a problem instance, albeit to a far lesser degree than the linear model. A refinement process akin to that of a guided local search was applied to the model in order to resolve poor matchings generated by the model. The resulting model M_5^R was then used to more accurately predict a suitable search strategy for a problem instance.

The performance of M_5^R , with respect to predicting a search strategy from \bar{P} , is able to approximate the performance of the Hypothesis 3.1.1 SAW GP algorithm. This performance was achieved even though there was a lack of exploitation bias for a problem class, which is an inherent feature in the design of the GP, and a high degree of parametrisation. The high degree of parametrisation allows a model to easily over-fit a particular subset of data, which will then lack generality. It was shown in results on unseen problem sets that although the performance is neutral with respect to the ORT default search, there are several instances in which a predicted strategy can successfully exploit a problem instance to outperform the ORT free search.

The neutral performance of M_5^R on the test data set may incline one to think that the experiment demonstrates that a deep learning model is either a poor predictor of performance, or that there is simply no free lunch in this context. This is not the case. M_5^R has reliably predicted both poor and good performance of the portfolio of search strategies. The problem is not in the predictions, but rather in the portfolio of strategies. The portfolio of search strategies is restricted to those in \bar{P} , which do not include the strategies present in the default ORT search. It is a natural

extension to add the sophisticated ORT default search as a candidate strategy in \bar{P} with the associated search metrics.

M_5^R demonstrates that within a problem class there are exploitations which can reliably be taken advantage of, if a suitable strategy exists. The model has also demonstrated that it can reliably match a strategy to CSP features, meaning that if the ORT default strategies were permitted in \bar{P} , equivalent solutions to the default search may be found for problem classes where a better search strategy was not found by the GP search. This suggests that the instances for which the search was beaten by the ORT strategy are less interesting than those instances in which it was predicted to outperform the ORT default search.

The size of \bar{P} and the limited search information as a result of a training data set consisting of searches conducted within the sampling limit are notable differences between the approach employed in this chapter and that typically utilised in the literature. It was demonstrated that the sampling limit was sufficient for many search attempts, although this may be increased for larger problem instances to adequately evaluate the quality of a candidate search strategy as part of future work. Furthermore, it is notable that the flaws already highlighted in the objective components measuring search quality are carried through to the predictive model. The objective to minimise the number of infeasible variable assignments (f_1) results in poor overall search for problems which do not admit a feasible incumbent or which require search near the root node. The predictive model is able to reproduce the poor results obtained by the GP. In a sense, this is a positive result as it suggests that should an informative objective term, such as the coverage achieved for a problem instance, be introduced as a search quality measurement, it is likely that a deep learning model will be able to predict an appropriate strategy for such problem — an interesting line of future work.

Lastly, this chapter contained a description of a foundational model for deep learning in portfolio selection using a simplistic representation scheme which obfuscates the inter-dependencies of a particular tree structure. For example, the trees $\frac{1}{x}$ and $\frac{x}{1}$ produce an identical feature description of the tree, although these two trees produce opposite rankings in respect of variable or value rankings. The representation used was demonstrated to be an adequate first approximation, but an interesting line of potential research would be to consider an LSTM model which may directly model the relationship of tree nodes based on their grammatical representation. This may yield two benefits: first, an improved model which may no longer require a refinement step to manage predictions over a large portfolio set, and secondly, the ability to query the model in the opposite direction. This second benefit alludes to the possibility of fixing an objective value, or multiple objective output values, as well as conditioning on the CSP features, in order to induce the tree of highest probability. The LSTM model is not constrained to select a tree structure already seen and may induce a tree structure which has not previously been evaluated. Such a process may then be coupled with a refinement step, as described in this chapter, to explore a search space of new tree structures iteratively, thus automating the process of both exploring the input and output search spaces.

CHAPTER 6

Conclusion

Contents

6.1	Dissertation summary	133
6.2	Appraisal of dissertation contributions	135
6.3	Suggestions for future work	136
6.3.1	<i>Problem-specific GP grammars</i>	136
6.3.2	<i>Alternate levels of branching strategy</i>	136
6.3.3	<i>Constraint-specific GP grammars</i>	136
6.3.4	<i>Faceting applications</i>	137
6.3.5	<i>Evolutionary seeding</i>	137
6.3.6	<i>Enhancements to the deep learning model</i>	138
6.3.7	<i>The addition of CP solvers to the portfolio</i>	139
6.4	A philosophical note	139

The salient points highlighted in the previous chapters are discussed in this chapter. Particular attention is afforded to insights that were gleaned from the relevant chapters which were non-intuitive at first, but yielded a interpretation through closer inspection. The novel contributions of the work in this dissertation are detailed and a summary of future potential research is elucidated.

6.1 Dissertation summary

The literature review presented in Chapter 2 covers, in a compact manner, the principal subject matter pertaining to the fields related to the topic of this dissertation. The principal fields are those of constraint programming, evolutionary algorithms, the algorithm selection problem and machine learning. Particular attention was afforded in the literature review to backtracking search, complete algorithms, genetic programming, machine learning and literary works which have provided a treatment of cooperative applications employing combinations thereof.

An outline of three research hypotheses, the research methodology and algorithmic implementation considerations was provided in Chapter 3.

Chapter 4 was devoted to the first two hypotheses, Hypothesis 3.1.1 and Hypothesis 3.1.2, aimed at assessing the ability of a GP to derive branching strategies for solving a class of CSP problems, employing two different objective schemes. A core assumption of the approach employed was

that a short search, consisting of a few seconds, provides sufficient information to determine the quality of a candidate branching scheme. A second assumption embedded in the sampling of candidate search strategies was that the metrics used to measure the quality of a candidate search are indicative of a high-quality search.

It was found that for smaller optimisation CSPs, the metrics defined worked reasonably well, and the GP was able to determine suitable branching strategies which outperformed the default ORT search. Not only did such strategies outperform the default search in respect of the training data, but similar trends were observed in the quality of the branching strategies on unseen problem instances of the same class. In instances where the quality of the strategy deteriorated, deeper analysis of the underlying CSP graphs revealed that the underlying structure of the problem no longer closely resembled those instances seen in the training data.

The GP was found to struggle with problem instances which required an infeasibility proof or concentrated search near the root node. This is not a deficiency in the GP itself, but rather a problem with employing a metric which attempts to maximise the search depth. An orthogonal approach is required when providing an infeasibility proof to that which was incentivised in the objective terms of the GP. A proposed rectification to the objective would be to include *coverage* as a search metric in addition to, or as opposed to, the search depth measurement for SAW and multi-objective schemes respectively.

An interesting outcome regarding the performance of the GP search itself, is that the multi-objective GP search typically found higher quality solutions than the SAW scheme. One may expect that a SAW scheme would embody a higher incentive to exploit the search as the one-dimensional objective provides a clear criterion for quality, resulting in higher-quality branching strategies. The result, however, was due to the fact that the search did not consider alternate strategies which may be good in another objective dimension, which when combined with a different candidate strategy, forms a better strategy. The myopic nature of the SAW scheme excludes many potentially beneficial strategies which the multi-objective scheme preserves when candidate solutions lie on the undominated frontier.

The number of generations between GP runs was held constant and it was expected that the additional exploration pressure applied through the multi-objective scheme would have required significantly more generations in order to match the performance of the SAW scheme. The multi-objective GP outperformed the SAW scheme on Hypothesis 3.1.1, but was unable, for either for the SAW or for multi-objective scheme, to match the performance obtained in Hypothesis 3.1.1 by the same scheme in the case of Hypothesis 3.1.2, even though the same strategies could, theoretically, be derived. This illustrates the effect of the increased search space in Hypothesis 3.1.2 where a significantly larger grammar was employed. The number of generations, and perhaps the population size, should be increased in order for the GP to operate optimally under such conditions. While the GP may not have been configured optimally for Hypothesis 3.1.2, it was still found that strategies which included graph features derived by Hypothesis 3.1.2 were utilised which outperformed the default ORT search strategy in both the training and test data sets.

The ORT free search strategy is a highly-tuned strategy which has been given much thought by its designers. The strategies developed by the GP range from the highly simplistic to the uninterpretable. In both instances, however, several strategies were found to be more effective than those employed by ORT, and provided consistent search quality within a problem class. This supports the intuition that a search strategy may be tailored to a particular problem structure.

Chapter 5 was devoted to the final hypothesis considered in this dissertation, Hypothesis 3.1.3 sought to verify whether a deep learning model may be considered a valuable asset for predicting the performance of a candidate search strategy for a particular CSP. It was found that a simple frequency-based representation for the tree structures, coupled with a deep neural network, was able to perform well and in a consistent manner. The purpose of Hypothesis 3.1.3 was to verify whether a deep learning model is able to produce an adequate prediction of performance, given search data gathered within the sampling limit. The model demonstrated that it is indeed capable of achieving a small predictive error, subject to a brief guided local search procedure to refine higher-order terms in the model. The implications of the success of this simple representation scheme are discussed later in this chapter.

6.2 Appraisal of dissertation contributions

A methodology has been presented which employs metaheuristic optimisation to create and measure the quality of candidate branching strategies to be applied to a CP search in order to resolve a CSP. The methodology applied is, to the best of the author's knowledge, the first instance of such a metaheuristic search which is consistent with the classic 'Koza' GP formulation. This is also the first instance where a multi-objective GP search approach has been employed to construct candidate search strategies which was demonstrated to outperform a SAW scheme, commonly employed in the literature.

An arithmetic ranking function was proposed in this dissertation which is consistent with similar approaches in the literature. A comparison of the CSP features available to the GP when building ranking functions was also investigated. It was found that there is merit in considering additional graph features, but that the GP configuration parameters should be updated in line with the larger search space.

A central question in this dissertation, although not an explicit hypothesis, was whether a strategy developed for a particular problem class can be demonstrated to be effective on unseen problem instances of the same class. It was demonstrated that the efficacy of branching strategies can indeed be carried forward to problems of the same class, when the constraint structure of the class is consistent. This is an important contribution as it demonstrates not only that the process of searching for a candidate branching strategy can be automated, but also that once a strategy has been shown to work within a class, a reliable search strategy can be reused for unseen problem instances within the same problem class. Verifying this extendibility of a search strategy within a particular problem class was a key motivation in deriving the topic at hand.

The CSP features used by top-tier portfolio solvers were replicated in this dissertation and applied to the training problem instances, thereby enabling the use of a deep learning model to predict a suitable branching strategy for a CSP instance. The deep learning model was demonstrated to have the predictive qualities which make the model a prime candidate for the task at hand. While the model employed did not exhibit a state-of-the-art architecture it demonstrated sufficient predictive qualities based on operator frequencies and CSP features, in spite of the number of parameters of the model. This suggests that a parameter-rich model which more closely models the tree structures will certainly outperform the approach demonstrated in this dissertation. A key contribution of the modelling approach employed is that a prediction can be performed based on search metrics within a sampling limit. This is an atypical approach in the literature which has demonstrated the ability to be an adequate predictor of future performance.

6.3 Suggestions for future work

The scope of this dissertation was intentionally restricted in order to thoroughly explore a series of well-defined hypotheses. Throughout the development and exploration process of this dissertation, several new research avenues have been identified. A description follows of potential future work in each of the relevant categories identified.

6.3.1 Problem-specific GP grammars

An arithmetic grammar was studied in this dissertation primarily as a by-product of the abstract level at which branching decisions are performed when solving a pure CSP. This does not prevent one from adopting an approach which employs information rooted in the context of the problem being solved when treating a problem which is not cast as a CSP. An example, one may treat a CVRP directly and provide a GP grammar which permits adorning graph nodes with attributes such as vehicles or stops. Furthermore, properties of the sub-graph (such as the unconstrained graph of stops, excluding capacity constraints) may also be attributed to nodes, such as the near-optimal stop sequence. Information such as this may permit a search strategy which employs the properties of the high-level graph in a meaningful way in order to trial different search tactics against a problem instance.

6.3.2 Alternate levels of branching strategy

A single strategy approach was adopted to resolve a CSP in this dissertation. As demonstrated by the ORT default search, multiple strategies may, at times, be beneficial to the overall search performance. There are multiple possibilities that may be explored so as to automate a multi-tiered strategy approach. There is existing precedence in the GP literature surrounding ADFs, which provides a mechanism for reusing existing GP structures that have been found to be effective. An approach may be to employ ADFs in combination with switching mechanisms which select between strategies based on rules, which may also be derived as a function of static or dynamic features of the CSP being solved. This leads to the exploration of nested strategies, or micro-portfolios of strategies, being composed to a master strategy. It would be interesting to explore what benefit this may yield in the context of resolving CSPs.

6.3.3 Constraint-specific GP grammars

The approach to resolving CSPs in this dissertation has been direct, attempting to find a solution technique for a problem instance. An alternate method would be to rather consider the constraints present in a CSP and evolve strategies which are applicable to such constraints. A way in which this may be accomplished is by considering global constraint types that span collections of nodes. Strategies for global constraints can then be developed independently of one another and combined in problems where multiple constraint types are present. This approach inverts the hierarchy of this dissertation where a single search strategy is applied to the problem, from the top down, whereas tailoring strategies to specific constraint types suggests that one may design a CP solver from the bottom up.

6.3.4 Faceting applications

The cutting plane technique is a remarkable approach to optimisation. The work of Applegate *et al.*[10], showcased in Concorde, demonstrates that if sufficient high-quality methodologies are developed to identify potential cutting planes for a specific constraint structure (in the case of the TSP, the circuit constraint), that although heuristics may be employed to identify facets on the polytope, a final integer solution, if found, is optimal. Some cuts are trivial to identify in polynomial time, such as subtour cuts. It is not known for many more complex classes of cuts, such as comb inequalities with five or more teeth, whether non-pseudo polynomial time algorithms exist which may separate such inequalities.

The suggestion for a line of research pertaining to cutting planes is not to resolve theoretical questions related to the computational complexity, but rather to focus on *searching* for the functional form of potential inequalities which may admit a simple heuristic solution. This would require a two-phase approach: a GP to propose a functional form for a particular cut, and a second phase to verify whether a separation for the form can be found. Verifying whether the functional form is *valid* in terms of the underlying polytope can be achieved in a number of ways. A probabilistic approach employing a partial enumeration of the feasible solution space, which, if any solution in this set is restricted by the functional form and a heuristic separation is found, is an invalid functional form. Another method would be to exploit the LP duality theory to verify whether a proposed cut is valid, although this approach would not guarantee that all invalid cuts are identified initially, and may require completing an optimality proof, in a MIP context, to verify the validity.

The state-of-the-art in modern MIP engines is advanced integer lifting, or lift-and-project procedures. Such procedures have a tendency to approximate the functional form of the exponential formulations when presented with compact formulations. The compact MTZ CVRP and exponential DFJ CVRP formulations provide a good example in which this can be verified algebraically. As such, alternative heuristic lifting procedures may be investigated by employing the approach of this dissertation to explore the benefit of exploiting structural properties of a formulation to apply cuts (MIP), *nogoods* (CP) or clauses (SAT) to a formulation in order to reduce the size of a search tree at the root node, thereby potentially improving the overall search performance.

6.3.5 Evolutionary seeding

It is typical in evolutionary algorithms that once a problem domain is better understood, be-spoke seeding heuristics may be employed to create diverse, high-quality initial solutions for a problem. The approach employed in this dissertation limited the evolutionary algorithm to random starting solutions, which are typically of poor quality. This is not a failure of the approach as it is often the case when little is known about the underlying problem structure *a priori*. It was, however, shown in this dissertation that a deep learning model is able to act as a reasonable predictor of the performance of candidate search strategies, which suggests that perhaps a deep learning model may be able to create a starting population for a GP search. There are potential shortcomings induced by creating a seeded population, such as introducing biases that the GP may be unable to resolve in the resulting search. It may also, however, lead to more refined or complex strategies being introduced. If convergence is attained more rapidly in the GP as a result of seeds, multiple shorter GP runs may be a sensible approach to address the local optimum bias introduced by the deep learning model. An empirical study on this topic would be interesting to explore.

6.3.6 Enhancements to the deep learning model

The focus in this dissertation was on modelling the results of samples of searches conducted with varying branching strategies. For each of these observations, the maximum amount of search time was restricted to 10 seconds. The motivation for this standardisation of sampling time was based on the simplification of control parameters for the experiments conducted. It would however be reasonable to investigate the effect of varying the sampling limit proportionately to certain properties of the CSP.

Furthermore, the deep learning model is capable of including the sampling time employed for an observation in order to calibrate the model. This may yield two benefits. Firstly, being able to more accurately predict future performance of searches given a time budget. Such a time budget was held constant in training data provided to the learning model in this dissertation. Given that a search strategy is run on a problem instance employing varying time budgets, the time budget and objectives obtained would thus be the only values differentiating two such rows of data from one another. This may assist a deep learning model to make longer term predictions more accurately. The second benefit would be to model in the opposite direction, querying the neural network in reverse to reveal the amount of search time estimated to achieve a particular result. This information may be useful in the context of portfolio optimisation, whereby multiple strategies, each being allocated a limited time budget, may be attempted in order to resolve a CSP. An overarching theme of this line of research would be that a single model is trained which may be queried in multiple ways to assist in portfolio selection and optimisation.

A single output neural network was considered in this dissertation, motivated by the resemblance to a classic linear regression model, a well-understood statistical modelling technique. It would be possible in the context of neural networks to utilise the same hidden layer structure, but produce multiple outputs, which are similar to a multivariate linear regression. This would allow the model to learn the relationship between input features and different objective components. Once again, by querying the network in reverse, it may be possible to determine which features, or interactions thereof, influence the components of the objective function. This is an insight which is difficult to glean through direct inspection.

A decomposition of the objective terms may also permit the selection of strategies in a portfolio optimisation context which are able to hedge against weaknesses of different strategies. An example of this would be a strategy which attempts to maximise the feasibility of the graph, and another which attempts to identify infeasible cliques. It would be difficult to discern two such strategies from one another unless the output metrics are modelled in a direct fashion, rather than through an aggregation scheme. This could potentially assist in the construction of more sophisticated portfolio optimisers.

A simpler extension to the deep learning model would be to add sampled searches by the ORT default search strategy. This could be achieved by adding an additional dummy variable to the input data, indicating whether the strategy is a GP derived strategy or the ORT default strategy. This would permit the predictive algorithm to select between the default search and an alternative search which has demonstrated its efficacy on a particular CSP. The expectation would be that a set of predictions may be made by the deep learning model dominating the results of all best GP results, as tested by Hypothesis 3.1.1, Hypothesis 3.1.2, SAW and multi-objective, and that of the default ORT search strategy.

6.3.7 The addition of CP solvers to the portfolio

Initial prototyping work in this dissertation was conducted in CPO, with the final set of results being concluded against the ORT CP solver. It was observed during testing that certain constraint types had more effective implementations in CPO than in ORT, and that the opposite was also true. It is not possible to directly view the constraint propagation techniques employed by CPO, but a possible reason for the difference in performance would be that more efficient or advanced propagation techniques are implemented in the solver which are specifically related to the circuit constraint. The ability to leverage a solver which has implemented a more efficient constraint propagation technique for a given constraint type is often the motivation behind utilising multiple differing solvers in a portfolio-solver approach to the ASP, the intuition being that portfolio predictions attempt to match problem instances to solvers which may be preferable for CSP instances. An interesting extension to the study presented in this dissertation would be to consider the addition of other solvers to the GP search, which is able to tailor strategies which work well in one solver, but not another. Understanding whether the performance of such decision making would remain effective would be an interesting study to consider.

6.4 A philosophical note

In the last 60 years, the state-of-the-art in adversarial game playing has shifted from hand-crafted strategies to strategies which are learnt by advanced machine learning implementations. The efficacy of the state-of-the-art has evolved from computers being able to match human intelligence in *Drafts* during the 1960s, *Backgammon* during the 1970s and *Chess* in a game dubbed *man vs. machine* that captivated the world in 1997 between *Deep Blue* and Garry Kasparov, the world Chess champion at the time. The strategies employed by the programs to defeat human intelligence in all these instances hinges on backtracking search, in a *minimax* context. In the case of Chess, known strategies used by experts were provided to the search program in order to produce high-quality solutions in order to emulate an effective opponent. It was estimated at the time, that in order for a computer to beat a human at *Go*, a board game with a considerably higher branching factor of 250 compared to the 35 of Chess, that such a feat may possibly only be achieved after the year 2030. At the time, children could easily beat the best Go computer programs available.

In 2015, the computer program *AlphaGo*, developed by *DeepMind Technologies*, beat the 9-dan professional Go player Lee Sedol. Since then, AlphaGo has been extended, reformulated and termed *AlphaZero*, which is capable of superhuman performance, beating the best human and computer algorithms in both Chess and Go by leveraging an algorithmic approach [164], utilising only self-play and no hand-crafted strategies or endgame tables. At the core of the approach employed by the DeepMind researchers is a neural network which learns the quality of decisions, through experience, in a reinforcement learning manner. The philosophy of this approach is one whereby the researcher acknowledges that using words, lines of code, or equations to describe a suitable strategy will only result in a distillation of a near-optimal strategy to a point where it is no longer representative or effective. A near-optimal strategy may itself not be understood in human terms, but the methodologies for deriving such a complex algorithm are steadily being improved upon.

In the same way that a list of ingredients are chanted around the cauldron in Shakespeare's *Macbeth* to conjure up what can only be described as magic — the trend in machine learning certainly appears to be rapidly moving away from that of understanding the mechanism, to that of refining the recipe. This is not a criticism, merely an observation. Effective search strategies

were found in this dissertation which could not be rationalised into words or fully comprehended. As researchers, we have a bias to strive toward providing a minimal explanation to what is observed. Arthur C Clark's third "law" aptly states that "*any sufficiently advanced technology is indistinguishable from magic.*" A great privilege and responsibility has been bestowed upon us as operations researchers to work with magic and discover new incantations each day.

References

- [1] AAMODT A & PLAZA E, 1994, *Case-based reasoning: Foundational issues, methodological variations, and system approaches*, AI Communications, **7(1)**, pp. 39–59.
- [2] AARTS E, AARTS EH & LENSTRA JK, 2003, *Local search in combinatorial optimization*, Princeton University Press, Princeton (NJ).
- [3] ABADI M, AGARWAL A, BARHAM P, BREVDO E, CHEN Z, CITRO C, CORRADO GS, DAVIS A, DEAN J, DEVIN M, GHEMAWAT S, GOODFELLOW I, HARP A, IRVING G, ISARD M, JIA Y, JOZEFOWICZ R, KAISER L, KUDLUR M, LEVENBERG J, MANÉ D, MONGA R, MOORE S, MURRAY D, OLAH C, SCHUSTER M, SHLENS J, STEINER B, SUTSKEVER I, TALWAR K, TUCKER P, VANHOUCHE V, VASUDEVAN V, VIÉGAS F, VINYALS O, WARDEN P, WATTENBERG M, WICKE M, YU Y & ZHENG X, 2015, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, [Online], [Cited September 2019], Available from <https://www.tensorflow.org/>.
- [4] AHUJA RK, LIU J, ORLIN JB, SHARMA D & SHUGHART LA, 2005, *Solving real-life locomotive-scheduling problems*, Transportation Science, **39(4)**, pp. 503–517.
- [5] AMADINI R, GABBRIELLI M & MAURO J, 2013, *Features for building CSP portfolio solvers*, arXiv preprint arXiv:1308.0227.
- [6] AMADINI R, GABBRIELLI M & MAURO J, 2018, *SUNNY-CP and the MiniZinc challenge*, Theory and Practice of Logic Programming, **18(1)**, pp. 81–96.
- [7] AMADINI R, GABBRIELLI M & MAURO J, 2015, *SUNNY-CP: A sequential CP portfolio solver*, Proceedings of the 30th Annual ACM Symposium on Applied Computing, New York (NY), pp. 1861–1867.
- [8] AMADINI R, GABBRIELLI M & MAURO J, 2014, *SUNNY: A lazy portfolio approach for constraint solving*, Theory and Practice of Logic Programming, **14(4-5)**, pp. 509–524.
- [9] ANGELINE PJ, 1996, *An investigation into the sensitivity of genetic programming to the frequency of leaf selection during subtree crossover*, Proceedings of the 1st annual conference on genetic programming, MIT Press, Cambridge (MA), pp. 21–29.
- [10] APPLGATE DL, BIXBY RE, CHVATAL V & COOK WJ, 2011, *The traveling salesman problem: A computational study*, Princeton University Press, Princeton (NJ).
- [11] BACCHUS F & VAN RUN P, 1995, *Dynamic variable ordering in CSPs*, Proceedings of the International Conference on Principles and Practice of Constraint Programming, Cassis, pp. 258–275.
- [12] BAIN S, THORNTON J & SATTAR A, 2004, *Methods of automatic algorithm generation*, Proceedings of the PRICAI 2004: Trends in Artificial Intelligence, Auckland, New Zealand, pp. 144–153.

- [13] BAIN S, THORNTON J & SATTAR A, 2005, *Evolving variable-ordering heuristics for constrained optimisation*, Proceedings of the International Conference on Principles and Practice of Constraint Programming, Sitges, pp. 732–736.
- [14] BARNHART C, JOHNSON EL, NEMHAUSER GL, SAVELSBERGH MW & VANCE PH, 1998, *Branch-and-price: Column generation for solving huge integer programs*, Operations Research, **46(3)**, pp. 316–329.
- [15] BARRAT A, BARTHELEMY M, PASTOR-SATORRAS R & VESPIGNANI A, 2004, *The architecture of complex weighted networks*, Proceedings of the National Academy of Sciences, **101(11)**, pp. 3747–3752.
- [16] BAYARDO RJ & MIRANKER DP, 1996, *A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem*, Proceedings of the National Conference on Artificial Intelligence, Portland (OR), pp. 298–304.
- [17] BAZ M, HUNSAKER B, BROOKS P & GOSAVI A, 2007, *Automated tuning of optimization software parameters*, (Unpublished) Technical Report, TR2007-7, University of Pittsburgh, Pittsburgh (PA).
- [18] BEAZLEY DM, 1996, *SWIG: An easy to use tool for integrating scripting languages with C and C++*, Proceedings of the 4th Conference on USENIX Tcl/Tk Workshop, Monterey (CA).
- [19] BELLMAN R, 1966, *Dynamic programming*, Science, **153(3731)**, pp. 34–37.
- [20] BENNETTO RA, 2013, *Dynamic train scheduling in an uncongested rail network*, MSc thesis, University of the Witwatersrand, Johannesburg.
- [21] BERGSTRA JS, BARDENET R, BENGIO Y & KÉGL B, 2011, *Algorithms for hyper-parameter optimization*, Proceedings of the Advances in Neural Information Processing Systems, Granada, pp. 2546–2554.
- [22] BERGSTRA J & BENGIO Y, 2012, *Random search for hyper-parameter optimization*, Journal of Machine Learning Research, **13**, pp. 281–305.
- [23] BESSIÈRE C, 1994, *Arc-consistency and arc-consistency again*, Artificial Intelligence, **65(1)**, pp. 179–190.
- [24] BESSIÈRE C, 2006, *Constraint propagation*, Handbook of Constraint Programming, **2**, pp. 29–83.
- [25] BESSIÈRE C & RÉGIN JC, 1996, *MAC and combined heuristics: Two reasons to forsake FC (and CBJ?) on hard problems*, Proceedings of the International Conference on Principles and Practice of Constraint Programming, Cambridge (MA), pp. 61–75.
- [26] BESSIÈRE C & RÉGIN JC, 2001, *Refining the basic constraint propagation algorithm*, Proceedings of the International Joint Conferences on Artificial Intelligence, Seattle (WA), pp. 309–315.
- [27] BESSIÈRE C, RÉGIN JC, YAP RH & ZHANG Y, 2005, *An optimal coarse-grained arc consistency algorithm*, Artificial Intelligence, **165(2)**, pp. 165–185.
- [28] BIANCHI L, DORIGO M, GAMBARDILLA LM & GUTJAHR WJ, 2009, *A survey on meta-heuristics for stochastic combinatorial optimization*, Natural Computing, **8(2)**, pp. 239–287.
- [29] BISHOP CM, 2006, *Pattern recognition and machine learning*, Springer-Verlag, New York (NY).
- [30] BIXBY RE, 2012, *A brief history of linear and mixed-integer programming computation*, Documenta Mathematica, Extra Volume “Optimization Stories”, pp. 107–121.

-
- [31] BOTTOU L, 1991, *Stochastic gradient learning in neural networks*, Proceedings of Neuro-Nîmes 91, EC2, **91(8)**.
- [32] BOUSSAÏD I, LEPAGNOT J & SIARRY P, 2013, *A survey on optimization metaheuristics*, Information Sciences, **237**, pp. 82–117.
- [33] BOUSSEMART F, HEMERY F, LECOUTRE C & SAIS L, 2004, *Boosting systematic search by weighting constraints*, Proceedings of the 16th European Conference on Artificial Intelligence, Valencia, pp. 146–150.
- [34] BRÉLAZ D, 1979, *New methods to color the vertices of a graph*, Communications of the ACM, **22(4)**, pp. 251–256.
- [35] BRIN S & PAGE L, 1998, *The anatomy of a large-scale hypertextual web search engine*, Computer Networks and ISDN Systems, **30(1-7)**, pp. 107–117.
- [36] BROWN DE, HUNTLEY CL & SPILLANE AR, 1989, *A parallel genetic heuristic for the quadratic assignment problem*, Proceedings of the 3rd International Conference on Genetic Algorithms, Fairfax (VA), pp. 406–415.
- [37] BROWNE CB, POWLEY E, WHITEHOUSE D, LUCAS SM, COWLING PI, ROHLFSHAGEN P, TAVENER S, PEREZ D, SAMOTHRAKIS S & COLTON S, 2012, *A survey of Monte Carlo tree search methods*, IEEE Transactions on Computational Intelligence and AI in Games, **4(1)**, pp. 1–43.
- [38] BURKE EK, HYDE MR & KENDALL G, 2012, *Grammatical evolution of local search heuristics*, IEEE Transactions on Evolutionary Computation, **16(3)**, pp. 406–417.
- [39] BURT RS, 2004, *Structural holes and good ideas*, American Journal of Sociology, **110(2)**, pp. 349–399.
- [40] CAUWET ML, LIU J & TEYTAUD O, 2014, *Algorithm portfolios for noisy optimization: Compare solvers early*, Proceedings of the International Conference on Learning and Intelligent Optimization, Gainesville (FL), pp. 1–15.
- [41] CHEN X & VAN BEEK P, 2001, *Conflict-directed backjumping revisited*, Journal of Artificial Intelligence Research, **14**, pp. 53–81.
- [42] CHOLLET F, 2015, *Keras*, [Online], [Cited September 2019], Available from <https://keras.io>.
- [43] COATES A, HUVAL B, WANG T, WU D, CATANZARO B & ANDREW N, 2013, *Deep learning with COTS HPC systems*, Proceedings of the 30th International Conference on Machine Learning, Atlanta (GA), pp. 1337–1345.
- [44] COLLOBERT R, BENGIO S & MARITHOZ J, 2002, *Torch: A modular machine learning software library*, [Online], [Cited September 2019], Available from <http://torch.ch/>.
- [45] CSARDI G & NEPUSZ T, 2006, *The igraph software package for complex network research*, Complex Systems, **1695(5)**, pp. 1–9.
- [46] D’HAESELEER P, 1994, *Context preserving crossover in genetic programming*, Proceedings of the First IEEE Conference on Evolutionary Computation, Orlando (FL), pp. 256–261.
- [47] DA SILVA MD & TAVARES HL, 2015, *Redis essentials*, Packt Publishing, Birmingham.
- [48] DANTZIG G, FULKERSON R & JOHNSON S, 1954, *Solution of a large-scale traveling-salesman problem*, Journal of the Operations Research Society of America, **2(4)**, pp. 393–410.

- [49] DEAN J, CORRADO G, MONGA R, CHEN K, DEVIN M, MAO M, SENIOR A, TUCKER P, YANG K & LE QV, 2012, *Large scale distributed deep networks*, Proceedings of the Advances in Neural Information Processing Systems, Lake Tahoe (NV), pp. 1223–1231.
- [50] DEB K, 2001, *Multi-objective optimization using evolutionary algorithms*, John Wiley & Sons, New York (NY).
- [51] DEB K, PRATAP A, AGARWAL S & MEYARIVAN T, 2002, *A fast and elitist multiobjective genetic algorithm: NSGA-II*, IEEE transactions on Evolutionary Computation, **6(2)**, pp. 182–197.
- [52] DECHTER R, 1990, *Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition*, Artificial Intelligence, **41(3)**, pp. 273–312.
- [53] DECHTER R & PEARL J, 1987, *Network-based heuristics for constraint-satisfaction problems*, Artificial Intelligence, **34(1)**, pp. 1–38.
- [54] DI GASPERO L, RENDL A & URLI T, 2013, *A hybrid ACO+ CP for balancing bicycle sharing systems*, Hybrid Metaheuristics, **1**, pp. 198–212.
- [55] DORIGO M & GAMBARDELLA LM, 1997, *Ant colonies for the travelling salesman problem*, Bio Systems, **43(2)**, pp. 73–81.
- [56] DORIGO M & GAMBARDELLA LM, 1997, *Ant colony system: A cooperative learning approach to the traveling salesman problem*, IEEE Transactions on Evolutionary Computation, **1(1)**, pp. 53–66.
- [57] EPSTEIN SL, FREUDER EC, WALLACE R, MOROZOV A & SAMUELS B, 2002, *The adaptive constraint engine*, Proceedings of the International Conference on Principles and Practice of Constraint Programming, Ithaca (NY), pp. 525–540.
- [58] FISCHETTI M & MONACI M, 2014, *Exploiting erraticism in search*, Operations Research, **62(1)**, pp. 114–122.
- [59] FISTER JR. I, YANG XS, FISTER I, BREST J & FISTER D, 2013, *A brief review of nature-inspired algorithms for optimization*, Elektrotehnikski vestnik, **80(3)**, pp. 116–122.
- [60] FREUDER EC, 1997, *In pursuit of the holy grail*, Constraints, **2(1)**, pp. 57–61.
- [61] FREUDER EC & QUINN MJ, 1985, *Taking advantage of stable sets of variables in constraint satisfaction problems*, Proceedings of the International Joint Conferences on Artificial Intelligence, Los Angeles (CA), pp. 1076–1078.
- [62] FROST D & DECHTER R, 1995, *Look-ahead value ordering for constraint satisfaction problems*, Proceedings of the International Joint Conferences on Artificial Intelligence, Quebec, pp. 572–578.
- [63] FUKUNAGA A, 2002, *Automated discovery of composite SAT variable-selection heuristics*, Proceedings of the 18th National Conference on Artificial Intelligence, Menlo Park (CA), pp. 641–648.
- [64] GAGLIOLO M & SCHMIDHUBER J, 2006, *Dynamic algorithm portfolios*, Annals of Mathematics and Artificial Intelligence, **47**, pp. 3–4.
- [65] GAGLIOLO M & SCHMIDHUBER J, 2006, *Learning dynamic algorithm portfolios*, Annals of Mathematics and Artificial Intelligence, **47(3-4)**, pp. 295–328.
- [66] GASCHNIG J, 1974, *A constraint satisfaction method for inference making*, Proceedings of the 12th Annual Allerton Conference on Circuit Systems Theory, Urbana (IL), pp. 866–874.

- [67] GEELEN PA, 1992, *Dual viewpoint heuristics for binary constraint satisfaction problems*, Proceedings of the 10th European Conference on Artificial Intelligence, Vienna, pp. 31–35.
- [68] GENT IP, MACINTYRE E, PRESSER P, SMITH BM & WALSH T, 1996, *An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem*, Proceedings of the International Conference on Principles and Practice of Constraint Programming, Cambridge (MA), pp. 179–193.
- [69] GINSBERG ML, 1993, *Dynamic backtracking*, Journal of Artificial Intelligence Research, **1**, pp. 25–46.
- [70] GINSBERG ML, FRANK M, HALPIN MP & TORRANCE MC, 1990, *Search lessons learned from crossword puzzles*, Proceedings of the 8th National Conference on Artificial Intelligence, Boston (MA), pp. 210–215.
- [71] GLEIXNER A, BASTUBBE M, EIFLER L, GALLY T, GAMRATH G, GOTTWALD RL, HENDEL G, HOJNY C, KOCH T, LÜBBECKE ME, MAHER SJ, MILTENBERGER M, MÜLLER B, PFETSCH ME, PUCHERT C, REHFELDT D, SCHLÖSSER F, SCHUBERT C, SERRANO F, SHINANO Y, VIERNICKEL JM, WALTER M, WEGSCHEIDER F, WITT JT & WITZIG J, 2018, *The SCIP Optimization Suite 6.0*, (Unpublished) Technical Report 18-26, Zuse Institute Berlin, Berlin.
- [72] GLOVER F, 1986, *Future paths for integer programming and links to artificial intelligence*, Computers and Operations Research, **13(5)**, pp. 533–549.
- [73] GOLDBERG DE, 1989, *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley Publishing Company, Boston (MA).
- [74] GOLOMB S & BAUMERT L, 1965, *Backtrack programming*, Journal of the ACM, **12(4)**, pp. 516–524.
- [75] GOMES CP, SELMAN B & KAUTZ H, 1998, *Boosting combinatorial search through randomization*, Proceedings of the 15th National/10th Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence, Madison (WI), pp. 431–437.
- [76] GOMES CP & SELMAN B, 1997, *Algorithm portfolio design: Theory vs. practice*, Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence, San Francisco (CA), pp. 190–197.
- [77] GOMES CP & SELMAN B, 2001, *Algorithm portfolios*, Artificial Intelligence, **126(1-2)**, pp. 43–62.
- [78] GREFENSTETTE J, GOPAL R, ROSMAITA B & VAN GUCHT D, 1985, *Genetic algorithms for the traveling salesman problem*, Proceedings of the First International Conference on Genetic Algorithms and their Applications, Pittsburgh (PA), pp. 160–168.
- [79] HAERDER T & REUTER A, 1983, *Principles of transaction-oriented database recovery*, ACM Computing Surveys, **15(4)**, pp. 287–317.
- [80] HAGE P & HARARY F, 1995, *Eccentricity and centrality in networks*, Social Networks, **17(1)**, pp. 57–63.
- [81] HARALICK RM & ELLIOTT GL, 1980, *Increasing tree search efficiency for constraint satisfaction problems*, Artificial Intelligence, **14(3)**, pp. 263–313.
- [82] HARRIES K & SMITH P, 1997, *Exploring alternative operators and search strategies in genetic programming*, Genetic Programming, **97**, pp. 147–155.
- [83] HARTIGAN JA & WONG MA, 1979, *Algorithm AS 136: A k-means clustering algorithm*, Journal of the Royal Statistical Society, Series C (Applied Statistics), **28(1)**, pp. 100–108.

- [84] HARVEY WD, 1995, *Nonsystematic backtracking search*, PhD thesis, Stanford University, Stanford (CA).
- [85] HOCHREITER S & SCHMIDHUBER J, 1997, *Long short-term memory*, *Neural Computation*, **9(8)**, pp. 1735–1780.
- [86] HOOKER JN, 1995, *Testing heuristics: We have it all wrong*, *Journal of Heuristics*, **1(1)**, pp. 33–42.
- [87] HUBERMAN BA, LUKOSE RM & HOGG T, 1997, *An economics approach to hard computational problems*, *Science*, **275(5296)**, pp. 51–54.
- [88] HULUBEI T & O’SULLIVAN B, 2005, *Search heuristics and heavy-tailed behaviour*, *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, Sitges, pp. 328–342.
- [89] HUTTER F, HOOS HH & LEYTON-BROWN K, 2011, *Sequential model-based optimization for general algorithm configuration*, *Proceedings of the International Conference on Learning and Intelligent Optimization*, Rome, pp. 507–523.
- [90] IBA H, SATO T & DEGARIS H, 1995, *Recombination guidance for numerical genetic programming*, *Proceedings of the IEEE International Conference on Evolutionary Computation*, Perth, p. 97.
- [91] IBM, 2015, *ILOG CPLEX CP Optimizer, V12.3*, [Online], [Cited September 2019], Available from www.cplex.com.
- [92] JAFFAR J & MAHER MJ, 1994, *Constraint logic programming: A survey*, *Journal of Logic Programming*, **19**, pp. 503–581.
- [93] JEFFERSON C, MIGUEL I, HNICH B, WALSH T & GENT IP, *CSPLib Language MiniZinc*, [Online], [Cited September 2019], Available from <http://www.csplib.org/Languages/MiniZinc>.
- [94] JEFFERSON C, MIGUEL I, HNICH B, WALSH T & GENT IP, 1999, *CSPLib: A problem library for constraints*, [Online], [Cited September 2019], Available from <http://www.csplib.org>.
- [95] JÉGOU P & TERRIOUX C, 2003, *Hybrid backtracking bounded by tree-decomposition of constraint networks*, *Artificial Intelligence*, **146(1)**, pp. 43–75.
- [96] KAELBLING LP, LITTMAN ML & MOORE AW, 1996, *Reinforcement learning: A survey*, *Journal of Artificial Intelligence Research*, **4**, pp. 237–285.
- [97] KARP RM, 1972, *Reducibility among combinatorial problems*, Springer, Boston (MA).
- [98] KATSIRELOS G & BACCHUS F, 2005, *Generalized nogoods in CSPs*, *Proceedings of the 20th National Conference on Artificial Intelligence*, Pittsburgh (PA), pp. 390–396.
- [99] KAUTZ H, HORVITZ E, RUAN Y, GOMES C & SELMAN B, 2002, *Dynamic restart policies*, *Proceedings of the 18th National Conference on Artificial Intelligence*, Alberta, pp. 674–681.
- [100] KENNEDY J & EBERHART R, 1995, *Particle swarm optimization*, *Proceedings of the IEEE International Conference on Neural Networks*, Perth, pp. 1942–1948.
- [101] KHICHANE M, ALBERT P & SOLNON C, 2008, *Integration of ACO in a constraint programming language*, *Proceedings of the International Conference on Ant Colony Optimization and Swarm Intelligence*, Brussels, pp. 84–95.
- [102] KINGMA DP & BA J, 2014, *Adam: A method for stochastic optimization*, *Proceedings of the 3rd International Conference on Learning Representations*, San Diego (CA).

-
- [103] KINNEAR KE, 1994, *Fitness landscapes and difficulty in genetic programming*, Proceedings of the First IEEE Conference on Evolutionary Computation, Orlando (FL), pp. 142–147.
- [104] KIRKPATRICK S, 1984, *Optimization by simulated annealing: Quantitative studies*, Journal of Statistical Physics, **34**(5-6), pp. 975–986.
- [105] KLEINBERG JM, 1999, *Authoritative sources in a hyperlinked environment*, Journal of the ACM, **46**(5), pp. 604–632.
- [106] KNOWLES J & CORNE D, 1999, *The Pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation*, Proceedings of the Congress on Evolutionary Computation, Washington (DC), pp. 98–105.
- [107] KOTTHOFF L, 2012, *Algorithm selection for combinatorial search problems: A survey*, AI Magazine, **35**(3), pp. 1–37.
- [108] KOTTHOFF L, GENT I & MIGUEL I, 2012, *An evaluation of machine learning in algorithm selection for search problems*, AI Communications, **25**(3), pp. 257–270.
- [109] KOZA JR, 1998, *Genetic programming IV*, Springer US, Boston (MA).
- [110] KOZA JR, 1990, *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems*, (Unpublished) Technical Report, Stanford (CA).
- [111] KOZA JR, 1992, *Genetic programming: On the programming of computers by means of natural selection*, MIT Press, Cambridge (MA).
- [112] KRIZHEVSKY A, SUTSKEVER I & HINTON GE, 2012, *Imagenet classification with deep convolutional neural networks*, Proceedings of the Advances in Neural Information Processing Systems, Lake Tahoe (NV), pp. 1097–1105.
- [113] KUNG HT, LUCCIO F & PREPARATA FP, 1975, *On finding the maxima of a set of vectors*, Journal of the ACM, **22**(4), pp. 469–476.
- [114] LAND AH & DOIG AG, 1960, *An automatic method of solving discrete programming problems*, Econometrica, **28**(3), pp. 497–520.
- [115] LANGDON WB, 1999, *Size fair and homologous tree genetic programming crossovers*, Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation, Orlando (FL), pp. 1092–1097.
- [116] LARRANAGA P, KUIJPERS CMH, MURGA RH, INZA I & DIZDAREVIC S, 1999, *Genetic algorithms for the travelling salesman problem: A review of representations and operators*, Artificial Intelligence Review, **13**(2), pp. 129–170.
- [117] LAWLER EL & WOOD DE, 1966, *Branch-and-bound methods: A survey*, Operations Research, **14**(4), pp. 699–719.
- [118] LE QV, NGIAM J, COATES A, LAHIRI A, PROCHNOW B & NG AY, 2011, *On optimization methods for deep learning*, Proceedings of the 28th International Conference on Machine Learning, Bellevue (WA), pp. 265–272.
- [119] LECOUTRE C, BOUSSEMART F & HEMERY F, 2004, *Backjump-based techniques versus conflict-directed heuristics*, Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence, Boca Raton (FL), pp. 549–557.
- [120] LECUN Y, BOSER B, DENKER JS, HENDERSON D, HOWARD RE, HUBBARD W & JACKEL LD, 1989, *Backpropagation applied to handwritten zip code recognition*, Neural Computation, **1**(4), pp. 541–551.

- [121] LEYTON-BROWN K, NUDELMAN E & SHOHAM Y, 2002, *Learning the empirical hardness of optimization problems: The case of combinatorial auctions*, Proceedings of the International Conference on Principles and Practice of Constraint Programming, Ithaca (NY), pp. 556–572.
- [122] LI W & VAN BEEK P, 2004, *Guiding real-world SAT solving with dynamic hypergraph separator decomposition*, Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence, Boca Raton (FL), pp. 542–548.
- [123] LIBERATORE P, 2000, *On the complexity of choosing the branching literal in DPLL*, Artificial Intelligence, **116(1-2)**, pp. 315–326.
- [124] LOREGGIA A, MALITSKY Y, SAMULOWITZ H & SARASWAT V, 2016, *Deep learning for algorithm portfolios*, Proceedings of the 13th AAI Conference on Artificial Intelligence, Phoenix (AZ), pp. 1280–1286.
- [125] LUBY M, SINCLAIR A & ZUCKERMAN D, 1993, *Optimal speedup of Las Vegas algorithms*, The 2nd Israel Symposium on Theory and Computing Systems, pp. 128–133.
- [126] LUKE S, 2009, *Essentials of metaheuristics*, 2nd Edition, Lulu, Department of Computer Science, George Mason University, Fairfax (VA).
- [127] LUKE S, PANAIT L, BALAN G, PAUS S, SKOLICKI Z, BASSETT J, HUBLEY R & CHIRCOP A, 2006, *Ecj: A java-based evolutionary computation research system*, [Online], [Cited September 2019], Available from <http://cs.gmu.edu/ecjlab/projects/ecj>.
- [128] MACKWORTH AK, 1977, *Consistency in networks of relations*, Artificial Intelligence, **8(1)**, pp. 99–118.
- [129] MALITSKY Y, SABHARWAL A, SAMULOWITZ H & SELLMANN M, 2013, *Algorithm portfolios based on cost-sensitive hierarchical clustering*, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, pp. 608–614.
- [130] MALITSKY Y, SABHARWAL A, SAMULOWITZ H & SELLMANN M, 2011, *Non-model-based algorithm portfolios for SAT*, Proceedings of the International Conference on Theory and Applications of Satisfiability Testing, Ann Arbor (MI), pp. 369–370.
- [131] MCGREGOR JJ, 1979, *Relational consistency algorithms and their application in finding subgraph and graph isomorphisms*, Information Sciences, **19(3)**, pp. 229–250.
- [132] MCKAY B, WILLIS MJ & BARTON GW, 1995, *Using a tree structured genetic algorithm to perform symbolic regression*, Proceedings of the First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, Sheffield, pp. 487–492.
- [133] MIKOLOV T, KARAFIÁT M, BURGET L, ČERNOCKÝ J & KHUDANPUR S, 2010, *Recurrent neural network based language model*, Proceedings of the 11th Annual Conference of the International Speech Communication Association, Makuhari, pp. 1045–1048.
- [134] MILLER CE, TUCKER AW & ZEMLIN RA, 1960, *Integer programming formulation of traveling salesman problems*, Journal of the ACM, **7(4)**, pp. 326–329.
- [135] MINTON S, 1996, *Automatically configuring constraint satisfaction programs: A case study*, Constraints, **1(1-2)**, pp. 7–43.
- [136] MITCHELL TM, 1997, *Machine learning*, McGraw-Hill, New York (NY).
- [137] MOHR R & HENDERSON TC, 1986, *Arc and path consistency revisited*, Artificial Intelligence, **28(2)**, pp. 225–233.
- [138] MONTANARI U & ROSSI F, 1991, *Constraint relaxation may be perfect*, Artificial Intelligence, **48(2)**, pp. 143–170.

- [139] MOSCATO P, 1989, *On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms*, Caltech Concurrent Computation Program, C3P Report 826, California Institute of Technology, Pasadena (CA).
- [140] MOSKEWICZ MW, MADIGAN CF, ZHAO Y, ZHANG L & MALIK S, 2001, *Chaff: Engineering an efficient SAT solver*, Proceedings of the 38th Annual Design Automation Conference, Las Vegas (NV), pp. 530–535.
- [141] NAGATA Y & KOBAYASHI S, 2013, *A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem*, INFORMS Journal on Computing, **25(2)**, pp. 346–363.
- [142] NEWMAN ME, 2016, *Mathematics of networks*, The New Palgrave Dictionary of Economics, pp. 1–8.
- [143] NUDEL B, 1983, *Consistent-labeling problems and their algorithms: Expected-complexities and theory-based heuristics*, Artificial Intelligence, **21(1-2)**, pp. 135–178.
- [144] O'MAHONY E, HEBRARD E, HOLLAND A, NUGENT C & O'SULLIVAN B, 2008, *Using case-based reasoning in an algorithm portfolio for constraint solving*, Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science, Cork City.
- [145] PENG F, TANG K, CHEN G & YAO X, 2010, *Population-based algorithm portfolios for numerical optimization*, IEEE Transactions on Evolutionary Computation, **14(5)**, pp. 782–800.
- [146] PERRON L & FURNON V, 2019, *OR-Tools*, [Online], [Cited September 2019], Available from <https://developers.google.com/optimization/>.
- [147] POLI R & LANGDON WB, 1997, *A new schema theorem for genetic programming with one-point crossover and point mutation*, Cognitive Science Research Papers, University of Birmingham CSRP, Birmingham.
- [148] POLI R & LANGDON WB, 1998, *On the search properties of different crossover operators in genetic programming*, Proceedings of Genetic Programming'98, Madison (WI), pp. 293–301.
- [149] POLI R, LANGDON WB & DIGNUM S, 2007, *On the limiting distribution of program sizes in tree-based genetic programming*, Proceedings of the European Conference on Genetic Programming, Valencia, pp. 193–204.
- [150] POLI R, LANGDON W & MCPHEE N, 2008, *A field guide to genetic programming*, [Online], [Cited September 2019], Available from <http://www.gp-field-guide.org.uk>.
- [151] PRESTWICH S, 2000, *A hybrid search architecture applied to hard random 3-SAT and low-autocorrelation binary sequences*, Proceedings of the International Conference on Principles and Practice of Constraint Programming, Singapore, pp. 337–352.
- [152] PROSSER P, 1993, *Hybrid algorithms for the constraint satisfaction problem*, Computational Intelligence, **9(3)**, pp. 268–299.
- [153] R CORE TEAM, 2018, *R: A language and environment for statistical computing*, [Online], [Cited September 2019], Available from <https://www.R-project.org/>.
- [154] RADEMEYER AL, 2013, *Algorithmic approaches to solving multi-period sales force and delivery vehicle master routing problems*, PhD thesis, University of the Witwatersrand, Johannesburg.
- [155] RASMUSSEN RV & TRICK MA, 2007, *A Benders approach for the constrained minimum break problem*, European Journal of Operational Research, **177(1)**, pp. 198–213.

- [156] REFALO P, 2004, *Impact-based search strategies for constraint programming*, Proceedings of the International Conference on Principles and Practice of Constraint Programming, Toronto (ON), pp. 557–571.
- [157] RIBEIRO MT, SINGH S & GUESTRIN C, 2016, *Why should I trust you?: Explaining the predictions of any classifier*, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1135–1144.
- [158] ROSVALL M & BERGSTROM CT, 2008, *Maps of random walks on complex networks reveal community structure*, Proceedings of the National Academy of Sciences, **105**(4), pp. 1118–1123.
- [159] SABIN D & FREUDER EC, 1994, *Contradicting conventional wisdom in constraint satisfaction*, Proceedings of the International Workshop on Principles and Practice of Constraint Programming, Orcas Island (WA), pp. 10–20.
- [160] SABIN D & FREUDER EC, 1997, *Understanding and improving the MAC algorithm*, Proceedings of the International Conference on Principles and Practice of Constraint Programming, Linz, pp. 167–181.
- [161] SCHIEX T & VERFAILLIE G, 1994, *Nogood recording for static and dynamic constraint satisfaction problems*, International Journal on Artificial Intelligence Tools, **3**(2), pp. 187–207.
- [162] SCHOENAUER M, SEBAG M, JOUVE F, LAMY B & MAITOURNAM H, 1996, *Evolutionary identification of macro-mechanical models*, Advances in Genetic Programming, **2**, pp. 467–488.
- [163] SCHUURMANS D & SOUTHEY F, 2001, *Local search characteristics of incomplete SAT procedures*, Artificial Intelligence, **132**(2), pp. 121–150.
- [164] SILVER D, HUBERT T, SCHRITTWIESER J, ANTONOGLIOU I, LAI M, GUEZ A, LANCTOT M, SIFRE L, KUMARAN D, GRAEPEL T, LILICRAP T, SIMONYAN K & HASSABIS D, 2018, *A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play*, Science, **362**(6419), pp. 1140–1144.
- [165] SÖRENSEN K, 2015, *Metaheuristics — the metaphor exposed*, International Transactions in Operational Research, **22**(1), pp. 3–18.
- [166] SRINIVAS N & DEB K, 1994, *Muiltiobjective optimization using nondominated sorting in genetic algorithms*, Evolutionary Computation, **2**(3), pp. 221–248.
- [167] SRIVASTAVA N, HINTON G, KRIZHEVSKY A, SUTSKEVER I & SALAKHUTDINOV R, 2014, *Dropout: A simple way to prevent neural networks from overfitting*, Journal of Machine Learning Research, **15**, pp. 1929–1958.
- [168] STALLMAN RM & SUSSMAN GJ, 1977, *Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis*, Artificial Intelligence, **9**(2), pp. 135–196.
- [169] STEWART TJ, 1992, *A critical survey on the status of multiple criteria decision making theory and practice*, Omega, **20**(5-6), pp. 569–586.
- [170] ŠTRUMBELJ E & KONONENKO I, 2014, *Explaining prediction models and individual predictions with feature contributions*, Knowledge and Information Systems, **41**(3), pp. 647–665.
- [171] STUCKEY PJ, FEYDY T, SCHUTT A, TACK G & FISCHER J, 2014, *The minizinc challenge 2008–2013*, AI Magazine, **35**(2), pp. 55–60.

- [172] SUTTON RS & BARTO AG, 1998, *Introduction to reinforcement learning*, MIT Press Cambridge (MA).
- [173] TANG K, PENG F, CHEN G & YAO X, 2014, *Population-based algorithm portfolios with automated constituent algorithms selection*, Information Sciences, **279**, pp. 94–104.
- [174] TESAURO G, 1995, *Temporal difference learning and TD-Gammon*, Communications of the ACM, **38(3)**, pp. 58–68.
- [175] THEANO DEVELOPMENT TEAM, 2016, *Theano: A Python framework for fast computation of mathematical expressions*, [Online], [Cited September 2019], Available from <http://arxiv.org/abs/1605.02688>.
- [176] TOTH P & VIGO D, 2002, *The vehicle routing problem*, SIAM, Philadelphia (PA).
- [177] TRICK MA, 2011, *Sports scheduling*, Hybrid Optimization, **45**, pp. 489–508.
- [178] TSANG E, 2014, *Foundations of constraint satisfaction: The classic text*, BoD–Books on Demand, Essex.
- [179] VAN BEEK P, 2006, *Backtracking search algorithms*, Handbook of Constraint Programming, **2**, pp. 85–134.
- [180] VAN DEN AKKER JM, HOOGEVEEN JA & VAN DE VELDE SL, 1999, *Parallel machine scheduling by column generation*, Operations Research, **47(6)**, pp. 862–872.
- [181] VILÍM P, LABORIE P & SHAW P, 2015, *Failure-directed search for constraint-based scheduling*, Proceedings of the International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Barcelona, pp. 437–453.
- [182] WALLACE RJ, 2005, *Factor analytic studies of CSP heuristics*, Proceedings of the International Conference on Principles and Practice of Constraint Programming, Sitges, pp. 712–726.
- [183] WATKINS CJ & DAYAN P, 1992, *Q-learning*, Machine Learning, **8(3-4)**, pp. 279–292.
- [184] WILLIAMS VV, 2014, *Multiplying matrices in $O(n^{2.373})$ time*, Stanford University, Stanford (CA).
- [185] WOLPERT DH & MACREADY WG, 1997, *No free lunch theorems for optimization*, IEEE Transactions on Evolutionary Computation, **1(1)**, pp. 67–82.
- [186] XING Z & ZHANG W, 2005, *MaxSolver: An efficient exact algorithm for (weighted) maximum satisfiability*, Artificial Intelligence, **164(1)**, pp. 47–80.
- [187] XU L, HUTTER F, HOOS HH & LEYTON-BROWN K, 2008, *SATzilla: Portfolio-based algorithm selection for SAT*, Journal of Artificial Intelligence Research, **32**, pp. 565–606.
- [188] YAN X & SU X, 2009, *Linear regression analysis: Theory and computing*, World Scientific, Washington (DC).
- [189] ZABIH R, 1990, *Some applications of graph bandwidth to constraint satisfaction problems*, Proceedings of the 8th National Conference on Artificial Intelligence, Boston (MA), pp. 46–51.
- [190] ZHANG H, 2002, *A randomisation strategy for combinatorial search*, Proceedings of the International Symposium on AI and Math, Fort Lauderdale (FL).
- [191] ZHANG Y & YAP RH, 2001, *Making AC-3 an optimal algorithm*, Proceedings of the International Joint Conferences on Artificial Intelligence, Seattle (WA), pp. 316–321.
- [192] ZHAO Z, ZHENG S, LI C, SUN J, CHANG L & CHICLANA F, 2018, *A comparative study on community detection methods in complex networks*, Journal of Intelligent and Fuzzy Systems, **35(1)**, pp. 1077–1086.

- [193] ZITZLER E, 1999, *Evolutionary algorithms for multiobjective optimization: methods and applications*, Shaker, Ithaca (NY).
- [194] ZIVAN R & MEISELS A, 2007, *Conflict directed backjumping for max-CSPs*, Proceedings of the International Joint Conferences on Artificial Intelligence, Hyderabad, pp. 198–204.