KATHOLIEKE UNIVERSITEIT
LEUVEN

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Science
Department of Computer Science

Intelligent Hyper-heuristics: A Tool for Solving Generic Optimisation Problems

Mustafa MISIR

Dissertation presented in partial
fulfillment of the requirements for
the degree of Doctor
in Science

September 2012

Intelligent Hyper-heuristics: A Tool for Solving Generic Optimisation Problems

Mustafa MISIR

Jury:
Prof. dr. Paul Igodt, chair
Prof. dr. Patrick De Causmaecker, supervisor
Prof. dr. ir. Greet Vanden Berghe, co-supervisor
Dr. Katja Verbeeck, co-supervisor
Prof. dr. Danny De Schreye
Prof. dr. ir. Stefan Vandewalle
Dr. Christian Blum
(Universitat Politècnica de Catalunya)
Dr. Ender Özcan
(University of Nottingham)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Science

September 2012

D/2012/10.705/78
ISBN 978-90-8649-561-0

Preface

I would like to first thank my promotors, Patrick De Causmaecker, Greet Vanden Berghe and Katja Verbeeck for providing me the PhD opportunity to pursue my goal of being a researcher. Their encouragement and guidance throughout my PhD helped me to look at my research subject from different perspectives and delivering new ideas. Besides my promotors, I would like to thank all the members of my PhD examination commmittee, Danny De Schreye, Stefan Vandewalle, Christian Blum and Ender Özcan. Their valuable comments helped much to improve my dissertation.

Thanks all the members of the CODeS research group for the friendly environment.

I would like to particularly thank all my colleagues that I collaborated with from CODeS, namely Burak Bilgin, Peter Demeester, Pieter Smet, Wim Vancroonenburg and Tony Wauters. Each of these collaborations provided significant contributions to the dissertation.

The PC cluster set up by Jannes Verstichel and Wim Vancroonenburg was a lifesaver to run all the extensive tests reported in the dissertation.

Additionally, it would not be possible to publish the dissertation without Erik Van Achter and Luke Connolly's help on the language of the text.

Finally, I would like to thank my family for their continuous support during my whole life.

Abstract

Designing a dedicated search and optimisation algorithm is a time-consuming process requiring an in-depth analysis of the problem. The resulting algorithm is expected to be effective for solving a given set of target problem instances. However, since the algorithm is dedicated, it is hard to adapt and to apply to other problems. Meta-heuristics were brought in to cope with this drawback. Nevertheless, in most of the meta-heuristic studies, the employed meta-heuristics have been implemented as rather problem-dependent methodologies. *Hyper-heuristics* furnish problem-independent management opportunities differently from such search and optimisation algorithms.

The present dissertation focuses on the *generality* of hyper-heuristics. It thereby aims at designing intelligent hyper-heuristics so that generality is facilitated. While most works on hyper-heuristics make use of the term generality in describing the potential for solving various problems, the performance changes across different domains have only rarely been reported. Additionally, there are other generality related elements such as the performance variations over distinct heuristic sets, that are usually ignored. This means that there is no study fully discussing generality questions while providing a hyper-heuristic design capable of addressing them.

To this end, the factors affecting the hyper-heuristics' generality are determined and several novel hyper-heuristic components are developed based on these factors. Then, the hyper-heuristics using the new components are tested across various problem domains on different heuristic sets, while also varying the experimental limits. First, each developed hyper-heuristic is applied to only one problem domain. The performance of these hyper-heuristics is compared with other algorithms encountered in the literature. The information gathered during these experiments is used later on to design a highly adaptive, intelligent selection hyper-heuristic.

The ultimate result of the present PhD research is called the *Generic Intelligent*

Hyper-heuristic (GIHH). It is equipped with multiple online adaptive hyper-heuristic procedures and decision mechanisms for simultaneously coordinating them. GIHH is expected to evolve for different search environments without human intervention. A simplified version of GIHH is tested via a series of experiments on three problems from practice to measure its generality level. A comprehensive performance analysis is conducted using a group of selection hyper-heuristics only involving heuristic selection and move acceptance mechanisms from the literature. The analysis provides strong conclusions about when a hyper-heuristic with certain characteristics has advantages or disadvantages.

Finally, GIHH is tested on other challenging combinatorial optimisation problems under different empirical conditions. The computational results indicate that GIHH is effective in solving the target instances from distinct problem domains. Additionally, GIHH won the first international cross domain heuristic search challenge 2011 against 19 high-level algorithms developed by the other academic competitors. The winning hyper-heuristic was then used to investigate the performance and contribution of low-level heuristics while simultaneously solving three problems with routing and rostering characteristics. This completely new application of a hyper-heuristic offers promising perspectives for supporting dedicated heuristic development.

Beknopte samenvatting

Het ontwerp van een probleemspecifiek zoekalgoritme is tijdrovend en vergt een diepgaande analyse. Van het resultaat wordt immers verwacht dat het doeltreffend is voor het oplossen van een groot aantal probleemgevallen. Daar het gaat om een probleemspecifiek algoritme, is het moeilijk aanpasbaar en tevens weinig bruikbaar voor andere gelijkaardige problemen. Om dit nadeel te omzeilen worden meta-heuristieken gebruikt. Echter, die meta-heuristieken zijn vaak geïmplementeerd als tamelijk probleem afhankelijke methoden. Hyper-heuristieken bieden probleem-onafhankelijk beheer van zoekcomponenten en precies daardoor zijn ze algemeen toepasbaar.

Het voorliggend proefschrift richt zich op de *generality* van hyper-heuristieken. Het beoogt het ontwerpen van intelligente hyper-heuristieken,om die algemeenheid te bewerkstelligen. Terwijl de meeste studies over hyper-heuristieken gebruik maken van de term *generality* om te beschrijven dat ze diverse problemen kunnen aanpakken, worden de performantieveranderingen over de verschillende domeinen maar zelden besproken. Daarnaast zijn er andere aan algemeenheid gerelateerde elementen meestal genegeerd. Dit betekent dat er tot op heden geen studie bestaat die de vragen omtrent *generality* diepgaand bespreekt en daarbij hyperheuristieken voorstelt die in staat zijn daarop in te spelen.

Daartoe worden de factoren die de *generality* van hyper-heuristieken beïnvloeden vastgelegd en wordt een aantal nieuwe componenten ontwikkeld voor hyperheuristieken op basis van deze eerder genoemde factoren. Vervolgens worden deze dan getest in verschillende probleemdomeinen en op verschillende heuristische sets, terwijl de experimentele limiet varieert. Eerst wordt elke ontwikkelde hyper-heuristiek toegepast op enkel één probleemdomein. De performantie van deze hyper-heuristieken wordt dan vergeleken met andere algoritmen uit de vakliteratuur. De op deze manier verkregen informatie wordt later gebruikt om een hoogst aanpasbare, intelligente hyper-heuristiek te ontwerpen.

Het ultieme resultaat van het doctoraal proefschift is de *Generic Intelligent*

Hyper-heuristic (GIHH). Deze is toegerust met meerdere adaptieve hyper-heuristieke procedures en besluitvormingsmechanismen voor gelijktijdige coördinatie van de heuristieke set. Naar alle waarschijnlijkheid zal de GIHH zich aan verschillende zoekomgevingen aanpassen zonder menselijke tussenkomst. Een vereenvoudigde versie van de GIHH wordt getest op zijn algemeenheid door middel van een reeks experimenten op drie problemen uit de praktijk. Tevens wordt een uitgebreide prestatie-analyse uitgevoerd met behulp van een groep van selectie hyper-heuristieken waarbij alleen *heuristic selection* en *move acceptance* mechanismen uit de vakliteratuur worden gebruikt. De analyse laat toe sterke conclusies te trekken over het gedrag van een hyper-heuristiek met bepaalde eigenschappen gedurende het zoekproces.

Tot slot wordt de GIHH getest op andere uitdagende combinatorische optimalisatieprobleemstelling telkens onder andere empirische omstandigheden. De resultaten geven aan dat de GIHH bruikbaar is om de problemen uit verschillende domeinen op te lossen. Er dient hier vermeld dat de GIHH de eerste Cross domain heuristic challenge 2011 won van 19 andere hoog niveau algoritmen. De winnende hyperheuristiek werd vervolgens gebruikt om de prestaties en bijdragen van low level heuristieken te onderzoeken tijdens het oplossen van drie problemen met routing en rostering kenmerken. Deze volledig nieuwe toepassing van een hyper-heuristiek biedt veelbelovende perspectieven voor de ondersteuning van de ontwikkeling van probleemspecifieke heuristieken.

Contents

Abstract iii

Contents vii

List of Figures xiii

List of Tables xvii

1 Introduction 1

1.1 Hyper-heuristics 2

1.2 Contributions 5

1.3 Structure of the dissertation 8

2 Literature Survey 11

2.1 Selection strategies in hyper-heuristics 11

2.1.1 Evolutionary algorithms 12

2.1.2 Ant colony optimisation 18

2.1.3 Tabu search 21

2.1.4 Case-based reasoning 22

2.1.5 Variable neighbourhood search 23

2.1.6 Reward-based mechanisms 23

2.2 Generation strategies in hyper-heuristics 24
2.2.1 Generation using primitive elements 24
2.2.2 Generation using problem-dependent elements 26
2.2.3 Generation via hybridisation 27
2.3 Move acceptance 27
2.3.1 Threshold accepting 28
2.4 Distributed hyper-heuristics 31
2.5 Comparison and analysis 33
2.6 Problems 38

3 Problem Domains **47**

3.1 Travelling tournament problem 48
3.1.1 Problem instances 48
3.1.2 Low-level heuristics 50
3.2 Structured problems with routing and rostering 50
3.2.1 General problem characteristics 51
3.2.2 Home care scheduling 53
3.2.3 Security personnel routing and rostering 56
3.2.4 Maintenance personnel scheduling 58
3.3 Ready-mixed concrete delivery problem 59
3.3.1 Problem instances 63
3.3.2 Low-level heuristic set 63
3.4 Patient admission scheduling problem 65
3.4.1 Problem instances 66
3.4.2 Low-level heuristic set 1 66
3.4.3 Low-level heuristic set 2 66
3.5 Nurse rostering problem 67

3.5.1 Problem instances 68
3.5.2 Low-level heuristics 68
3.6 Problems in HyFlex 71
3.6.1 1D Bin packing 71
3.6.2 Max SAT 72
3.6.3 Permutation flowshop scheduling 72
3.6.4 Personnel scheduling 73
3.6.5 Travelling Salesman 74
3.6.6 Vehicle routing 75
3.7 Conclusion 75

4 Selection Hyper-heuristics with Different Sub-mechanisms 77

4.1 A selection hyper-heuristic with learning automata and iteration limited threshold accepting 79
4.1.1 Heuristic selection: learning automata 80
4.1.2 Move acceptance: iteration limited threshold accepting . 82
4.1.3 Performance of hyper-heuristic sub-mechanisms: L_R, LR_R-I, ILTA 84
4.2 Effect of learning-based hyper-heuristic components 91
4.2.1 Tested selection hyper-heuristics 91
4.2.2 Performance of the hyper-heuristics with new heuristic selection and mentoring strategies 96
4.3 A selection hyper-heuristic with heuristic selection and adaptive acceptance 99
4.3.1 Heuristic (subset) selection: a dynamic heuristic set strategy 100
4.3.2 Move acceptance: adaptive iteration limited threshold accepting 104
4.3.3 Performance of hyper-heuristic sub-mechanisms: DHS, AILTA 104

4.4 A selection hyper-heuristic with a new move acceptance criterion 110
4.4.1 Tested selection hyper-heuristics 110
4.4.2 Experiments 112
4.5 Conclusion 118

5 Generality of Hyper-heuristics 121
5.1 A selection hyper-heuristic framework 122
5.1.1 Heuristic selection 123
5.1.2 Mentoring 124
5.1.3 Move acceptance 126
5.2 Experimental settings 128
5.2.1 Tested hyper-heuristics 128
5.2.2 Heuristic sets 128
5.3 Case studies 130
5.3.1 The home care scheduling problem 130
5.3.2 The nurse rostering problem 136
5.3.3 The patient admission scheduling problem 143
5.4 Conclusion 148
5.4.1 Effect of heuristic selection 150
5.4.2 Effect of move acceptance 150

6 An Intelligent Selection Hyper-heuristic 153
6.1 Design principles 157
6.2 Management 158
6.2.1 Adaptive dynamic heuristic set strategy 158
6.3 Mentoring 162
6.3.1 Relay hybridisation 162
6.3.2 Parameter adaptation of the parametric heuristics . . . 164

6.4 Move acceptance 165
6.4.1 Adaptive iteration limited list-based threshold accepting 165
6.5 HyFlex 169
6.6 Computational results 170
6.6.1 Improvement provided based on execution time 177
6.6.2 Adaptive dynamic heuristic set strategy 179
6.6.3 Relay hybridisation 182
6.6.4 Adaptive iteration limited list-based threshold accepting 185
6.6.5 Re-initialisation 188
6.6.6 Cross-domain heuristic search challenge (CHeSC 2011) . 188
6.7 Conclusion 190

7 A Selection Hyper-heuristic as Analysis Tool 193
7.1 Low-level heuristics 194
7.2 Computational results 196
7.2.1 Performance of the low-level heuristics 198
7.2.2 Exploring new heuristics 200
7.3 Conclusion 201

8 Conclusion 203
8.1 Discussion 204
8.2 Future research 206

Bibliography 209

Publications List 247

Biography 253

List of Figures

1.1 A traditional selection hyper-heuristic framework [227] 3

1.2 Best fitness improvement over time 4

3.1 A home care scheduling problem example 54

3.2 A ready-mixed concrete delivery problem example 60

3.3 Heuristic-type distributions and heuristic set sizes for each problem domain (each column shows the number of heuristics and their types) 71

4.1 A learning automaton put into a feedback loop with its environment [323] 80

4.2 Solution samples from the SR-ILTA hyper-heuristic on a TTP instance (NL12) 82

4.3 Fitness landscapes for two heuristics 83

4.4 An example of the prohibition mechanism with tabu duration of 2 ($d = 2$) 102

4.5 Extending the acceptable search space region for worsening solutions by increasing the range value from $R_1 \mapsto R_n$ 105

4.6 Sample QI values during the first 250 phases over the hh by SR-AILTA $R_1 = 1.003$ $R_n = 1.007$ with $d = 1$ and $ph_{iter} = 2500$ 108

4.7 Best fitness, current fitness and threshold level changes by SR-AILLA-F during the run on p2011-01-27 (run 3) 115

4.8 Total start time exceeded in minutes for each problem instance after 10 minutes and 1 hour of execution time by SR-AILLA-F (for the first runs) 117

4.9 The number of vehicles and the number of deliveries for each solution regarding each problem instance found by SR-AILLA-F (for the first runs). The chart above belongs to the solutions generated after 10 minutes and the chart below shows the solutions found after 1 hour 118

4.10 The best fitness changes during the run on p2011-01-18 (run 7) 119

4.11 Improvement performance of the low-level heuristic on p2011-02-15 (run 5) 120

5.1 A selection hyper-heuristic framework 123

5.2 Tournament selection process (a heuristic is applied for a number of times to visit sampling solutions in a solution space) 125

5.3 Hill climbing process (a heuristic is applied for a number of times like a hill climber) 126

5.4 The number of iterations spent on each heuristic set for the HCSP by choosing heuristics in a uniformly random manner . . 130

5.5 Number of new best solutions per iteration for each heuristic set for the HCSP (ADHS-AILLA on $ll3$, based on 1 hour test results) 131

5.6 The significantly best hyper-heuristics on each heuristic set for the home care scheduling problem after 10 minutes (left) and 1 hour (right) (Consecutive elements on the x axis refer to ADHS and SR respectively) 132

5.7 Average ranking of the hyper-heuristics on the HCSP after 10 minutes. Each graph represents the results obtained with the heuristic set. They are ordered from left to right, top to bottom: $HS_1 \rightarrow HS_9$ 133

5.8 Average ranking of the hyper-heuristics on the HCSP after 1 hour. Each graph represents the results obtained with the heuristic set. They are ordered from left to right, top to bottom: $HS_1 \rightarrow HS_9$ 134

5.9 Box plot for average ranks of the hyper-heuristics after 10 minutes (left) and 50 minutes (right). For each acceptance, two consecutive boxes denote ADHS and SR 135

5.10 The number of iterations spent on each heuristic set for the NRP by choosing heuristics in a uniformly random manner 137

5.11 Number of new best solutions over iteration charts on each heuristic set for the NRP (SR-GD on *long_late*02, 1 hour test results were used) 138

5.12 The significantly best hyper-heuristics on each heuristic set for the NRP after 10 minutes (left) and 1 hour (right) (Consecutive elements on the x axis refer to ADHS and SR respectively) . . 138

5.13 Average ranking of the hyper-heuristics on the NRP after 10 minutes. Each graph represents the results obtained with the heuristic set. They are ordered from left to right, top to bottom: $HS_1 \rightarrow HS_9$ 139

5.14 Average ranking of the hyper-heuristics on the NRP after 1 hour. Each graph represents the results obtained with the heuristic set. They are ordered from left to right, top to bottom: $HS_1 \rightarrow HS_9$ 140

5.15 Box plot for average ranks of the hyper-heuristics on the nurse rostering problem after 10 minutes (left) and 1 hour (right). For each acceptance, two consecutive boxes denote ADHS and SR . . 141

5.16 The number of iterations spent on each heuristic set for the PASP by choosing heuristics in a uniformly random manner 143

5.17 Number of new best solutions per spent iterations charts on each heuristic set for the PASP. ADHS-AILLA on $Inst3$, 50 minutes test results were used 144

5.18 The significantly best hyper-heuristics on each heuristic set for the PASP after 10 minutes (left) and 50 minutes (right). Consecutive elements on the x axis refer to ADHS and SR respectively . . . 145

5.19 Average ranking of the hyper-heuristics on the PASP after 10 minutes. Each graph represents the results obtained on a heuristic set. They are ordered from left to right, top to bottom: $HS_1 \rightarrow HS_9$ 146

5.20 Average ranking of the hyper-heuristics on the PASP after 50 minutes. Each graph represents the results obtained on a heuristic set. They are ordered from left to right, top to bottom: $HS_1 \rightarrow HS_9$ 147

5.21 Box plot for average ranks of the hyper-heuristics on the PASP after 10 minutes (left) and 50 minutes (right). For each acceptance, two consecutive boxes belong to ADHS and SR . . 148

6.1 Generalised intelligent hyper-heuristic (GIHH) 156

6.2 The number of calls for each heuristic on each problem domain by ADHS-AILLA with 10 minutes of execution time (a run on one sample instance represents each domain) 180

6.3 The heuristic set size changes on each problem domain by ADHS-AILLA with 1 hour of execution time (a run on one sample instance representing each domain was used) 181

6.4 QI changes of the bin packing heuristics by ADHS-AILLA with 10 minutes of execution time (the red dotted lines show avg) . 183

6.5 Heuristic pairs yielding new best solutions by relay hybridisation using ADHS-AILLA for 1 hour of execution time (squares show the first applied heuristics, circles indicate the heuristics applied afterwards) 184

6.6 Number of iterations ratio between relay hybridisation and single heuristic selection (relay/single) 186

6.7 Iteration limit (k) changes on each problem domain (a run on one sample instance representing each domain was used) 187

6.8 Changes on the best fitness found by ADHS-AILLA on each problem domain 189

7.1 Comparison between the average speed of the low-level heuristics on each problem domain (from black to white in counter-clockwise direction: $LLH_0 \rightarrow LLH_{11}$) 195

7.2 Average operator speed for each problem domain 196

7.3 Solutions visited in the course of the search process for each problem 197

7.4 Number of iterations spent per time for each heuristic on each problem domain (each graph belongs to one problem domain ordered as HCSP, SPRR, MPSP from top to bottom) 198

7.5 Number of new best and improving solutions found by each heurstic for each problem domain (each graph belongs to one problem domain ordered as HCSP, SPRR, MPSP from top to bottom) 199

7.6 Heuristic pairs found new best solutions (squares and circles represent the consecutively applied heuristics respectively) 201

List of Tables

1.1 The problems studied in the dissertation 7

2.1 Meta-heuristics used within/as hyper-heuristics 13
2.2 Learning in heuristic selection 14
2.3 Learning in heuristic selection (Table 2.2 continues) 15
2.4 Learning in heuristic generation 25
2.5 Move acceptance strategies 40
2.6 Scheduling datasets solved by hyper-heuristics 41
2.7 Scheduling datasets solved by hyper-heuristics (Table 2.6 continues) 42
2.8 Timetabling problem datasets solved by hyper-heuristics 43
2.9 Routing problem datasets solved by hyper-heuristics 44
2.10 Cutting and packing problem datasets solved by hyper-heuristics 45
2.11 Decision problem datasets solved by hyper-heuristics 46
2.12 Problems solved in a multi-objective fashion 46

3.1 The tested problem domains (RW: real-world, AB: academic benchmarks) 47
3.2 The distance matrix of instance NL8 as an example 49
3.3 An optimal solution for NL8 49

3.4 The home care scheduling problem instances (*hh*, *ll2*, *ll3*) were taken from [183]. The other instances (*hh1*, *ll11*, *ll21*) are the modified versions of the original instances in [183] 54

3.5 Workforce related constraints in the SPRR 56

3.6 Overview of problem characteristics 57

3.7 The security personnel routing and rostering problem instances 58

3.8 Workforce related constraints in the MPSP 59

3.9 The maintenance personnel scheduling problem instances . . . 59

3.10 The details of the RMC instances 64

3.11 The patient admission scheduling problem instances 66

3.12 The nurse rostering problem instances with 50 nurses 69

3.13 Bin packing instances [178] (the column "Best" shows the number of bins used. The solutions with * are optimal) 72

3.14 Max SAT instances [178] (the solutions with $*$ are optimal and > 0 shows the corresponding solutions with a fitness value larger than zero) 73

3.15 Permutation flowshop scheduling instances [178] 73

3.16 Personnel scheduling instances [178] 74

3.17 Travelling salesman instances [178] 74

3.18 Vehicle routing problem instances [178] 75

4.1 Contributions of the chapter 78

4.2 Results of the SR hyper-heuristic for the NL instances (˜0 means almost zero) 85

4.3 The best results among the L_R-I and LR_R-I (L_R-I + Restarting) hyper-heuristics for the NL instances (In the λ line; **B**: Both L_R-I & LR_R-I, **R**: LR_R-I), (˜0 means almost zero) 85

4.4 Results of the SR hyper-heuristic for the Super instances 85

4.5 The best results among the L_R-I and LR_R-I (L_R-I + Restarting) hyper-heuristics for the Super instances (In the λ line; **B**: Both L_R-I & LR_R-I, **R**: LR_R-I) 86

4.6 Ranking corresponding to the average results among the hyper-heuristics with L_R-I, LR_R-I (L_R-I + Restarting) and SR heuristic selection mechanisms for the NL instances (lower rank values are better) 87

4.7 Ranking corresponding to the average results among the hyper-heuristics with L_R-I, LR_R-I (L_R-I + Restarting) and SR heuristic selection mechanisms for the Super instances (lower rank values are better) 88

4.8 T-Test results for the learning hyper-heuristics and the SR based hyper-heuristic on the NL and Super instances 88

4.9 Comparison between the best results obtained by the learning automata hyper-heuristics (LAHHs) and the state of the art results (LB: Lower Bound) 89

4.10 TTP solution methods compared, adapted from [98] 90

4.11 The tested hyper-heuristics on the patient admission scheduling problem (LATE(200) refers to late acceptance with a fixed list size 200, LATE(Var) refers to late acceptance tested with different list size values) 93

4.12 The average best results for the hyper-heuristics (the best results are in bold) 97

4.13 The best results for the hyper-heuristics (the best results are in bold) 98

4.14 The average ranking of the average best results 98

4.15 The average ranking for the best results 99

4.16 Average fitness values with ranks of different hyper-heuristics over 6 HCSP benchmark instances 106

4.17 Average fitness values with ranks of different hyper-heuristics over 6 HCSP benchmark instances 107

4.18 The Average Best Results for the HCSP instances 108

4.19 The Best Results for the HCSP instances 109

4.20 SR-AILTA with $R_1 = 1.003$ $R_n = 1.01$ with $ph_{iter} = 2500$ and $d = \{1, 2, 3, 4, 5, 6\}$ 110

4.21 Ranking of the hyper-heuristics based on their average performance 113

4.22 Average fitness values with standard deviations for each hyper-heuristic 114

4.23 Average fitness values with standard deviations for the top two hyper-heuristics with 1 hour of execution time (improvement difference as percentage between 10 minutes and 1 hour of execution time shown under %(10m-1h)) 116

5.1 Heuristic sets used for the experiments (n refers to the number of parametric heuristics used) 129

5.2 The best results on the HCSP instances (The best results are from [232]) 135

5.3 The performance of different neighbourhood sampling rules using ADHS-AILLA on the HCSP. $\approx$ denotes similar performance, $\gg$ refers to significantly better performance 136

5.4 The best results on the NRP instances. The best known results are from [216] and each of these best known solutions was found after around 10-14 hours of a run. # means that several hyper-heuristics found the solutions with the given quality 142

5.5 The performance of different neighbourhood sampling rules using SR-GD on the NRP. $\approx$ denotes similar performance, $\gg$ refers to significantly better performance 142

5.6 The best known results and the best known hyper-heuristic results on the PASP instances are from [94] and [46] 148

5.7 The performance of different neighbourhood sampling rules using ADHS-AILLA on the PASP. $\approx$ denotes similar performance, $\gg$ refers to significantly better performance 149

6.1 Learning rates used for the adaptation of the heuristics' parameters (Algorithm 6) 165

6.2 The experimental results with 10 minutes of execution time (Re-initialisation is available only for ADHS-AILLA) 171

6.3 The experimental results with 1 hour of execution time (Reinitialisation is available only for ADHS-AILLA) 172

6.4 The experimental results by SR hyper-heuristics with 1 hour of execution time 173

6.5 The scores of the tested hyper-heuristics after 10 minutes of execution based on the CHeSC scoring system (the highest possible score is 680). The results of SR hyper-heuristics except SR-AILLA were taken from [236] 175

6.6 The scores of the tested hyper-heuristics after 1 hour of execution based on the CHeSC scoring system (the highest possible score is 680) 176

6.7 the % difference between the best known solutions and the best solutions found by ADHS-AILLA after 1 hour 178

6.8 The % performance difference between 10 minutes and 1 hour cases based on their average results by ADHS-AILLA 178

6.9 CHeSC 2011 competition scores (Algorithms are sorted from the best to the worst based on their overall scores) 190

Chapter 1

Introduction

Problem solving is an ordinary process in our daily lives. An effective way, a solution strategy, is being looked for to solve a problem. The effectiveness of a solution strategy should be measured with respect to the requirements and expectations from a solution. For instance, if a solution is required urgently, the speed of a solution strategy can be considered the primary measure for its effectiveness. While solving a problem, a solution strategy needs to take into account the goals called ***objectives*** in this work. A problem's objective may be determining whether there exists an answer for a particular instance, for example, whether or not it is possible to travel from location A to location B within 30 minutes using the available transportation opportunities. This type of questions are called ***decision*** problems. Besides decision problems, searching a solution among many for a problem is studied under ***optimisation*** problems. It can be exemplified as what is the fastest way for travelling between location A and location B. The answer to this question is described as an ***optimal*** solution. ***Exact*** algorithms such as linear/integer programming methods guarantee optimal solutions. However, finding an optimal solution is extremely hard for many optimisation problems. Therefore, near-optimal solutions are usually acceptable rather than finding an optimal solution. A ***fitness function*** measures the quality of the solutions in terms of the problem objectives. Additionally, it may be an issue for exact algorithms to quickly return a solution. ***Heuristic*** strategies, which are rules of thumbs for discovering good quality solutions in reasonable time, speed up a search process. Although they cannot provide any optimality guarantee, innumerable studies already showed their effectiveness for finding near-optimal solutions in a fast manner.

Meta-heuristics [259] are generalised heuristic methods. There are several

meta-heuristic methods such as tabu search [160, 161], applied to a vast range of problem domains. Most of the meta-heuristics have been designed based on certain existing processes and events from nature. Simulated annealing [200], for example, is a meta-heuristic based on the idea of heating and cooling processes for shaping solid materials. Ant colony optimisation [124] simulates the ants' behaviour for finding food.

Despite their general applicability characteristics, meta-heuristics have often been tailored to the target problem domains. Therefore, it is a time-consuming process to transform a meta-heuristic implemented so that it can be applied to another problem. Algorithms perform differently on distinct problems and distinct instances of one problem. *Algorithm selection* [290] strategies have been studied to effectively solve different problem instances by choosing the best algorithm among multiple options. Whatever the usefulness of such approaches, they miss the possible improvement opportunities due to varying algorithm performance during a search process. ***Hyper-heuristics*** follow a deeper selection approach by managing a number of low-level search strategies during a search. The motivation of hyper-heuristics is to take advantage of the strengths of different algorithms [62].

1.1 Hyper-heuristics

According to the initial hyper-heuristic definitions, a hyper-heuristic is described as a method operating across a set of given low-level heuristics for a problem. In other words, they perform by selecting appropriate heuristics for solving a particular problem. This definition was extended by automatically building low-level heuristics rather than using the existing ones. These hyper-heuristic types are called *selection hyper-heuristics* and *generation hyper-heuristics* respectively. Furthermore, hyper-heuristics have been sub-categorised with respect to the type of the provided feedback mechanisms and the nature of the heuristic search space [66]. Three categories of feedback mechanisms are considered, namely hyper-heuristics with *online learning*, those with *offline learning* and finally those implying *no learning* at all. The learning process takes place during the search process in online learning. Choice function [105] and reinforcement learning [245, 267] are widely used examples of online learning. In contrast, offline learning refers to learning before the actual search starts. Particularly case based reasoning [78] and learning classifier systems [219, 293] work in an offline manner. In addition, some hyper-heuristic components without any learning device are available. One such example is the simple random heuristic selection mechanism [105]. Additionally, hyper-heuristics are categorised with respect to the nature of the given heuristics. Both *selection hyper-heuristics* and

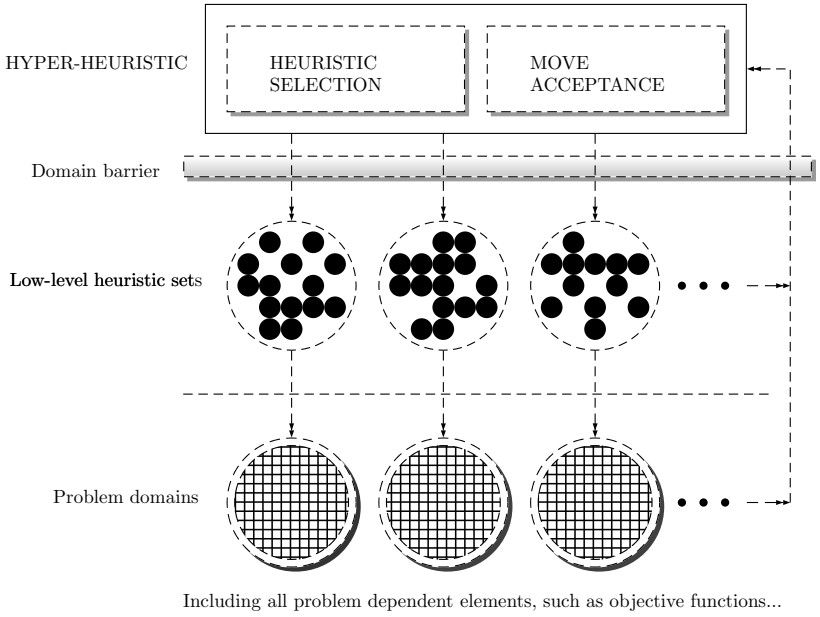

Figure 1.1: A traditional selection hyper-heuristic framework [227]

generation hyper-heuristics are considered in the context. The selection hyper-heuristic designs usually contain meta-heuristic components such as tabu search [73], genetic algorithms [168], simulated annealing [127, 71] and ant colony optimisation [69]. Genetic programming is the usually preferred approach for the heuristic generation process [63, 67, 153, 25, 68].

The type of low-level heuristics are also mentioned for further categorisation. The available options for this sub-category are *constructive* and *perturbative* heuristics. On one hand, constructive heuristics are used to build complete solutions via partially constructing solutions using different heuristics. On the other hand, perturbative heuristics make changes to the existing solutions. These operations are usually performed on the complete solutions.

Selection hyper-heuristics with *online learning* capabilities that are searching over the *perturbative* low-level heuristics are the subject of the present research. A traditional selection hyper-heuristic framework is depicted in Figure 1.1. In this framework, a hyper-heuristic is composed of two sub-mechanisms, namely *heuristic selection* and *move acceptance*. The hyper-heuristic search process starts by the selection mechanism for choosing a heuristic and then applying it to a solution. Afterwards, some problem-independent information is sent to the

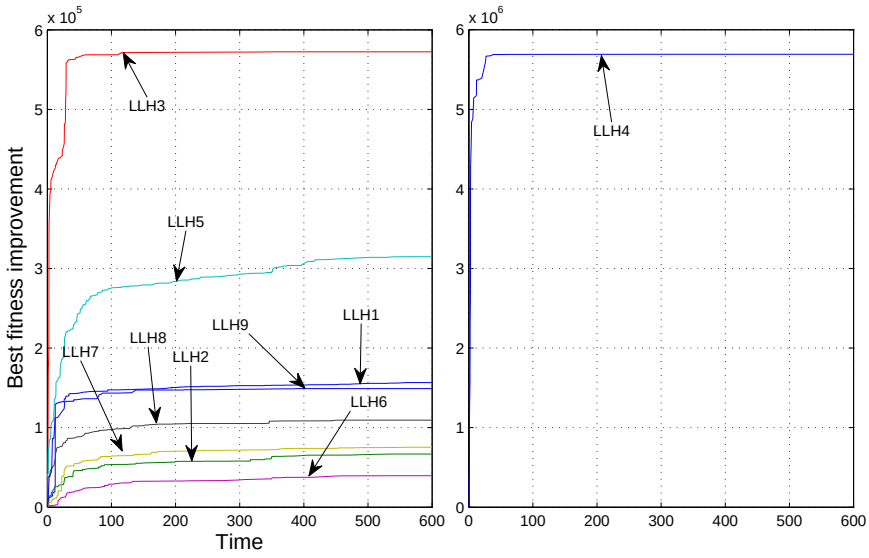

Figure 1.2: Best fitness improvement over time

move acceptance criterion to decide whether to accept or reject the new solution. The solution quality is widely used for assessing the acceptability of a solution. For performing these consecutive procedures, a set of low-level heuristics should be available. Each heuristic can be applied to the current solution of the target problem. The duty of a selection hyper-heuristic here is to determine a strategy for choosing the most appropriate heuristics during different phases of a search process of the search space as well as the evaluation of these decisions rather than directly solving a problem instance.

Figure 1.2 illustrates the advantage of selection hyper-heuristics when solving a complex problem from practice. It shows the performance of 9 low-level heuristics for the ready-mixed concrete delivery problem with respect to their best fitness improvement capabilities. The graph on the left explains the behaviour of the 8 heuristics for the aforementioned performance criterion. The graph on the right presents the heuristic (LLH_4) with the highest improvement over the best fitness solutions found during the search. Although LLH_4 provides the highest improvement, it is incapable of improving the solutions after a short time period. Some of the other heuristics, such as LLH_5, continue the improvement process throughout the search. However, when these heuristics are used separately, it appears impossible to find solutions with a quality similar to what can be obtained when combining all the heuristics. Hence, using multiple algorithms together is expected to perform better than using each one individually.

Due to maintaining multiple low-level heuristics and due to hyper-heuristics' problem-independent characteristics, ***generality*** is considered the key concept behind hyper-heuristics. The idea is to design a high-level approach, intelligent enough to determine the strengths and weaknesses of a set of low-level heuristics [61]. The resulting approach is expected to efficiently solve different computationally hard problems. However, there is no strict definition of generality in the context of hyper-heuristics. Therefore, a hyper-heuristic strategy addressing various generality related issues is not available.

> The ***aim*** of this study is to design a hyper-heuristic that can deliver effective performance across different heuristic sets while solving distinct problems.

1.2 Contributions

The present research investigates various aspects of selection hyper-heuristics, addresses several generality related issues and discusses their advantages and disadvantages. More specifically, new heuristic selection mechanisms with heuristic exclusion strategies, mostly supported by certain learning devices, are introduced. These methods additionally involve online parameter adaptation methods and heuristic hybridisation approaches as new mentoring elements. Likewise, new acceptance criteria are developed. The hyper-heuristics using these sub-mechanisms are additionally supported by some restarting and re-initialisation strategies. Furthermore, these new methods and existing hyper-heuristic strategies are used to measure the generality level of selection hyper-heuristics.

New heuristic selection mechanisms: The heuristic selection mechanisms devised in the dissertation involve two types of methods. The first type considers the pure heuristic selection strategies. A learning automaton was employed to determine online selection probabilities of given low-level heuristics. Another method dividing the search process into phases was introduced. It considers learning automata and simple random for choosing heuristics during different phases. This means that it combines a learning based selection strategy and a naive approach selecting heuristics randomly. Additonally, a generalised version of the random descent heuristic selection mechanism, i.e. random descent iteration, was proposed. Besides these heuristic selection mechanisms, the dynamic heuristic set strategy and its adaptive version perform a heuristic exclusion procedure before the selection operation. The approaches from this part of the research resulted in publications of the following subjects.

- Learning automata [240, 241, 233, 235, 236, 227]
- Learning automata with simple random [226]
- Random descent iteration [226]
- Dynamic heuristic set [232, 325, 46]
- Adaptive dynamic heuristic set [233, 228, 235, 236]

New mentoring strategies: Mentoring strategies have been studied to use low-level heuristics in a more efficient way. Adapting the parameters of the parametric heuristics and determining effective heuristic hybridisations are the two main mentoring approaches. A learning automaton based parameter adaptation strategy was introduced for specifying the number of neighbouring solutions to be visited. Additionally, the parameters affecting the number of steps performed by a heuristic per iteration were adapted using a generic reward-penalty technique. Finally, a relay hybridisation algorithm was developed to identify proficient heuristic pairs. The developed mentoring approaches were introduced in the following publications.

- Parameter adaptation
  - Learning automata [226]
  - A reward-penalty strategy [235, 230]
- Relay hybridisation [233, 235, 236]

New move acceptance criteria: The move acceptance mechanisms determine the intensification-diversification behaviour of hyper-heuristics. Threshold acceptance approaches use threshold values to decide on the acceptability of the visited solutions. In the present dissertation, a simple and novel acceptance criterion, i.e. iteration limited threshold accepting, was developed. This idea has been improved by employing an adaptive strategy in correspondence with the hardness of finding new best solutions. Then, the adaptiveness of this new method was elevated by adaptive threshold values. For this purpose, the quality of recently visited solutions is used as threshold values. The following list of publications cover the new acceptance mechanisms.

- Iteration limited threshold accepting [240, 241]
- Adaptive iteration limited threshold accepting [237, 232]
- Adaptive iteration limited list-based threshold accepting [233, 229, 228, 235, 236, 227]

An investigation on the generality level of selection hyper-heuristics: All the aforementioned hyper-heuristic sub-mechanisms were combined as selection hyper-heuristics and tested across various combinatorial optimisation problems. These problem domains consist of common problems, such as scheduling, timetabling, routing, rostering, packing problems. In addition, some structured problems were addressed, i.e. problems with characteristics of more than one of the traditional optimisation problems. The problems and publications covering the experimental results produced are listed in Table 1.1.

Problems	**Published results**
Home care scheduling	Mısır et al, 2009[237]; Mısır et al, 2010[232]; Mısır et al, *under review*[227]
Security personnel routing and rostering	Smet et al, 2011[306]; Mısır et al, 2011[228], Mısır et al, *under review*[227]
Maintenance personnel scheduling	Mısır et al, *under review*[227]
Ready-mixed concrete delivery	Vancroonenburg et al, 2011[326]; Mısır et al, 2011[229]
Travelling tournament	Mısır et al, 2009[240]; Mısır et al, *to appear*[241]
Patient admission scheduling	Vancroonenburg et al, 2010[325]; Bilgin et al, 2010[47]; Bilgin et al, 2012[46]; Mısır et al, *under review*[231]
Maximum satisfiability	Mısır et al, 2011a,b[233, 239]; Mısır et al, *to appear*[236]; Mısır et al, 2012a,b,c[234, 235, 230]
Bin packing	Mısır et al, 2011a,b[233, 239]; Mısır et al, *to appear*[236]; Mısır et al, 2012a,b,c[234, 235, 230]
Nurse rostering	Bilgin et al, 2010[47]; Mısır et al, 2011a,b[233, 239]; Mısır et al, *to appear*[236]; Mısır et al, 2012a,b,c[234, 235, 230]
Permutation flowshop scheduling	Mısır et al, 2011a,b[233, 239]; Mısır et al, *to appear*[236]; Mısır et al, 2012a,b,c[234, 235, 230]
Travelling salesman	Mısır et al, *to appear*[236]; Mısır et al, 2012a,b,c[234, 235, 230]
Vehicle routing	Mısır et al, *to appear*[236]; Mısır et al, 2012a,b,c[234, 235, 230]

Table 1.1: The problems studied in the dissertation

Additionally, different heuristic sets with varying speed and improvement characteristics were used. Especially the study presented in Chapter 4, employed a set of hyper-heuristics using distinct heuristic sets with different execution time limits. The computational results were analysed and the generality level of the tested hyper-heuristics was investigated.

A complete hyper-heuristic framework for generality: The experience earned while developing the above ideas is used to design an intelligent hyper-heuristic [235] for the purpose of generality. A detailed analysis on the performance of this hyper-heuristic and on the adaptiveness of the hyper-heuristic sub-components is presented. This hyper-heuristic won the first international cross domain heuristic search challenge (CHeSC) 2011. The winning hyper-heuristic is additionally used as a heuristic performance analysis tool on a number of problem domains with the same heuristic set.

1.3 Structure of the dissertation

Chapter 2 presents a detailed literature survey on hyper-heuristics. The heuristic selection mechanisms and move acceptance criteria reported in these papers on hyper-heuristics are foregrounded. The work on generation hyper-heuristics is explained as well as hyper-heuristics executed in distributed environments. After that, the comparison and analysis studies on hyper-heuristics are discussed. Finally, the hyper-heuristic problem domains together with the datasets used for the experiments are listed.

Chapter 3 exhibits the problem domains considered. Each problem is briefly defined. The low-level heuristics used for solving these problems are described.

Chapter 4 discusses a number of new hyper-heuristics with some learning devices and adaptive components. The performance and behaviour analysis of the introduced hyper-heuristics is investigated on different, arbitrarily selected, combinatorial optimisation problems. The first hyper-heuristic involves a reinforcement learning selection strategy as well as a new acceptance criterion. The performance and the behaviour of the proposed approach is investigated on the travelling tournament problem. The next application is the home care scheduling problem. A hyper-heuristic using a temporary heuristic exclusion procedure together with an adaptive version of the previous acceptance method is discussed in detail. Hyper-heuristics using different sub-mechanisms with learning and adaptive behaviour are then proposed. They are tested on the patient admission scheduling problem. The last method considers a highly search space dependent move acceptance mechanism together with an adaptive version of the previous heuristic exclusion approach. This hyper-heuristic is tested on the ready-mixed concrete delivery problem. The tested problems are arbitrarily selected.

Chapter 5 reports a study on the generality level of selection hyper-heuristics. For this purpose, a large set of hyper-heuristics combining various selection and acceptance mechanisms are used. A number of heuristic sets are generated. The

purpose is to measure the performance of these hyper-heuristics on different heuristic sets. Three related problem domains have been selected for experiments. All these domains exhibit rostering and routing components, thus belong to the category of structured problems. The set consists of home care scheduling, nurse rostering and patient admission scheduling problems. The generality of the hyper-heuristics and the changes on the tested hyper-heuristics' performance are discussed.

Chapter 6 explains the final hyper-heuristic design. The design principles for the purpose of generality are presented. A group of hyper-heuristic sub-mechanisms based on these principles are shown. Subsequently, these sub-mechanisms together with certain control mechanisms are integrated into hyper-heuristics. A detailed empirical analysis on a high-level search framework, i.e. HyFlex, is exhibited.

Chapter 7 demonstrates a novel strategy using the hyper-heuristic presented in Chapter 5 as an analysis tool. Three related problem domains including the home care scheduling, security personnel routing and rostering and maintenance personnel scheduling problems, used for the experiments are explained. A general heuristic set is developed. The requirements for solving these problems are analysed and performance of the same low-level heuristics on the different problems sharing certain common characteristics is shown.

Chapter 8, finally, provides a general summary of the dissertation as well as some design recommendations and directions for future research.

Chapter 2

Literature Survey

This chapter presents a detailed survey on hyper-heuristics. The structure is mainly based on the distinction in the hyper-heuristics' selection and generation characteristics [61]. Different running strategies including hyper-heuristics performing sequentially and in parallel, are presented. In addition, the problem domains used to measure the hyper-heuristics' performance are discussed.

2.1 Selection strategies in hyper-heuristics

Selection hyper-heuristics aim at choosing the most appropriate heuristics for different parts of the search space in a problem-independent manner. Numerous approaches were introduced in order to perform this selection task. This section presents the studies focusing on the heuristic selection process.

Table 2.1 lists the meta-heuristics used as hyper-heuristics or embedded into hyper-heuristics. Among these methods, there are single-point search techniques such as tabu search, simulated annealing, variable neighbourhood search, iterated local search and greedy randomized adaptive search procedure, as well as population-based approaches including evolutionary algorithms, ant colony optimisation, particle swarm optimisation and scatter search.

Table 2.2 and Table 2.3 provide an overview of the heuristic selection mechanisms with different learning characteristics. Among them, simple random is a naive, simple but effective selection method used in several studies even though it does not accommodate any learning components. There are also learning based methods with short-term memory such as greedy, random descent and

random permutation descent. Additionally, relatively complex learning methods for the selection process were introduced. Choice function is a widely used example from this category of the selection approaches. Reward-based scoring methods also perform online learning with respect to the performance of the low-level heuristics. In addition, certain evolutionary algorithms such as genetic algorithms and genetic programming have been considered heuristic selection mechanisms. A popular swarm intelligence algorithm, i.e. ant colony optimisation, has been implemented in different ways for better hyper-heuristic performance. Besides all these methods, algorithms with offline learning characteristics like case-based reasoning and learning classifier systems, have been studied to train or devise high-level selection strategies.

2.1.1 Evolutionary algorithms

Evolutionary approaches have been successfully applied to numerous optimisation problems. Due to their effective performance, they have been additionally employed as high-level search strategies. In [143], a genetic algorithm based algorithm called a hybrid GA/heuristic was proposed to solve the open-shop scheduling problem. The heuristic set consisted of 8 low-level heuristics. In each chromosome, one heuristic and one task from a particular job were encoded next to each other. Each heuristic was used to schedule the task placed next to. The approach showed better performance than scheduling the tasks using only one predetermined heuristic. It also yielded new best solutions for the tested instances.

Effect of online learning

Adaptation of hyper-heuristics online is useful to deliver better performance due to the differentiating nature of search spaces. This means that it is important to behave differently while searching on distinct search regions. Thus, any adaptive hyper-heuristic component designed appropriately promises superior performance. The following studies present the effect of adaptation mechanisms improving the performance of genetic algorithms based hyper-heuristics.

The performance of another GA-based hyper-heuristic called hyper-GA was investigated on the trainer scheduling problem in [104]. During the experiments, two different representations, direct and indirect GA, were used. In the direct representation, each gene was used as a trainer with his/her details in a chromosome. Also, this direct representation was used within memetic algorithms (MAs) [243] by applying local search after GA operations, namely crossover and mutation. On the other hand, each gene belonging to the indirect

Meta-heuristics	**References**
Tabu search	Burke et al, 2003[73]; Burke and Soubeiga, 2003[84]; Han and Kendall, 2003[168]; Burke et al, 2005[83]; Kendall and Hussin, 2005[191]; Hussin, 2005[177]; Burke et al, 2007[76]; Burke et al, 2008[29]; Qu and Burke, 2009[280]; Remde et al, 2009[288]; Keleş et al, 2010[187]; Keleş et al, 2010[186]
Simulated annealing	Bai and Kendall, 2005[31]; Anagnostopoulos et al, 2006[14]; Dowsland et al, 2007[127]; Bai et al, 2007[27]; Burke et al, 2008[70]; Anagnostopoulos et al, 2006[15]; Bai et al, 2012[26]; Bilgin et al, 2012[46]
Variable neighbourhood search	Qu and Burke, 2005[279]; Remde et al, 2007[286]; Qu and Burke, 2009[280]; Hsiao et al, 2012[176]
Iterated local search	Qu and Burke, 2009[280]; Burke et al, 2010[57], Ochoa et al, 2012[258], Walker et al, 2012[331]
Ant colony optimisation	Burke et al, 2003a,b[72, 59]; Burke et al, 2005[69]; Cuesta-Cañada et al, 2005[114]; Chen et al, 2007[98]; O'Brien, 2007[254]; Burke et al, 2007[79]; Aziz and Kendall, 2009[20]; Keleş et al, 2010[187]; Ergin et al, 2011a,b[138, 139]; Hernandez et al, 2011[174]; Ren et al, 2011[289]; Khamassi et al, 2011a,b[196, 195]; Nunez and Ceballos, 2011[252]
Genetic algorithms	Fang et al, 1994[143]; Cowling et al, 2002a,b[103, 104]; Han and Kendall, 2003a,b[167, 168]; Ross et al, 2003[291]; Ochoa et al, 2009[257]; Fatima and Bader-El-Den, 2010[144]
Genetic programming	Burke et al, 2006[63]; Burke et al, 2007[67]; Bader-El-Den and Poli, 2008a,b[217, 22]; Fukunaga, 2008[153]; Keller and Poli, 2008a,b,c[188, 189, 190]; Bader-El-Den and Poli, 2009[23]; Bader-El-Den et al, 2009[25]; Allen et al, 2009[12]; Tolay and Kumar, 2009[324]; Burke et al, 2009[65]; Burke et al, 2010[68]; Fatima and Bader-El-Den, 2010[144]; Hauptman et al, 2010[171]; Abednego and Hendratmo, 2011[3]; Koohestani and Poli, 2011[202]; Nguyen et al, 2011[250], Drake et al, 2012[130]
Greedy randomized adaptive search procedure	Jiang et al, 2011[181]; Anagnostopoulos and Koulinas, 2011[16]
Particle swarm optimisation	Ahmed et al, 2011[5], Alinia Ahandani et al, *to appear*[11]
Scatter search	Cano-Belmán et al, 2009[89]

Table 2.1: Meta-heuristics used within/as hyper-heuristics

representation, represented a heuristic among 12 heuristics. Heuristics residing in a chromosome were applied consecutively. The performance of hyper-GA was superior in comparison to the GA and MA with direct encoding, each low-level

Heuristic Selection	Type	References
Simple random	No learning	Cowling et al, 2001[105], Cowling et al, 2002[108], Ayob and Kendall, 2003[19], Kendall and Mohamad, 2004a,b[193, 194], Bai and Kendall, 2005[31], Anagnostopoulos et al, 2006[14], Bilgin et al, 2006[49], Remde et al, 2007[286], Burke et al, 2008[77], Ouelhadj and Petrovic, 2008[260], Özcan et al, 2008[263], Bilgin et al, 2009[45], Maden et al, 2009[218], Ouelhadj and Petrovic, 2009[261], Ouelhadj et al, 2009[262], Özcan et al, 2009[264], Grobler et al, 2010[166], Kiraz and Topçuoğlu, 2010[198], Berberoğlu and Uyar, 2011[41], Kalender et al, 2012[184], Wauters et al, 2012[336]
Greedy	Online	Cowling et al, 2001[105], Bilgin et al, 2006[49], Remde et al, 2007[286], Özcan et al, 2008[263], Maden et al, 2009[218], Berberoğlu and Uyar, 2011[41], Özcan and Kheiri, 2011[265], Kalender et al, 2012[184]
Greedy gradient	Online	Kalender et al, 2012[184]
Random descent	Online	Cowling et al, 2001[105], Özcan et al, 2008[263], Maden et al, 2009[218], Özcan et al, 2009[264], Kiraz and Topçuoğlu, 2010[198], Berberoğlu and Uyar, 2011[41]
Random permutation	No learning	Cowling et al, 2001[105], Özcan et al, 2008[263], Kiraz and Topçuoğlu, 2010[198], Berberoğlu and Uyar, 2011[41]
Random permutation descent	Online	Cowling et al, 2001[105], Özcan et al, 2008[263], Kiraz and Topçuoğlu, 2010[198], Berberoğlu and Uyar, 2011[41]

Table 2.2: Learning in heuristic selection

heuristic and some greedy heuristics.

hyper-GA was extended by investigating adaptive length chromosomes, i.e. ALChyper-GA, in [103]. The approach was proposed to provide a more effective heuristic list. Therefore, effective heuristics were added while poor heuristics were removed from chromosomes. In addition, the most optimal length for chromosomes was aimed to be determined. Adding and removing operations were mostly used for a set of genes or heuristics. Also, a penalty function was used to prevent lengthy chromosomes. The empirical results demonstrated that ALChyper-GA was better than its predecessor, i.e. hyper-GA, for most of the problem instances. Moreover, it has been concluded that hyper-GA with a priori information concerning the optimal chromosome length is expected to

Heuristic Selection	Type	References
Choice function	Online	Cowling and Soubeiga, 2000[109], Cowling et al, 2001a,b[105, 106], Cowling et al, 2002a,b[107, 108], Rattadilok et al, 2004[283], Bai and Kendall, 2005[31], Rattadilok et al, 2005[284], Bilgin et al, 2006[49], Özcan et al, 2008[263], Crawford et al, 2009[112], Özcan et al, 2009[264], Gibbs et al, 2010[159], Kiraz and Topçuoğlu, 2010[198], Crawford et al, 2011[113], Berberoğlu and Uyar, 2011[41], Blazewicz et al, 2011[51], Drake et al, 2011[128], Kiraz et al, 2011[199], Bilgin et al, 2012[46], Kalender et al, 2012[184], Drake et al, 2012[129]
Case-based reasoning	Offline	Burke et al, 2002[75], Petrovic and Qu, 2002[271], Burke et al, 2006[78]
Reinforcement learning	Online	Nareyek, 2003[245], Bai et al, 2007[28], Özcan et al, 2009[264], Gibbs et al, 2010[159], Özcan et al, 2010[267], Obit et al, 2011[253], Sin, 2011[304], Bai et al, 2012[26], Di Gaspero and Urli, 2012[123], Remde et al, 2012[287], Sin and Kham, 2012[305]
Genetic algorithm	Online	Cowling et al, 2002a,b[104, 103], Han and Kendall, 2003[167, 168]
	Offline	Ross et al, 2003[291]
Tabu search	Online	Burke et al, 2003a,b[73, 87], Burke and Soubeiga, 2003[84], Burke et al, 2005[83], Hussin, 2005[177], Kendall and Hussin, 2005[191], Bai et al, 2007[29], Burke et ak, 2007[76], Dowsland et al, 2007[127], Qu and Burke, 2009[280], Remde et al, 2009[288], Keleş et al, 2010a,b[187, 186], Blazewicz et al, 2011[51]
Genetic programming	Online	Nguyen et al, 2011[250]
Ant colony optimisation	Online	Cuesta-Cañada et al, 2005[114], Burke et al, 2005[69], Chen et al, 2007[98], Keles et al, 2010[187], Hernandez et al, 2011[174], Ren et al, 2011[289], Khamassi et al, 2011[196]
Learning classifier system	Offline	Ross et al, 2002[293], Marín-Blázquez and Schulenburg, 2007[219]
Adaptive operator selection	Online	Ochoa et al, 2012[258], Walker et al, 2012[331]

Table 2.3: Learning in heuristic selection (Table 2.2 continues)

a better approach than ALChyper-GA. However, it should be noted that this information cannot be retrieved without offline tests.

The extended work called ALChyper-GA with tabu method (hyper-TGA) [168] was proposed as a new hyper-heuristic strategy. The motivation was to temporarily eliminate the poor low-level heuristics. The tabu approach was embedded by adding an extra variable maintaining the information about not

calling the indexed heuristic for a determined number of cycles into each gene. The optimal list length was determined by testing the list lengths ranging from 1 to 10. It was eventually set to 5. Results of these experiments showed that hyper-TGA produces better solutions than the previously mentioned two hyper-GA methods. Additionally, the results by the other hyper-GA based hyper-heuristic, i.e. guided hyper-GA [167], were examined as the best hyper-GA approach in the literature. The main goal was again to provide an efficient mechanism for adding and removing heuristics based on their performance in chromosomes. In order to identify the optimal chromosome length, two operations were used: 1) removing the worst-block of genes and 2) adding the best-block of genes under certain conditions. Unlike the aforementioned GA based hyper-heuristics, guided hyper-GA was performed on an additional problem domain, i.e. project presentation scheduling. The computational results were compared to the other hyper-GAs and a hyper-heuristic with choice function [108]. Guided hyper-GA performed better than these algorithms.

Effect of offline learning

Besides online learning and adaptation, there exists approaches with offline learning delivering high quality solutions for various problems. A successful application of an offline learning strategy in the context of genetic algorithms was presented in [291]. In this study, a messy-GA based hyper-heuristic was introduced and tested on the 1D bin packing problem by using four basic constructive heuristics. The representation of a chromosome consisted of a number of blocks. Each block involves the information about the proportion of items remaining to be packed and a heuristic to place these items. While applying a steady state genetic algorithm, two crossover and three mutation operators were used. The experiments were carried out with 40 chromosomes and 75% of the instances were used to train the hyper-heuristic. The results were compared to the results of each low-level heuristic by itself and some learning classifier XCS based hyper-heuristics [293, 300]. The messy-GA performed better than each individual heuristic and similar to XCS based hyper-heuristics.

Forming specialised heuristics using different problem specific components

Although there is a hyper-heuristic categorisation as selection and generation hyper-heuristics, it is possible to consider all hyper-heuristics as heuristic generation strategies. The reason is that the outcome of a selection hyper-heuristic can be described as a list of heuristics showing the heuristics should be applied during a search process. Hyper-heuristics with genetic algorithms already result in lists of heuristics accommodated in chromosomes to perturb or

construct solutions. The following studies present the usage of genetic algorithms to form effective chromosomes involving problem specific sub-components of heuristics.

Another GA-based hyper-heuristic for the 2D-regular stock cutting problem was proposed in [318]. Each fixed-length chromosome consisted of a sequence of heuristics from three types, namely selection heuristics, placement heuristics, and rotation procedures. A sequence involved a group of four elements representing one selection heuristic, one placement heuristic, one rotation method and the number of items to be placed. The tests were conducted on 28 instances from the literature and on 6 randomly generated instances. The hyper-heuristic found the optimal solutions for most of these instances. Furthermore, it yielded new best solutions for some of the instances.

In [317], a similar strategy to [291] was employed. It is based on variable-length chromosomes involving a set of blocks referring to the percentage of pieces to be placed in four distinct size categories. The fifth element in a block showed the total percentage of all the pieces to be packed. The sixth element denotes one selection-placement heuristic combination out of 40. Two crossover and three mutation operators were used to manipulate and produce new chromosomes. After running the proposed strategy on a set of training instances, the best state-action pairs were determined. The empirical results demonstrated that the devised method performed better than the best single heuristic for each tested instance.

Effect of parallelisation

Parallelisation of algorithms have been studied to fasten search processes by using more CPU power. In [212], a memetic algorithm was executed in a parallel hyper-heuristic island-based model for a 2D bin packing problem. The idea was to run multiple memetic algorithms with different configurations on distinct processors, i.e. islands. The quality of the solutions discovered after each run was considered the score of the corresponding memetic algorithm configuration. A run with a high score determined the next configuration to be executed on the idle island. 18 mutation operators using three mutation operators with different parameter values were employed. Recombination was either with or without a crossover. 36 configurations were generated with these mutation-crossover pairs. The experiments were carried out through a master island and 16 execution islands. Besides that, an asynchronous migration scheme was employed to exchange solutions between islands. The computational results demonstrated the superior performance of this island-based model compared to each of the aforementioned 36 configurations.

Fitness landscape analysis

Analysing a fitness landscape provides useful information that can be employed to improve the performance of any search and optimisation algorithm. This information can be more interesting for hyper-heuristics, since they perform search across a set of heuristics rather than a solution space. In [257], a fitness landscape analysis was performed for selection hyper-heuristics. Experiments were carried out on the hybrid flow shop scheduling problem for two objective functions. 13 dispatching rules were used as low-level heuristics to schedule jobs. A genetic algorithm with elitism was used as a hyper-heuristic to determine competent heuristic sequences. One crossover operator and one mutation operator were designated to perform changes on these sequences. Different aspects of hyper-heuristic landscapes were presented including valleys, multimodality, neutrality and positional bias.

Performance on dynamic problems

In [157], an evolutionary hyper-heuristic, EH-DVRP, was studied for the dynamic vehicle routing problem. A sequence of constructive and repairing heuristic pairs were used to build a solution. Each constructive heuristic was considered together with a constructive-improvement heuristic. The latter consists of information about the number of customers to be inserted, an ordering heuristic and an improvement heuristic. Ordering heuristics were used to order the customers before the construction process starts. The improvement heuristics were employed to improve the constructed part of the solution. The corresponding repairing heuristic placed in a sequence element was used to improve the whole solution constructed until that moment. Each of these sequences was employed in a population. The hyper-heuristic chooses and applies one operator out of four with equal probability to generate new heuristic sequences. The computational results demonstrated the competitive performance of the proposed approach with limited tuning compared to other heuristic strategies.

2.1.2 Ant colony optimisation

Ant colony optimisation is a nature-inspired algorithm simulating ants or their colonies. It is applied for finding a path for food to solve hard combinatorial optimization problems [124, 125]. One was used within a hyper-heuristic to construct good sequences of heuristics for the project presentation scheduling problem in [69]. 8 low-level heuristics were employed for the solution construction process. The ants move towards to the best vertex, referring to a heuristic.

This means that, if an ant finds a new best solution at a vertex, then all the others go to that vertex. In addition, the moves that are applied at successive iterations, were determined based on a probability value, namely *pheromone*. It indicates one particular heuristic can be applied after another for the purpose of intensification. In the experiments, hyper-heuristics with 3, 4 and 5 ants were used. Additionally, all moves and only improving move acceptance criteria were employed. The experimental results showed that the hyper-heuristic with the only improving acceptance criterion produced competitive or better results against the selection hyper-heuristics using choice function. Furthermore, the hyper-heuristic with only 3 ants was the best performing configuration.

Another ant colony optimisation based hyper-heuristic was studied and applied to the 2D bin packing problem in [114]. A number of low-level heuristics was defined, each composed of the number of items to be placed, an item rotation element, an item ordering heuristic, a bin selection heuristic and an item placement heuristic. The combination of these heuristics was considered a hyper-heuristic and 25 pheromone matrices were used as transition matrices for specifying efficient paths to follow for ants. The experiments indicated that the generated heuristic combinations usually perform better than the single heuristics. Their performance was similar performance as the meta-heuristic strategies from the literature.

A hyper-heuristic based on ant colony optimisation was applied to a set of travelling tournament problem benchmarks in [98]. The approach discovered two optimal solutions for two small benchmarks including 4 and 6 teams. On the other hand, for the benchmarks with 8, 10, 12, 14 and 16 teams, it found no optimal solution, but produced promising feasible solutions in a reasonable execution time.

Ant colony optimisation was utilised as a hyper-heuristic for routing lightpaths on the subject of the routing and wavelength assignment problem in [187]. In this setting, 5 low-level heuristics were located at the vertices and each was responsible for the routing of a particular lightpath. The proposed approach with 6 ants was compared to a tabu search based hyper-heuristic [76] and the low-level heuristics. This hyper-heuristic performed better than the tabu search based hyper-heuristic and some of the low-level heuristics with respect to the best solutions found. Its performance was also superior with respect to the spent execution time of the tested approaches. For the success on finding feasible solutions, both hyper-heuristics performed better than the low-level heuristics.

An ant based hyper-heuristic with space reduction (AHSAR) was introduced in [289]. This hyper-heuristic considers two heuristic subsets including four heuristics visiting only improving or equal quality solutions and six heuristics diversifying the search process. Each ant chooses a heuristic pair from these

two heuristic subsets. The selection process is based on a probabilistic selection strategy in connection with the heuristics' performance and speed. A subset of best ants according to their solutions' qualities was determined. The heuristics chosen by these ants were rewarded to be more frequently selected. Visited heuristics were used to construct solutions. The experimental results indicated that the developed algorithm was able to generate high quality results for the p-median problem compared to a group of meta-heuristics from the literature. The hyper-heuristic was also faster than these meta-heuristics.

A hyper-heuristic based on ant-Q learning method was proposed to solve 2D cutting stock problem using five low-level heuristics [196]. This algorithm follows a solution construction process as the aforementioned ant based hyper-heuristics. While constructing solutions, the pheromone values were updated using a reinforcement learning strategy. The update operation was performed with respect to the performance of the heuristic pairs applied at consecutive iterations. Each time all the ants constructed complete solutions, a global pheromone was updated. The devised algorithm delivered superior results than some heuristic mechanisms from the literature and similar results as two GA hyper-heuristics [318] and an ant colony approach [114].

In [138], four hyper-heuristics were introduced to solve the survivable virtual topology design problem. These hyper-heuristics were designed based on evolutionary algorithms, ant colony optimisation, adaptive iterative constructive search and simulated annealing. Evolutionary algorithms based hyper-heuristic used perturbative heuristics. Basic evolutionary operators were used to change or generate individuals. The ant colony optimisation based hyper-heuristic also performed search across a population while using constructive heuristics. The ant colony optimisation algorithm was the elitist ant system. A simulated annealing based hyper-heuristic accommodates only one solution using perturbative heuristics. The contribution of simulated annealing was decrementing the mutation probability over time. The adaptive iterative constructive search based hyper-heuristic was another single-point search algorithm using constructive heuristics. This method can be considered ant colony optimisation with only one ant. The aim of the hyper-heuristics was to determine a heuristic list, each one was responsible to choose a bidirectional lightpath in the context of the target problem. 3 low-level heuristics were employed for the heuristic set. The proposed hyper-heuristics were compared with each low-level heuristic separately and a random solution generation approach. The experimental results showed that the hyper-heuristics were capable of managing the given heuristics while providing better results. Besides, the ant colony optimisation based hyper-heuristic outperformed the other tested hyper-heuristics. In [139], a flow-deviation method [150] was used to improve the solutions generated by this ant colony optimisation based hyper-heuristic. This method focused

on balancing the traffic load on the lightpaths. The computational results demonstrated the success of this approach for finding feasible solutions.

2.1.3 Tabu search

Tabu search [160, 161] is an optimization algorithm and a local search technique used to solve combinatorial optimization problems. The logic is to prevent cyclic moves by using a tabu list maintaining forbidden moves. This approach was embedded in a selection hype-heuristic in [73]. The devised method accommodated a variable length tabu list for temporarily excluding heuristics that could not improve solutions. Additionally, a reinforcement learning based scoring method was employed to increase or decrease the score of heuristics with respect to their performance. The heuristic with the highest score was applied to improve the solution at hand and all the generated solutions were accepted. This new hyper-heuristic strategy was applied to the nurse rostering and university course timetabling problems using 9 and 6 low-level heuristics, respectively. The hyper-heuristic delivered better results than a problem-specific genetic algorithm [8] according to their performance on finding feasible solutions for the nurse rostering problem. The hyper-heuristic's performance for the quality of the solutions found was not as successful as in the case of finding feasible solutions when compared to the genetic algorithm. However, the results were still promising. For another problem domain, i.e. university course timetabling, it was compared to a random restart local search [308] and an ant algorithm [308]. The required number of iterations for finding good quality solutions was lower then the compared algorithms. For the feasibility of the solutions, the hyper-heuristic together with the ant algorithm always found feasible solutions while this was not the case for the random restart local search. In addition, for small and some medium size instances, the hyper-heuristic performed better than the ant algorithm, but its performance was poorer on some medium and large size instances.

Multi-objective optimisation

The same tabu search idea was used in a multi-objective setting for solving the office space allocation and university course timetabling problems in [83]. The tabu search based hyper-heuristic was implemented in three different ways considering multiple objectives. Single tabu random uniform performed the hyper-heuristic by choosing one objective randomly. Single tabu roulette-wheel investigated a similar method by choosing the objective element with respect to the distance of each objective value to its optimal value. Multiple tabu roulette-wheel accommodated a separate tabu list for each objective differently

than its single objective version. A hyper-heuristic choosing heuristics randomly was additionally used to show the effect of the learning components. The results on the first application domain indicated that the single tabu roulette-wheel approach performed best. In addition, the performance of this hyper-heuristic was significantly similar to a dedicated algorithm, i.e. population-based annealing [82]. For the course timetabling problem, the single tabu roulette-wheel and multiple tabu roulette-wheel approaches were the best. In comparison with the results from three algorithms presented in the single objective tabu search based hyper-heuristic study [73], the results were competitive.

2.1.4 Case-based reasoning

The key idea behind case-based reasoning [1, 208] is analysing previously generated problem-solution pairs and examining similarities between these pairs. Case-based reasoning can be explained in four main steps. The first step is *retrieve*: retrieve from a case base source cases similar to a target case. Afterwards, *reuse*: reuse the retrieved cases to solve the target case. The third step is *revise*: revise the retrieved case to solve the target case if it is necessary. Finally, *retain*: retain the revised case into the case base.

Case-based reasoning was assigned as a heuristic selection strategy for hyper-heuristics in [75, 271]. A similarity measure function was used to determine similarities between source and target cases for the course timetabling problem. The retrieved information was used to determine the best low-level heuristics among 5 for each case. A two-stage knowledge discovery technique was employed for case-based reasoning. In the first stage, certain problem features and their weights were adapted by adjusting their values, removing irrelevant features and adding new features. Only problem related cases were gathered in a case base in the latter stage. Then, the case base was trained by *Leave-One-Out* strategy that removes a source case and adds it back if the number of successful retrievals for target cases decreases. In [75], during experiments, two initial case bases, *OneSet* with 45 source cases and *TwoSet* with 90 source cases were generated for training purposes. The experimental results concluded that *TwoSet* was a better initial case base than *OneSet*, since, having more cases was useful for more information. In [271], case-based reasoning was considered with additional search mechanisms, namely tabu search and hill climbing methods. They perform a search on two different group of features. The first group involves some simple and pure features such as the number of courses. The other group consists of combinations of them. The same similarity measure function was employed. The tabu search method was more effective for determining the best feature set for case-based reasoning than the hill climbing algorithm. In [78], a hyper-heuristic based on case-based reasoning was used to solve exam

timetabling problems in addition to the university course timetabling problem. For the exam timetabling problem, 4 low-level heuristics were utilised. The proposed hyper-heuristic found better solutions than each of these low-level heuristics.

2.1.5 Variable neighbourhood search

Variable neighbourhood search is a meta-heuristic technique for providing an efficient search strategy using multiple neighbourhoods [242]. During a search process on a neighbourhood, a variable neighbourhood search algorithm may continue to search over another neighbourhood for continuously improving solutions. In this sense, these algorithms behave similarly to selection hyper-heuristics apart from their problem-dependent search characteristics. In [279], a variable neighbourhood search algorithm was used as a hyper-heuristic to manage a set of heuristics for the exam timetabling problem. This approach was devised on a graph based hyper-heuristic [76] that aimed at determining effective constructive heuristic sequences to build a complete solution. The same 6 graph-based heuristics were used as in [76]. Two neighbourhood structures were studied: VNS1 that randomly changes two to five heuristics in a sequence and VNS2 that change the same possible number of heuristics as VNS1HH located consecutive to each other. The computational results demonstrated that VNS1 performed better than VNS2. Additionally, the comparison to other algorithms from the literature, tabu search, steepest descent and iterated local search, indicated that the iterated local search was superior than the other methods. VNS1 also showed the best performance on certain exam timetabling problem instances.

2.1.6 Reward-based mechanisms

Reward-based mechanisms are frequently used for efficiently selecting heuristics. In [245, 267], a simple reward-based strategy adapting the score of low-level heuristics based on their improvement capabilities was studied. Different update options were used for the score adaptation process. In addition, some score based heuristic selection rules were used. For the heuristic selection process based on the heuristics' scores, two approaches were utilised in [245]. The first approach was a fair random choice that chooses heuristics according to their selection probabilities determined based on their scores. The other selection method was choosing the heuristics with the maximum score. Their performance on the Orc quest problem and the logistics domain problem indicated the superiority of the latter selection method. The maximum score based selection was also used

while solving the exam timetabling problem in [267]. Additionally, a random selection strategy among the heuristics with a score better or equal compared to the average of all the heuristics' scores was employed. The selection of the heuristics with the maximum score was again the best strategy although there is no statistically significant performance difference between both the selection approaches.

A similar scoring strategy was used to divide the heuristic set into two in [266]. This study was carried out with a hyper-heuristic framework, F_C [263]. In the framework, the purpose was to apply a worsening heuristic and then executing a predetermined hill climber. Instead of having this information beforehand, heuristics with a score below the overall average were considered a deteriorating heuristic. The heuristics with higher scores than the average were counted as hill climbers. The experimental results on a set of mathematical benchmark functions indicated that this naive strategy was capable of determining different heuristic types.

2.2 Generation strategies in hyper-heuristics

Table 2.4 provides methods for automatically generating heuristics. The studies on this subject commonly use genetic programming to build new heuristics in tree structures. There are other simple studies with heuristic generation characteristics. In these studies, the idea is to find good heuristic pairs showing effective performance when they are applied in a relay fashion. Some of them operate in an offline manner while the others perform the heuristic generation process online. There is also research on methods without learning but using some predetermined setting like memetic algorithms or iterated local search. During the generation process, the elements used to build a heuristic varies for different studies. These elements can be listed as: primitive elements, problem-dependent elements and heuristics. The following subsections investigate these studies with respect to elements used for the generation process.

2.2.1 Generation using primitive elements

Genetic programming was utilised to generate trees representing heuristics to solve the online 1D bin packing problem in [63]. Each heuristic was used to decide whether or not to place the items coming one by one in bins. A number of arithmetic operators and a group of terminals regarding the features of a solution space and an instance, were employed as the tree elements. The generated heuristics with two genetic operators showed similar performance to an effective

Heuristic Generation	Generation Elements	Type	References
Genetic programming	Primitive	Offline	Burke et al, 2006[63], Burke et al, 2007[67], Bader-El-Den and Poli, 2008[22], Allen et al, 2009[12], Burke et al, 2010[68], Hauptman et al, 2010[171], Abednego and Hendratmo, 2011[3], Burke et al, 2012[86]
Genetic programming	Problem-dependent	Offline	Bader-El-Den and Poli, 2007[21], Bader-El-Den and Poli, 2008[217], Bader-El-Den and Poli, 2009a,b[23, 24], Bader-El-Den et al, 2009[25], Tolay and Kumar, 2009[324], Pillay, 2010[272], Drake et al, 2012[130]
Genetic programming	Heuristics	Online	Nguyen et al, 2011[250]
Differential evolution	Primitive	Offline	Sotelo-Figueroa et al, 2013[311]
Memetic hybridisation	Heuristics	No learning	Bilgin et al, 2006[49], Özcan et al, 2008[263], Özcan et al, 2009[269], Berberoğlu and Uyar, 2010[40], Burke et al, 2010[57]
Memetic hybridisation	Heuristics	Online	Özcan et al, 2009[266]

Table 2.4: Learning in heuristic generation

human designed heuristic, i.e. first-fit. In [67], a similar method was employed to build heuristics trained on a set of randomly generated instances for different item distributions. The experimental results showed that the similarity between the item distributions of the training and test instances provide advantage to the proposed generation hyper-heuristic. In that sense, the constructed heuristics found the same or better results than the best fit heuristic for most of the instances.

In [21], a grammar was developed based on the components of SAT heuristics from the literature. The generated heuristics solved more instances than two effective local search heuristics.

In [3], a single dynamic machine production scheduling problem (single machine scheduling) from the metal industry was addressed by a hyper-heuristic. The idea was to generate new heuristics, i.e. dispatching rules, using certain heuristic components via genetic programming. 6 problem-dependent elements were used as the terminals and 4 functions were employed as the functions for the genetic programming trees. The generated heuristics were reached to comparable results to the heuristics from the literature.

A genetic programming based heuristic generation strategy for the rush hour and free cell games was introduced in [171]. Two types of trees, i.e. condition and value trees, were accommodated. 4 and 2 basic functions were employed for these trees, respectively. The experimental results showed that the evolved policies were effective on both the solution quality and speed.

Unlike the previous studies, in [85] the goal was to build local search heuristics for the offline 1D bin packing problem using grammatical evolution. The suggested grammar consists of three main categories, namely selecting bins, removing items from the selected bins and repacking the removed items. Each evolved heuristic was applied to a solution constructed by the first-fit heuristic on an instance in order to measure the heuristic's quality. This process was repeated 100 times and the new solution found at each step was accepted if there was improvement. The quality of the solutions found by the proposed approach was near to the lower bounds for the test instances.

Two evolutionary hyper-heuristics were tested on a set of exam timetabling problem instances in [272]. The first hyper-heuristic based on a linear representation aimed at determining effective constructive heuristic sequences to schedule exams one by one. These sequences had variable lengths and they were improved using a mutation operator and a crossover operator. The second hyper-heuristic was designed relying on a hierarchical representation involving parse trees composed of certain conditional and logical operators. The population involved such tress and similarly a mutational and a crossover operators were utilised to reproduce better trees. Each tree simply compared two exams and decided which one should be scheduled first depending on certain features, i.e. saturation degree and highest cost. The experimental results on 8 Toronto instances indicated that the hyper-heuristic based on the linear representation was more effective than the hyper-heuristic based on the hierarchical representation.

In [86], a genetic programming based generalised heuristic generation strategy was proposed for 1D, 2D, 3D knapsack and bin packing problems. Heuristics were evolved in tree forms composed of basic mathematical operations and terminal elements. The generated heuristics were constructive and involve an item selection procedure along with an item placement strategy. These heuristics delivered competitive results to the state of the art, human designed heuristics and the best known solutions for the target problem instances.

2.2.2 Generation using problem-dependent elements

In [153], a genetic programming based system, i.e. CLASS, was introduced for generating effective and fast local search heuristics for SAT. Several problem-

specific functions and terminal elements were identified for the individuals of a genetic programming approach. At each generation, individual expressions were randomly selected based on their ranks and determined by Whitley's linear rank-based selection function [337]. A particular composition operator was used to generate 10 new expressions using two individuals. In order to prevent expressions with large tree depths, the sub-trees with a depth greater than a certain bound were replaced with randomly generated sub-trees. Each individual was tested on a set of instances. The proposed approach was tested twice with 5 different depth bounds. The best heuristics generated at each run were compared to the best human-designed methods from the literature. The experimental results demonstrated the competitive performance of the automatically generated heuristics.

Another genetic programming based hyper-heuristic inspired from the pareto converging genetic algorithm [206] was studied in [324]. The goal was to solve the bi-objective graph colouring problem. For genetic programming, 2 arithmetic operators, 3 logical operators together with 4 problem-dependent terminals were employed. The performance of the generated heuristics was comparable to the problem-specific human-designed heuristics.

2.2.3 Generation via hybridisation

Heuristic combinations might yield superior results than the individual heuristics. Some strategies such as memetic algorithms and iterated local search are based on this idea. In [275], heuristic sequences composed of ordering heuristics were evolved for the exam timetabling problem using genetic programming. For this purpose, 5 low-level heuristics were employed. The heuristics in a sequence were used to construct a timetable by using each heuristic for assigning an exam. 2 mutation and 3 crossover operators were used to manipulate the heuristic sequences. The proposed approach outperformed 4 other hyper-heuristics from the literature on most of the test instances. Although the generated heuristic combinations that can only construct solutions were not state of the art, their performance was promising when compared to the best know solutions found by the algorithms involving both constructive and improvement capabilities.

2.3 Move acceptance

Table 2.5 lists a number of move acceptance criteria that can be used in hyper-heuristics. Due to the major influence of the move acceptance part on a hyper-heuristic's performance, a move acceptance criterion should be carefully

selected or designed. The aim is generally to employ an acceptance mechanism that determines the diversification characteristics of a hyper-heuristic. For this purpose, threshold accepting approaches have been usually preferred. Studies on threshold accepting based move acceptance criteria are explained in the following sub-section.

2.3.1 Threshold accepting

Simulated annealing

Annealing is a process for changing the state of materials such as metals via heating and cooling. The material is first heated to transform it into a high energy state with freely moving atoms, so that its shape can be easily transformed. Then, it is slowly cooled down to transform it into a solid structure via non-moving atoms. Simulated annealing [200] was introduced based on this idea for solving optimisation problems. In this approach, solutions are repeated as states and their qualities are considered energy level of the states. Lower energy states are always accepted while higher energy states are accepted based on a probability which depends on energy and temperature. Simulated annealing was used as a move acceptance criterion in hyper-heuristics with the simple random heuristic selection mechanism in [31]. It accepts improving or equal moves. If the new solution is a worsening solution, acceptance or rejection of the solution is decided based on the *metropolis probability* [224]. Temperature cooling schedule was facilitated using a temperature parameter decreasing over time. After each temperature update, it was used for a fixed number of iterations before the next update. Two temperature initialisation strategies to determine good initial values were used. 12 low-level heuristics were accommodated in the heuristic set for the planogram problem for shelf space allocation. The hyper-heuristic with the simple random heuristic selection mechanism and simulated annealing acceptance criterion was superior than some simulated annealing algorithms, hyper-heuristics and a greedy approach. Another hyper-heuristic study using simulated annealing was proposed to solve a sports timetabling problem for generating a feasible double round robin tournament, i.e. travelling tournament problem, [14]. Similarly, simulated annealing was employed as an acceptance criterion and simple random was used for the heuristic selection process. In this study, simulated annealing was modified with a reheating mechanism by using the temperature value while the most recent best solution was found. The proposed hyper-heuristic found new state of the art solutions for the large instances.

In [26], a learning based hyper-heuristic using simulated annealing with reheating was proposed for the nurse rostering, university course timetabling and 1D bin

packing problems. Unlike the aforementioned hyper-heuristics with simulated annealing, a simple reinforcement learning strategy was used as heuristic selection. In this approach, reheating operations again occurred by using the temperature at which the best solution was found. Particularly, after a number of iterations without improvement, the current temperature was set to the best temperature. If no improving solutions were found, then the current temperature was increased. The results on the nurse rostering problem using 9 low-level heuristics showed that the hyper-heuristic performed better than the indirect GA approach [9]. For the course timetabling problem using 3 low-level heuristics, experiments were performed for two different stopping conditions. When only a small number of iterations was allowed, the hyper-heuristic produced competitive results to an ant colony optimisation algorithm [308], a tabu search based hyper-heuristic [73], a variable neighbourhood search meta-heuristic and a graph-based hyper-heuristic [76]. For more iterations, the hyper-heuristic's performance was better than these algorithms. Additionally, the performance of this hyper-heuristic was ranked fourth among algorithms competing in the first international timetabling competition[1]. The computational results using 5 low-level heuristics for the 1D bin packing problem demonstrated promising performance.

A tabu search based hyper-heuristic [73] with simulated annealing was studied to solve the shipper rationalisation problem in [127]. The acceptance criterion also had a reheating strategy [126]. 21 low-level heuristics based on certain neighbourhoods and sampling policies were employed. The effective performance of the hyper-heuristic was shown on the variations of real world data sets.

In [29], a variety of algorithms consisting of heuristics, meta-heuristics and hyper-heuristics were applied to solve the fresh produce inventory control and shelf space allocation problem. A tabu search based hyper-heuristic [73], a simulated annealing hyper-heuristic [31] and a tabu search with simulated annealing hyper-heuristic were used. 4 low-level heuristics were employed in the heuristic set. Additionally, other algorithms including GRASP, a simulated annealing algorithm and 4 greedy heuristics were executed for the experiments. The simulated annealing hyper-heuristic and the hyper-heuristic using tabu search and simulated annealing together usually found higher quality results than the other tested algorithms as well as the multi-start generalised reduced gradient algorithm [32] from the literature.

In [45], selection hyper-heuristics were applied for solving a real world nurse rostering problem. Two selection hyper-heuristics using simulated annealing and great deluge were used. These acceptance criteria were combined with the simple random heuristic selection mechanism. These methods were additionally

[1] `http://www.idsia.ch/Files/ttcomp2002/`

compared to a variable neighbourhood search approach. 6 low-level heuristics with the tournament selection sampling strategy were employed in the heuristic set. The computational results on the instances from different wards with varying nurse availabilities and requirements showed the superior performance of the hyper-heuristic using simulated annealing.

Great deluge

Great deluge is another threshold based acceptance strategy proposed by [131]. A selection heuristic with the simple random heuristic selection mechanism and great deluge acceptance criterion was studied in [193]. The move acceptance method accepts solutions with the objective value lower than a parameter decreasing over time. The hyper-heuristic has reached the calculated lower bounds of 11 benchmarks for the channel assignment problem. In addition, it generated better than or equal quality results as the best solutions for 17 benchmarks.

In [43], the greedy heuristic selection approach was combined with the extended great deluge move acceptance [222] mechanism for solving the job scheduling problem in a grid environment. A group of hybridised algorithms combining a genetic algorithm with three meta-heuristics, namely a hill climber, simulated annealing and tabu search, were used in the heuristic set. The computational results indicated the superior performance of the hyper-heuristic against a genetic algorithm and each of these hybrid meta-heuristics.

In [305], a heuristic selection mechanism based on reinforcement learning was used together with three variants of the great deluge move acceptance criterion consisting of the flex deluge [55], non-linear great deluge [253] and extended great deluge [222] acceptance mechanisms. A constructive graph colouring heuristic was used to generate a feasible initial solution. 4 perturbative heuristics were employed as the low-level heuristics. The studied hyper-heuristics were applied to a set of randomly generated instances with less than 100 exams, with at least 500 exams and the Toronto benchmarks. The hyper-heuristic with the extended great deluge [304] achieved better performance than the other tested hyper-heuristics with respect to the best solution found after 10 runs and to the average of the best results.

Record-to-record travel

Record-to-record travel accepts worsening solutions based on a threshold value determined by the quality of the current best solution and a deviation element

[131]. This method was used as a move acceptance criterion in hyper-heuristics to solve the channel assignment problem in [194]. Additionally, the monte carlo, all moves and only improving move acceptance criteria were employed for performance comparison. Simple random was combined with these acceptance mechanisms. 4 low-level heuristics were used in the heuristic set. Experimental results showed that the record-to-record travel move acceptance mechanism provided better or equal solutions compared to the other experimented methods.

Late acceptance

In [264], the late acceptance [56] criterion combined with a number of heuristic selection mechanisms has been applied to the exam timetabling problem. Simple random, greedy, reinforcement learning, reinforcement learning with tabu search and choice function were employed as selection mechanisms. The heuristic set was composed of 4 perturbative low-level heuristics. Simple random together with late acceptance composed the best hyper-heuristic among the tested ones and the previous best one, i.e. a choice function-simulated annealing hyper-heuristic studied in [49].

2.4 Distributed hyper-heuristics

Hybrid algorithms and algorithms running on multiple CPUs or PCs are the most widely used methods to facilitate higher performance than traditional approaches. A number of studies focused on the latter option via running hyper-heuristics in a distributed setting.

Effect of communication type: *synchronous* vs. *asynchronous*

Application of algorithms in a parallel environment requires additional components. One of the main components is related to the communication held between each individual algorithm. This communication can be exemplified as sharing solutions found by each algorithm. This type of communication takes place after each algorithm finalises its search process concerning certain time or iteration limited constraints. Differently, algorithms may communicate with each other or via a coordinator method when it is required. These two communication strategies have been studied as *synchronous* and *asynchronous* approaches. The studies presented in this section illustrate the effect and usage of these two approaches.

A parametric choice function employed for the distributed hyper-heuristic approaches using multiple processors simultaneously was introduced in [283, 284]. These approaches were *Hierarchical Distributed Hyper-heuristic* and *Hybrid-agent Distributed hyper-heuristic*. In the hierarchical distributed version, a hyper-heuristic, i.e. *controller* having a predetermined number of best solutions was placed at a processor. Its aim was assigning some tasks to a number of other processors. Each processor used a heuristic and applied it to a solution. Whenever a processor finished a given task, a modified solution was returned to the controller. Then, new tasks were assigned to the available processors. These operations were asynchronously performed. In the second strategy, HADHH, HDHHs were used as separate agents communicating with each other to exchange their good solutions. A heuristic set consisting of 1 diversifying and 6 intensifying heuristics was used. Increasing number of processors improved the speed of finding solutions. It was shown that the number of branches for changing the behaviour of the hyper-heuristics between diversification and intensification should also be increased proportionally to the number of processors.

A traditional heuristic selection approach was used as a distributed hyper-heuristic considering each low-level heuristic a separate agent (LLHA) in a synchronous environment [260]. These agents were managed by a higher level agent, i.e. a hyper-heuristic agent (HHA). The process was started by creating an initial solution and distributing it to each of LLHA. Then, each LLHA applied its own heuristic to this solution for a while. After that, each resulting best solution found by LLHAs was transferred to HHA for distributing the overall new best solution to LLHAs. Various selection hyper-heuristics were employed within this distributed hyper-heuristic framework. These hyper-heuristics consisted of greedy-all moves, greedy-only improving, descent-all moves, descent-only improving and a tabu search based hyper-heuristic. 4 low-level heuristics were used to solve the permutation flow-shop scheduling problem. The distributed hyper-heuristic based on tabu search delivered the best performance among the tested distributed hyper-heuristics. It was superior than two hyper-heuristics using simple random and a constructive heuristic, i.e. NEH, [248] which was also used to construct initial solutions for the hyper-heuristics. Another distributed hyper-heuristic approach with two types of cooperation, synchronous and asynchronous, was studied in [261]. Again 4 low-level heuristics for the permutation flowshop scheduling problem were employed. Each of these heuristics was considered a separate agent and a cooperative hyper-heuristic was utilised to manage the best solutions found by each heuristic. If a new best solution was found by a heuristic, it was sent to the coordinator in the asynchronous version. The coordinator replaced the current global best solution with this solution if it was better than the global best. If the solution was worse then the global best, a decision was given about whether to accept or reject adding it to the solution pool based on a

move acceptance mechanism. If the solution is rejected, the best solution in the pool (which has not been sent before) is sent to this heuristic to continue the search. For the experiments, only improving (OI), all moves (AM), tabu search (TS), simulated annealing (SA), and great deluge (GD) move acceptance strategies were employed. In the synchronous version, this communication related operations were performed depending on certain iterations for each heuristic at the same time. The experimental results indicated that the cooperative hyper-heuristics outperformed the sequential hyper-heuristics in the test, namely GR-OI, GR-AM, GR-TS, SR-SA and SR-GD. Moreover, the asynchronous version performed better than the cooperative search. In [262], a similar asynchronous hyper-heuristic, but in a multi-level search context, was studied. In this case, a number of hyper-heuristics, GR-OI, GR-AM, SR-TS, SR-SA and SR-GD, were assigned as agents. There was no mechanism for accepting worse solutions than the global best of the solution pool by the coordinator agent. The experimental results on a set of permutation flowshop scheduling instances demonstrated that the designed approach outperforms each separate hyper-heuristic.

2.5 Comparison and analysis

The comparison of several hyper-heuristics and their analysis were studied in a limited number of articles. Some of these studies focused on using simple or easy-to-implement hyper-heuristics to make a comparison on a particular problem domain. Other studies investigated the characteristics, such as the effect of heuristic selection, the effect of move acceptance and the effect of low-level heuristics on the performance of hyper-heuristics.

Effect of learning in heuristic selection

In [105], various selection hyper-heuristics were proposed. Among them, simple random chooses a heuristic at each iteration. Random descent chooses a heuristic randomly and constantly applies it if the solution is improved. A variant of random descent, namely random permutation descent, determines a permutation of heuristics randomly and applies them like random descent based on the permutation. The greedy selection mechanism tries all the available heuristics and then uses the best generated solution. In addition to these simple selection mechanisms, a learning based selection strategy, i.e. *choice function*, was introduced. It aims at learning the best heuristic dynamically and effective heuristic pairs that can be applied at consecutive iterations. The all moves and only improving acceptance criteria were combined with these selection

mechanisms, The experimental results on the sales summit scheduling problem using 10 low-level heuristics indicated that the variants of choice function performed superior with the all moves acceptance strategy to the other tested algorithms.

Several selection hyper-heuristics from the literature were applied to the exam timetabling problem and the mathematical function optimisation problem in [49]. The hyper-heuristics used 7 heuristic selection mechanisms, i.e. SR, RD, RP, RPD, GR, TABU, CF and 5 move acceptance strategies, i.e. AM, OI, IE, SA, GD. On the function optimisation problem instances, the hyper-heuristics with choice function outperformed the other selection mechanisms, yet no signification performance difference yielded. The improving or equal acceptance strategy performed significantly better than the rest of the move acceptance strategies. Besides, there was no hyper-heuristic delivering the best solutions across all the test functions. For the exam timetabling problem, the CF-MC hyper-heuristic outperformed the other hyper-heuristics. Besides that, hyper-heuristics utilising all moves and only improving acceptance criteria performed worse.

Comparison of hyper-heuristics for problem solving

In [286], a number of hyper-heuristics were employed to solve the resource constrained scheduling problem. This problem was considered the selection of a task to schedule and the selection of resources for this task. For the initial selection process, tasks were ordered based on one task order method out of 8 options and a randomly selected task among the top 2 in the ordered tasks was returned. Then, one among 16 resource selectors chose a subset of resources based on the requirements of the selected task. The best resource was determined by the exhaustive search. HyperRandom (SR-OI) and HyperGreedy (GR-OI) selection hyper-heuristics were used to solve the target problem. Task order method-resource selector pairs were the low-level heuristics. Besides these hyper-heuristics, a reduced variable neighbourhood search algorithm [169] was tested using each task order method-resource selector pair and a genetic algorithm [102]. The hyper-heuristics and rVNS performed better than the genetic algorithm. Among these three algorithms, HyperGreedy was superior when it was allowed to run longer. A tabu search based hyper-heuristic using the binary exponential back off (BEBO) strategy was applied to the same problem in [288]. The tabu search here was used to adapt the tabu tenures for low-level heuristics. 9 task order methods and 27 resource selectors were used and 243 heuristics were generated in total. The hyper-heuristics from the earlier study were used with their heuristic sets and using the new one. In addition, the proposed approach was used together with different values indicating the

number of heuristics that should be kept in the heuristic set. Moreover, the traditional tabu search hyper-heuristic [73] was used with different fixed tabu tenure values and with randomly changing tabu tenures. The proposed approach delivered better results than the tested traditional tabu search hyper-heuristics. Its results were also better than the tested algorithms in their earlier study. In comparison to the current best method, HyperGreedy, the new approach found better results with only two parameter settings concerning the size of the heuristic set. Even though its performance was worse than HyperGreedy for the other parameter configurations, it found near quality solutions faster. In [287], the algorithms investigated in [286, 288] were applied to the same problem. Additionally, a reinforcement learning based hyper-heuristic [245] scoring heuristics according to their performance was used together with the only improving move acceptance criterion. Moreover, a heuristic set reduction algorithm, i.e. step-by-step reduction [95], and a random hyper-heuristic was employed for comparison purposes. The same low-level heuristics were used as in [288]. With certain specific parameter settings, the reinforcement learning strategy and the random hyper-heuristics yielded comparable performance to the best performing approaches.

A selection hyper-heuristic was studied to solve the unit commitment problem in [40]. 7 low-level heuristics including 4 mutation operators and 3 hill climbers were utilised. Random permutation descent [105] was employed as heuristic selection mechanism. The F_B [263] framework was used for selection. In this framework, if the chosen heuristic was a mutation operator, then a predetermined hill climber was applied afterwards. Only improving [105] was utilised as the acceptance criterion. The computational results on a set of benchmark and real world instances showed robustness of the hyper-heuristic. The solutions found by the hyper-heuristic were the best or near to the best among the solutions found by the existing algorithms from the literature.

A group of selection hyper-heuristics were studied for scheduling football fixtures in [159]. The performance of 12 hyper-heuristics that were combinations of 3 heuristic selection mechanisms and 4 move acceptance criteria was investigated across a set of real-world instances. Choice function and simple random were employed as heuristic selection mechanisms. Additionally, a reinforcement learning based approach, i.e. QV-Learning, was introduced as a heuristic selection mechanism. All moves, improving or equal, great deluge and simulated annealing were used as acceptance criteria. 6 low-level heuristics composed the heuristic set. Particularly the RL-GD and RL-SA hyper-heuristics outperformed the other hyper-heuristics in the experiments and produced better results than the best known solutions for the tested instances.

In [198], a number of hyper-heuristics from the literature were applied to the dynamic generalised assignment problem. The hyper-heuristics were

used together with the memory/search (MS) algorithm [53] that considers a memory population for exploitation and a search population for exploration. 6 low-level heuristics were employed in the heuristic set operating on the search population. Choice function, simple random, random descent, random permutation and random permutation descent were used as the heuristic selection mechanisms. They were combined with all moves and only improving move acceptance strategies to build 10 hyper-heuristics. At each step, each hyper-heuristic was applied to the solutions in the search population to generate new solutions. Regarding the memory population, the binary tournament selection, one-point crossover and swap mutation were applied and the generational replacement strategy was used to determine the next generation. The hyper-heuristics delivered better results than the original MS algorithm with relatively longer running time. Additionally, hyper-heuristics with learning and descent characteristics performed better than the other hyper-heuristics. For the performance of the move acceptance criteria, the hyper-heuristics with AM were usually better than the hyper-heuristics with OI.

In [41], a number of selection hyper-heuristics were applied to the short-term electrical power generation scheduling problem. Simple random, random descent, random permutation, random permutation descent greedy and choice function heuristic selection mechanisms were employed. These heuristic selection approaches were combined with four move acceptance criteria, i,e. all moves, only improving, improving or equal and great deluge. 7 low-level heuristics [185] including a number of mutation operators and hill climbers were used in the heuristic set. The hyper-heuristic with the random permutation descent heuristic selection mechanism and only improving move acceptance criterion delivered the best performance.

A group of hyper-heuristics were studied to predict DNA sequences in [51]. Four choice function based heuristic selection mechanisms, one tabu search based selection mechanism were combined with all moves acceptance criterion as selection hyper-heuristics. In addition, a similar hyper-heuristic to the tabu search based hyper-heuristic but with a probabilistic heuristic selection strategy along with a simulated annealing acceptance criterion was studied as the fifth hyper-heuristic. The heuristic set for the target problem was composed of 5 low-level heuristics. The performance of the aforementioned hyper-heuristics was promising. For higher performance, they used the same hyper-heuristics for another representation with an updated and extended heuristic set involving 14 low-level heuristics. The experiments were repeated with different heuristic set combinations from these 14 low-level heuristics. The experimental results showed that one of the choice function based hyper-heuristics and the hyper-heuristic with simulated annealing were the best performing hyper-heuristics. They also indicated that the different heuristic set configurations were useful

for different settings.

In [46], a group of heuristic selection mechanisms and acceptance criteria were combined as selection hyper-heuristics applied two health health care problems, namely nurse rostering and patient admission scheduling. The heuristic sets employed for each problem domain were considered with different tournament factors. The value of a tournament factor indicates the number of sampling solutions taken from the neighbourhood of the current solution and the best of them was returned. The selection mechanisms were choice function, simple random and dynamic heuristic set strategy with random selection. The acceptance criteria were only improving, improving or equal, simulated annealing and great deluge. Dynamic heuristic set strategy with great deluge delivered the best results on the patient admission scheduling problem instances. However, its results were not significantly different from the four other hyper-heuristics using great deluge as acceptance criterion. In the case of the nurse rostering problem, choice function-great deluge hyper-heuristics showed the best performance.

In [119], simple random was combined with five move acceptance criteria including improving or equal, simulated annealing, great deluge, late acceptance and late acceptance with steepest descent to solve a set of exam timetabling instances from three datasets: uncapacitated Toronto dataset, capacitated ITC 2007 dataset and capacitated KAHO Sint-Lieven dataset. Six low-level heuristics were designed for solving these problems. Four other low-level heuristics including three heuristics for the additional soft constraints of the ITC 2007 exam timetabling problem and one kempe chain heuristic were used. The number of neighbouring solutions visited by each heuristic at each step was limited to a value for differentiating heuristic sets in the context of tournament selection. Experiments were repeated using different heuristics among the aforementioned heuristics. A hyper-heuristic with simulated annealing delivered new state of the art results on some of the Toronto benchmarks. Some of the hyper-heuristics used late acceptance and simulated annealing for solution feasibility and improvement respectively within two phases. They found promising results on the second test dataset. For the KAHO dataset, most of the tested hyper-heuristics yielded satisfactory results.

Better heuristic application

A study for maintaining a list with solutions to be used as the second parent solutions for crossover operators was carried out in [128]. The idea was to facilitate better performance for single-point search based hyper-heuristics while solving a set of the multidimensional 0-1 knapsack problem instances. The proposed approach was carried out at the hyper-heuristic level using

problem-independent information and at problem level using problem-dependent information. The first method updates the list using a second parent solution when a new best solution is found. The second strategy considers a list generated in an offline manner using the solutions generated by some problem specific heuristics. Nine hyper-heuristics that were the combination of three heuristic selection mechanisms, simple random, choice function, reinforcement learning, and three acceptance criteria, only improving, late acceptance, simulated annealing, were employed. These hyper-heuristics were used in the F_C hyper-heuristic framework [263], that apply a chosen mutational heuristic first and apply a predetermined hill-climber afterwards. Three crossover operators and two mutational heuristics were employed as actual mutational heuristics in the framework and a problem specific hill climber was used. The computational results indicated that controlling crossover operators at the problem level performs better than controlling them at the hyper-heuristic level. In addition, the hyper-heuristic with the proposed crossover controlling strategies delivered promising results on the tested instances.

2.6 Problems

Optimisation and *decision* problems have been widely studied in hyper-heuristic studies. Optimisation aims at minimising or maximising given set of objectives. On the other hand, decision problems focus on finding an answer to a yes or no question, e.g. whether a given statement or goal holds. It is also possible to transform an optimisation problem to a decision problem, and vice versa. An optimisation problem can be reduced to a decision problem by defining an objective value, e.g. the objective value of the best solution, beforehand. A decision problem can be handled as an optimisation problem by looking for a solution with given quality for the decision problem. As an illustration, SAT is a decision problem and Max SAT is its optimisation version.

Hyper-heuristics have been applied to problems from a wide range of domains. Most of these problems were optimisation problems. Only a limited number of hyper-heuristics investigated decision problems. In the present chapter, the optimisation problems are grouped in four main categories, namely *scheduling*, *timetabling*, *routing* and *cutting/packing problems*. Table 2.6 & 2.7, Table 2.8, Table 2.9 and Table 2.10 present all these optimisation problems respectively. These tables additionally provide the problem instances used for the experiments as well as the references. A variety of real-world and benchmark problems give the opportunity to compare any search and optimisation algorithm with these hyper-heuristic approaches.

Table 2.11 demonstrates the decision problem, 3-SAT, along with the datasets that were solved using hyper-heuristics. The number of studies is very limited. Thus, applying hyper-heuristics to different decision problems can be considered an open research area.

In addition, Table 2.12 provides the problem domains addressed as multi-objective optimisation problems in the hyper-heuristic studies. Although the number of objectives considered in these problems is small, they show the potential of hyper-heuristics for solving multi-objective optimisation problems or single-objective optimisation problems in a multi-objective setting by considering constraints as objectives.

[1] http://www.cs.nott.ac.uk/∼jds/research/spacedata.html
[2] http://www.kuleuven-kortrijk.be/nrpcompetition
[3] http://allserv.kahosl.be/∼burak/project.html
[4] http://www.cs.nott.ac.uk/∼tec/NRP/
[5] http://allserv.kahosl.be/∼peter/pas/
[6] http://mat.gsia.cmu.edu/TOURN/
[7] http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/dwt/dwt.html
[8] http://www.maxsat.udl.cat/09/
[9] http://www.cril.univ-artois.fr/SAT07/
[10] http://www.cril.univ-artois.fr/SAT09/
[11] http://mat.gsia.cmu.edu/COLOR/instances.html
[12] ftp://ftp.mie.utoronto.ca/pub/carter/testprob/
[13] http://www.cs.qub.ac.uk/itc2007/
[14] http://www.asap.cs.nott.ac.uk/external/resources/
[15] http://www.idsia.ch/Files/ttcomp2002/
[16] http://www.metaheuristics.net
[17] http://people.brunel.ac.uk/∼mastjjb/jeb/orlib/tableinfo.html
[18] http://www.sintef.no/Projectweb/TOP/VRPTW/Solomon-benchmark/
[19] http://www.sintef.no/Projectweb/TOP/VRPTW/Homberger-benchmark/
[20] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
[21] http://people.brunel.ac.uk/∼mastjjb/jeb/orlib/binpackinfo.html
[22] http://www.wiwi.uni-jena.de/Entscheidung/binpp/index.htm
[23] http://paginas.fe.up.pt/∼esicup/
[24] http://www.cs.nott.ac.uk/∼mvh/resources.php
[25] http://marvin.cs.uidaho.edu/∼heckendo/Gecco08Contest/2DGAProblem/
[26] http://elib.zib.de/pub/Packages/mp-testdata/ip/sac94-suite/
[27] http://people.brunel.ac.uk/∼mastjjb/jeb/orlib/mknapinfo.html
[28] http://hces.bus.olemiss.edu/tools.html
[29] http://www.or.deis.unibo.it/research_pages/ORinstances/2CBP.html
[30] ftp://cermsem.univ-paris1.fr/pub/CERMSEM/hifi/2Dcutting/
[31] http://www.cs.ubc.ca/∼hoos/SATLIB/benchm.html
[32] http://www.in.tu-clausthal.de/∼gottlieb/benchmarks/3sat
[33] http://www.cril.univ-artois.fr/SAT07/

Move Acceptance	References
All moves	Cowling et al, 2001a,b[105, 106], Burke et al, 2003[73], Kendall and Mohamad, 2004[194], Bai and Kendall, 2005[31], Burke et al, 2005[69], Bilgin et al, 2006[49], Ouelhadj and Petrovic, 2008[260], Özcan et al, 2008[263], Maden et al, 2009[218], Gibbs et al, 2010[159], Kiraz and Topçuoğlu, 2010[198], Berberoğlu and Uyar, 2011[41], Wauters et al, 2012[336]
Only improving	Cowling et al, 2001a,b[105, 106], Kendall and Mohamad, 2004[194], Bai and Kendall, 2005[31], Burke et al, 2005[69], Bilgin et al, 2006[49], Remde et al, 2007[286], Ouelhadj and Petrovic, 2008[260], Özcan et al, 2008[263], Maden et al, 2009[218], Kiraz and Topçuoğlu, 2010[198], Berberoğlu and Uyar, 2011[41], Burke et al, 2011[85], Remde et al, 2012[287]
Improving or equal	Ayob and Kendall, 2003[19], Bilgin et al, 2006[49], Özcan et al, 2008[263], Özcan et al, 2009[269], Berberoğlu and Uyar, 2010[40], Gibbs, 2010[159], Berberoğlu and Uyar, 2011[41], Ren et al, 2011[289], Demeester et al, 2012[119], Wauters et al, 2012[336]
Naive acceptance	Burke et al, 2010[57], Özcan and Kheiri, 2011[265]
Adaptive acceptance	Burke et al, 2010[57]
Monte carlo	Glover and Laguna, 1993[162]
Linear monte carlo	Ayob and Kendall, 2003[19]
Exponential monte carlo	Ayob and Kendall, 2003[19]
Exponential monte carlo with counter	Ayob and Kendall, 2003[19]
Great deluge	Kendall and Mohamad, 2004[193], Bilgin et al, 2006[49], Özcan et al, 2008[263], Bilgin et al, 2009[45], Gibbs et al, 2010[159], Özcan et al, 2010[267], Berberoğlu and Uyar, 2011[41], Bilgin et al, 2012[46], Demeester et al, 2012[119], Wauters et al, 2012[336]
Non-linear great deluge	Obit et al, 2011[253], Sin and Kham, 2012[305]
Extended great deluge	McMullan, 2007[222], Bhanu and Gopalan, 2008[43], Sin, 2011[304], Sin and Kham, 2012[305]
Flex deluge	Burke and Bykov, 2006[55], Sin and Kham, 2012[305]
Record-to-record travel	Kendall and Mohamad, 2004[194]
Late acceptance	Burke and Bykov, 2008[56], Özcan et al, 2009[264], Demeester et al, 2012[119]
Simulated annealing	Bai and Kendall, 2005[31], Bilgin et al, 2006[49], Bai et al, 2007[29], Özcan et al, 2008[263], Bilgin et al, 2009[45], Gibbs et al, 2010[159], Bai et al, 2010[30], Burke et al, 2010[71], Bilgin et al,[46], Demeester et al, 2012[119], Kalender et al, 2012[184], Wauters et al, 2012[336]
Simulated annealing with re-heating	Anagnostopoulos et al, 2006[14], Dowsland et al, 2007[127], Burke et al, 2010[71], Jiang et al, 2011[182], Bai et al, 2012[26]
Group decision making	Mısır, 2008[225]

Table 2.5: Move acceptance strategies

Data set	Problem	References
U. Nottingham[1] , Nottingham Trent U.[1]	Office space allocation	Burke et al, 2003[87], Burke et al, 2005[83]
U. Nottingham [108]	Presentation scheduling	Cowling et al, 2002[108], Han and Kendall, 2003[167], Burke et al, 2005[69]
NRC 2010[2]	Nurse rostering	Vancroonenburg et al, 2010[325], Bilgin et al, 2010[47], Bilgin et al, 2012[46]
A UK hospital	Nurse rostering	Cowling et al, 2002[107], Burke et al, 2003[73], Bai et al, 2012[26]
A Belgian hospital[3] [44]	Nurse rostering	Bilgin et al, 2009[45]
Benchmarks[4]	Nurse rostering	Burke et al, 2010[57], Nguyen et al, 2011[250], Özcan and Kheiri, 2011[265], Di Gaspero and Urli, 2012[123], Drake et al, 2012[129], Hsiao et al, 2012[176], Kalender et al, 2012[184], Kubalík, 2012[205]
A UK company	Sales summit scheduling	Cowling et al, 2011a,b[105, 106]
Instance generator	Trainer scheduling	Cowling et al, 2002a,b[103, 104], Han and Kendall, 2003a,b[168, 167]
Instance generator[5]	Patient admission scheduling	Bilgin et al, 2008[48], Vancroonenburg et al, 2010[325], Bilgin et al, 2010[46]
Instance generator [314]	Open-shop scheduling problem	Fang et al, 1994[143]
Instance generator	Hybrid flowshop scheduling	Vázquez-Rodríguez and Salhi, 2007[329]
Instance generator	Hybrid flowshop scheduling	Ochoa et al, 2009[257]
Instance generator [314]	Permutation flowshop scheduling	Ouelhadj and Petrovic, 2008[260], Ouelhadj and Petrovic, 2009[261], Ouelhadj et al, 2009[262], Burke et al, 2010[57], Vázquez Rodríguez and Ochoa, 2010[327], Nguyen et al, 2011[250], Özcan and Kheiri, 2011[265], Di Gaspero and Urli, 2012[123], Drake et al, 2012[129], Hsiao et al, 2012[176], Kalender et al, 2012[184], Kubalík, 2012[205]
Real-world	Single machine scheduling	Abednego and Hendratmo, 2011[3]
Instance generator	Job shop scheduling	Vázquez-Rodríguez and Petrovic, 2010[328]
Instance generator [314]	Job shop scheduling	Bittle and Fox, 2009[50]
A metal company	Job shop scheduling	Castrillon et al, 2010[92]
Instance generator	Shelf space allocation	Bai and Kendall, 2005[31]
Instance generator [32]	Shelf space allocation	Bai et al, 2007[29]
Benchmark instances [185]	Power generation scheduling	Berberoğlu and Uyar, 2010[40], Berberoğlu and Uyar, 2011[41]
	Grid scheduling	Bhanu and Gopalan, 2008[43]
Instance generator [102]	Resource constrained scheduling	Remde et al, 2007[286], Remde et al, 2009[288], Remde et al, 2012[287]

Table 2.6: Scheduling datasets solved by hyper-heuristics

Data set	Problem	References
English football leagues	Fixture scheduling	Gibbs et al, 2010[159]
US national baseball league [6]	Travelling tournament	Anagnostopoulos et al, 2006[14], Chen et al, 2007[98], Pérez and Riff, 2011[270]
Rugby league[6]	Travelling tournament	Pérez and Riff, 2011[270]
Instance generator [96]	Channel assignment	Kendall and Mohamad, 2004a,b[193, 194]
NYTUN [295], TLN [13], HANOI [152], TRP[303]	WDS design optimisation	Raad et al, 2010[282]
Harwell-Boeing[7]	Bandwidth reduction	Koohestani and Poli, 2011[202]
Max Sat Eval. 2009[8], SAT 2007 Comp.[9], SAT 2009 Comp.[10]	Max SAT	Özcan and Kheiri, 2011[265], Di Gaspero and Urli, 2011[122], Nguyen et al, 2011[250], Drake et al, 2012[129], Hsiao et al, 2012[176], Kalender et al, 2012[184], Kubalík, 2012[205]
Generated using [149]	Map coloring	Bittle and Fox, 2009[50]
DIMACS[11]	Graph coloring	Tolay and Kumar, 2009[324]
ORLIB [35], TSPLIB [285]	p-median	Ren et al, 2011[289]
BB [52], ES [135], Random [302], Rat [302], Virus [302], DNA-random1 & DNA-random2 [334], Pfam [147], Real dataset [251], Simulated dataset [251]	Longest common subsequence	Tabataba and Mousavi, 2012[313]

Table 2.7: Scheduling datasets solved by hyper-heuristics (Table 2.6 continues)

Data set	Problem	References
Toronto[12] [91]	Exam timetabling	Terashima-Marín et al, 1999[320], Ross et al, 2004[292], Burke et al, 2005[60], Qu and Burke, 2005[279], Yang and Petrovic, 2005[340], Bilgin et al, 2006[49], Pillay and Banzhaf, 2007[275], Burke et al, 2007[76], Mısır, 2008[225], Qu et al, 2009[281], Özcan et al, 2009[264], Pillay and Banzhaf, 2009[276], Bader-El-Den et al, 2007[25], Qu and Burke, 2009[280], Burke et al, 2010[80], Fatima and Bader-El-Den, 2010[144], Li et al, 2010[213], Pillay, 2010[272], Özcan et al, 2010[267], Burke et al, 2010[71], Sabar et al, 2011[294], Abdul-Rahman et al, 2011[2], Sin, 2011[304], Demeester et al, 2012[119], Pillay, 2012[274], Sin and Kham, 2012[305] , Burke et al, *to appear*[81], Alinia Ahandani et al, *to appear*[11]
U. Technology MARA	Exam timetabling	Kendall and Hussin, 2005[191]
Yeditepe U.	Exam timetabling	Bilgin et al, 2006[49], Mısır, 2008[225], Özcan et al, 2010[267]
ICT - BUITEMS	Exam timetabling	Ahmed et al, 2011[5]
KAHO Sint-Lieven	Exam timetabling	Demeester et al, 2012[119]
ITC 2007[13]	Exam timetabling	Burke et al, 2010[80], Sabar et al, 2011[294], Demeester et al, 2012[119], Burke et al, *to appear*[81]
Instance generator[14]	Exam timetabling	Burke et al, 2005[60], Burke et al, 2006[78], Sin and Kham, 2012[305]
ITC 2003[15]	Course timetabling	Rattadilok et al, 2004[283], Rattadilok et al, 2005[284], Bai et al, 2007[28], Bai et al, 2012[26]
Instance generator [308]	Course timetabling	Burke et al, 2003[73], Burke et al, 2005[83], Burke et al, 2007[76], Obit et al, 2011[253], Bai et al, 2012[26]
Instance generator	Course timetabling	Burke et al, 2002[75], Petrovic and Qu, 2002[271]
Instance generator[16]	Course timetabling	Qu and Burke, 2009[280]
U. Nottingham, Nottingham Trent U.	Course timetabling	Burke et al, 2005[83]
Yeditepe U., Instance generator	Course timetabling	Kalender et al, 2012[184]
Intituto Tecnologio de Leon	Course timetabling	Soria-Alcaraz et al, 2012[310]
Balochistan U. of IT [7]	Course timetabling	Ahmed et al, 2011[6]
OR-library[17] [4]	School timetabling	Pillay, 2010[273]

Table 2.8: Timetabling problem datasets solved by hyper-heuristics

Data set	Problem	References
Christofides [99]	Vehicle routing	Meignan et al, 2010[223]
Christofides [99], Taillard [315], Fisher [148]	Vehicle routing	Garrido and Castro, 2009[155]
Solomon[18] [309], Gehring-Homberger[19]	Vehicle routing	Drake et al, 2012[129], Kalender et al, 2012[184], Kubalík, 2012[205], Ochoa et al, 2012[258], Walker et al, 2012[331]
Kilby [197]	Dynamic vehicle routing	Garrido and Riff, 2009[157]
TSPLIB[20]	Travelling salesman	Keller and Poli, 2008a,b[189, 190], Terrazas et al, 2010[321, 322], Kalender et al, 2012[184], Kubalík, 2012[205], Drake et al, 2012[129]
Instance generator	Competitive travelling salesman	Kendall and Li, 2012[192]
Instance generator	Semantic query routing	Hernández et al, 2011[174]
Instance generator	Routing and wavelength assignment	Keleş et al, 2010[187]
Instance generator	Survival virtual topology design	Ergin et al, 2011a,b[138, 139],

Table 2.9: Routing problem datasets solved by hyper-heuristics

Data set	Problem	References
OR-library[21] [142]	1D Bin packing	Ross et al, 2002[293], Ross et al, 2003[291], Burke et al, 2006[63], Burke et al, 2011[85], Jiang et al, 2011[182], López-Camacho et al, 2011[215], Özcan and Kheiri, 2011[265], Di Gaspero and Urli, 2011[122], Nguyen et al, 2011[250], Bai et al, 2012[26], Burke et al, 2012[86], Drake et al, 2012[129], Hsiao et al, 2012[176], Kalender et al, 2012[184], Kubalík, 2012[205]
Scholl[22] [299]	1D bin packing	Burke et al, 2011[85], Jiang et al, 2011[182], López-Camacho et al, 2011[215], Bai et al, 2012[26], Burke et al, 2012[86]
Wäscher & Gau [335]	1D bin packing	López-Camacho et al, 2011[215]
Dual distribution[23]	1D bin packing	Hyde et al, 2009[180], Burke et al, 2010[64]
Schoenfield[23] [298], Hyde[24]	1D bin packing	Özcan and Kheiri, 2011[265], Di Gaspero and Urli, 2011[122], Nguyen et al, 2011[250], Drake et al, 2012[129], Hsiao et al, 2012[176], Kalender et al, 2012[184], Kubalík, 2012(-Schoenfield)[205], Sotelo-Figueroa et al, 2013(-Hyde)[311]
Instance generator	Online 1D bin packing	Özcan and Parkes, 2011[268]
Beng [38]	2D bin packing	Burke et al, 2012[86]
Instance generator [319], Instance generator	2D bin packing	López-Camacho et al, 2011[215]
Okp [145], Wang [332], Ep [136]	2D knapsack	Burke et al, 2012[86]
Ngcut [34], Gcut [33], Cgcut [100]	2D bin packing/knapsack	Burke et al, 2012[86]
GECCO'08 contest[25]	2D bin packing	León et al, 2009[212]
Ep3D [136], Thpack8, BandR	3D knapsack	Burke et al, 2012[86]
Thpack9	3D bin packing/knapsack	Burke et al, 2012[86]
SAC-94[26], Glover-Kochenberg[27]	Multi-dimensional knapsack	Drake et al, 2011[128]
OR-library[28]	Multi-dimensional knapsack	Drake et al, 2011[128], Drake et al, 2012[130]
Instance generator [333]	2D strip packing	De Armas et al, 2011[116]
Hopper instances [175]	2D strip packing	Garrido and Riff, 2007[156], Araya et el, 2008[17]
DEIS[29] , Hifi[30]	2D stock cutting	De Armas et al, 2011[116]
Instance generator, [151]	2D irregular cutting stock	Gomez and Terashima-Marín, 2010[163]
A cosmetic company	Shipper rationalisation	Dowsland et al, 2007[127]

Table 2.10: Cutting and packing problem datasets solved by hyper-heuristics

Data set	Problem	References
SATLIB[31]	3-SAT	Bader-El-Den and Poli, 2007a,b[21, 217], Fukunaga, 2008[153], Bader-El-Den and Poli, 2009[23], Hyde et al, 2009[180], Swan et al, 2011[312]
Gotlieb[32] [164]	3-SAT	Fukunaga, 2008[153]
SAT2007 competition[33]	3-SAT	Bader-El-Den and Poli, 2009[23]

Table 2.11: Decision problem datasets solved by hyper-heuristics

Problem	# of objectives	References
Office space allocation	2	Burke et al, 2003[87], Burke et al, 2005[83]
University course timetabling	3	Burke et al, 2005[83]
Graph coloring	2	Tolay and Kumar, 2009[324]
Stacked NN optimisation	2	Furtuna et al, 2011[154]
Strip packing	2	De Armas et al, 2011[116]
Stock cutting	2	De Armas et al, 2011[116]
DTLZ [118]	3	McClymont and Keedwell, 2011[221]
Job shop scheduling	2, 3, 4	Vázquez-Rodríguez and Petrovic, 2010[328]
WDS design optimisation	2	Raad et al, 2010[282]
2D irregular cutting stock	2	Gomez and Terashima-Marín, 2010[163]

Table 2.12: Problems solved in a multi-objective fashion

Chapter 3

Problem Domains

In order to gain evidence of the general applicability of the hyper-heuristics, a wide variety of problem domains and problem instances have been subjected to the experimentation. This chapter lists the hyper-heuristic application domains, which cover both *real-world* and *academic* problems, studied in the dissertation. The problems are summarised in Table 3.1 and the corresponding low-level heuristics used for each problem are additionally detailed.

Problem	**Type**	**Source**
Home care scheduling	RW	Zealand Care [183]
Security personnel routing and rostering	RW	FIT[1]
Maintenance personnel scheduling	RW	
Ready-mixed concrete delivery	RW	ICORDA[2]
Travelling tournament	AB	US NLB[3] & Super 14 RL[3]
Nurse rostering	AB	NRC 2010[4] [170]
Patient admission scheduling	AB	Instance generator[5]
Maximum satisfiability	AB	HyFlex [255]
1D bin packing	AB	
Personnel scheduling	RW/AB	
Permutation flowshop scheduling	AB	
Travelling salesman	AB	
Vehicle routing	AB	

Table 3.1: The tested problem domains (RW: real-world, AB: academic benchmarks)

The hyper-heuristics do not restrict the problem to particular classes or particular sizes. In what follows, we briefly describe the targeted problem domains and modelling components that are below the domain barrier.

3.1 Travelling tournament problem

The travelling tournament problem (TTP) is a challenging sport timetabling problem requiring a feasible home/away schedule for a double round robin tournament [133]. In small or middle-size countries, teams return to their home city after away games. However in large countries, cities are too far away from each other, therefore returning home between games is often not an option. Hence, this problem is composed of small travelling salesman problems with additional constraints. The optimisation process of this problem involves finding the shortest travelling distance aggregated for all the teams.

The solutions found for the travelling tournament problem should satisfy two constraints: 1) (c_1) *a team may not play more than three consecutive games at home or away* and 2) (c_2) *for all the teams t_i and t_j, game t_i–t_j cannot be followed by game t_j–t_i.*

The objective function used here is determined by $f = d \times (1 + \sum_{i=1}^{2} w_i c_i)$. f should be minimised whilst respecting the constraints. d is the total travelling distance, c_i is the violation of constraint i, w_i is the corresponding weight. Both weights (w_1, w_2) are set to 10.

A mathematical model for the travelling tournament problem was presented in [210].

3.1.1 Problem instances

The travelling tournament problem instances used in this study were derived from the US National League Baseball and the Super 14 Rugby League. The first group of instances (NL) consists {4,6,8,10,12,14,16} teams and the second set of instances (Super) involves {4,6,8,10,12,14} teams.

[1]Fascinating IT Solutions: `http://www.fit.be`

[2]ICORDA NV: `http://www.icorda.be`

[3]TTP instances and their best solutions with lower bound values: `http://mat.gsia.cmu.edu/TOURN/`

[4]Nurse rostering competition 2010: `http://www.kuleuven-kulak.be/nrpcompetition`

[5]`http://allserv.kahosl.be/~peter/pas/`

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
t_1	0	745	665	929	605	521	370	587
t_2	745	0	80	337	1090	315	567	712
t_3	665	80	0	380	1020	257	501	664
t_4	929	337	380	0	1380	408	622	646
t_5	605	1090	1020	1380	0	1010	957	1190
t_6	521	315	257	408	1010	0	253	410
t_7	370	567	501	622	957	253	0	250
t_8	587	712	664	646	1190	410	250	0

Table 3.2: The distance matrix of instance NL8 as an example

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
Round 1	5	-6	-4	3	-1	2	-8	7
Round 2	4	-7	-8	-1	6	-5	2	3
Round 3	6	-8	-7	-5	4	-1	3	2
Round 4	-7	4	5	-2	-3	8	1	-6
Round 5	-6	5	7	8	-2	1	-3	-4
Round 6	-8	7	-6	5	-4	3	-2	1
Round 7	3	-5	-1	7	2	-8	-4	6
Round 8	2	-1	-5	-8	3	-7	6	4
Round 9	-3	8	1	-7	-6	5	4	-2
Round 10	-2	1	8	-6	-7	4	5	-3
Round 11	-4	3	-2	1	-8	7	-6	5
Round 12	8	-4	6	2	7	-3	-5	-1
Round 13	7	6	4	-3	8	-2	-1	-5
Round 14	-5	-3	2	6	1	-4	8	-7

Table 3.3: An optimal solution for NL8

In Table 3.2, an example of an 8-team TTP instance with travelling distances between teams (t_i) is given. The problem instances used in this study contain symmetric distances. This means that the distance between two teams is not depending on the travel direction. An example solution for the games among 8 teams during 14 rounds (# of rounds $= 2 \times (n-1)$ for n teams) are given in Table 3.3. For instance, the entry for t_2 in Round 7 shows that t_2 will play an away game at t_5's location (away games are characterised by a minus ($-$) sign).

3.1.2 Low-level heuristics

In order to handle the problem constraints and optimise the travelling objective, 5 low-level heuristics are used in the heuristic set. These heuristics include:

- $\boldsymbol{LLH_1}$: Swap home/away games between teams t_i and t_j
- $\boldsymbol{LLH_2}$: Swap the schedules of team t_i and t_j
- $\boldsymbol{LLH_3}$: Swap two rounds by exchanging the games assigned to r_i with those assigned to r_j
- $\boldsymbol{LLH_4}$: Swap games of t_i and t_j for a given round r
- $\boldsymbol{LLH_5}$: Swap games of a team t in rounds r_i and r_j

3.2 Structured problems with routing and rostering

Structured problems are hard problems involving more than one problem. This section presents three structured problems with routing and rostering characteristics, in particular *home care scheduling*, *security personnel routing and rostering* and *maintenance personnel scheduling*.

Each of these specific problems incorporates characteristics of the general problem and some additional constraints which differentiate the problems from one another. These additional constraints are what makes it hard for a single general purpose approach to find good solutions for each problem.

In what follows, the characteristics of the general problem as well as problem specific constraints are discussed. Additionally, the objective function of each problem is presented.

3.2.1 General problem characteristics

Consider the following general, structured problem:

$$\begin{aligned} min \quad & f_{\mathrm{VR}}(X) + f_{\mathrm{PR}}(X) \\ s.t. \quad & BX' \leq C \\ & DX'' \leq E \end{aligned} \tag{3.1}$$

Here X represents the structured problem composed of the vehicle routing and personnel scheduling sub-problems, represented by the constraint sets X' and X'' respectively.

The objective to be minimised is composed of two parts, $f_{\mathrm{VR}}(X)$ and $f_{\mathrm{PR}}(X)$, representing the objective functions of the sub-problems. The objective function of the present general problem is composed of three objectives: total travel time, idle time and violations of soft constraints. These objectives are combined in a weighted sum function. Note that only the components of the objective function are common whereas the weight associated with each soft constraint varies among the different problems and furthermore even among instances of a specific problem.

The general model presented in Equation 3.1 contains two sets of hard constraints. The constant vectors B, C, D and E are used to represent the constraints of the vehicle routing and personnel scheduling problem. The vehicle routing constraints include common constraints from the (MD)VRPTW[6] literature that guarantee feasible routes, e.g. providing sufficient travel time between two different visits, and that a route should start and end at the same depot. The personnel scheduling constraints mostly pertain to execution of the activities by the resources, e.g. maximum one activity can be performed by a resource at any one time and an activity locked to a resource cannot be assigned to another resource. Finally, one hard constraint states that all activities must be assigned.

There exists one soft constraint in the general problem concerning the time windows of the activities. The penalty incurred when violating this constraint increases linearly with the degree to which the time window is violated.

The aforementioned six constraints form the general problems at hand. The particular problems are usually characterised by additional soft constraints. Each problem adds extra elements to the general problem discussed in Section

[6]Vehicle routing problem with time windows

3.2.1. These additional constraints or objectives that distinguish the problems from each other. Table 3.6 shows an overview of the shared properties alongside the particular characteristics of each problem.

These problems all exhibit similar characteristics, the principle being that they are 'structured', i.e. they combine characteristics of different combinatorial optimisation problems. Examples of this kind of problems are the lock scheduling problem [330], the tool switching problem [110] and the travelling tournament problem [134]. The structured problem under investigation combines elements from the multi-depot vehicle routing problem with time windows (MDVRPTW) [101] and personnel rostering [58, 140]. The crux of the problem consists of assigning activities to resources (e.g. visits to patients assigned nurses). Each activity is associated with a geographic location (e.g. the patient's home) and one or more time windows during which the activity should be completed. The goal is to construct efficient routes in which all activities are performed, while also respecting the specified requirements and preferences of the activities where possible. This general integrated problem has been discussed in various domains such as home health care, security and maintenance.

Regarding home health care, the goal is typically to construct shortest routes for nurses whereby they visit patients at different geographic locations while respecting a set of constraints [232]. In the literature, different versions of this problem have been discussed. A decision support system in which the utilisation of nurses is maximised over a five day scheduling period is reported on in [36]. Their problem is modelled as a VRPTW with a central depot and additional constraints concerning route construction, nurse availability and patient visitation requirements. Within the work of [42], continuity of care as well as staff and patient satisfaction in the evaluation of solutions were considered. The scheduling problems deal with day-to-day operations. A successful implementation of a decision support system for scheduling home health care nurses was described in the study of [141]. Besides constraints guaranteeing continuity of care, additional constraints are included in their model to account for various practical restrictions. In contrast to the previously discussed literature, it was assumed that each nurse starts and ends their tour at home rather than in a central depot in [10].

In the case of security guard patrols, the focus is more on the routing aspect of the integrated problem rather than on other characteristics such as personnel availability or time windows of jobs. A tabu search meta-heuristic used for finding patrol routes on the road network of an estate was presented in [338]. Their model is based on the Chinese postmen and rural postmen problems. A problem in which both the number of guards and total distance travelled in performing routine inspections should be minimised was discussed in [88]. Furthermore, some robustness must be incorporated to account for alarm calls.

Concerning the staff, there is only one constraint stating that the number of requests assigned to each employee should be limited.

Routing and scheduling in the context of maintenance is closely related to the problem in the home health care context. The emphasis is placed both on routing and personnel rostering. In the work of [203], a problem was presented in which the skill set of employees plays an important role in deciding which teams are put together whenever a task cannot be completed by a single technician. The objective is to minimise the number of time window violations and the number of service tasks outsourced to contractors for one day. In [204], both the offline and online version of a routing and scheduling problem at a large car service company are described. Their objective is to minimise the waiting times for customers and keep the operation costs (e.g. tour lengths and overtime expenditure) as low as possible. The considered scheduling horizon also spans one day.

The mathematical model presented in [183] can be used as the base model for these three problems.

3.2.2 Home care scheduling

In the home care scheduling problem (HCSP), the activities to be performed are visits to patients. These must be carried out by a set of the available resources (home care nurses). This problem emphasises the patient-nurse relationship, i.e. patients have preferred nurses and vice versa. Connected activities, to be carried out at the same time or starting within the same defined period, occur in the HCSP. Particular care to patients requiring more than one nurse can be modelled using this concept. A HCSP example is depicted in Figure 3.1

No additional objectives or hard constraints exist when compared to the general problem. However, there are a number of additional soft constraints. Due to the aforementioned characteristics of the HCSP, violations of nurses' and patients' preferences with regard to one another are considered penalties. Also, the violation of a connected task incurs a penalty, e.g. when one of two connected activities does not start within the defined time interval. Furthermore, two personnel scheduling soft constraints exist stating that nurses should be qualified for the performed activities and the start of the nurses' shift should be respected. Finally, there exists a soft constraint stating that priorities of visits should be respected as much as possible.

Generally the problem addresses day-to-day operations, incurring a planning horizon of only one day, thus implying that no complex time-related personnel

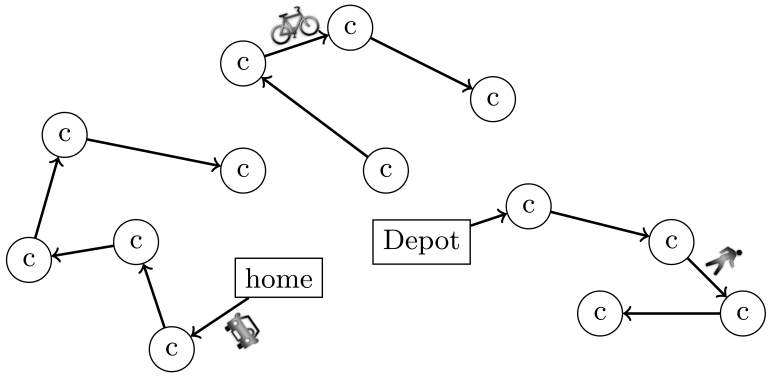

Figure 3.1: A home care scheduling problem example

scheduling constraints exist. The tested HCSP instances are profiled in Table 3.4.

Bench.	Num. of carers	Num. of tasks	Num. of patients
hh	15	154	74
hh1	15	150	74
ll11	9	104	59
ll2	7	61	30
ll21	7	60	30
ll3	7	61	30

Table 3.4: The home care scheduling problem instances (*hh*, *ll2*, *ll3*) were taken from [183]. The other instances (*hh1*, *ll11*, *ll21*) are the modified versions of the original instances in [183]

Low-level heuristic set 1 The following 6 low-level heuristics were developed for the problem. They are used in the experiments reported in Chapter 4.

- $\boldsymbol{LLH_1}$: Swap visits between two routes
- $\boldsymbol{LLH_2}$: Swap visits within a route
- $\boldsymbol{LLH_3}$: Move a visit to the best place within another route

- $\boldsymbol{LLH_4}$: Select a route and find the visit that provides the best improvement when removed and move it to the best place within another route
- $\boldsymbol{LLH_5}$: Select a route and find the visit that provides the best improvement when removed and move it to the best place within the best route that is determined based on an individual objective value of each route
- $\boldsymbol{LLH_6}$: Select a route and find the visit that provides the best improvement when removed and move it to the best place within the most idle route

Low-level heuristic set 2 The following 12 low-level heuristics were developed for the experiments in Chapter 5, Section 5.3.1.

- $\boldsymbol{LLH_1}$: Swap two randomly selected visits between two randomly selected routes
- $\boldsymbol{LLH_2}$: Swap two randomly selected visits in a randomly selected route
- $\boldsymbol{LLH_3}$: Swap two sets of randomly selected consecutive visits between two randomly selected routes
- $\boldsymbol{LLH_4}$: Move a randomly selected visit from a randomly selected route to another randomly selected route
- $\boldsymbol{LLH_5}$: Move the most conflicting visit from a randomly selected route to another randomly selected route
- $\boldsymbol{LLH_6}$: Move the most conflicting visit from a randomly selected route to the best route
- $\boldsymbol{LLH_7}$: Move the most conflicting visit from a randomly selected route to the route with the worst idle time
- $\boldsymbol{LLH_8}$: Move a set of randomly selected consecutive visits from a randomly selected route to another randomly selected route
- $\boldsymbol{LLH_9}$: Move a randomly selected visit in a randomly selected route
- $\boldsymbol{LLH_{10}}$: Move a set of randomly selected consecutive visits in a randomly selected route

Maximum number of consecutive working days	6
Maximum consecutive working time per day (minutes)	720
Maximum working time per week (hours)	37
Maximum working time per month (hours)	175 or 190
Minimum rest time between two working days (hours)	5
Maximum number of consecutive working weekends	2
Preferred number of days worked per week	5

Table 3.5: Workforce related constraints in the SPRR

- $\boldsymbol{LLH_{11}}$: Reverse the visits between two randomly selected visits in a randomly selected route
- $\boldsymbol{LLH_{12}}$: Scramble the visits between two randomly selected visits in a randomly selected route

3.2.3 Security personnel routing and rostering

The security personnel routing and rostering problem (SPRR) concerns the assignment of security tasks to a set of available guards [228]. An intriguing characteristic of this particular problem is that the generated solutions should present some randomness, i.e. it should be impossible to predict the route for a certain period based on previous scheduling periods. This property of the SPRR can be considered an objective. It is rather complex to model this kind of requirement. As an alternative, employing a non-deterministic algorithm can provide sufficiently random solutions.

One final hard constraint is added to the general problem model stating that fixed activities should be respected, i.e. jobs a priori assigned a specific start and end time should be respected. Furthermore, a number of personnel scheduling soft constraints are added. These constraints concern assigning qualified personnel to their respective activities (similar to the HCSP), assigning guards to tasks which are on the list of preferred geographic regions and also a set of workforce related constraints. Table 3.5 visualises a detailed overview of the latter constraints.

The scheduling horizon for the SPRR is ordinarily much longer than for the other problems and can span up to one month. This is a direct result of the more predictable nature of the jobs being scheduled. In contrast to patients for home health care, the demand for security jobs typically does not vary much over time, allowing for a longer scheduling period.

	HCSP	**SPRR**	**MPSP**
Objective function	min travel time min idle time min violations of soft constraints	min travel time min waiting time min violations of soft constraints max randomness of routes	min travel time min waiting time min violations of soft constraints
Hard constraints	assign all tasks perform one task at a time routes must be feasible routes start and end at same location respect locked tasks	assign all tasks perform one task at a time routes must be feasible routes start and end at same location respect locked tasks respect fixed tasks	assign all tasks perform one task at a time routes must be feasible routes start and end at same location respect locked tasks respect fixed tasks assign qualified personnel
Soft constraints	respect time window no assignments outside duty start and end time assign qualified personnel assign preferred carers to tasks assign preferred tasks to carers respect connected tasks respect priorities	respect time windows no assignments outside feasible regions assign qualified personnel respect max consecutive working days respect max consecutive working time per day respect max working time per week respect max working time per month respect min rest time between two working days respect max consecutive working weekends respect preferred number of working days per week	respect time windows no assignments outside feasible regions no assignments outside specified working hours respect max consecutive working days respect max consecutive working time per day respect max working time per week respect max working time per month respect min rest time between two working days respect max consecutive working weekends respect preferred number of working days per week respect priorities

Table 3.6: Overview of problem characteristics

Table 3.7 shows the monthly SPRR problem instances provided by a software company[1]. The instances are downloadable from `http://allserv.kahosl.be/~pieter/securityguards.html`. Each individual instance has a different number of guards and a number of visits. These instances have very tight time windows and therefore the number of instances was doubled artificially by extending time windows of the visits for each instance.

Bench.	Num. of guards	Num. of tasks
district0	62	1560
district1	348	4217
district2	120	1714
district3	113	2193
district4	192	5252
district5	389	5139

Table 3.7: The security personnel routing and rostering problem instances

3.2.4 Maintenance personnel scheduling

The maintenance personnel scheduling problem (MPSP) assigns technicians to various tasks such as installing new devices or repairing malfunctioning infrastructure. Typically these tasks have time windows specified at a day-level, with the earliest and latest start date. Times are not included in the time windows and are set depending on the the availability of technicians.

This problem adds two additional hard constraints to the general problem. First, the particular date at which a task is to be executed can be determined a priori. If this is the case, the predefined date of execution should be respected. Second, each assigned technician should have the required qualifications. Besides the hard constraints, the MPSP includes two additional soft constraints. The first one states that technicians should only work within their feasible geographic regions. The second presents a set of workforce related restrictions, as shown in Table 3.8.

Table 3.9 lists the tested MPSP instances with their details. The instances can be obtained from `http://allserv.kahosl.be/~pieter/maintenancesched.html` They are based on real world data provided by the same software company[1].

The scheduling horizon of this problem is relatively short, typically one week, due to the unpredictable nature of maintenance tasks. Ordinarily, a task

[1] Fascinating IT Solutions (`http://www.fit.be`)

Maximum consecutive working time per day (minutes)	480
Minimum rest time between two working days (hours)	12
Maximum travelling time between two locations (minutes)	60

Table 3.8: Workforce related constraints in the MPSP

Bench.	**Num. of technicians**	**Num. of tasks**
inst0	12	141
inst1	12	113
inst2	9	110
inst3	10	100
inst4	11	142

Table 3.9: The maintenance personnel scheduling problem instances

constitutes a call from a customer. To satisfy the customer's request as much as possible, a visit is scheduled for within the next week. The scheduling horizon is therefore limited to the same week. As a consequence of this relatively short scheduling period, the number of workforce related constraints is rather small. Note that we consider here the offline version of the MPSP. When analysing the problem description it becomes clear that the MPSP in an online context presents another relevant problem.

3.3 Ready-mixed concrete delivery problem

The ready-mix concrete (RMC) delivery problem is an example of complex vehicle routing and scheduling encountered in the construction sector. It is essentially concerned with scheduling deliveries of ready-mixed concrete (RMC) from central production centres to customers' construction yards, using a large heterogeneous fleet of trucks. Due to the perishable nature of RMC, these deliveries must be performed under strict time constraints as the RMC quality may decline (or even solidify) during transport. Furthermore, customers typically require a large amount of RMC that cannot be delivered by a single truck, thus necessitating multiple truck deliveries. To guarantee the quality of the concrete structure, large time gaps between sequential deliveries are to be avoided.

The concrete delivery problem represents both the primary characteristics of scheduling and routing problems and has been accessed in a limited number

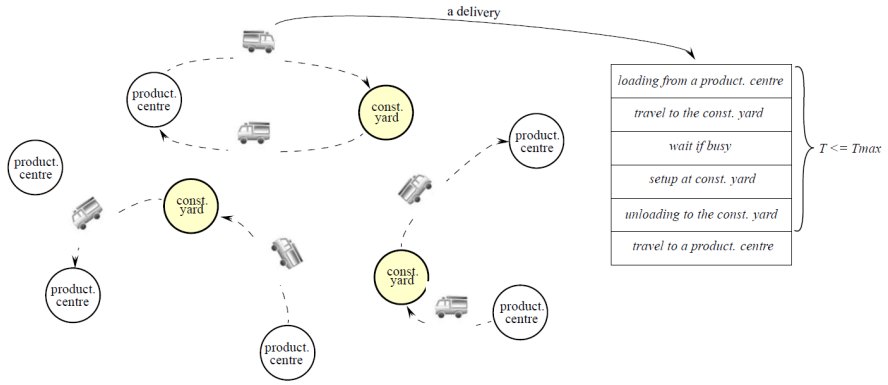

Figure 3.2: A ready-mixed concrete delivery problem example

of studies. A decision support system was developed to solve the ready-mixed concrete delivery problem consisting of routing for the pumping vehicles and scheduling for concrete delivery [220]. In [146], genetic algorithms were used to determine the dispatching sequence of trucks and a simulation technique was employed to detect timing information regarding each operation. Different neural network models based on feed-forward and Elman recurrent networks were proposed in [165]. The problem was decomposed into the assignment of orders to the production centres and the assignment of trucks in [246, 247]. Finally, a two-phase approach involving genetic algorithms and a construction heuristic was designed for solving the decomposed problem. Both a mixed-integer programming model and local search strategy, with large real-world problem solving capability, were studied in [18]. In [297], the problem was modelled as an integer multicommodity network flow problem while at the same time an improvement strategy using variable neighbourhood search was studied. In [296], a hybrid approach involving variable neighbourhood search and mixed integer linear programming, supported by a very large neighbourhood search strategy, was proposed. This study also presents a mathematical model.

Figure 3.2 shows an example of a simple RMC delivery problem. In what follows, we describe the three main elements of the real world problem.

Production centres

RMC is loaded into the trucks at production centres. Loading RMC requires a certain amount of time, which is specific to the equipment used at the production centre.

Vehicles

Vehicles (trucks) are characterised by their volume, pumpline length and vehicle type. They are the main carriers of RMC that pick up from any of the production centres and deliver to construction yards. They are assumed to be present at the first production centre. They always take an amount of concrete that is equal to their capacity. The sequence involved in making a delivery is detailed below:

1. Load an amount of RMC at the production centre. Vehicles loading concrete require some time (which is specific to the production centre) and only one vehicle may perform loading operations at any one time. The loading operations therefore should not overlap.
2. Drive towards the construction yard.
3. Wait at the construction yard until the unloading can be set up.
4. Set up equipment at the construction yard. The setup process takes an amount of time specific to the construction yard. This setup may overlap with a delivery already in progress.
5. Unload the RMC at the discharge rate specified by the customer. Only one vehicle can perform unloading at any time, so unloading operations must not overlap.
6. Return to a production centre for the next delivery.

Only a limited amount of time may pass between loading RMC at a production centre and unloading it at a construction yard in order to control its viscosity. Therefore, each delivery must finish unloading before a specified amount of time, T_{max}. This possibly limits the amount of RMC that can be delivered by a vehicle, depending on the discharge rate used at the construction yard. This may lead to waste, given that each vehicle should be filled to its maximum capacity when leaving the production centre.

Orders

An order o is defined by its order quantity q_o and a discharge rate d_o at which the concrete should be poured. The discharge rate determines the unloading time for deliveries: unloading a quantity q at a discharge rate d takes q/d time units. Each order also has an earliest start time, before which no unloading operations may be performed. Some required properties for the delivery vehicles can be specified per order:

- A customer can require a pumpline of certain length for pouring concrete.
- Certain vehicle types may be prohibited by the customer. For example, some vehicles may be too large to enter the construction yard. Any vehicle matching the required properties for an order can deliver concrete to that customer.

Objective function

Symbol	Description
o	Order
d	Delivery
$\mathcal{O}$	Set of all orders
$\mathcal{D}$	Set of all deliveries
$\mathcal{D}_o$	Set of all deliveries for order o
C_o	Requested concrete amount for order o
$C_{d,o}$	Capacity of the truck handling delivery d for order o
W_o	Total waste for order o ($\sum_{d\in\mathcal{D}_o} C_{d,o} - C_o$)
$\mathcal{V}$	Set of all trucks
L_o	Lateness of the first delivery for order o
$t_{lag,d}$	Time lag for delivery d (time gap between unloading end time of previous delivery and unloading start time of delivery d)
$psOK_{o,d}$	
T_v	Total travelling time of a truck v
α_i	Weight for constraint i

Minimise:

$$f_{objective}=\sum_{o\in\mathcal{O}}\left(\alpha_1\cdot L_o+\alpha_2 W_o+\alpha_3\sum_{d\in\mathcal{D}_o} psOK_{o,d}+\alpha_4\sum_{d\in\mathcal{D}_o} t_{lag,d}\right)+\alpha_5\sum_{v\in\mathcal{V}} T_v \tag{3.2}$$

The RMC problem considered in this study aims at minimising the objective function consisting of the following components:

- per-order lateness of the *first* delivery, weighted by α_1,
- per-order wasted concrete, weighted by α_2; i.e. the total delivered volume of RMC minus the ordered quantity,

- for each delivery, whether the RMC was loaded at the preferred production centre, α_3,
- per-order lag between subsequent deliveries, weighted by α_4,
- total travelling time, weighted by α_5.

The weights of the objective function were set after consultation with an RMC company: $\alpha_1 = 10, \alpha_2 = 10, \alpha_3 = 1, \alpha_4 = 20, \alpha_5 = 20$. Additionally, the maximum time permitted for completing a delivery (T_{max}) was set to 100 minutes.

Initial solution

An initial solution is constructed at random. The process is initiated by sorting the orders on their earliest start time in an increasing manner. Whilst sorting the orders, vehicles are randomly selected. If a vehicle is not currently assigned for pick-up, a starting station is selected. According to the required travelling time to the order's location and permitted transfer time limit, the maximum amount of concrete that can contribute to the order is determined. Based on these random selections, new deliveries are introduced until the total amount of concrete requested by the order is satisfied. The same procedure is repeated for subsequent orders.

3.3.1 Problem instances

Table 3.10 provides the 26 real-world ready-mixed concrete delivery problem instances provided by a software company[7]. The instances can be downloaded from `http://allserv.kahosl.be/~mustafa.misir/rmc.html`.

3.3.2 Low-level heuristic set

Nine low-level heuristics were implemented for the RMC problem. After applying a selected heuristic, it is necessary to execute a method to maintain the solution feasible. This method simply reassigns time information whilst respecting the T_{max} constraint and updates the number of deliveries if required.

- $\boldsymbol{LLH_1}$: change the return station of a vehicle that is responsible for a randomly selected delivery

[7] ICORDA NV: `www.icorda.be`

Instances	# orders	Amount(m^3)	# vehicles
p2011-01-10	6	80.75	28
p2011-01-11	17	598.75	34
p2011-01-12	26	748.25	34
p2011-01-13	15	581.95	31
p2011-01-14	21	474.95	32
p2011-01-18	17	395.25	28
p2011-01-19	18	436	28
p2011-01-20	14	309.25	28
p2011-01-21	17	308	28
p2011-01-25	13	280.25	28
p2011-01-26	15	389.8	28
p2011-01-27	19	610.25	28
p2011-01-28	19	546.5	29
p2011-02-01	18	299.85	28
p2011-02-02	16	333.25	28
p2011-02-03	19	422	28
p2011-02-07	17	317.25	28
p2011-02-08	14	233.5	31
p2011-02-09	17	588.25	28
p2011-02-10	16	507	28
p2011-02-11	19	651.25	30
p2011-02-14	15	233.95	32
p2011-02-15	24	1195.7	28
p2011-02-16	20	554.6	28
p2011-02-18	24	796.5	28
p2011-02-21	19	747.25	30

Table 3.10: The details of the RMC instances

- $\boldsymbol{LLH_2}$: change the vehicle of a randomly selected delivery
- $\boldsymbol{LLH_3}$: ruin all the deliveries of an order and recreate them
- $\boldsymbol{LLH_4}$: change the position of an order within the sequence of all orders
- $\boldsymbol{LLH_5}$: move a delivery between deliveries belonging to an order
- $\boldsymbol{LLH_6}$: change the amount to deliver for a randomly selected delivery
- $\boldsymbol{LLH_7}$: change the loading station for a randomly selected delivery
- $\boldsymbol{LLH_8}$: swap two vehicles between two randomly selected deliveries belonging to a randomly selected order

- $\boldsymbol{LLH_9}$: swap two vehicles between two randomly selected deliveries belonging to two randomly selected orders

3.4 Patient admission scheduling problem

The patient admission scheduling problem (PASP) considers assigning a number of patients to rooms for a period of time required for each patient while respecting his/her medical needs. Various studies on the PASP have dealt with different aspects of the problem using quite different solution techniques. Multi-neighbourhood local search [93, 94], tabu search [121] were studied to solve the target PASPs. We also applied hyper-heuristics as a high level solution strategy in [93].

The hard constraints for this problem can be enumerated as follows:

- The admission and discharge date of the patient are fixed and cannot be changed during the search
- Every patient is attributed a length-of-stay and this is always one contiguous period
- Every patient must be assigned to a bed during his/her stay
- The capacity of a room cannot be exceeded

The related soft constraints are shown in the following list:

- The gender of the patients assigned to one room at the same time should be the same
- The number of room transfers during a patient's stay should be minimised
- The ward to which the patient is assigned should be in accordance with his/her pathology
- The patient's room should be equipped with the mandatory or preferred room properties of her/his pathology
- The patient's room should satisfy the specialism of his/her pathology
- The patient's room preference should be satisfied

The objective is to minimise all the violations of the aforementioned soft constraints.

A mathematical model for this problem is available in [46].

Inst.	# departments	# rooms	# beds	# patients
Inst0	6	150	447	660
Inst1	4	98	286	652
Inst2	6	151	465	755
Inst3	5	131	395	708
Inst4	6	155	471	746
Inst5	4	102	325	587
Inst6	4	104	313	685

Table 3.11: The patient admission scheduling problem instances

3.4.1 Problem instances

Table 3.11 shows the PASP instances used in the experiments.

3.4.2 Low-level heuristic set 1

The following 4 low-level heuristics were used in Chapter 4, Section 4.2.

- $\boldsymbol{LLH_1}$: Swap beds of the two patients in the same ward
- $\boldsymbol{LLH_2}$: Swap beds of the two patients in different wards
- $\boldsymbol{LLH_3}$: Transfer the whole schedule of a patient to empty beds in the same ward
- $\boldsymbol{LLH_4}$: Transfer the whole schedule of a patient to empty beds in another ward

3.4.3 Low-level heuristic set 2

The following 11 low-level heuristics were developed for the problem. They are part of the experiments in Chapter 5, Section 5.3.3.

- $\boldsymbol{LLH_1}$: Swap all the bed assignments of a randomly selected patient with the beds of randomly selected patients
- $\boldsymbol{LLH_2}$: Transfer all the bed assignments of a randomly selected patient to randomly selected empty beds

- $\boldsymbol{LLH_3}$: Swap all the bed assignments between two randomly selected patients
- $\boldsymbol{LLH_4}$: Swap all the bed assignments of a randomly selected patient with randomly selected occupied beds. Transfer the remaining assignments to the randomly selected beds
- $\boldsymbol{LLH_5}$: Transfer all the bed assignments of a randomly selected patient to a randomly selected empty bed
- $\boldsymbol{LLH_6}$: Swap a randomly selected bed assignment with another bed while respecting room properties
- $\boldsymbol{LLH_7}$: Swap a randomly selected bed assignment with another bed while respecting room preferences of the corresponding patient
- $\boldsymbol{LLH_8}$: Swap a randomly selected bed assignment with another bed while respecting the room specialism
- $\boldsymbol{LLH_9}$: Swap a randomly selected bed assignment of a randomly selected patient with another bed while respecting room properties
- $\boldsymbol{LLH_{10}}$: Swap two randomly selected beds
- $\boldsymbol{LLH_{11}}$: Transfer all the patients in a randomly selected room to another randomly selected room

3.5 Nurse rostering problem

The nurse rostering problem (NRP) is a hard combinatorial optimisation problem. It deals with scheduling nurses with respect to given requirements concerning certain contractual constraints. Variable neighbourhood search [74] and hyper-heuristics [73, 46] are the example studies related to this chapter. The problem subject to this study is from the first international nurse rostering competition [170].

The hard constraints regarding solution feasibility are:

- Assignment of nurses to cover all the nurse requirements
- Assignment of each nurse to at most one shift per day

The soft constraints that need to be minimised are presented as follows:

- The number of days more than the maximum number of working days for nurses
- The number of days less than the minimum number of working days for nurses
- The number of days more than the maximum number of consecutive working days for nurses
- The number of days less than the minimum number of consecutive working days for nurses
- The number of days more than the maximum free days for nurses
- The number of days less than the minimum free days for nurses
- The number of assignments for the cases such that a nurse works only one weekend day for a particular weekend
- The number of assignments violating certain shift pair patterns for nurses
- The number of unpreferred shift patterns assigned to the nurses

The corresponding objective function returns the total number of violations over all the soft constraints.

A mathematical model for this problem is available in [46].

3.5.1 Problem instances

The problem instances were taken from the nurse rostering competition [170]. For the experiments, 10 long instances (Table 3.12) were employed. The experiments were conducted for 10 minutes and 1 hour of execution time.

3.5.2 Low-level heuristics

In total, 29 low-level heuristics were developed to manage for the hyper-heuristics.

- $\boldsymbol{LLH_1}$: Swap a randomly selected day between two randomly selected nurses
- $\boldsymbol{LLH_2}$: Swap a number of days between two randomly selected nurses

Instances	Best
long_late01	235
long_late02	229
long_late03	220
long_late04	221
long_late05	83
long_hidden01	346
long_hidden02	89
long_hidden03	38
long_hidden04	22
long_hidden05	41

Table 3.12: The nurse rostering problem instances with 50 nurses

- $\boldsymbol{LLH_3}$: Swap a number of consecutive days between two randomly selected nurses
- $\boldsymbol{LLH_4}$: Swap one weekdays between two randomly selected nurses
- $\boldsymbol{LLH_5}$: Swap all the weekend days between two randomly selected nurses
- $\boldsymbol{LLH_6}$: Swap all the weekdays between two randomly selected nurses
- $\boldsymbol{LLH_7}$: Swap two randomly selected weekends from two randomly selected nurses
- $\boldsymbol{LLH_8}$: Swap one randomly selected day from a randomly selected weekend between two randomly selected nurses
- $\boldsymbol{LLH_9}$: Swap all the weekend days by passing one weekend after each one between two randomly selected nurses
- $\boldsymbol{LLH_{10}}$: Swap all the days between randomly selected two days between two randomly selected nurses
- $\boldsymbol{LLH_{11}}$: Swap a randomly selected day between the most conflicting nurse and the least conflicting nurse
- $\boldsymbol{LLH_{12}}$: Swap a randomly selected day between the most conflicting nurse and a randomly selected nurse
- $\boldsymbol{LLH_{13}}$: Swap a randomly selected day between a randomly selected nurse with more than average conflict and a randomly selected nurse with less than the average conflict

- $\boldsymbol{LLH_{14}}$: Swap a number of days between the most conflicting and the less conflicting nurse
- $\boldsymbol{LLH_{15}}$: Swap a number of days between the most conflicting and a randomly selected nurse
- $\boldsymbol{LLH_{16}}$: Swap a number of days between a randomly selected nurse with more than average conflicts and a randomly selected nurse with less than the average
- $\boldsymbol{LLH_{17}}$: Move a randomly selected day from the most conflicting nurse to a randomly selected nurse
- $\boldsymbol{LLH_{18}}$: Move a randomly selected day from a randomly selected nurse to the most conflicting nurse
- $\boldsymbol{LLH_{19}}$: Move a randomly selected day from a randomly selected nurse with more than average conflict to a randomly selected nurse with less than the average conflict
- $\boldsymbol{LLH_{20}}$: Move a randomly selected day from a randomly selected nurse with less than the average conflict to a randomly selected nurse with more than average conflict
- $\boldsymbol{LLH_{21}}$: Move a randomly selected day from a randomly selected nurse with more than average conflict to a randomly selected nurse
- $\boldsymbol{LLH_{22}}$: Move a randomly selected day from a randomly selected nurse to a randomly selected nurse with more than average conflict
- $\boldsymbol{LLH_{23}}$: Move a randomly selected day from a randomly selected nurse to another randomly selected nurse
- $\boldsymbol{LLH_{24}}$: Move all the weekend days from a randomly selected to another randomly selected nurse
- $\boldsymbol{LLH_{25}}$: Swap a randomly selected day between two randomly selected nurses
- $\boldsymbol{LLH_{26}}$: Swap two randomly selected consecutive days between two randomly selected nurses
- $\boldsymbol{LLH_{27}}$: Swap three randomly selected consecutive days between two randomly selected nurses
- $\boldsymbol{LLH_{28}}$: Swap four randomly selected consecutive days between two randomly selected nurses
- $\boldsymbol{LLH_{29}}$: Swap five randomly selected consecutive days between two randomly selected nurses

3.6 Problems in HyFlex

HyFlex is a software framework supporting hyper-heuristic development. It provides a number of problem domains and a low-level heuristic set for each problem. The current version of HyFlex involves six problems, namely 1D bin packing, max SAT, permutation flowshop scheduling, personnel scheduling, travelling salesman and vehicle routing problems. For each problem domain, a heuristic set involving distinct numbers of heuristics from aforementioned types is available. The heuristic type distributions of each heuristic set are shown in Figure 3.3. The following subsections describe the problem instances provided in HyFlex. More details about the instances are available in [178, 255].

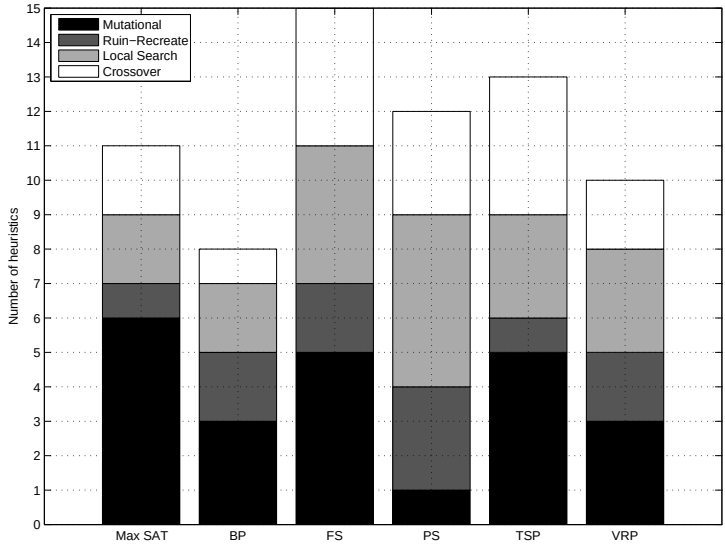

Figure 3.3: Heuristic-type distributions and heuristic set sizes for each problem domain (each column shows the number of heuristics and their types)

3.6.1 1D Bin packing

The 1D bin packing problem aims at minimising the number of homogeneous bins required for placing a set of given items. Equation 3.3 shows the fitness function included in HyFlex, n refers to the number of bins used, C is the capacity of a bin and $fullness_i$ means the total size of the elements in bin i. The fitness function considers the fullness of a bin while the objective is to minimise n to make the improvement easier.

$$f = 1 - \left(\frac{\sum_{i=1}^{n} (fullness_i/C)^2}{n} \right) \tag{3.3}$$

Table 3.13 shows the instances for this domain.

Inst.	Name	Best
0	falkenauer/u1000-00	399*
1	falkenauer/u1000-01	406*
2	schoenfieldhard/BPP14	62
3	schoenfieldhard/BPP832	60
4	10-30/instance1	N/A
5	10-30/instance2	N/A
6	triples1002/instance1	N/A
7	triples2004/instance1	N/A
8	test/testdual4/binpack0	N/A
9	test/testdual7/binpack0	N/A
10	50-90/instance1	N/A
11	test/testdual10/binpack0	N/A

Table 3.13: Bin packing instances [178] (the column "Best" shows the number of bins used. The solutions with * are optimal)

A simple mathematical model for the bin packing problem was shown in [179].

3.6.2 Max SAT

The max SAT problem deals with minimising the number of broken clauses of a boolean formula in a conjunctive normal form. The problem instances in the testbed are given in Table 3.14. All the instances were taken from the SAT competitions.

A mathematical model for the max SAT problem was discussed in [117].

3.6.3 Permutation flowshop scheduling

The permutation flowshop scheduling problem requires minimising the completion time of the last processed job (makespan) while assigning a set of jobs to a group of machines. Table 3.15 presents the related problem instances provided by HyFlex.

Inst.	Name	Best
0	contest02-Mat26.sat05-457.reshuffled-07	> 0
1	hidden-k3-s0-r5-n700-01-S2069048075.sat05-488.reshuffled-07	N/A
2	hidden-k3-s0-r5-n700-02-S350203913.sat05-486.reshuffled-07	N/A
3	parity-games/instance-n3-i3-pp	0*
4	parity-games/instance-n3-i3-pp-ci-ce	0*
5	parity-games/instance-n3-i4-pp-ci-ce	N/A
6	HG-3SAT-V250-C1000-1	6
7	HG-3SAT-V250-C1000-2	6
8	HG-3SAT-V300-C1200-2	8
9	MAXCUT/SPINGLASS/t7pm3-9999	N/A
10	jarvisalo/eq.atree.braun.8.unsat	> 0
11	HG-3SAT-V300-C1200-4	7

Table 3.14: Max SAT instances [178] (the solutions with $*$ are optimal and > 0 shows the corresponding solutions with a fitness value larger than zero)

Inst.	Name	Best
0	100x20/1	6202
1	100x20/2	6183
2	100x20/3	6271
3	100x20/4	6269
4	100x20/5	6314
5	200x10/2	10480
6	200x10/3	10922
7	500x20/1	26059
8	500x20/2	26520
9	500x20/4	26456
10	200x20/1	11181
11	500x20/3	26371

Table 3.15: Permutation flowshop scheduling instances [178]

A mathematical model for the permutation flowshop scheduling problem was given in [97].

3.6.4 Personnel scheduling

The personnel scheduling problem requires the assignment of employees to certain timeslots for a given planning period. The instances tested here belong

to the nurse rostering domain and they are listed with the values of the best known solutions in Table 3.16.

Inst.	Name	Best
0	BCV-3.46.1	3280
1	BCV-A.12.2	1953
2	ORTEC02	270
3	Ikegami-3Shift-DATA1	2
4	Ikegami-3Shift-DATA1.1	3
5	Ikegami-3Shift-DATA1.2	3
6	CHILD-A2	1095
7	ERRVH-A	2142
8	ERRVH-B	3121
9	MER-A	9017
10	BCV-A.12.1	1294
11	ORTEC01	270

Table 3.16: Personnel scheduling instances [178]

A detailed problem description with a mathematical model is available in [115].

3.6.5 Travelling Salesman

The objective of the travelling salesman problem is to find the shortest route for visiting a set of cities and returning to the starting city. The travelling salesman problem instances are illustrated in Table 3.17.

Inst.	Name	Best (Optimal)
0	pr299	48191
1	pr439	107217
2	rat575	6773
3	u724	41910
4	rat783	8806
5	pcb1173	56892
6	d1291	50801
7	u2152	64253
8	usa13509	19982859
9	d18512	645238

Table 3.17: Travelling salesman instances [178]

More details about the travelling salesman problem including a mathematical model was provided in [37].

3.6.6 Vehicle routing

The vehicle routing problem deals with visiting a number of locations within their given time windows using a small number of vehicles. That is, the objective is to minimise the total number of vehicles used (n) and the total distance travelled ($\sum d_i$). Equation 3.4 represents the fitness function to be minimised.

$$f = n * 1000 + \sum_{i=1}^{n} d_i \tag{3.4}$$

Table 3.18 provides the vehicle routing instances.

Inst.	Name	Best	# Vehicles	Distance
0	rc207	4061.14	3	1061.14
1	r101	20645.79	19	1645.79
2	rc103	12261.67	11	1261.67
3	r201	5252.37	4	1252.37
4	r106	13251.98	12	1251.98
5	c1-10-1	142478.95	100	42478.95
6	rc2-10-1	83373.15	20	63373.15
7	r1-10-1	153904.23	100	53904.23
8	c1-10-8	135499.59	93	42499.59
9	rc1-10-5	136631.89	90	46631.89

Table 3.18: Vehicle routing problem instances [178]

A mathematical model for the vehicle routing problem was presented in [201].

3.7 Conclusion

The problem independent nature of hyper-heuristics is their main motivational characteristic differentiating them from traditional search and optimisation algorithms. Therefore, it is possible to apply a hyper-heuristic to any problem without additional effort on the hyper-heuristic side. Thus, we employed

various real-world and benchmark problems to measure the performance and the generality of our hyper-heuristics. No particular rule was taken into account while determining these problems. Additionally, we developed or used the existing low-level heuristics for the target problems. It should also be that there were no limitations on the heuristic sets regarding the number of heuristics, the heuristics' improvement capabilities and speed. Hence, our testbed includes a diverse range of problems with different heuristics sets. This chapter explains the problems used in our testbed and presents the low-level heuristics for solving them. Hence, the chapter can be considered a general information showing the possibility of choosing any problem using any heuristic set for the application of a hyper-heuristic. It additionally gives insights about problem modelling and the characteristics together with the implementation details about low-level heuristics.

Chapter 4

Selection Hyper-heuristics with Different Sub-mechanisms

A hyper-heuristic is composed of a number of sub-mechanisms. It is possible to separately design these sub-mechanisms and combine them as new hyper-heuristics. This means that it is unnecessary to design a hyper-heuristic as a single piece. There are a large number of studies on selection hyper-heuristics focusing on either heuristic selection or move acceptance. For the heuristic selection process, the usually preferred approach is using a learning device [66]. In case of online learning, the idea is to determine successful heuristics based on their performance while solving a problem instance. The acceptance criteria generally aim at balancing intensification and diversification. This balance can be achieved by using search space dependent approaches. The variety of approaches for any other hyper-heuristic sub-mechanisms can be designed in the same way. Analysing the behaviour of new hyper-heuristic sub-mechanisms could help to determine the hyper-heuristic requirements from the hyper-heuristic side. The analysis results could be used to address their usefulness and help to devise new hyper-heuristic sub-mechanisms.

This chapter introduces new hyper-heuristic sub-mechanisms and tests them in order to get insights in their performance. Table 4.1 presents the new hyper-heuristic components developed and their application domains. These components consist of the heuristic selection mechanisms and move acceptance criteria. All the selection methods accommodate learning approaches for

choosing heuristics. Unlike these methods, a heuristic subset selection or heuristic exclusion strategy was additionally introduced. The acceptance mechanisms share the same basic idea but their adaptiveness levels differ. Furthermore, an approach for a new hyper-heuristic component type, i.e. *mentoring*, was introduced.

The common characteristics of these approaches are *learning* and *adaptation*. Different hyper-heuristics were built using these components together with other mechanisms from the literature. Each of the hyper-heuristics was applied to the instances of one problem to quickly acquire knowledge about how hyper-heuristic sub-mechanisms works.

Selection	**Mentoring**	**Acceptance**	**Application**
LA	-	ILTA	Travelling tournament
LA, LA+SR, RDI	LA	-	Patient admission scheduling
DHS	-	AILTA	Home care scheduling
-	-	AILLA-F	Ready-mixed concrete delivery

Table 4.1: Contributions of the chapter

The chapter is a combination of the following publications:

M. Mısır, T. Wauters, K. Verbeeck, G. Vanden Berghe, A Hyper-heuristic with Learning Automata for the Traveling Tournament Problem, *Metaheuristics: Intelligent Decision Making*, *to appear*.

M. Mısır, T. Wauters, K. Verbeeck, G. Vanden Berghe, A New Learning Hyper-heuristic for the Traveling Tournament Problem, in *Proceedings of the 8th Metaheuristics International Conference (MIC'09)*, Hamburg, Germany, 2009.

M. Mısır, B. Bilgin, P. Demeester, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, A Hyper-heuristic Approach to the Patient Admission Scheduling Problem, in *Proceedings of the 35th International Conference of Operational Research Applied to Health Services (ORAHS'09)*, Leuven, Belgium, 2009.

M. Mısır, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, Hyper-heuristics with a Dynamic Heuristic Set for the Home Care Scheduling Problem, in *Proceedings of the IEEE Congress on Evolutionary Computation (IEEE CEC'10)*, p.2875–2882, Barcelona, Spain, 2010.

M. Mısır, W. Vancroonenburg, K. Verbeeck, G. Vanden Berghe, A Selection Hyper-heuristic for Scheduling Deliveries of Ready-Mixed Concrete, in *Proceedings of the 9th Metaheuristics International Conference (MIC'11)*, Udine, Italy, 2011.

4.1 A selection hyper-heuristic with learning automata and iteration limited threshold accepting

This section introduces both a new learning based selection mechanism and a new acceptance criterion. A learning automaton (LA) [244] is used as the heuristic selection mechanism. The objective of an automaton is to learn optimal actions based on their past experience. The internal state of the learning device is described as a probability distribution according to which actions should be chosen. These probabilities are adjusted with some reinforcement schemes corresponding to the success or failure of the actions. A common update scheme for learning automata is the linear reward-penalty scheme. The philosophy behind is to increase the probability of an action when it results in a success and to decrease it when the response is a failure. The field is well-founded in the sense that the theory shows interesting convergence results. In the present study, the action space of the learning automata is composed of low-level heuristics. The goal of the learning device is to learn selecting appropriate low-level heuristics for a problem instance at hand.

Besides the heuristic selection mechanism, a new move acceptance criterion, i.e. iteration limited threshold accepting (ILTA), is introduced. The idea is to accept a worsening solution under certain conditions, namely 1) after a predefined number of worsening moves has been considered and 2) if the quality of the worsening solution is within a range of the current best solution.

The proposed hyper-heuristic is tested on the challenging travelling tournament problem presented in Chapter 3, Section 3.1. The empirical results demonstrate that the proposed hyper-heuristic, LA-ILTA, consistently outperforms the hyper-heuristic with simple random even on a small-sized heuristic set. Additionally, this simple yet general method easily found high quality solutions for the known travelling tournament problem benchmarks.

The learning automata based selection mechanism and the new acceptance criterion are detailed in Sections 4.1.1 and 4.1.2 respectively. The experimental results are shown and discussed in Section 4.1.3.

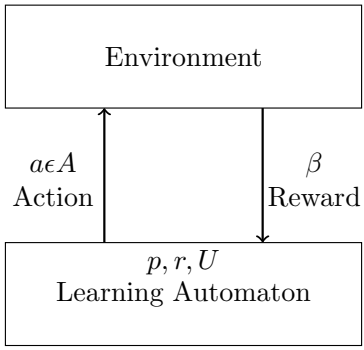

Figure 4.1: A learning automaton put into a feedback loop with its environment [323]

4.1.1 Heuristic selection: learning automata

Formally, a learning automaton is described by a quadruple $\{A, \beta, p, U\}$, where $A = \{a_1, \ldots, a_n\}$ is the action set the automaton can perform, p is the probability distribution over these actions, $\beta(t)$ is a random variable between 0 and 1 representing the environmental response, and U is a learning scheme used to update p [323].

A single automaton is connected with its environment in a feedback loop (Figure 4.1). Actions chosen by the automaton are given as input to the environment and the environmental response to this action serves as input to the automaton. Several automaton update schemes with different properties have been studied, such as *linear reward-penalty*, *linear reward-inaction* and *linear reward-ϵ-penalty*. These schemes determine how the action probabilities are updated. The general

update scheme (U) is given by:

$$\begin{aligned} p_i(t+1) &= p_i(t) + \lambda_1\, \beta(t)(1-p_i(t)) \\ &\quad -\lambda_2(1-\beta(t))p_i(t) \\ &\quad \text{if } a_i \text{ is the action taken at time step } t \end{aligned} \tag{4.1}$$

$$\begin{aligned} p_j(t+1) &= p_j(t) - \lambda_1\, \beta(t)p_j(t) \\ &\quad +\lambda_2(1-\beta(t))[(r-1)^{-1} - p_j(t)] \\ &\quad \text{if } a_j \neq a_i \end{aligned} \tag{4.2}$$

with r the number of actions of the action set A. The constants λ_1 and λ_2 are the reward and penalty parameters respectively. When $\lambda_1 = \lambda_2$, the algorithm is referred to as linear reward-penalty (L_{R-P}). In case of $\lambda_2 = 0$, it is referred to as linear reward-inaction (L_{R-I}). If λ_2 is smaller than λ_1, it is called linear reward-ϵ-penalty ($L_{R-\epsilon P}$).

In this study, actions are considered low-level heuristics. The selection probabilities of the heuristics are adapted based on their performance. Equations 4.1 and 4.2 are used for the update operations. When a move results in a better solution than the best solution found so far, the probability of the corresponding heuristic is updated with a reward of 1. Otherwise, the reward signal is 0.

Additionally, in order to investigate the effect of neglecting information gathered about the performance of the low-level heuristics during the earlier iterations, a restarting mechanism is employed. It simply resets the heuristics' probabilities (p_i) to their initial values ($1/r$). The logic behind this approach depends on the strictly varying search requirements. In particular, improving in the beginning of a search process is a lot easier than improving a solution during further iterations. The situation is depicted in Figure 4.2. It illustrates the quality change of the best solution generated by the SR-ILTA hyper-heuristic with simple random while solving a travelling tournament problem instance. The hardness of improving a solution can be clearly seen after very fast and effective improvement stage. Hence, it is likely that a policy learnt during earlier stages of a search becomes misleading in later stages. In such a case, restarting the learning process would be useful.

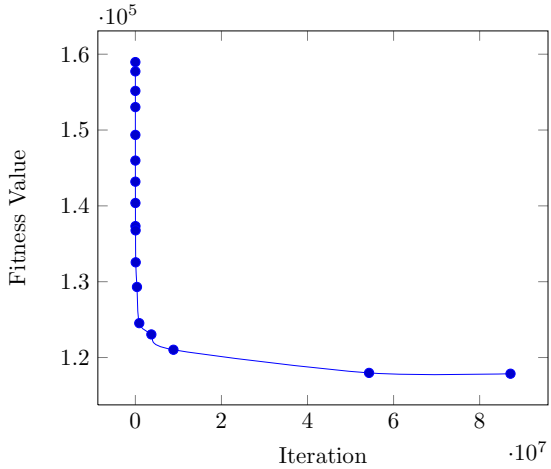

Figure 4.2: Solution samples from the SR-ILTA hyper-heuristic on a TTP instance (NL12)

4.1.2 Move acceptance: iteration limited threshold accepting

Iteration limited threshold accepting (ILTA) is a move acceptance mechanism for efficiently balancing the intensification and diversification behaviour of a hyper-heuristic. The pseudo-code of ILTA is given in Algorithm 1.

Algorithm 1: ILTA Move Acceptance

Input: $k \geq 0$, $R \in (1.00 : \infty)$
if $f(S') < f(S)$ **then**
  $S \leftarrow S'$
  $w\_iterations = 0$
else if $f(S') = f(S)$ **then**
  $S \leftarrow S'$
else
  $w\_iterations = w\_iterations + 1$
  if $w\_iterations \geq k$ *and* $f(S') < f(S_b) \times R$ **then**
    $S \leftarrow S'$ and $w\_iterations = 0$
  end
end

When a heuristic is selected and applied to generate a new solution S', its

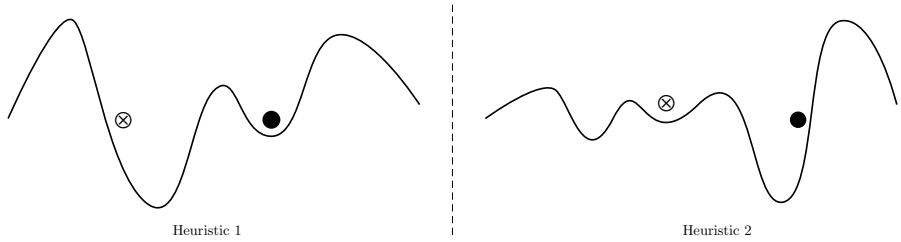

Figure 4.3: Fitness landscapes for two heuristics

objective function value $f(S')$ is compared to the objective function value of the current solution $f(S)$. If the new solution is non-worsening, the new solution is accepted and it replaces the current one. However, when the new solution is worse than the current solution, the acceptance criterion checks whether this worsening move is good enough to accept. Move acceptance mechanisms consisting of a diversification strategy may accept a worsening move at any step of a search process. ILTA acts much more selective before accepting a worsening move. In ILTA, a worsening move can only be accepted after a predefined number of worsening moves (k) was generated consecutively. For instance, if 100 worsening moves ($w\_iterations = 100$) were generated with no new best solution, it is plausible to think that the current solution is not good enough to be improved. It makes sense to accept a worsening move then. In that case, a worsening move is accepted given that the new solution's objective function value $f(S')$ is within a certain range R of the current best solution's objective function value $f(S_b)$. For example, choosing $R = 1.05$ means that a solution will be accepted within a range of 5% from the current best objective function value. R was introduced for preventing the search to go to much worse solutions, e.g. the solutions that can be found during earlier iterations.

The motivation behind limiting the number of iterations for accepting worsening solutions is related to avoiding a *local minimum* that a set of fitness landscapes have in common. In addition, the search should explore *neutral pathways or plateaus* [307], solutions that have the same objective values, efficiently in order to discover possible new best solutions. Each low-level heuristic generates a fitness landscape itself and each solution available in one landscape is available in the others supposing that the representation spaces are the same for different low-level heuristics, but the neighbouring relations differ. Figure 4.3 visualises two virtual fitness landscapes for two heuristics. Two solutions are represented as circles on each landscape. Each of these solutions is a local minimum on one of the landscapes. However, each local minimum can be easily improved by changing heuristics.

A naive selection mechanism like simple random, can be sufficient to overcome such problems. However, suppose that the current solution is a common local minimum of both landscapes. In such situations, swapping landscapes will not be helpful. Accepting a worsening solution can be a meaningful strategy for exploring promising search regions.. Unfortunately, fitness landscapes do not usually look that simple. Each represented solution has lots of neighbours and checking all of them whenever encountering a possible local minimum would be a time consuming process. Instead of that, we suggest to provide a reasonable number of iterations for the improvement attempt. If after the given number of iterations, the solution has not improved, then it is quite reasonable to assume that the solution is at a common local minimum. It makes then sense to explore the search space further by accepting worsening moves.

4.1.3 Performance of hyper-heuristic sub-mechanisms: L_R, LR_R-I, ILTA

The experiments were carried out on Pentium Core 2 Duo 3 GHz PCs with 3.23 GB memory using the JPPF grid computing platform[1]. Each hyper-heuristic was tested 10 times for each travelling tournament problem instance. The learning automata based selection mechanism was tested with different values of λ_1, namely $\{0.001, 0.002, 0.003, 0.005, 0.0075, 0.01\}$. The minimum number of consecutive worsening solutions (k) in ILTA was set to 100. R was set to $\{1.2, 1.2, 1.04, 1.04, 1.02, 1.02, 1.01\}$ for the NL4 $\mapsto$ NL16 and $\{1.2, 1.2, 1.1, 1.02, 1.015, 1.01\}$ for the Super4 $\mapsto$ Super14 instances. In addition, distinct time limits as stopping conditions were used for different TTP instances, namely 5 minutes for the NL4–6 and the Super4–6 instances, 30 minutes for the NL8 and Super8 instances and 1 hour for the rest (NL10–16 and Super10–14).

The R and k values were determined after a number of preliminary experiments. The restarting iteration value was based on the rate of improvement corresponding to the first 10^7 iterations as shown in Figure 4.2. The learning rates were not tuned beforehand so that the effect of different learning rates on the hyper-heuristic's performance could be investigated.

Comparison of LA and SR heuristic selection

Table 4.2, 4.3, 4.4 and 4.5 present the performance of the tested hyper-heuristics for each TTP instance. In the tables, we consider *AVG*: Average Objective Function Value, *MIN*: Minimum Objective Function Value, *MAX*: Maximum

[1] Java Parallel Processing Framework: `http://www.jppf.org/`

SR	**NL4**	**NL6**	**NL8**	**NL10**	**NL12**	**NL14**	**NL16**
AVG	8276	23916	39813	60226	115320	202610	286772
MIN	**8276**	**23916**	**39721**	59727	113222	201076	283133
MAX	8276	23916	40155	61336	116725	205078	289480
STD	0	0	181	468	1201	1550	2591
TIME	˜0	0.314	8	2037	2702	2365	2807
ITER	9,00E+00	1,39E+05	2,73E+06	5,17E+08	4,95E+08	3,17E+08	2,95E+08

Table 4.2: Results of the SR hyper-heuristic for the NL instances (˜0 means almost zero)

LA	**NL4**	**NL6**	**NL8**	**NL10**	**NL12**	**NL14**	**NL16**
AVG	8276	23916	39802	60046	115828	201256	288113
MIN	**8276**	**23916**	**39721**	**59583**	**112873**	**196058**	**279330**
MAX	8276	23916	40155	60780	117816	206009	293329
STD	0	0	172	335	1313	2779	4267
TIME	˜0	0.062	0.265	760	3508	1583	1726
ITER	1.90E+01	1.34E+03	8.23E+04	1.94E+08	6.35E+08	2.14E+08	1.79E+08
λ	ALL	0.002(B)	0.002(B)	0.001	0.003	0.002	0.0075(R)

Table 4.3: The best results among the L_R-I and LR_R-I (L_R-I + Restarting) hyper-heuristics for the NL instances (In the λ line; **B**: Both L_R-I & LR_R-I, **R**: LR_R-I), (˜0 means almost zero)

Objective Function Value, *STD*: Standard Deviation, *TIME*: Elapsed CPU Time in seconds to reach to the corresponding best result, *ITER*: Number of Iterations to reach to the corresponding best result.

In Table 4.2 and Table 4.3, the results for the NL instances are given. For all the instances, *L_R-I* and *LR_R-I* performed better than the hyper-heuristic with simple random, SR-ILTA. The results provided in Table 4.4 and Table 4.5 also show similar performance on the Super instances. However, the learning rate (λ) of each best hyper-heuristicwith learning automata for finding the best solution among all the trials may differ. These cases are illustrated in the last rows of the LAHHs tables. This situation also occurs regarding the average performance of the hyper-heuristics. Table 4.6 and 4.7 show the hyper-heuristics' ranking[2].

[2]MS Excel Rank Function

SR	**Super4**	**Super6**	**Super8**	**Super10**	**Super12**	**Super14**
AVG	71033	130365	182626	325888	472829	630751
MIN	**63405**	**130365**	**182409**	322761	469276	607925
MAX	88833	130365	184581	329789	475067	648648
STD	12283	0	687	2256	1822	13908
TIME	3.063	0.016	10	756	3147	2742
ITER	1.90E+01	2.11E+03	2.85E+06	1.81E+08	5.42E+08	3.50E+08

Table 4.4: Results of the SR hyper-heuristic for the Super instances

LA	**Super4**	**Super6**	**Super8**	**Super10**	**Super12**	**Super14**
AVG	71033	130365	182975	327152	475899	634535
MIN	**63405**	**130365**	**182409**	**318421**	**467267**	**599296**
MAX	88833	130365	184098	342514	485559	646073
STD	12283	0	558	6295	5626	13963
TIME	~0	0.031	1	1731	3422	1610
ITER	8.00E+00	9.41E+02	1.49E+05	3.59E+08	5.12E+08	2.08E+08
λ	ALL	0.01(B)	0.003(B)	0.002	0.005	0.001

Table 4.5: The best results among the L_R-I and LR_R-I (L_R-I + Restarting) hyper-heuristics for the Super instances (In the λ line; **B**: Both L_R-I & LR_R-I, **R**: LR_R-I)

The rank of each hyper-heuristic was calculated as the average of all the ranks among all the TTP benchmark instances. It can be recognised that a fixed learning rate (λ_1) is not a good strategy to solve all the instances. It can even cause the LAHHs to perform worse than SR, as Table 4.6 shows. Therefore, the key point for increasing the performance of LAHHs is determining the right learning rates.

As we mentioned before, learning heuristic selection can be hard due to the limited room for improvement when using a small heuristic set. A similar note was raised concerning a reinforcement learning scoring strategy for selecting heuristics in [77]. It was demonstrated that SR may perform better than the learning based selection mechanism with poor scoring schemes. A similar issue can be raised concerning the learning rate of learning automata. The experimental results showed that SR may perform better than LA with respect to the heuristic selection. Employing inappropriate learning rates for a problem instance is the main reason behind the superior performance of SR.

Additionally, a restarting mechanism can further improve the performance of pure LAHHs. The proposed restarting mechanism provides such improvement even if it is quite simple. Again in Table 4.6 and 4.7, the learning hyper-heuristics with restarting (LR_R-I) are usually better than the pure version with respect to average performance. The underlying reason behind this improvement is that the performance of a heuristic can change over different search regions. Thus, restarting the learning process for different regions with distinct characteristics can provide easy adaptation under different search conditions.

In Table 4.8, the result of a statistical comparison between the learning hyper-heuristics and the hyper-heuristic with SR is given. The average performance of the hyper-heuristics indicates that it is not possible to state a significant performance difference between the LA and SR hyper-heuristics based on a T-Test within 95% confidence interval. However, LA in general performs better as

λ_1	L_R-I	NL4	NL6	NL8	NL10	NL12	NL14	NL16	RANK
0.001	AVG	8276	23916	39808	60046	115481	201505	288491	**4.57**
	STD	0	0	183	335	947	2916	3513	
0.002	AVG	8276	23916	39802	60573	115587	201256	289220	**6.14**
	STD	0	0	172	738	918	3093	2299	
0.003	AVG	8276	23916	40038	60510	115828	202762	290719	**9.00**
	STD	0	0	307	727	1313	3838	3675	
0.005	AVG	8276	23916	39908	60422	116174	203694	290009	**8.86**
	STD	0	0	288	541	1867	2100	3681	
0.075	AVG	8276	23916	39912	60606	116286	204712	288205	**9.43**
	STD	0	0	210	644	1125	3097	2929	
0.01	AVG	8276	23916	39836	60727	117256	205294	293704	**10.36**
	STD	0	0	252	965	854	2305	1615	
λ_1	LR_R-I	NL4	NL6	NL8	NL10	NL12	NL14	NL16	
0.001	AVG	8276	23916	39836	60563	115469	201234	288881	**6.07**
	STD	0	0	252	714	957	3296	3370	
0.002	AVG	8276	23916	39813	60449	116141	201984	285319	**5.64**
	STD	0	0	181	1066	862	1911	3580	
0.003	AVG	8276	23916	40097	60295	115242	202092	288040	**5.57**
	STD	0	0	303	631	592	3644	4431	
0.005	AVG	8276	23916	39848	60375	115958	202584	289355	**7.29**
	STD	0	0	283	716	1895	2642	3399	
0.075	AVG	8276	23916	39895	60454	115796	203060	288113	**7.00**
	STD	0	0	224	721	945	2334	1928	
0.01	AVG	8276	23916	39770	60503	115806	203429	288375	**6.57**
	STD	0	0	136	696	386	3593	3366	
SR		NL4	NL6	NL8	NL10	NL12	NL14	NL16	
	AVG	8276	23916	39813	60226	115320	202610	286772	**4.50**
	STD	0	0	136	696	386	3593	3366	

Table 4.6: Ranking corresponding to the average results among the hyper-heuristics with L_R-I, LR_R-I (L_R-I + Restarting) and SR heuristic selection mechanisms for the NL instances (lower rank values are better)

a heuristic selection than SR. On the other hand, the T-Test shows a statistically significant difference between LA and SR with respect to the best solutions found out of 10 trails.

Comparison with other methods

The hyper-heuristic results have been compared to the best results in the literature. They are optimal for the small TTP instances (NL4–8 and Super4–8). The results found in this study are not state of the art on larger instances. The results on Super12 and Super14 instances were the best as listed in a TTP instance website[3]. It should be noted that the results were produced using short execution times compared to some of the other approaches shown in Table 4.9 and Table 4.10.

[3] `http://mat.gsia.cmu.edu/TOURN/`

λ_1	L_R-I	Super4	Super6	Super8	Super10	Super12	Super14	RANK
0.001	AVG	71033	130365	182615	327268	476371	634535	**7.25**
	STD	12283	0	346	4836	5618	13963	
0.002	AVG	71033	130365	183629	327152	477237	626643	**8.00**
	STD	12283	0	27	6295	5900	12918	
0.003	AVG	71033	130365	182975	327588	481131	630109	**9.33**
	STD	12283	0	558	4605	12226	14375	
0.005	AVG	71033	130365	182533	327599	475899	629428	**6.17**
	STD	12283	0	274	6753	5626	17992	
0.075	AVG	71033	130365	183585	327604	480109	635947	**10.50**
	STD	12283	0	1551	5012	6688	11880	
0.01	AVG	71033	130365	182699	327523	478001	646711	**8.83**
	STD	12283	0	428	3116	5570	16954	
λ_1	LR_R-I	Super4	Super6	Super8	Super10	Super12	Super14	
0.001	AVG	71033	130365	182615	325961	476084	625490	**5.08**
	STD	12283	0	346	5327	3519	12591	
0.002	AVG	71033	130365	183255	325379	475779	615274	**5.17**
	STD	12283	0	989	5562	3028	9147	
0.003	AVG	71033	130365	182938	325407	475980	622210	**5.33**
	STD	12283	0	552	1896	5777	15024	
0.005	AVG	71033	130365	182727	327033	473861	629087	**5.67**
	STD	12283	0	411	5941	2478	8345	
0.075	AVG	71033	130365	182787	324098	477943	632881	**7.00**
	STD	12283	0	713	2041	5857	3083	
0.01	AVG	71033	130365	182953	326273	477175	629564	**7.33**
	STD	12283	0	807	3863	2906	12415	
SR		Super4	Super6	Super8	Super10	Super12	Super14	
	AVG	71033	130365	182626	325888	472829	630751	**5.33**
	STD	12833	0	687	2256	1822	13908	

Table 4.7: Ranking corresponding to the average results among the hyper-heuristics with L_R-I, LR_R-I (L_R-I + Restarting) and SR heuristic selection mechanisms for the Super instances (lower rank values are better)

AVG | BEST	LAHH	SR	LAHH	SR
NL4	8276	8276	8276	8276
NL6	23916	23916	23916	23916
NL8	39770	39813	39721	39721
NL10	60046	60226	59583	59727
NL12	115242	115320	112873	113222
NL14	201234	202610	196058	201076
NL16	285319	286772	279330	283133
Super4	71033	71033	63405	63405
Super6	130365	130365	130365	130365
Super8	182533	182626	182409	182409
Super10	324098	325888	318421	322761
Super12	473861	472829	467267	469276
Super14	615274	639751	599296	607925
T-TEST	**2,64E-01**		**3,13E-02**	

Table 4.8: T-Test results for the learning hyper-heuristics and the SR based hyper-heuristic on the NL and Super instances

TTP Inst.	LAHH	Best	Difference (%)	LB
NL4	**8276**	**8276**	0,00%	8276
NL6	**23916**	**23916**	0,00%	23916
NL8	**39721**	**39721**	0,00%	39721
NL10	59583	59436	0,25%	58831
NL12	112873	110729	1,88%	108244
NL14	196058	188728	3,88%	182797
NL16	279330	261687	6,74%	249477
Super4	**63405**	**63405**	0,00%	63405
Super6	**130365**	**130365**	0,00%	130365
Super8	**182409**	**182409**	0,00%	182409
Super10	318421	316329	0,66%	316329
Super12	467267	463876	0,73%	452597
Super14	599296	571632	4,84%	557070

Table 4.9: Comparison between the best results obtained by the learning automata hyper-heuristics (LAHHs) and the state of the art results (LB: Lower Bound)

Author(s)	Method	NL4	NL6	NL8	NL10	NL12	NL14	NL16
Easton et al. [133]	Linear Programming (LP)	**8276**	**23916**	441113				312623
Benoist et al. [39]	A combination of constraint programming and lagrange relaxation	**8276**	**23916**	42517	68691	143655	301113	437273
Cardemil [90]	Tabu Search	**8276**	**23916**	40416	66037	125803	205894	308413
Zhang [341]	Unknown (data from TTP website)	**8276**	24073	39947	61608	119012	207075	293175
Shen and Zhang [301]	'Greedy big step' Meta-heuristic			39776	61679	117888	206274	281660
Lim et al. [214]	Simulated Annealing and Hill-climbing	**8276**	**23916**	**39721**	59821	115089	196363	274673
LangFord [207]	Unknown (data from TTP website)				**59436**	112298	190056	272902
Crauwels and Oudheusden [111]	Ant Colony Optimisation with Local Improvement	**8276**	**23916**	40797	67640	128909	238507	346530
Anagnostopoulos et al. [14]	Simulated Annealing	**8276**	**23916**	**39721**	59583	111248	188728	263772
Gaspero and Schaerf [158]	Composite Neighborhood Tabu Search Approach				59583	111483	190174	270063
Chen et al. [98]	Ant-Based Hyper-heuristic	**8276**	**23916**	40361	65168	123752	225169	321037
Van Hentenryck and Vergados [173]	Population-Based Simulated Annealing					**110729**	**188728**	**261687**
This study	Learning Automata Hyper-heuristics with Iteration Limited Threshold Accepting	**8276**	**23916**	**39721**	59583	112873	196058	279330

Table 4.10: TTP solution methods compared, adapted from [98]

Finally, we separately compare our results to those obtained by other TTP solution methods. Hyper-heuristic approaches have been applied to the TTP before [14, 98, 173]. As can be seen from Table 4.10, the best results were generated by these hyper-heuristic techniques. The SR as a heuristic selection mechanism and simulated annealing (SA) as a move acceptance criterion were used to construct a hyper-heuristic structure in [14]. The authors of [173], which is an extension of [14], generated the state of the art results for most of the NL instances. Compared to our approach, they have a complex move acceptance mechanism requiring quite some parameter tuning, and their experiments took much more time than ours. For instance, the best result for NL16 was found after almost 4 days in [14], be it with slower PCs than ours. We are aware that it is not fair to compare the results that were generated using different computers, but the execution time of 4 days is more than 1 hour (our execution time for NL16) considering the computers' configurations. Another hyper-heuristic for the TTP was proposed in [98]. A population based hyper-heuristic using ant colony optimisation was applied to the NL instances. It found the best results for the NL4-6 and good enough feasible solutions for the rest. Compared to [14], the execution times were smaller. The results found here are better than those in [98]. Consequently, these results show that hyper-heuristics have a great potential for solving hard combinatorial optimisation problems like the travelling tournament problem.

4.2 Effect of learning-based hyper-heuristic components

In this section, several selection hyper-heuristics with different sub-mechanisms are investigated. The experiments are conducted on the instances of the patient admission scheduling problem, with a computation time limit of one hour. The problem details are presented in Chapter 3, Section 3.4.

4.2.1 Tested selection hyper-heuristics

Table 4.11 gives the details of 15 selection hyper-heuristics applied to the patient admission scheduling problem. The heuristic selection mechanisms are categorised based on the deployment of a learning strategy. The first six hyper-heuristics ($HH1 \rightarrow HH6$) consist of learning automata [323] to manage the heuristic selection process. HH9 and HH10 employ a simpler heuristic selection mechanism, i.e. random descent iteration. Learning automata provide a continuing learning strategy in order to measure the quality of heuristics

according to their performance during a search. On the other hand, random descent iteration learns based on only one move and after i iterations it forgets this information. $HH11 \rightarrow HH15$ randomly select heuristics. Unlike the other heuristic selection approaches, HH7 and HH8 use a new hybrid selection strategy. This hybrid approach combines learning automata and simple random. For move acceptance, learning characteristics are similarly taken into account. Among them, simulated annealing behaves adaptively for diversification without a learning component. Late acceptance learns the threshold values by updating a list with the accepted solutions' fitness values. Additionally, each low-level heuristic used here requires a parameter denoting the number of neighbouring solutions that will be visited at each iteration. This type of heuristics were studied in [48]. The corresponding parameters were set before the search process starts and this value was kept the same during the whole search process. It is expected that adaptiveness of these parameters would be more effective than static values. Therefore, a learning automaton is used to specify the best parameter value during a run. $HH2$, $HH4$, $HH12$ and $HH14$ use this parameter adaptation strategy while the rest of the hyper-heuristics apply a fixed value.

HH	Heuristic selection		Move acceptance		Solution sampling		
	Learning	**No Learning**	**Learning**	**No Learning**	**Learning**	**No Learning**	**# of samples**
HH1	LA	-	-	SA	-	Fixed	4
HH2	LA	-	-	SA	LA	-	4
HH3	LA	-	-	SA	-	Fixed	8
HH4	LA	-	-	SA	LA	-	8
HH5	LA	-	LATE(200)	-	-	Fixed	4
HH6	LA	-	LATE(200)	-	-	Fixed	8
HH7	LA	SR	-	SA	-	Fixed	4
HH8	LA	SR	-	SA	-	Fixed	8
HH9	RDI	-	-	SA	-	Fixed	4
HH10	RDI	-	-	SA	-	Fixed	8
HH11	-	SR	LATE(Var)	-	-	Fixed	4
HH12	-	SR	-	SA	LA	-	4
HH13	-	SR	-	SA	LA	-	8
HH14	-	SR	-	SA	-	Fixed	4
HH15	-	SR	-	SA	-	Fixed	4

Table 4.11: The tested hyper-heuristics on the patient admission scheduling problem (LATE(200) refers to late acceptance with a fixed list size 200, LATE(Var) refers to late acceptance tested with different list size values)

Table 4.11 illustrates all the tested hyper-heuristics with their sub-mechanisms. The main focus is to show the effect of learning mechanisms and of some variations with respect to the move acceptance part.

Heuristic selection and mentoring: learning automata

A learning automaton was also employed as a heuristic selection mechanism in HH1 $\rightarrow$ HH8. Additionally, HH7 and HH8 use a combined heuristic selection mechanism accommodating a learning automaton and simple random rather than using only learning automata. In these hyper-heuristics, the selection procedure is divided into three phases. A percentage value (P) is defined to identify when simple random works and when the learning automaton is applied. Simple random is used to choose heuristics during the first $P\%$ of the total execution time. Afterwards, the second phase proceeds until the last $P\%$ of the time is reached. Finally, simple random is applied during the last $P\%$ of the total execution time The difference between HH7 and HH8 is the update mechanism of the probability vector during the first phase. In HH7, the probabilities are already updated during the first simple random phase. In contrast, the probability vector in HH8 is only updated during the learning automata phase.

A solution sampling strategy [48] was employed for the parametric heuristics. When a heuristic is called, it visits a number of neighbouring solutions and returns the best one. In the literature, the term tournament factor denotes the number of neighbouring solutions to consider. The value of the tournament factor is usually predetermined. A learning automaton is employed so that a tournament factor value for each low-level heuristic can be learning while performing a search. Particularly, the learning automaton determines tournament factors proportionally to the maintained probability vector. A lower bound and some scaling values based on this lower bound are initially set. The tournament factors are then updated with respect to their probabilities considering the average probability ($1/number\_of\_heuristics$). For instance, if the probability vector concerning 4 heuristics is $\{0.5, 0.25, 0.125, 0.125\}$ and the lower bound is set to 4, then their tournament factors are set to $\{8, 4, 4, 4\}$ respectively. If the heuristic selection mechanism is a learning automaton, then both learning components use the same probability vector. Otherwise, the learning automaton is used only for the adaptation of the tournament factors.

Heuristic selection: random descent iteration

Random descent [105] chooses a heuristic randomly at each selection phase. If the selected heuristic improves the solution at hand, it is automatically applied in the next step. This selection mechanism is generalised by using an additional parameter (i) denoting the number of consecutive iterations of an improving heuristic to be applied. Therefore, if a selected heuristic improves a solution, then it will be applied for i consecutive steps without evaluating its improvement capabilities.

Move acceptance criteria: simulated annealing and late acceptance

Simulated annealing and late acceptance are two move acceptance criteria from the literature. The simulated annealing acceptance method has been implemented based on [48]. It accepts improving solutions for intensification. Equal quality solutions are accepted for diversification. Worsening solutions are also accepted if their fitness qualities are lower than a linearly decreasing threshold value. The second move acceptance criterion is a relatively new approach, i.e. late acceptance proposed by [56]. The pseudocode of this approach is presented in Algorithm 2. The idea is to maintain a list of fitness values using the previously acceptance solutions. It requires only one parameter, the list length (L). A fitness value is selected from the list using the current iteration number (I) and the list length at each iteration. The index returns from (I mod L) is used to choose the fitness value ($f(S_v)$) for deciding whether to accept or reject new solutions. Solutions with lower fitness values than the chosen fitness values are accepted and the list is updated.

Algorithm 2: Late acceptance

Input: list length L, number of iterations elapsed $I = 1$

```
1 v = I mod L
2 if f(S') ≤ f(S_v) then
3 |   S ← S'
4 |   f(S_v) = f(S)
5 |   I++
end
```

4.2.2 Performance of the hyper-heuristics with new heuristic selection and mentoring strategies

15 hyper-heuristics were tested on the patient admission scheduling problem using Pentium Core 2 Duo 3 GHz PCs with 3.23 GB memory. We carried out 10 trials per instance with 1 hour of execution time. Six different learning rates $\lambda_1 = \{0.0001, 0.00025, 0.0005, 0.001, 0.002, 0.003\}$ were used for the learning automata. P was set to 10 for the combined simple random-learning automata hyper-heuristics. Different iteration values were employed for the random descent iteration move acceptance: $i = \{1, 3, 5, 10, 30, 50\}$. In addition, the list length of the late acceptance mechanism, was set to $L = \{100,200,500,\}$. The experiments were conducting using a heuristic set with the 4 low-level heuristics described in Chapter 3, Section 3.4.2.

Table 4.12 presents the average best results for different hyper-heuristics. These results show that learning-based hyper-heuristics components are sufficient to beat the simple random (SR)-simulated annealing (SA) hyper-heuristic, which was the best hyper-heuristic [48], ($HH14$, $HH15$). Table 4.13 demonstrates the best results after 10 trials. The tested approaches were still superior to SR-SA.

In Table 4.14, a ranking of the average best results concerning all the tested hyper-heuristics is presented. Among all these hyper-heuristics, we see that HH1, with average ranking 4.29, is the best hyper-heuristic for solving the patient admission scheduling problem. In HH1, the learning automata perform well with tournament factor 4. Also, the comparison between HH1 vs. HH2 and HH3 vs. HH4 show that letting learning automata control the probability vector and adapt the tournament factor at the same time, does not seem a good strategy for the current settings. On the other hand, using learning automata for adapting the tournament factors only, helps to increase the performance. This is apparent in HH12 vs. HH14 and HH13 vs. HH15. Concerning the move acceptance mechanisms, if we compare HH11 with HH14, we can see that it is possible to find better results with late acceptance than with simulated annealing. However, we cannot show significant performance differences between them. It should be noticed that the simulated annealing algorithm does not require any user-defined parameter.

In Table 4.15, the best results of the hyper-heuristics are compared for each patient admission scheduling instance. HH7 with ranking 4.93 is the best strategy for finding the best solutions after 10 runs. HH1 is the third best hyper-heuristic, when considering the overall best solutions. However, it is impossible to point at a statistically significant difference. HH7, shows the effect of employing learning automata for heuristic selection and the performance increment due to the division of the heuristic selection process into three phases.

	Inst0	**Inst1**	**Inst2**	**Inst3**	**Inst4**	**Inst5**	**Inst6**
HH1	892.88	800.42	1350.52	**884,22**	1542.46	**651.60**	927.94
STD	17.17	23.65	39.54	**21.88**	42.86	**3.87**	23.26
HH2	901.80	817.26	1370.90	887.10	1538.54	651.76	928.90
STD	17.36	47.48	24.66	22.57	44.01	4.53	12.78
HH3	897.78	806.16	1354.78	891.80	1549.32	651.76	928.36
STD	17.16	29.11	24.40	14.50	44.47	3.96	23.70
HH4	899.90	806.56	1360.52	901.14	1542.80	652.24	933.74
STD	26.20	37.85	18.42	16.92	31.46	5.05	10.21
HH5	918.62	795.20	**1307.04**	894.90	1473.26	664.32	939.52
STD	28.38	23.97	**26.53**	17.80	43.27	4.73	10.74
HH6	911.24	807.38	1311.96	895.92	**1466.20**	665.04	947.48
STD	22.46	23.00	33.29	25.93	**25.94**	5.03	33.93
HH7	893.16	797.56	1362.32	885.02	1537.22	652.32	**926.78**
STD	27.81	29.18	36.72	20.27	49.73	3.64	**27.26**
HH8	895.44	802.90	1355.84	895.78	1541.42	652.56	936.16
STD	16.78	16.71	43.27	15.50	33.66	6.58	19.55
HH9	910.38	**792.88**	1360.10	895.42	1535.50	652.72	941.78
STD	15.53	**27.67**	36.86	33.51	34.44	6.81	21.49
HH10	900.58	798.22	1359.50	905.28	1537.68	654.08	935.96
STD	24.31	21.72	26.97	15.04	57.52	3.23	32.06
HH11	919.74	796.06	1326.02	900.02	1478.30	660.72	949.70
STD	29.15	22.90	16.13	23.62	28.94	4.70	23.39
HH12	**890.96**	798.24	1367.16	893,18	1545.90	652.32	927.02
STD	**17.47**	23.94	29.85	29.96	35.55	3.20	15.68
HH13	892.02	807.36	1356.92	892.62	1549.56	655.36	931.80
STD	16.51	26.85	39.28	23.05	37.82	4.54	23.06
HH14	920.30	821.96	1362.70	894.42	1541.46	654.80	941.66
STD	12.36	24.99	23.69	20.95	32.44	6.38	27.58
HH15	919.12	819.64	1372.34	915.42	1537.48	655.20	938.06
STD	30.73	35.76	50.89	24.63	45.14	4.73	16.70

Table 4.12: The average best results for the hyper-heuristics (the best results are in bold)

However, another learning automata-simple random (HH8) combination does not perform that well. HH7 updates the probability vector even in the first phase, where SR is used as the heuristic selection mechanism. Therefore, the learning process starts from the beginning even though the information learnt is not used during the first phase. Since a learning automaton is an incremental learning strategy, initial changes to the probability vector can be misleading. However, even if the information learnt in the first phase can be misleading, it is helpful to find the best (optimal) heuristic among all the heuristics utilised.

A Wilcoxon test with a 95% confidence interval was used to check the performance difference between HH1 and the other hyper-heuristics. HH1 appears to be significantly better than HH3, HH4, HH8, HH13, HH14 and

	Inst0	**Inst1**	**Inst2**	**Inst3**	**Inst4**	**Inst5**	**Inst6**
HH1	859.2	757.6	1302.6	855.4	1486.2	642.4	890.2
HH2	878.6	763.0	1309.0	856.0	1472.0	644.8	887.8
HH3	863.0	755.2	1310.8	862.2	1471.6	643.2	897.6
HH4	861.2	752.0	1313.0	869.6	1463.4	641.6	896.6
HH5	879.6	753.4	**1249.4**	861.0	**1402.0**	651.2	896.2
HH6	864.0	751.8	1252.2	853.2	1433.0	656.0	907.2
HH7	863.2	743.2	1294.4	858.4	1450.8	644.8	**880.2**
HH8	869.6	758.0	1306.4	860.8	1474.0	643.2	899.8
HH9	**858.2**	**723.2**	1300.4	860.0	1481.2	**640.0**	902.6
HH10	869.0	741.6	1297.2	860.0	1438.2	644.8	899.0
HH11	878.6	768.8	1287.8	863.0	1421.0	654.4	923.2
HH12	864.4	758.2	1312.2	**847.2**	1455.0	642.4	890.0
HH13	861.6	741.6	1313.4	862.6	1492.4	641.6	895.2
HH14	902.8	771.2	1330.0	861.2	1496.0	645.6	913.2
HH15	879.0	763.6	1314.0	886.4	1460.2	647.2	920.2

Table 4.13: The best results for the hyper-heuristics (the best results are in bold)

	Inst0	**Inst1**	**Inst2**	**Inst3**	**Inst4**	**Inst5**	**Inst6**	**AVG**
HH1	3	7	4	1	11	1	3	**4.29**
HH2	9	13	14	3	8	2.5	5	7.79
HH3	6	9	5	4	14	2.5	4	6.36
HH4	7	10	10	13	12	4	7	9
HH5	12	2	1	8	2	14	11	7.14
HH6	11	12	2	11	1	15	14	9.43
HH7	4	4	11	2	5	5.5	1	4.64
HH8	5	8	6	10	9	7	9	7.71
HH9	10	1	9	9	4	8	13	7.71
HH10	8	5	8	14	7	9	8	8.43
HH11	14	3	3	12	3	13	15	9
HH12	1	6	13	6	13	5.5	2	6.64
HH13	2	11	7	5	15	12	6	8.29
HH14	15	15	12	7	10	10	12	11.57
HH15	13	14	15	15	6	11	10	12

Table 4.14: The average ranking of the average best results

	Inst0	**Inst1**	**Inst2**	**Inst3**	**Inst4**	**Inst5**	**Inst6**	**AVG**
HH1	2	9	7	3	13	4,5	4	6,07
HH2	11,5	12	9	4	10	9,5	2	8,29
HH3	5	8	10	11	9	6,5	8	8,21
HH4	3	6	12	14	8	2,5	7	7,5
HH5	14	7	1	9	1	13	6	7,29
HH6	7	5	2	2	3	15	12	6,57
HH7	6	4	4	5	5	9,5	1	**4,93**
HH8	10	10	8	8	11	6,5	10	9,07
HH9	1	1	6	6	12	1	11	5,43
HH10	9	2	5	7	4	8	9	6,29
HH11	11,5	14	3	13	2	14	15	10,36
HH12	8	11	11	1	6	4,5	3	6,36
HH13	4	3	13	12	14	2,5	5	7,64
HH14	15	15	15	10	15	11	13	13,43
HH15	13	13	14	15	7	12	14	12,57

Table 4.15: The average ranking for the best results

HH15. However, although HH1 works better than the other hyper-heuristics, there is no a statistical difference.

4.3 A selection hyper-heuristic with heuristic selection and adaptive acceptance

In this section, we propose a simple learning mechanism to exclude poor heuristics for a short period. The strategy was used within a number of hyper-heuristics and tested over a set of home care scheduling problem instances. The home care scheduling problem is a relatively new vehicle routing variant described in Chapter 3, Section 3.2.2. The experimental results show that excluding worse performing heuristics is useful to find elite heuristic subsets in different phases of a search.

The proposed dynamic heuristic set strategy is explained in Section 4.3.1. Then, the devised move acceptance criterion is presented in Section 4.3.2. Next, the experimental results considering the performance of the proposed mechanisms are discussed in Section 4.3.3.

4.3.1 Heuristic (subset) selection: a dynamic heuristic set strategy

In this study, we developed a simple learning approach, i.e. dynamic heuristic set (DHS) strategy, to exclude some heuristics for a number of phases. Phases are determined by a number of iterations. After n iterations, a snapshot of the heuristics' performance is taken. Based on that knowledge, some heuristics are excluded. The pseudocode of the learning strategy is presented in Algorithm 3. n is the number of heuristics. d shows the fixed tabu duration value. num_of_iter refers to the current iteration number; ph_{iter} means the phase length in terms of the number of iterations during the current phase; $count(i)_{newBest}$ indicates the number of new best solutions found by heuristic i during the current phase; imp_i is the total improvement provided by heuristic i during the current phase; wrs_i is the total worsening caused by heuristic i during the current phase; t_i shows the spent total time by heuristic i during the current phase; $TDList(i)$ represents the tabu duration of heuristic i.

Heuristics are scored based on their performance in [266]. The average of all scores was used to divide the heuristic set into two. Heuristics with a higher or equal score than the average value are considered to be improving heuristics. Heuristics with a lower value than the average are considered worsening heuristics. Instead of directly using such a scoring approach, we will learn a quality index (QI) for each heuristic. This value expresses the quality of the heuristics during one phase and helps to determine performance differences. If the employed performance metric is different from zero for heuristic i, then its QI_i is set to a value based on its order among the other heuristics. Otherwise, QI_i is automatically set to 1. This means that better heuristics have higher the QI values. The highest possible value is equal to the number of heuristics (n) and the lowest value is 1. At the end of each phase, referring to a predetermined number of iterations, QI values are set and the average QI (avg) is calculated. For the calculation of avg, tabu heuristics additionally contribute with $QI = 1$. This helps the learning strategy to keep a number of heuristics non-tabu in any case. The heuristics with a QI value smaller than avg are excluded from the set for a pre-determined number of phases (d: tabu duration).

$$avg = \left\lfloor \left(\sum_{i}^{n} QI_i \right) / n \right\rfloor \tag{4.3}$$

Although the QI values rank the heuristics based on given performance criteria, they do not reflect the exact performances of the heuristics. However, QI values

Algorithm 3: The Learning Strategy

Input: $d > 0 \wedge ph_{iter} > 0$

```
1  if (num_of_iter%ph_iter) = 0 then
2      for i ← 1 to n do
3          M_1(i) = count(i)_newBest;
4          if t_i > 0 then
5              M_2(i) = imp_i/t_i;
6              M_3(i) = −(wrs_i/t_i);
7          else
8              M_2(i) = M_3(i) = 0;
           end
       end
9      for i ← 1 to n do
10         for j ← 1 to 3 do
11             if M_j(i) ≠ 0 then
12                 QI_i = 1;
13                 for k ← 1 to n do
14                     if i ≠ k and TDList(k) = 0 and QI_k = 0 then
15                         if M_j(i) > M_j(k) then
16                             QI_i = QI_i + 1;
17                         else if j = 1 and M_j(i) = M_j(k) and M_2(i) > M_2(k)
                           then
18                             QI_i = QI_i + 1;
                           end
                       end
                   end
19                 break;
               end
           end
20         if QI_i = 0 then
21             QI_i = 1;
           end
       end
22     for i ← 1 to n do
23         if TDList(i) = 0 then
24             if QI_i < avg(QI) then
25                 TDList(i) = d;
               end
26         else
27             TDList(i) = TDList(i) − 1;
           end
       end
   end
```

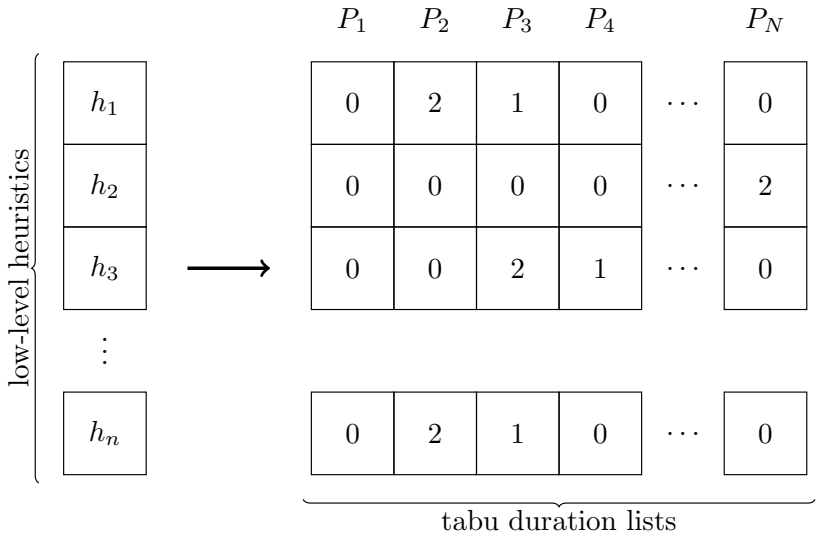

Figure 4.4: An example of the prohibition mechanism with tabu duration of 2 ($d = 2$)

are effective and accurate enough for the proposed learning strategy.

In Figure 4.4, an example of DHS is shown. The P_i denote the phases. At each phase, the tabu duration of each heuristic is updated if necessary. If a non-tabu heuristic is determined to be tabu for the next phase, its tabu duration is set to a fixed value (d). If a heuristic is tabu, its tabu duration is decremented by 1 after every phase until it is zero. In the given example, all the heuristics are non-tabu during P_1. During the second phase, h_1 and h_n have been set tabu. They are excluded from the heuristic set for the next two phases (tabu duration = 2). Their tabu durations are decreased by 1 and another heuristic, h_3, is excluded during the next phase. This procedure continues until the termination criteria are met.

Performance metrics involved in learning

Different elements were used to measure the performance of a heuristic. These elements are mainly related to improvement capabilities of the employed heuristics. However, taking only the improvement/worsening behaviour of the heuristics into account can be misleading. It can be useful to additionally consider the speed of the heuristics while generating a new solution.

Three dependent performance metrics were selected for determining tabu heuristics. M_1) The first comparison is made by checking the capabilities of the heuristics for improving the current best solution. Heuristics with a higher number of improvements over the current best solution have higher QI values. M_2) In the case of not providing any new best solutions, the QI values of the heuristics are determined based on their improvement capabilities per unit execution time. M_3) If some of the heuristics do not provide any improvement but worsen the solution, then the QI values of these heuristics are determined based on their worsening characteristics per unit execution time.

- M_1 : Number of new best solutions found
- M_2 : Total fitness improvement / execution time
- M_3 : Total fitness worsening / execution time

Whenever num_of_iter reaches ph_{iter} and its multiples, the three performance metrics are checked for the non-tabu heuristics. Based on these values, the QI value of each heuristic is determined as explained in Section 4.3.1. After that, some heuristics are made tabu for d phases and the tabu durations of the tabu heuristics are decreased by 1.

Size of the heuristic subsets

Based on the proposed learning strategy, the size of a heuristic subset or the number of non-tabu heuristics have limits. If none of the heuristics makes a change to a solution, then the QI value of each heuristic is set to 1. This means that avg is 1. Since $avg = 1$ is the lowest possible value, no heuristic will be excluded. On the other hand, if all the heuristics are non-tabu and all have a different performance, $avg = (n + 1)/2$. If $avg \geq 2$, at least one heuristic is excluded. From this limit, the formula 4.4 can be derived. x shows the total number of non-tabu heuristics. The total of the non-tabu heuristics' QI values is calculated as $(x(x + 1)/2)$. The total of the tabu heuristics' QI values is calculated as $(n - x)$, since each of their QI is 1.

$$\frac{(n-x) + x(x+1)/2}{n} \geq 2 \tag{4.4}$$

After simplification it turns into formula 4.5.

$$x(x-1) \geq 2n \tag{4.5}$$

If this formula is valid, then it is possible to exclude some heuristics. When it is violated, the size of the heuristic subset reaches its lowest limit. For instance, for a large enough tabu duration ($d = \infty$), which prevent the tabu heuristics from being added back to the heuristic set before the number of tabu heuristics reaches its maximum limit, the minimum size of a heuristic subset is 3 for n=5, 4 for n=10 and 14 for n=100. As seen from the given examples, the minimum heuristic set for small n is some kind of exclusion mechanism. On the other hand, for a larger n, it behaves like a temporary heuristic reduction strategy. For larger heuristic sets, higher tabu duration values should be used. Otherwise, the number of non-tabu heuristics will be too large for an elite heuristic set. The phase length should also be determined based on the number of heuristics. For instance, if $n = 100$, using $ph_{iter} = 1000$ is not a reasonable choice. At best, each heuristic will be called only 10 times. Thus, making a meaningful comparison between the heuristics will not be possible, especially if most of the heuristics are non-tabu.

4.3.2 Move acceptance: adaptive iteration limited threshold accepting

AILTA is an adaptive version of ILTA (presented in Section 4.1) that increases the value of R in case it is hard to improve the current best solution. For that purpose, another iteration limit is added. This limit determines when R should be increased. To prevent the value of R of becoming very large, an upper bound (R_n) for R is provided. Whenever the current best solution is improved, R is set to its initial value. The adaptation process is visualised in Figure 4.5.

4.3.3 Performance of hyper-heuristic sub-mechanisms: DHS, AILTA

Three perturbative selection hyper-heuristics were used in different configurations from those in [238]. All the tested hyper-heuristics use the Simple Random (SR) heuristic selection mechanism. Since SR selects heuristics in a random uniform manner, all the heuristics get equal opportunity to be explored. For the move acceptance part, one hyper-heuristic involves Improving or Equal (IE). The other two use Iteration Limited Threshold Accepting (ILTA) and Adaptive ILTA (AILTA) move acceptance strategies. The aim is to show performance changes over a number of hyper-heuristics.

We tested the hyper-heuristics over 6 home care scheduling problem instances on Pentium Core 2 Duo 3 GHz PCs with 3.23 GB memory. The tests are repeated

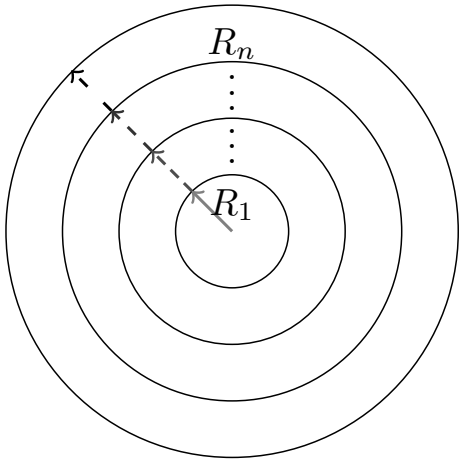

Figure 4.5: Extending the acceptable search space region for worsening solutions by increasing the range value from $R_1 \mapsto R_n$

10 times. The phase lengths are set to $\{1000, 2500, 5000, 10000\}$ iterations. The tabu duration is set to 1. For ILTA, k is 100. For AILTA, the second iteration limit is set to 5000 and the range value is adapted ($R_1 \mapsto R_n$) by an incrementing factor 1.001. The parameters regarding the move acceptance part determined after some preliminary experiments. No special parameter tuning process was applied.

The Table 4.16 and Table 4.17 provide the average fitness (AF) values and the average ranks (AR) belonging to different phase lengths concerning each hyper-heuristic. The experimental results with "N/A" denote the hyper-heuristics with no dynamic heuristic set.

Table 4.16(a) gives the results for SR-AILTA with R_1=1.003 and R_n=1.007 are given. The average rank values show that all the tested phase lengths improve the performance of the hyper-heuristic with an initially formed fixed heuristic set. Among them, 10000 looks like the best option. In Table 4.16(b), the ranking of the same hyper-heuristic with R_n=1.01 is presented. For this hyper-heuristic configuration, 2500 is a useful phase length for determining a dynamic heuristic set. The results found by the same hyper-heuristic with R_1=1.004 and R_n=1.007 are given in Table 4.16(c). Learning heuristic subsets for different phases on average generates superior results. All the tested phase lengths result in improvements. Among them, 2500 shows the best performance based on average ranks.

(a) SR-AILTA with $R_1 = 1.003$ $R_n = 1.007$

AF	PHASE LENGTH				
	N/A	1000	2500	5000	10000
hh	75121,94	74049,25	74831,22	74772,85	74378,76
hh1	43645,46	46726,00	44719,40	43466,85	42706,44
ll11	50458,76	47392,77	46408,61	47675,16	44787,40
ll2	9873,51	9650,13	9332,27	9420,47	9352,76
ll21	8732,83	9233,70	9174,38	9432,25	9330,76
ll3	8775,59	8843,06	8724,14	8688,68	8815,87
AR	3,67	3,50	2,50	3,00	**2,33**

(b) SR-AILTA with $R_1 = 1.003$ $R_n = 1.01$

AF	PHASE LENGTH				
	N/A	1000	2500	5000	10000
hh	74213,60	73995,66	74654,60	74808,14	74220,70
hh1	42654,80	43415,05	43942,63	43157,50	43001,02
ll11	45264,19	44730,06	44677,35	46197,30	44996,57
ll2	9712,77	9491,63	9326,69	9369,95	9322,01
ll21	8739,54	9194,20	8708,62	8935,51	8829,87
ll3	8636,19	8699,11	8500,31	8628,37	8789,47
AR	2,83	3,33	**2,33**	3,67	2,83

(c) SR-AILTA with $R_1 = 1.004$ $R_n = 1.007$

AF	PHASE LENGTH				
	N/A	1000	2500	5000	10000
hh	75685,42	74241,68	74200,68	74634,61	74019,18
hh1	43837,95	43675,98	43155,15	43448,52	43301,90
ll11	47291,00	47212,63	42774,24	44854,42	49183,66
ll2	9776,71	9488,96	9296,00	9393,11	9478,91
ll21	9057,60	8919,95	8947,66	9014,98	9086,86
ll3	8645,29	8574,99	8659,42	8716,54	8819,17
AR	4,17	2,67	**1,67**	3,00	3,50

Table 4.16: Average fitness values with ranks of different hyper-heuristics over 6 HCSP benchmark instances

(a) SR-ILTA with R=1.003

AF	PHASE LENGTH				
	N/A	**1000**	**2500**	**5000**	**10000**
hh	75958,19	78077,78	74911,15	78822,38	75234,02
hh1	50103,33	45625,38	45637,32	45234,76	45057,79
ll11	53289,69	46101,79	50750,60	47830,82	49135,72
ll2	10187,80	9703,55	9661,95	9652,46	9611,41
ll21	9016,99	9462,47	9474,32	9686,57	9595,09
ll3	8995,84	8984,76	9368,15	8998,98	8995,41
AR	3,67	2,50	3,33	3,33	**2,17**

(b) SR-ILTA with R=1.004

AF	PHASE LENGTH				
	N/A	**1000**	**2500**	**5000**	**10000**
hh	75155,32	77448,89	74835,87	75125,33	74513,35
hh1	44794,36	44407,52	44508,57	44662,96	44457,79
ll11	51109,00	48684,60	45194,42	45532,99	46548,59
ll2	10047,85	9599,21	9439,91	9685,93	9609,40
ll21	9303,31	9387,10	9433,68	9765,26	9233,92
ll3	8752,16	8793,18	8943,88	8933,82	9103,44
AR	3,67	2,83	**2,50**	3,50	**2,50**

(c) SR-IE

AF	PHASE LENGTH				
	N/A	**1000**	**2500**	**5000**	**10000**
hh	89286,77	87083,58	89609,35	88606,49	87620,99
hh1	54778,81	55659,93	53259,56	50893,50	53326,32
ll11	104670,45	76356,56	81872,82	85122,85	80558,46
ll2	10339,69	9826,82	9909,80	15365,12	15353,77
ll21	9418,10	9512,55	9816,59	9527,36	9507,30
ll3	9395,24	9570,64	9770,43	9784,44	9765,54
AR	3,00	**2,17**	3,50	3,67	2,67

Table 4.17: Average fitness values with ranks of different hyper-heuristics over 6 HCSP benchmark instances

Table 4.17(a) indicates the average performance of SR-ILTA with R=1.003. The ranking values based on the average fitness values show that all the phase length values are useful for determining elite heuristics subsets. Among them, 10000 performs best. In Table 4.17(b), the performance of the same hyper-heuristic with R=1.004 was improved using different phase lengths. Among them, 2500 and 10000 have the same average rank as the best options.

The last hyper-heuristic, SR-IE, was also tested for different phase lengths. For this hyper-heuristic, 1000 provides the best improvement with respect to the average ranks.

The comparisons are made based on a fixed phase length value. In the case of taking different phase lengths into account for the same hyper-heuristic configuration, the performance improvement coming from the learning strategy appears to be higher.

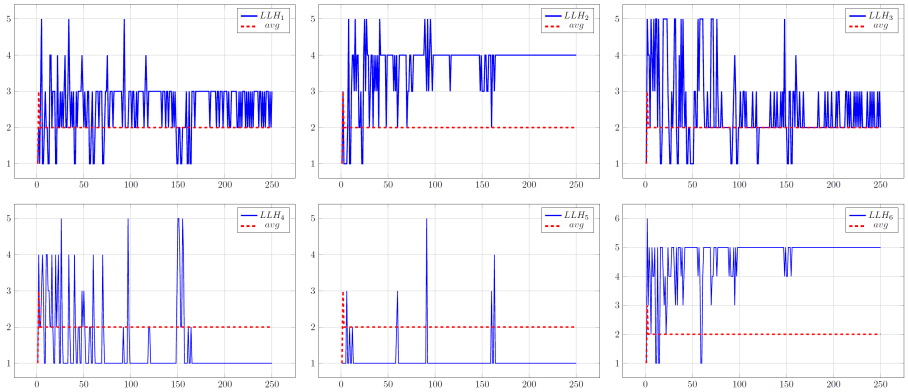

Figure 4.6: Sample QI values during the first 250 phases over the hh by SR-AILTA $R_1 = 1.003$ $R_n = 1.007$ with $d = 1$ and $ph_{iter} = 2500$

Bench.	HH	Phase Length	Fitness
hh	AILTA R_1=1.003 R_n=1.01	1000	73995,66
hh1	AILTA R_1=1.003 R_n=1.01	N/A	42654,80
ll11	AILTA R_1=1.004 R_n=1.007	2500	42774,24
ll2	AILTA R_1=1.004 R_n=1.007	2500	9296
ll21	AILTA R_1=1.003 R_n=1.01	2500	8708,62
ll3	AILTA R_1=1.003 R_n=1.01	2500	8500,31

Table 4.18: The Average Best Results for the HCSP instances

Bench.	**HH**	**Phase Length**	**Fitness**
hh	AILTA R_1=1.003 R_n=1.01	10000	71981,80
hh1	AILTA R_1=1.004 R_n=1.007	N/A	40644,60
ll11	AILTA R_1=1.003 R_n=1.007	2500	39046,80
ll2	AILTA R_1=1.004 R_n=1.007	1000	8926,25
ll21	AILTA R_1=1.003 R_n=1.01	2500	8372,40
	ILTA R=1.003	N/A	
ll3	AILTA R_1=1.003 R_n=1.01	2500	8047,15
	AILTA R_1=1.004 R_n=1.007	2500	

Table 4.19: The Best Results for the HCSP instances

Table 4.18 shows the average best results for the tested instances. All the average best results after 10 trails excluding *hh1* were improved. In addition, Table 4.19 provides some new best results. The current best result of the *hh1* could not be improved and for the *ll3* the previous best result is found again. For the rest, new best results were reached.

In Figure 4.6, sample QI values for each heuristic by SR-AILTA $R_1 = 1.003$ $R_n = 1.007$ with $ph_{iter} = 2500$ are presented. These values show how the performance of the heuristics changes in time. Especially, LLH_4 and LLH_5 perform worse than the other heuristics. This causes them to be frequently excluded from the heuristic set.

All these results show that the proposed learning strategy can provide improvement over different hyper-heuristics. SR-IE without a dynamic heuristic set has been improved by the learning strategy. For the other hyper-heuristics that perform significantly better than SR-IE, it is still possible to provide improvement via the dynamic heuristic set. That is, regardless of the characteristics of the hyper-heuristics, the learning strategy can help them to fill the gap between their current performance and their potential on a given heuristic set.

The computational results were generated using a tabu duration of 1. This means that each excluded heuristic returns to the heuristic set just after one phase. For that reason more experiments were carried out to show the effect of different tabu durations $d = \{2, 3, 4, 5, 6\}$. However, we did not see a significant performance difference among the hyper-heuristics with $d = \{1, 2, 3, 4\}$. The performance decrease starts with $d = 5$. With $d = 6$, the quality of the solutions decreases in a significant way. This shows that there can be a range of tabu durations behaving similarly. Outside of that range, the performance of the hyper-heuristic can decrease significantly.

AF	TABU DURATION					
	1	**2**	**3**	**4**	**5**	**6**
hh	74654.60	74294.45	73848.02	74457.14	73771.55	74643.89
hh1	43942.63	43213.85	43096.84	43675.20	43747.53	44921.41
ll11	44677.35	44080.03	42614.54	43730.65	45522.86	44714.58
ll2	9326.69	9410.14	9406.52	9356.36	9358.18	9372.62
ll21	8708.62	8733.55	8780.35	9168.57	8906.54	9154.57
ll3	8500.31	8604.94	8717.98	8574.86	8668.40	8610.33
AR	**3,00**	3,17	**3,00**	3,17	3,83	4,83

Table 4.20: SR-AILTA with $R_1 = 1.003$ $R_n = 1.01$ with $ph_{iter} = 2500$ and $d = \{1, 2, 3, 4, 5, 6\}$

4.4 A selection hyper-heuristic with a new move acceptance criterion

The present section presents a new move acceptance criterion, i.e. an adaptive list-based acceptance mechanism. Besides the new hyper-heuristic, a group of hyper-heuristics composed of certain referenced methods has been tested on the ready-mixed concrete delivery problem. A performance analysis has been carried out. The details about the problem are presented in Chapter 3, Section 3.3. Section 4.4.1 discusses both the new move acceptance criterion and the selection hyper-heuristics from the literature. The experimental results will be analysed in Section 4.4.2.

4.4.1 Tested selection hyper-heuristics

A hyper-heuristic using the simple random (SR) selection mechanism [105] and a new move acceptance strategy, namely adaptive iteration limited list-based threshold accepting with a fixed limit (AILLA-F), was constructed. SR randomly chooses heuristics and AILLA-F accepts solutions which are visited by the selected heuristics using a list of previously found best solutions ($best_{list}$). The literature describes similar strategies maintaining a history of fitness values as a list, for example list-based threshold accepting [209] and late acceptance [264].

AILLA-F maintains a fixed-sized (l) list of previously visited new best solutions. Thus, whenever a new best solution is found the new value is added to the beginning of the list, pushing out the oldest at the bottom. The underlying idea is to use proper threshold values on differing regions of the search space. Earlier

Algorithm 4: AILLA-F move acceptance

Input: threshold index $i = 1$, iteration limit $k \geq 0$, iteration limit to increase the threshold level $K \geq k$, list length $l > 0$; $k = 5$, $K = 125$, $l = 10$

for *j=0* **to** *l-1* **do** $best_{list}(j) = f(S_{initial})$
if $adapt\_iterations \geq K$ **then**
  if $i < l - 1$ **then**
    $i++$
  end
end
if $f(S') < f(S)$ **then**
  $S \leftarrow S'$
  $w\_iterations = 0$
  if $f(S') < f(S_b)$ **then**
    $i = 1$
    $S_b \leftarrow S'$
    $w\_iterations = adapt\_iterations = 0$
    $best_{list}.remove(last)$
    $best_{list}.add(0, f(S_b))$
  end
else if $f(S') = f(S)$ **then**
  $S \leftarrow S'$
else
  $w\_iterations++$
  $adapt\_iterations++$
  if $w\_iterations \geq k$ *and* $f(S') \leq best_{list}(i)$ **then**
    $S \leftarrow S'$ and $w\_iterations = 0$
  end
end

versions of AILLA-F [240, 232] determined a threshold level based on a constant value and the current best solution of the search. In AILLA-F the list size is given as a parameter to reduce its user-dependency, rather than using a constant value that directly affects the threshold value. Another critical component of AILLA-F is its iteration limit. It postpones the diversification decision by checking a fixed number of neighbouring solutions (k) before accepting a worse one. If no new best solution is found, it is assumed unlikely to be found during the current search process. Then a worsening solution is accepted based on the current threshold value. Moreover, if the employed threshold value is insufficient to discover new best solutions after K iterations, a larger value from the list is used as the threshold value. These threshold fluctuations continue

until either the largest value in the list is applied or a new best solution is found. The pseudocode of AILLA-F is presented in Algorithm 4. S denotes the current solution, S' is the new solution generated after the selected heuristic is applied and S_b refers to the current best solution. The function f demonstrates the quality of a given solution. *w_iterations* is the number of consecutively visited worsening solutions. *adapt_iterations* is a counter deciding whether the threshold level should be increased.

Hyper-heuristics using the same selection mechanisms with four other move acceptance mechanisms were used for comparison purposes. These mechanisms are simulated annealing (SA) [119], great deluge (GD) [119], late acceptance (LATE) [120] and improving or equal (IE) [19]. IE, as the name suggests, accepts only improving or equal quality solutions. The rest of the hyper-heuristics also use IE. In addition they may accept worsening solutions based on certain time related threshold values either those decreasing in time as in SA or GD, or using data gathered during the run like LATE.

4.4.2 Experiments

The group of hyper-heuristics (SR-AILLA-F, SR-SA, SR-GD, SR-LATE, SR-IE) were applied 10 times to the constructed initial solutions representing 26 real-world RMC problem instances (Chapter 3, Section 3.3.1). The total execution time was limited to 10 minutes for each run. The best two hyper-heuristics were additionally tested with 1 hour of execution time. The experiments were carried out using a Pentium Core 2 Duo 3 GHz PC with 3.23 GB memory.

Computational results

Table 4.22 shows the average fitness values of each hyper-heuristic on the tested instances whereas Table 4.21 indicates a simple ranking based on these average performance values. The hyper-heuristics are ranked SR-AILLA-F, SR-LATE, SR-SA, SR-IE and SR-GD from the best to the worst based on their average performance. Table 4.21 and Table 4.22 indicate that the two best hyper-heuristics use list-based threshold accepting strategies. This demonstrates the helpfulness of studying hyper-heuristics' past-behaviour when making further judgements on diversification. These two hyper-heuristics also perform significantly better than the other three hyper-heuristics with a confidence interval of 95% based on the Wilcoxon test. Although SR-AILLA-F outperforms SR-LATE for the majority of instances, the difference is not statistically significant when considering the overall performance. For only 5 instances SR-AILLA-F performs significantly better than SR-LATE.

Instances	**SR-AILLA-F**	**SR-LATE**	**SR-SA**	**SR-GD**	**SR-IE**
p2011-01-10	1.0	3.0	5.0	2.0	4.0
p2011-01-11	1.0	2.0	4.0	5.0	3.0
p2011-01-12	2.0	5.0	1.0	4.0	3.0
p2011-01-13	1.0	2.0	3.0	5.0	4.0
p2011-01-14	2.0	1.0	3.0	5.0	4.0
p2011-01-18	1.0	2.0	3.0	5.0	4.0
p2011-01-19	1.0	4.0	2.0	5.0	3.0
p2011-01-20	2.0	1.0	3.0	5.0	4.0
p2011-01-21	2.0	1.0	3.0	5.0	4.0
p2011-01-25	2.0	1.0	3.0	5.0	4.0
p2011-01-26	1.0	2.0	3.0	5.0	4.0
p2011-01-27	1.0	2.0	4.0	5.0	3.0
p2011-01-28	1.0	2.0	3.0	5.0	4.0
p2011-02-01	1.0	2.0	3.0	5.0	4.0
p2011-02-02	1.0	3.0	4.0	2.0	5.0
p2011-02-03	1.0	2.0	3.0	5.0	4.0
p2011-02-07	1.0	2.0	3.0	5.0	4.0
p2011-02-08	1.0	2.0	3.0	4.0	5.0
p2011-02-09	2.0	1.0	5.0	3.0	4.0
p2011-02-10	3.0	4.0	1.0	5.0	2.0
p2011-02-11	1.0	2.0	3.0	5.0	4.0
p2011-02-14	1.0	2.0	3.0	4.0	5.0
p2011-02-15	1.0	5.0	2.0	4.0	3.0
p2011-02-16	1.0	2.0	3.0	5.0	4.0
p2011-02-18	2.0	1.0	5.0	3.0	4.0
p2011-02-21	2.0	1.0	3.0	5.0	4.0
AVG	**1.38**	2.19	3.12	4.46	3.85

Table 4.21: Ranking of the hyper-heuristics based on their average performance

Instances	SR-AILLA-F		SR-LATE		SR-SA		SR-GD		SR-IE	
	AVG	STD	AVG	STD	AVG	STD	AVG	STD	AVG	STD
p2011-01-10	**3783.85**	0.52	3829.15	145.50	3874.75	192.90	3828.75	2105.76	3837.65	696.88
p2011-01-11	**42308.55**	517.84	42547.42	818.30	43376.02	1092.83	44097.92	53050.34	43054.42	49643.52
p2011-01-12	298283.35	15846.13	366837.02	37818.51	**270496.22**	23940.32	359933.61	23268.58	314874.22	29613.89
p2011-01-13	**33431.68**	1549.30	33787.85	1583.88	34073.59	1610.74	36643.53	3912.76	34654.69	2065.12
p2011-01-14	30653.32	1689.16	**30185.73**	1538.58	30804.45	2017.61	41820.34	3641.37	33580.48	3360.66
p2011-01-18	**68635.28**	3094.89	69583.50	2193.09	73818.34	3906.36	78536.53	138634.46	74115.77	4140.76
p2011-01-19	**95806.79**	14695.65	110216.07	6790.16	102069.11	9294.39	132954.89	8088.98	102966.30	8758.61
p2011-01-20	94802.50	4333.84	**90405.56**	5608.52	100130.03	8602.82	110974.18	4917.97	109269.06	11244.14
p2011-01-21	62751.37	1174.61	**62518.80**	1751.25	64076.47	2338.30	64976.16	1120.64	64783.31	3250.76
p2011-01-25	50591.05	2643.41	**48687.56**	1758.78	52140.23	1826.53	55967.50	112545.13	55205.67	2969.21
p2011-01-26	**71658.31**	2093.15	72209.63	1559.61	76691.32	2836.23	81691.65	94452.10	79853.11	112677.12
p2011-01-27	**78696.14**	4434.09	80972.00	1846.89	87373.60	9548.44	91418.55	3888.31	83162.08	2819.35
p2011-01-28	**40009.56**	1879.02	40461.41	1461.14	44268.95	2200.93	55732.31	4748.82	46977.88	3606.37
p2011-02-01	**38790.96**	1718.79	38967.25	1736.38	41940.66	2908.43	49864.53	2050.40	45459.96	3542.27
p2011-02-02	**24692.87**	550.05	24992.28	518.33	25171.48	508.80	24943.45	15600.81	25693.68	18044.14
p2011-02-03	**65749.17**	1745.81	65942.09	1493.24	67370.82	2444.06	73484.04	2072.34	70201.96	5123.40
p2011-02-07	**34654.58**	1010.02	34700.01	925.78	35017.62	1516.60	37427.25	89373.26	36707.73	78029.03
p2011-02-08	**18553.10**	87.29	18569.00	197.88	18676.10	316.24	18677.68	148.08	18855.70	288.38
p2011-02-09	39427.22	1179.49	**39088.37**	1194.40	40534.73	1763.98	39543.35	1266.16	39884.30	1156.13
p2011-02-10	66574.60	70.22	67437.70	779.76	**66513.10**	41.43	69441.40	23206.06	66550.80	24954.80
p2011-02-11	**51853.51**	3449.69	53825.82	2972.18	55322.35	4904.08	67621.69	3337.41	58926.42	4519.54
p2011-02-14	**59601.56**	2950.68	60426.73	1718.60	66759.76	9968.05	67664.46	2824.46	69954.17	8374.82
p2011-02-15	**591919.15**	26627.67	660075.33	22085.79	592718.33	41801.29	627683.22	30100.45	596502.97	48846.38
p2011-02-16	**104711.29**	3967.35	110962.91	4177.45	111031.32	7179.41	123034.98	12367.71	119981.34	9015.92
p2011-02-18	365809.87	61779.94	**357262.07**	11521.33	428190.88	38843.64	375636.81	419522.25	379902.35	578378.08
p2011-02-21	104091.79	15574.68	**92152.51**	3596.63	106756.11	12814.48	121561.69	11152.39	115964.24	10295.44

Table 4.22: Average fitness values with standard deviations for each hyper-heuristic

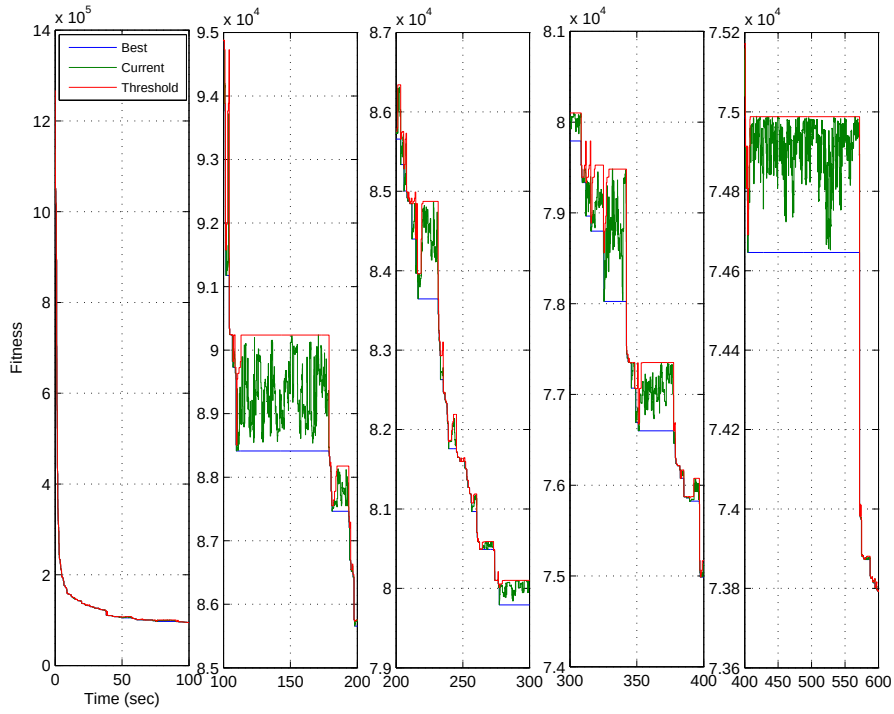

Figure 4.7: Best fitness, current fitness and threshold level changes by SR-AILLA-F during the run on p2011-01-27 (run 3)

Figure 4.7 shows the detailed progress of SR-AILLA-F on p2011-01-27. It shows the varying threshold levels, accepted worsening solutions and explored new best solutions for different time intervals. In different parts of the search space different threshold levels are required. Furthermore, in some cases, the threshold level is increased using the previously visited best fitness list accommodated by AILLA-F. Especially the second graph at 100-200 seconds and the fifth graph for 400-600 seconds show where the diversification stages struggle the most. AILLA-F spends more time freeing itself from the search regions surrounding the best solutions at these moments. Between these two time intervals diversification can be achieved relatively easily.

The top two hyper-heuristics from the 10-minutes experiments are now tested with 1 hour of execution time. The test results in Table 4.23 indicate that for longer execution times SR-AILLA-F still performs better than SR-LATE. The Wilcoxon test indicates, with 95% confidence, SR-AILLA-F and SR-LATE

Instances	SR-AILLA-F			SR-LATE		
	AVG	STD	%(10m-1h)	AVG	STD	%(10m-1h)
p2011-01-10	3783.75	0.85	0.00	**3782.55**	0.48	1.22
p2011-01-11	41457.12	299.84	2.01	**41325.45**	372.24	2.87
p2011-01-12	**217455.99**	20469.46	27.10	233072.96	18007.72	36.46
p2011-01-13	**31557.60**	1904.81	5.61	32171.19	2152.41	4.78
p2011-01-14	**27715.36**	2100.99	9.58	28106.35	2280.85	6.89
p2011-01-18	**64992.81**	1510.02	5.31	66909.32	719.75	3.84
p2011-01-19	**56455.19**	3325.83	41.07	59504.30	2387.96	46.01
p2011-01-20	**75576.76**	3438.29	20.28	83699.83	5580.40	7.42
p2011-01-21	**60749.08**	443.67	3.19	60806.60	902.09	2.74
p2011-01-25	**47507.55**	1864.27	6.09	48911.85	2855.42	-0.46
p2011-01-26	**68304.51**	1781.66	4.68	70836.96	1576.90	1.90
p2011-01-27	**71668.21**	1851.01	8.93	73507.16	1801.41	9.22
p2011-01-28	**34126.37**	1144.11	14.70	35419.53	2063.34	12.46
p2011-02-01	**35766.23**	647.63	7.80	36733.65	1456.72	5.73
p2011-02-02	**24418.18**	140.39	1.11	24588.97	320.00	1.61
p2011-02-03	**61222.13**	1524.57	6.89	63064.78	2312.47	4.36
p2011-02-07	33841.77	455.49	2.35	**33766.98**	498.59	2.69
p2011-02-08	**18463.90**	47.64	0.48	18467.70	46.08	0.55
p2011-02-09	37987.95	472.21	3.65	**37635.25**	209.55	3.67
p2011-02-10	**66541.90**	57.23	0.05	67450.50	792.42	-0.02
p2011-02-11	**49601.22**	553.27	4.34	50239.25	1472.44	6.66
p2011-02-14	**54578.45**	1187.21	8.43	56709.26	1558.92	6.15
p2011-02-15	500672.21	14764.59	15.42	**480298.53**	7223.04	27.24
p2011-02-16	**91456.18**	1608.41	12.66	92431.51	2230.97	16.70
p2011-02-18	300698.23	33498.32	17.80	**279121.50**	4453.98	21.87
p2011-02-21	**68985.05**	5659.49	33.73	72281.96	6161.38	21.56

Table 4.23: Average fitness values with standard deviations for the top two hyper-heuristics with 1 hour of execution time (improvement difference as percentage between 10 minutes and 1 hour of execution time shown under %(10m-1h))

performing significantly better on ten and four instances respectively. For the remaining instances neither of them performs significantly better than another. The third column for each hyper-heuristic in the table indicates that there is high fitness improvement potential between 10 minutes and 1 hour of execution time. The greatest degrees of improvement occurred on the problem instance, p2011-01-19. The first run of SR-AILLA-F returns solutions using 24 vehicles for 203 deliveries (10 minutes computation) and 22 vehicles for 201 deliveries (1 hour computation) as shown in Figure 4.9. Evidently the utilised resources are similar with the quality difference mostly arising from the lateness of the first deliveries as indicated in Figure 4.8. Therefore, employing specific heuristics aimed at improving this objective might be useful in providing continued improvements over long running times.

Figure 4.10 illustrates changes concerning the best fitness values obtained with the hyper-heuristics. In these results, SR-AILLA-F, SR-SA and SR-IE improve the initial solution quickly at the beginning of the search, whereas SR-LATE is comparatively slower. For SR-GD is delayed in comparison to other methods, yet it consistently improves the solution. During further iterations the performance of the hyper-heuristics changes. SR-LATE and SR-GD get close to the other hyper-heuristics' level of performance. SR-IE almost stops improving after the first 200 seconds because it lacks an effective diversification component.

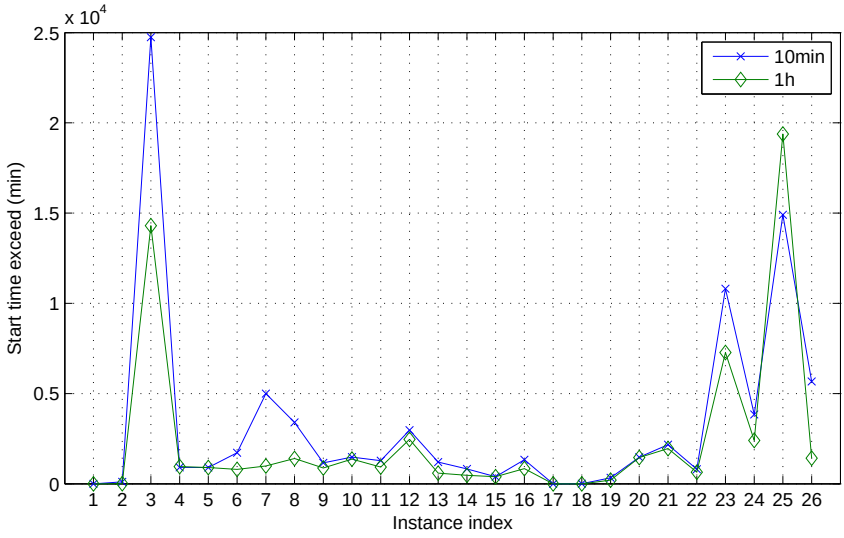

Figure 4.8: Total start time exceeded in minutes for each problem instance after 10 minutes and 1 hour of execution time by SR-AILLA-F (for the first runs)

Heuristic set Figure 4.11 shows the improvement behaviours of the heuristics during a single run. From this figure it can be deduced that each heuristic contributes to the improvement process. The graph on the left shows the number of improving solutions and the number that are new best solutions. LLH_1 and LLH_7 discover the highest number of improving and new best solutions. More than half of the improving solutions found by these heuristics are new best solutions. For the remaining heuristics, even if their improvement capabilities are comparatively worse based on the number of improving solutions found, the majority of their improving solutions represent new best solutions. However, these heuristics locate more worsening and equal quality solutions than the best two.

The graph on the right indicates the new best solutions found by each heuristic during the run. LLH_1, LLH_2, LLH_5 and LLH_7 are the best at finding new best solutions during the entire run. However the other heuristics are mostly useful during early iterations. In particular, LLH_4 finds new best solutions during the first 60 seconds and after that timeframe it reaches only one new best solution. Generally, the heuristic performance may change as the search continues. Thus, an intelligent selection mechanism may deliver better quality results.

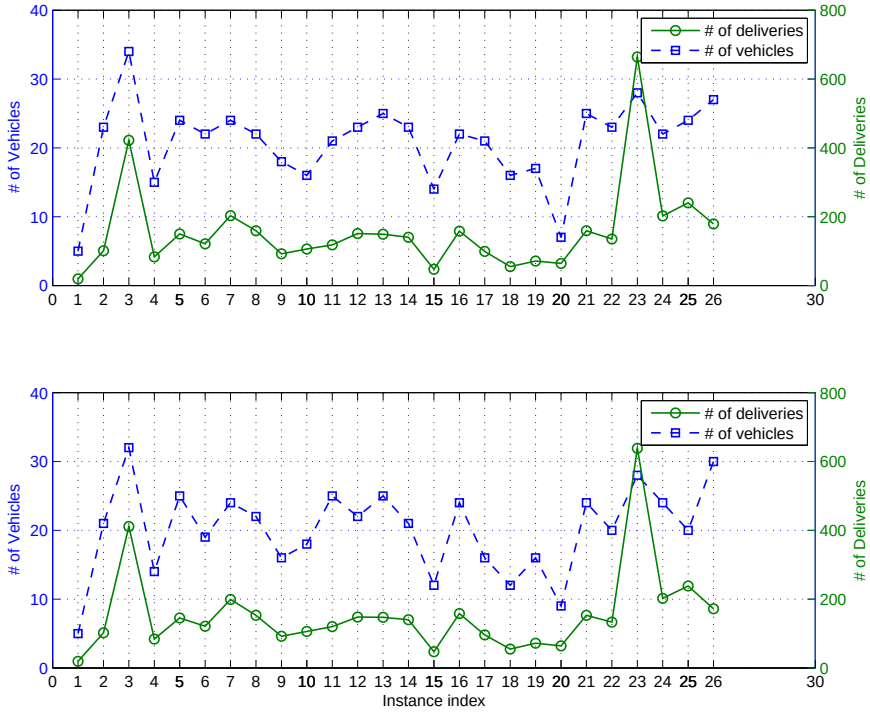

Figure 4.9: The number of vehicles and the number of deliveries for each solution regarding each problem instance found by SR-AILLA-F (for the first runs). The chart above belongs to the solutions generated after 10 minutes and the chart below shows the solutions found after 1 hour

4.5 Conclusion

The present chapter investigated various heuristic selection, move acceptance and mentoring mechanisms. Each hyper-heuristic using some of these sub-mechanisms was applied to one problem among the travelling tournament, home care scheduling, patient admission scheduling and ready-mixed concrete delivery problems. The analysis of these applications illustrated that learning and adaptation processes is a requirement to reach better quality results. In particular, the heuristic selection mechanism based on learning automata showed that using learning fastens the search beside finding higher quality solutions. In addition, generalising the random descent selection method's design delivered improving solutions by employing a learning component with a short-term

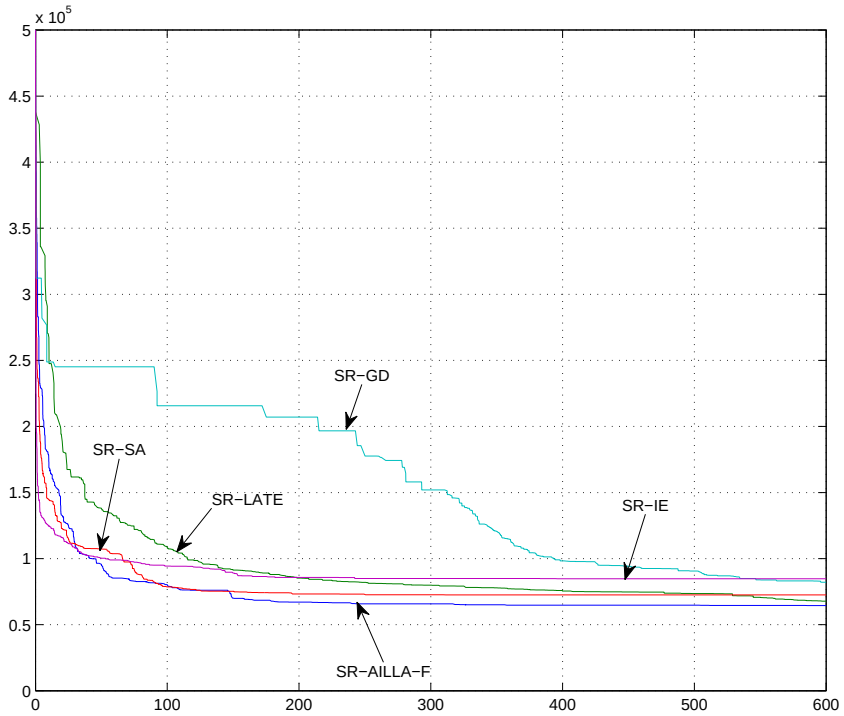

Figure 4.10: The best fitness changes during the run on p2011-01-18 (run 7)

memory. Moreover, the combination of learning automata and simple random showed that learning is required. However, at the same time, this combination exhibited that selecting heuristics randomly can provide further improvements during very easy and very hard to search regions of a search space. Rather than using a traditional selection approach, we additionally introduced a heuristic subset selection or heuristic exclusion strategy, i.e. dynamic heuristic set, to support the selection process. The performance of this approach showed that temporary heuristic exclusion methods can help to the search process by reducing the number of heuristic options.

The new approaches introduced for the acceptance strategy yielded that increasing the adaptiveness of the acceptance part provides better performance. In other words, increasing the dependency level of adaptiveness to a search space is an effective way of elevating the performance of a hyper-heuristic. The performance analysis of the proposed acceptance mechanisms, i.e. ILTA, AILTA and AILLA-F, provides the effect of adaptation in hyper-heuristics.

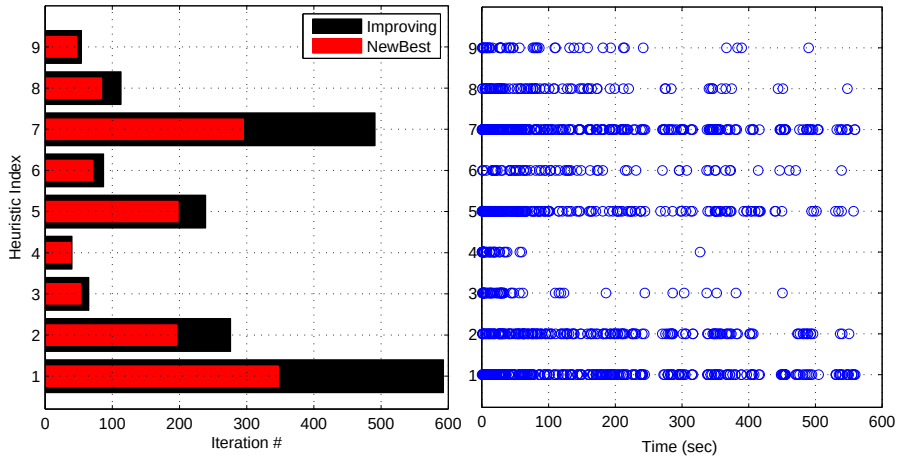

Figure 4.11: Improvement performance of the low-level heuristic on p2011-02-15 (run 5)

Besides these traditional hyper-heuristic sub-mechanisms, we introduced a new hyper-heuristic element, i.e. mentoring. We designed a learning based mentoring approach. The idea is to use learning automata for adapting the heuristics' parameters during a search. This simply indicated the performance changes of the heuristics while solving a problem. Thus, it is important to employ online heuristic parameter adaptation strategies for addressing the changing search space characteristics.

This chapter investigated the effect of learning and adaptation on different hyper-heuristic sub-mechanisms while solving distinct problems. The main conclusion is that it is helpful to use online and adaptive methods in a hyper-heuristic's sub-mechanisms, if these methods are properly designed for capturing changing search requirements. The computational results discussed for different hyper-heuristic sub-mechanisms can be used to determine design requirements for an effective hyper-heuristic. However, the chapter does not consider any generality related factors on the performance of hyper-heuristics. Hence, Chapter 5 discusses how the generality level of a hyper-heuristic is measured and provides an empirical analysis considering various generality related factors.

Chapter 5

Generality of Hyper-heuristics

Reusability is a desired feature for search and optimisation strategies. Low-level, problem dependent search mechanisms are rarely applied to different problems with no change. The generic characteristics of hyper-heuristics provide an opportunity for reusability. Thus, the aim of hyper-heuristics are expressed as *raising the level of generality* [62]. While most hyper-heuristic works employ the term generality in describing the potential for solving various problems, the performance changes across different domains are rarely reported. Furthermore, a performance study of hyper-heuristics purely on the topic of heuristic sets is uncommon. Similarly, experimental limits are generally ignored when comparing hyper-heuristics. Thus, defining generality factors and inspecting their effects on the performance of hyper-heuristics is a vital requirement.

The present chapter aims at investigating the performance of a set of selection hyper-heuristics with respect to:

1. a number of problem domains
2. different heuristic sets
3. distinct execution time limits.

The first empirical study helps to demonstrate the performance of a hyper-heuristic across different problems. The second setting is essential in examining the effect of heuristic sets with diverse characteristics. The final setting studies the influence of the optimality gap on heuristic selection. The problem domains considered here are:

- *Home care scheduling*: assigning a number of tasks required to be handled at different locations to a set of carers based on their availabilities, contracts and the patients' requirements
- *Nurse rostering*: assigning a number of nurses to shifts based on the workload and the nurses' contracts
- *Patient admission scheduling*: assigning a number of patients to a number of beds based on their availabilities and the patients' requirements.

For each problem domain, a set of parametric heuristics was employed to generate nine heuristic sets with different improvement and speed features. Each hyper-heuristic was tested on these heuristic sets under two different execution time limits. For this purpose, a set of hyper-heuristics was used to demonstrate which performs better on which type of heuristic set and on which problem after how much time. Thus, the outcome of this study can be used to determine the generality level of selection hyper-heuristics together with their advantages and disadvantages regarding certain experimental conditions before running them. The experimental results of this study indicate that different experimental conditions are required for different hyper-heuristics to reach their potential efficiently. Additionally, it is apparent that both the performance and the contribution of a hyper-heuristic's sub-mechanism depends on the other sub-mechanisms.

The following portion of the section presents the hyper-heuristic components, used for the experiments, in Section 5.1. The experimental settings are detailed in Section 5.2. In Section 5.3, case studies across a number of problems with different heuristic sets and distinct environmental settings are presented. The last section concludes the paper with a summary and remarks.

The context of this chapter was submitted to a journal:

> M. Mısır, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, An Investigation on the Generality Level of Selection Hyper-heuristics under Different Empirical Conditions, *under review*.

5.1 A selection hyper-heuristic framework

Figure 5.1 depicts a selection hyper-heuristic framework. This framework involves a selection mechanism to choose heuristics and an acceptance criterion to decide whether to accept or to reject the visited solutions. In addition to these traditional hyper-heuristic sub-mechanisms, a mentoring element is used.

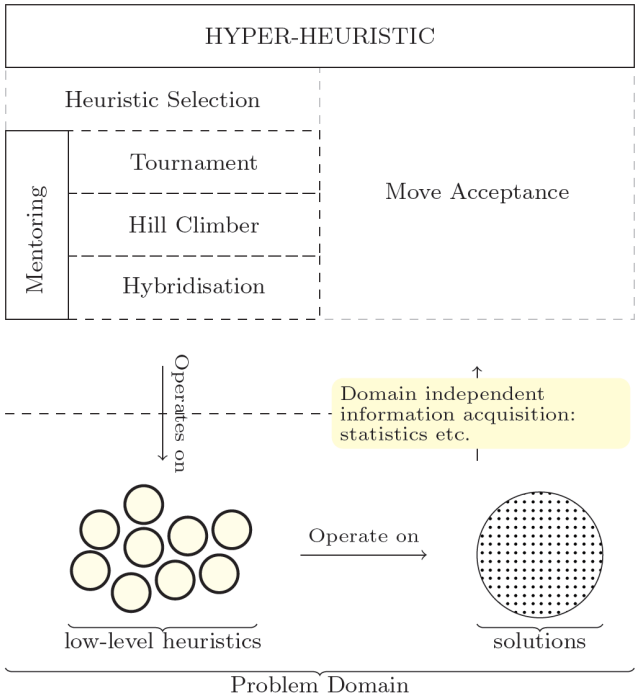

Figure 5.1: A selection hyper-heuristic framework

A variety of sub-mechanisms for the mentioned purposes were employed to build a large set of hyper-heuristics. These mechanisms are detailed in the following sub-sections.

5.1.1 Heuristic selection

A traditional selection hyper-heuristic processes two consecutive operations at each decision step. These processes begin by choosing one or more heuristics based a heuristic selection mechanism. The selected heuristics are applied to a solution for generating new solutions. To perform this task more efficiently, most of the heuristic selection strategies accommodate certain learning devices. These learning devices can also be used as heuristic exclusion strategies to determine which heuristics should not be chosen to make the heuristic selection process easier. In particular, we developed a heuristic exclusion mechanism, the dynamic heuristic set (DHS) [232] strategy, based on the online performance

of the low-level heuristics. The details of this approach are presented in Chapter 4, Section 4.3.1. We extended this approach by adding certain adaptive components, i.e. adaptive dynamic heuristic set (ADHS) [233] strategy. The phase length was continuously updated based on the speed of the heuristics available in the heuristic set. The fixed tabu duration was chosen independently for each heuristic according to their exclusion frequencies and the random selection strategy for selecting heuristics was replaced by a learning automaton. We applied the same approach with a different selection rule and a modified performance metric to distinct problems in [235]. The present study used the ADHS implementation provided in [233].

5.1.2 Mentoring

For differentiating the performance of a set of low-level heuristics, mentoring (explained in Chapter 4, Section 4.2.1) idea is employed. The following subsections present three mentoring methods.

Tournament selection

Tournament selection is a technique that aims at choosing individuals from a population in an evolutionary algorithm. The idea is to return the best individual among a number of randomly selected individuals. A number of studies exist about the tournament selection for sampling solutions while using perturbative selection hyper-heuristics [46]. The aim was to select a number of neighbouring solutions via utilisation of a specific low-level heuristic and choosing the best one with respect to solution qualities as shown in Figure 5.2. This solution selection strategy was automated to determine the best sampling factor for each heuristic and these values were adapted during the search in [226].

In the present study, this approach was applied in two different ways: BEST and FIRST_IMPROVING.

- **BEST**: Select the best solution among the samples. This is the basis of the current hyper-heuristic studies [46] on tournament heuristics.
- **FIRST_IMPROVING**: Select the first improving solution whilst selecting a fixed number of samples. If no improving solution can be found, return the first visited solution.

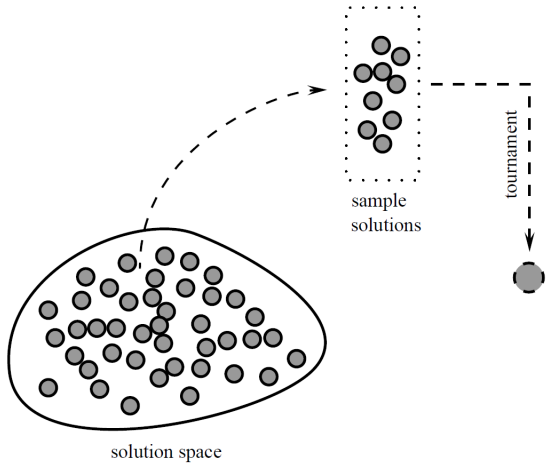

Figure 5.2: Tournament selection process (a heuristic is applied for a number of times to visit sampling solutions in a solution space)

Hill climbing

A hill climber is easily created by consecutively applying heuristics and accepting only if the new solutions are better or equal quality than the earlier solutions. Figure 5.3 visualises this process. Creating new hill climber in this way requires a sampling factor value denoting the number of neighbouring solutions that will be visited. This method aims to generate new heuristics with a high exploitation focus. Additionally, if no improving or equal quality solution is found, the first sampled solution is returned.

Hybridisation

In addition to the aforementioned mentoring strategies, new heuristics can be constructed via hybridisation. A hyper-heuristic framework that consecutively applies two heuristics was proposed in [263]. One of these heuristics is mutational and designated for deteriorating a solution. Afterwards, a predetermined hill climber is applied to enhance improvement over the solution visited by the mutational heuristic. A similar approach, applying first a worsening heuristic and then all available hill climbers, was employed by [57]. In our recent studies [233, 235], a heuristic list was maintained to keep heuristics performing effectively in pairs for each heuristic. These studies show combined heuristic behaviour to be helpful to exploit the potential of a given heuristic set. In the present study,

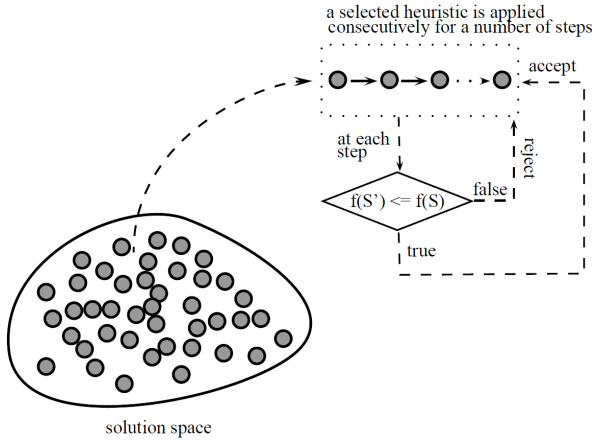

Figure 5.3: Hill climbing process (a heuristic is applied for a number of times like a hill climber)

hybridisation approaches are not considered since the number of experiments is already very high.

5.1.3 Move acceptance

Move acceptance mechanisms are very effective in increasing the performance of hyper-heuristics since they decide on the search trajectory to follow. They are responsible for balancing intensification and diversification within traditional hyper-heuristics. For this purpose, improving solutions visited by the selected heuristics are usually accepted and other solutions can also be accepted under certain conditions, thus diversifying the search process. Having decisions depend on the state of a search space is the preferred strategy behind most successful move acceptance implementations. For the experiments, seven move acceptance criteria with different search behaviours are employed, as explained in the following subsections.

Adaptive iteration limited list-based threshold accepting

In [233, 235], we introduced a new move acceptance criterion, i.e. adaptive iteration limited list-based threshold accepting (AILLA). AILLA aims at managing the search process according to the changing requirements and

characteristics of a search space. AILLA immediately accepts improving or equal quality solutions. For diversification purposes, worsening solutions are accepted under two conditions. The first being if a new best solution could not be found after a certain number of iterations regarding consecutive worsening solutions. The value of this iteration limit is adapted during search. The second condition is connected with a list accommodating the fitness values of recently visited new best solutions. These values are used as the threshold levels for accepting worsening solutions. The pseudocode of AILLA is available in [233].

Great deluge

Great deluge (GD) used for this study is subject to a linear version from [46].

Simulated annealing

In this study, the simulated annealing (SA) acceptance criterion is taken from [46].

Late acceptance

The LATE acceptance implementation subject to this study has been published in [120].

Improving or equal

The improving or equal (IE) [19] acceptance strategy always accepts better quality solutions. Also, the solutions with equal quality is accepted for diversification purposes. This method is expected to be useful especially if there are same quality solutions on different regions of a search landscape.

Only improving

The only improving (OI) [105] move acceptance criterion accepts only better quality solutions and does not provide any diversification opportunities.

All moves

The All move (AM) [105] acceptance method accepts every solution visited by the low-level heuristics.

5.2 Experimental settings

A series of experiments was conducted using Pentium Core 2 Duo 3 GHz PCs with 3.23 GB memory. Each experiment was repeated 10 times. All the available low-level heuristics for the target problem domains were used without any limitation. Due to the extremely large number of experiments, the number of instances per problem was required to be kept low.

5.2.1 Tested hyper-heuristics

In total 14 hyper-heuristics were tested. They comprise of the combinations of 2 heuristic selection strategies and 7 move acceptance mechanisms, where ADHS and simple random (SR) were used. SR chooses a heuristic in a uniformly random manner. Regarding move acceptance, the employed mechanisms include adaptive iteration limited list-based threshold accepting (AILLA), great deluge (GD), simulated annealing (SA), late acceptance (LATE), improving or equal (IE), only improving (OI) and all moves (AM). The first three acceptance mechanisms provide diversification strategies, which may lead to accepting worsening solutions under certain threshold values. The resulting hyper-heuristics with these sub-mechanisms are those: ADHS-AILLA, ADHS-GD, ADHS-SA, ADHS-LATE, ADHS-IE, ADHS-OI, ADHS-AM, SR-AILLA, SR-GD, SR-SA, SR-LATE, SR-IE, SR-OI, SR-AM.

5.2.2 Heuristic sets

The aforementioned 14 hyper-heuristics were applied to a number of problems to analyse performance and generality. Additionally, different parametrised heuristic sets with distinct heuristic distributions were used with the aforementioned selection types, i.e. BEST, FIRST_IMPROVING and HILL_CLIMBER. The parameter of each heuristic is a sampling factor referring to the number of solutions sampled each time the corresponding heuristic is applied. The heuristic sets determined with respect to the heuristics' sampling factors are explained as follows:

1. Experiments were carried with n low-level heuristics at sampling factor 4.
2. In addition to the heuristics with sampling factor 4, the same heuristics with sampling factor 1000 were added as time consuming heuristics. Thus, each heuristic set involves $2n$ heuristics.
3. The same heuristics with sampling factors $\{1, 4, 8, 16\}$ differentiate with respect to the number of sampling solutions visited at each decision step. In other words, there are n heuristics with sampling factor 1, n heuristics with sampling factor 4, n heuristics with sampling factor 8 and n heuristics with sampling factor 16 in each heuristic set.

Table 5.1 shows the heuristic sets used for the experiments. The number of heuristics and the sampling factor with the selection type choice for each heuristic are presented for each heuristic set.

Heuristic set	**Size**	**MENTORING**	
		Sampling factor	**Selection type**
HS_1	n	4	BEST
HS_2	n	4	FIRST_IMPROVING
HS_3	n	4	HILL_CLIMBER
HS_4	2n	4, 1000	BEST
HS_5	2n	4, 1000	FIRST_IMPROVING
HS_6	2n	4, 1000	HILL_CLIMBER
HS_7	4n	1,4,8,16	BEST
HS_8	4n	1,4,8,16	FIRST_IMPROVING
HS_9	4n	1,4,8,16	HILL_CLIMBER

Table 5.1: Heuristic sets used for the experiments (n refers to the number of parametric heuristics used)

5.3 Case studies

5.3.1 The home care scheduling problem

Problem instances

The experiments employ 6 HCSP instances (*hh*, *hh1*, *ll11*, *ll2*, *ll21*, *ll3*) detailed in Chapter 3, Section 3.2.2. The 12 low-level heuristics developed for the HCSP are also presented in Chapter 3.

Speed of the heuristic sets Figure 5.4 illustrates the speed of the heuristic sets when heuristics are randomly selected. Among these sets, $HS4$ and $HS6$ are slower in comparison with the rest of the heuristic sets. These two heuristic sets involve $2n$ heuristics including n heuristics with sampling factor 4 and n heuristics with sampling factor 1000. For the second group of n heuristics, whenever one of them is chosen, 1000 solutions are checked and this slows down the search. $HS5$, another heuristic set with a similar $2n$ heuristics, is faster than these two due to the $FIRST\_IMPROVING$ sampling rule.

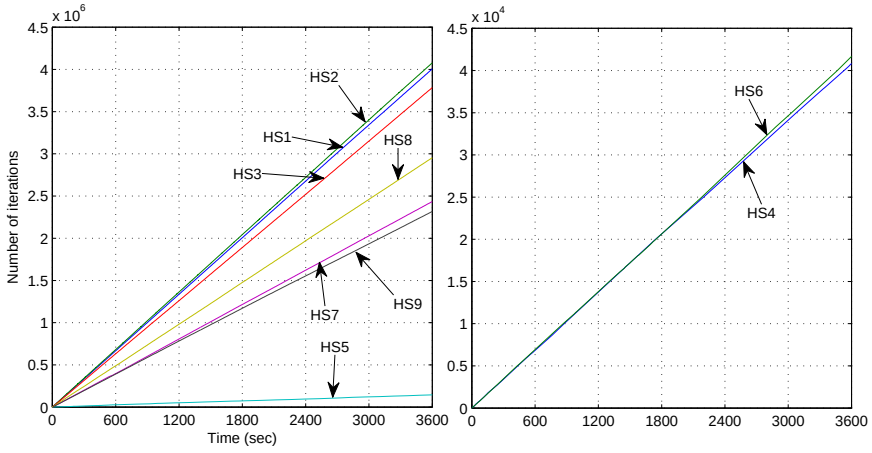

Figure 5.4: The number of iterations spent on each heuristic set for the HCSP by choosing heuristics in a uniformly random manner

Evolvability of the heuristic sets Figure 5.5 presents the number of new best solutions visited during an hour of execution time on each heuristic set by ADHS-AILLA. The highest number of new best solutions was found while

searching over the heuristic set $H1$, generating 100 new best solutions. This value decreases to 30 new best solutions in the case of $HS6$, one of the slowest heuristic sets with hill climbing capabilities. These values are very small with respect to the total number of iterations spent. Hence, for these search spaces, exploration oriented heuristic selection strategies are expected to perform well when compared with intelligent, exploitation oriented methods if there is enough time to run.

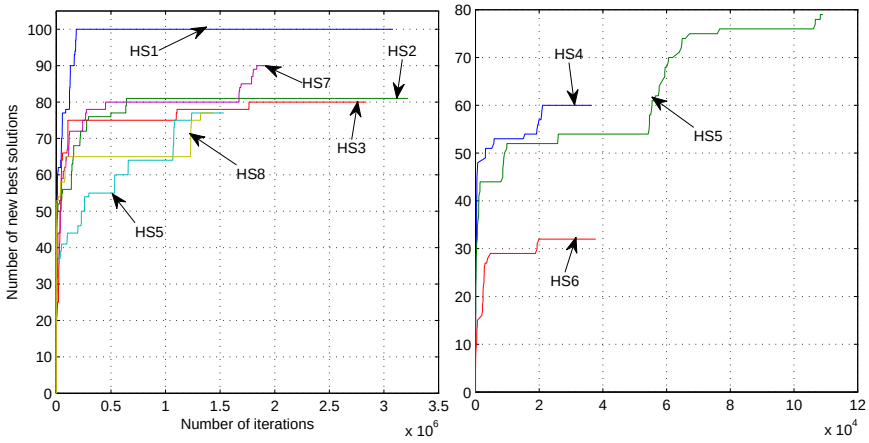

Figure 5.5: Number of new best solutions per iteration for each heuristic set for the HCSP (ADHS-AILLA on $ll3$, based on 1 hour test results)

Computational Results

Figure 5.6 presents the best performing hyper-heuristics on each heuristic set after both 10 minutes and 1 hour computation time. The performance difference was determined significant after a Wilcoxon test with a 95% confidence interval. In the 10-minute case, there was always more than one hyper-heuristic which could be considered the best on each heuristic set. Among all the tested hyper-heuristics, the hyper-heuristics with GD and AM were unable to deliver any best results across the heuristic sets. Contrastingly ADHS-AILLA found the best resulting solutions on all the heuristic sets, except on $HS6$. In addition, SR-AILLA showed the best performance regardless of which heuristic set was used. The other hyper-heuristics also showed effective performance, but on different heuristic sets. After 1 hour of execution ADHS-AILLA performed best on seven out of nine heuristic sets. SR-AILLA still found the best results across all the heuristic sets. The hyper-heuristics with GD and AM were not as successful as the other hyper-heuristics. Furthermore, the remaining hyper-heuristics showed

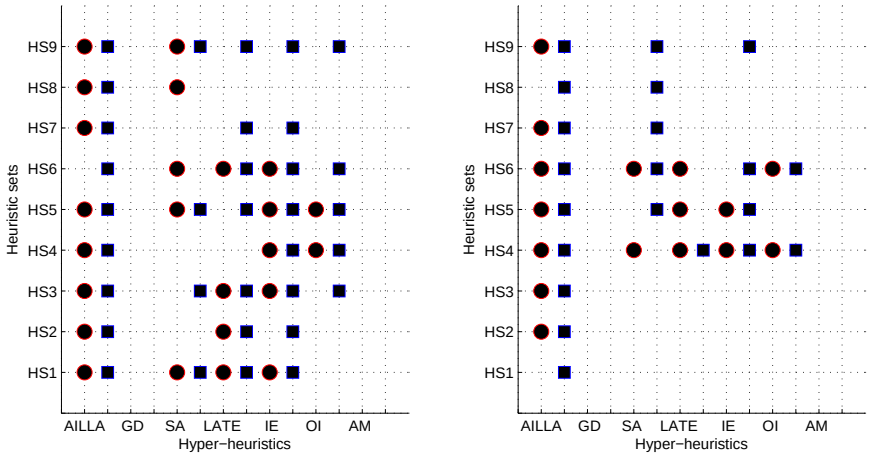

Figure 5.6: The significantly best hyper-heuristics on each heuristic set for the home care scheduling problem after 10 minutes (left) and 1 hour (right) (Consecutive elements on the x axis refer to ADHS and SR respectively)

a performance decrement. These performance changes occurred particularly for the first three heuristic sets, $HS1$, $HS2$, $HS3$. This clearly indicates that the experimental limitations such as execution time limit, should be taken into account while assessing the performance of different hyper-heuristics in terms of generality.

Figure 5.7 and 5.8 present the ranking of each hyper-heuristic after 10 minutes and 1 hour of execution respectively. One can see the superior performance of ADHS-AILLA and SR-AILLA from this simple ranking. The obvious performance difference between the hyper-heuristics with GD and AM compared to the other hyper-heuristics on each heuristic set is also easily visible. Regarding the heuristic selection methods, SR is generally better than ADHS with respect to the number of significant performance on the tested heuristics sets. The main reason could conceivably be the low number of possible improving iterations, limiting the effect of the learning process. evolvability characteristics of the HCSP's solution space regarding the number of new best solutions found during the whole search. The performance of SR was even better when the running time was 1 hour. Figure 5.9 depicts this situation better in box-plots. These show that the move acceptance criterion is more important than the heuristic selection mechanism for the HCSP. However, if the move criteria, such as GD and AM, are not adequate for this test setting, it is useful to employ ADHS to elevate their performance. Another important point is the performance of the hyper-heuristics with OI. Whilst these hyper-heuristics could not deliver significantly

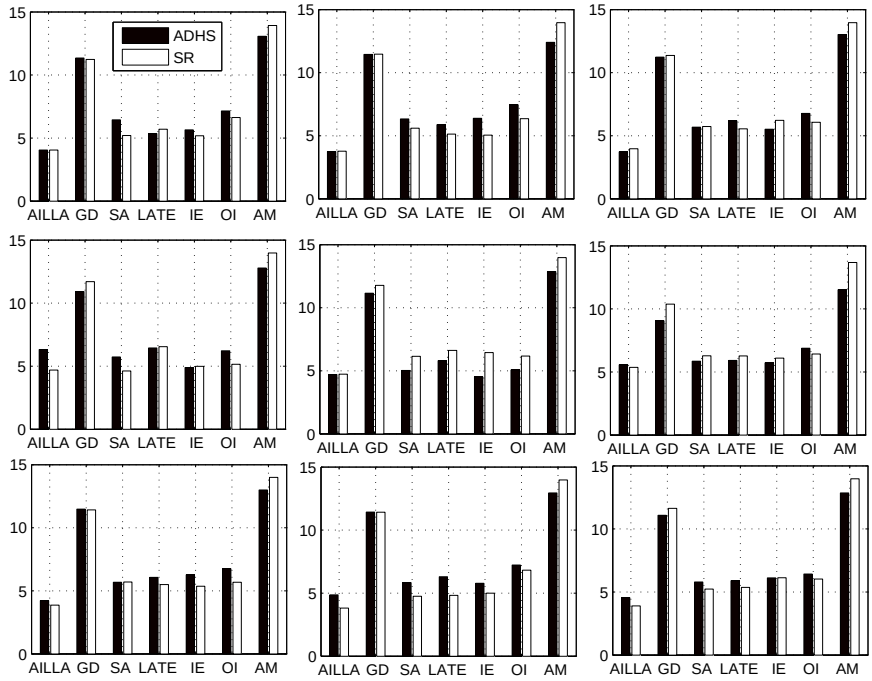

Figure 5.7: Average ranking of the hyper-heuristics on the HCSP after 10 minutes. Each graph represents the results obtained with the heuristic set. They are ordered from left to right, top to bottom: $HS_1 \rightarrow HS_9$

best results, their performance was not as bad as the hyper-heuristics with GD and AM. This indicates that it might be possible to find good quality solutions without a diversification strategy.

Best hyper-heuristic settings Table 5.2 presents the best results found by the tested hyper-heuristics. For the first two instances, SR-AILLA on HS1 and ADHS-AILLA on HS2 found the best solutions. For the $ll11$, $ll2$ and $ll3$ instances, new best solutions were found by SR-AILLA on HS2, ADHS-AILLA on HS9 and ADHS-AILLA on HS1. The best result found for $ll21$ has the same quality as its best known solution.

Which Heuristic Set is Better? Table 5.3 presents the best neighbourhood sampling rule employed for n, $2n$ and $4n$ heuristic sets using the results by ADHS-AILLA. The experimental results showed that there is no significant

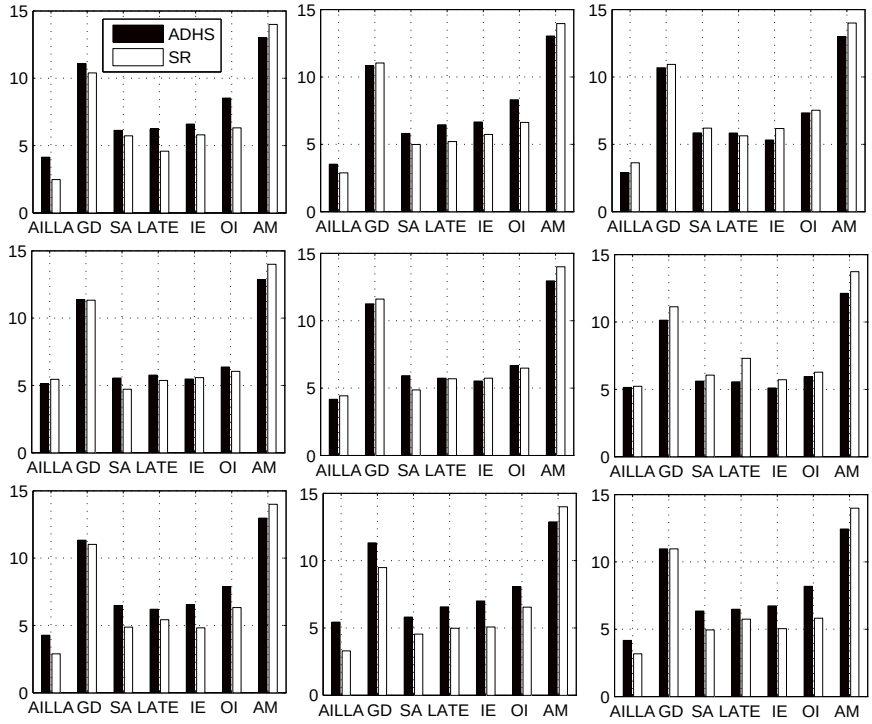

Figure 5.8: Average ranking of the hyper-heuristics on the HCSP after 1 hour. Each graph represents the results obtained with the heuristic set. They are ordered from left to right, top to bottom: $HS_1 \rightarrow HS_9$

performance difference between the sampling rules due to the execution time. In the case of the heuristic set with n heuristics $FIRST\_IMPROVING$ and $BEST$ had a similar effect on the performance of ADHS-AILLA. However, the heuristic sets using these two rules prove better choices than $HILL\_CLIMBER$ for the hyper-heuristic. The heuristic set of size $2n$ was managed more efficiently by using the $HILL\_CLIMBER$ sampling rule. For the other sampling rules, $FIRST\_IMPROVING$ produced significantly better results than $BEST$ on the heuristic set. The reason behind this performance difference is the low-evolvability of the search space whereas half of the heuristics require 1000 sampling solutions to check before returning one. Thus, $HILL\_CLIMBER$ helps the search process slowly yet in an efficient way. However, $FIRST\_IMPROVING$ speeds up the search process, while $BEST$ slows down the search speed and delivers poorer results than $HILL\_CLIMBER$. For the heuristic set of size $4n$, there is no significant

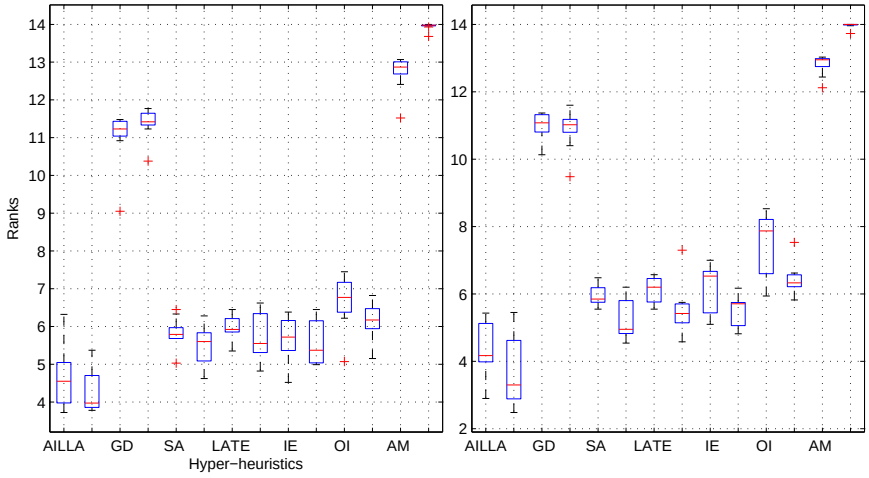

Figure 5.9: Box plot for average ranks of the hyper-heuristics after 10 minutes (left) and 50 minutes (right). For each acceptance, two consecutive boxes denote ADHS and SR

Inst.	Best known	Our best	Hyper-heuristic (Heuristic Set)
hh	**71981,80**	73048,20	SR-AILLA(HS1)
hh1	**40644,60**	41084,45	ADHS-AILLA(HS2)
ll11	39046,80	**35513,20**	SR-AILLA(HS2)
ll2	8926,25	**8919,95**	ADHS-AILLA(HS9)
ll21	**8372,40**	**8372,40**	ADHS-AILLA(HS1)
ll3	8047,12	**7806,95**	ADHS-AILLA(HS1)

Table 5.2: The best results on the HCSP instances (The best results are from [232])

performance difference between these rules.

Effect of the initial solution Although GD is an effective move acceptance strategy, it did not find promising results for the HCSP with the given heuristic sets. One of the reasons for GD's poor performance is the low quality of the initial solutions. These solutions are used to determine the threshold values for accepting worsening solutions. A similar and related reason is the highly destructible solution space of the HCSP. This means that when a heuristic is applied, the solution quality may dramatically change. Whenever a heuristic

10'	**Sampling rules**
n	$FIRST\_IMPROVING \approx BEST \gg HILL\_CLIMBER$
$2n$	$HILL\_CLIMBER \gg FIRST\_IMPROVING \gg BEST$
$4n$	$FIRST\_IMPROVING \approx BEST \approx HILL\_CLIMBER$
1h	
n	$FIRST\_IMPROVING \approx BEST \gg HILL\_CLIMBER$
$2n$	$HILL\_CLIMBER \gg FIRST\_IMPROVING \gg BEST$
$4n$	$BEST \approx HILL\_CLIMBER \approx FIRST\_IMPROVING$

Table 5.3: The performance of different neighbourhood sampling rules using ADHS-AILLA on the HCSP. $\approx$ denotes similar performance, $\gg$ refers to significantly better performance

is applied to a solution, it is likely that it leads to worse solutions. Thus, GD cannot maintain good quality solutions for long, particularly during early stages of the search process. Furthermore, the availability of the limited number of new best solutions that can be visited, compared to the total iterations spent, further worsens the performance of GD.

5.3.2 The nurse rostering problem

Problem instances

The problem instances were taken from the nurse rostering competition [170]. 10 long instances were selected (*long_late01*, *long_late02*, *long_late03*, *long_late04*, *long_late05*, *long_hidden01*, *long_hidden02*, *long_hidden03*, *long_hidden04*, *long_hidden05*). The 29 low-level heuristics presented in Chapter 3, Section 3.5.2, were used in the heuristic set. The experiments were conducted for both 10 minutes and 1 hour of execution time.

Speed of the heuristic sets Figure 5.10 illustrates the number of iterations per time graph when moves are performed by choosing heuristics in a uniformly random manner. $HS4$ and $HS6$ are again the slowest heuristic sets, similar to that occurred for the same type of heuristic sets on the other tested problem domains.

Evolvability of the heuristic sets Figure 5.5 illustrates the number of new best solutions found while searching the nurse rostering problem solution space. For $HS2$, the highest number of new best solutions was found. This value

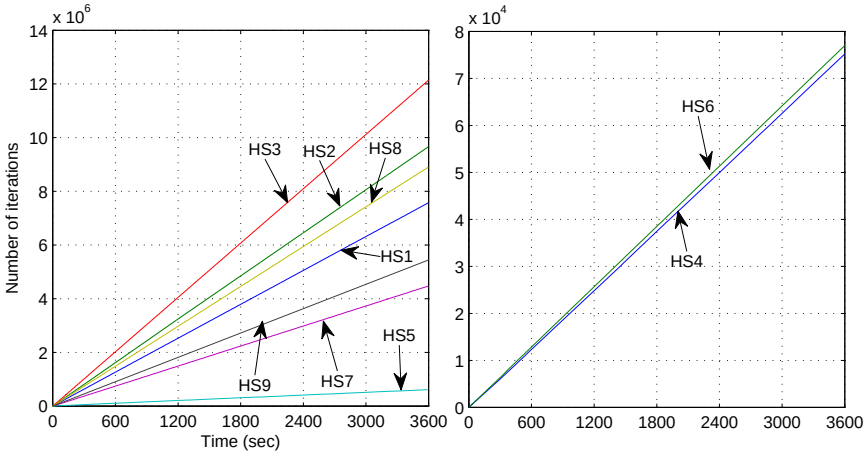

Figure 5.10: The number of iterations spent on each heuristic set for the NRP by choosing heuristics in a uniformly random manner

decreases through $HS1$, $HS8$, $HS5$, $HS7$, $HS3$, $HS9$, $HS4$ and $HS6$. The lowest number of new best solutions was found for the heuristic set with the $HILL\_CLIMBING$ sampling rule. This heuristic set is slow due to the heuristics with a large sampling factor, i.e. 1000. These values are small compared to the number of iterations spent in total. However, they are large enough to warrant better performance on certain heuristic sets by employing an intelligent heuristic selection strategy.

Computational results

Figure 5.12 shows the significantly best hyper-heuristic separately on each heuristic set. The best hyper-heuristics after 10 minutes and 1 hour of execution were similar. Among the tested hyper-heuristics, the hyper-heuristics utilising GD, i.e. ADHS-GD and SR-GD, were clearly the best approaches for this NRP given the low-level heuristic sets. For the 10 minute case, these hyper-heuristics were best for the heuristic sets $HS1 \rightsquigarrow HS8$. Regarding $HS9$ SR-AILLA, SR-IE and SR-OI were the best methods after 10 minutes, SR-AILLA and SR-OI on average performed best after 1 hour. For the two heuristic sets with the $HILL\_CLIMBER$ solution sampling rule, $HS3$ and $HS6$, almost all the tested hyper-heuristics performed similarly. The only exceptions for these heuristic sets are SR-IE for the 10 minute experiments and SR-OI for the 1 hour experiment. Clearly, most hyper-heuristics show a similar performance due to local optima visited with the $HILL\_CLIMBER$ sampling rule.

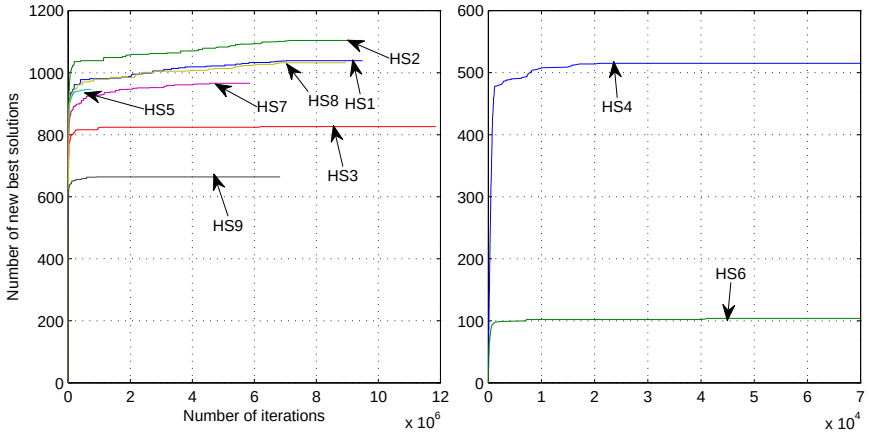

Figure 5.11: Number of new best solutions over iteration charts on each heuristic set for the NRP (SR-GD on *long_late*02, 1 hour test results were used)

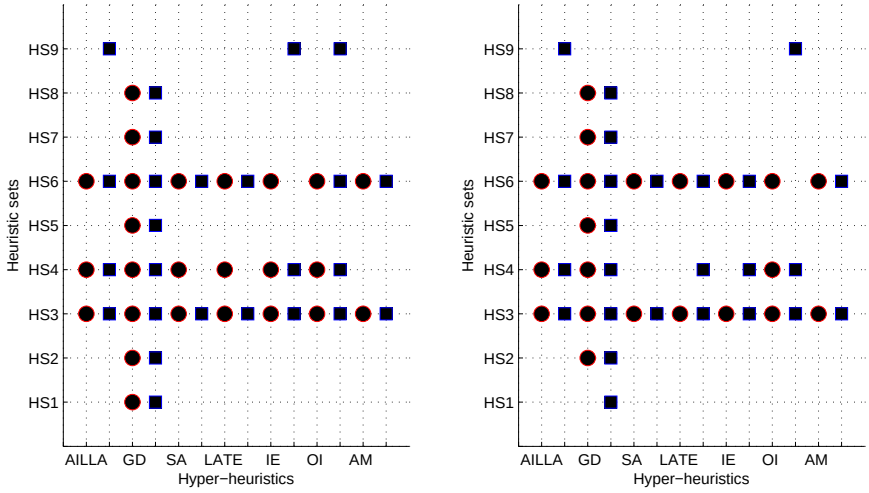

Figure 5.12: The significantly best hyper-heuristics on each heuristic set for the NRP after 10 minutes (left) and 1 hour (right) (Consecutive elements on the x axis refer to ADHS and SR respectively)

Figures 5.13 and 5.14 provide the ranking of each hyper-heuristic on each heuristic set. The superior performance of the hyper-heuristics with GD is also visible through these ranking graphs. Concerning the heuristic selection process, ADHS performs better than SR with most of the acceptance criteria,

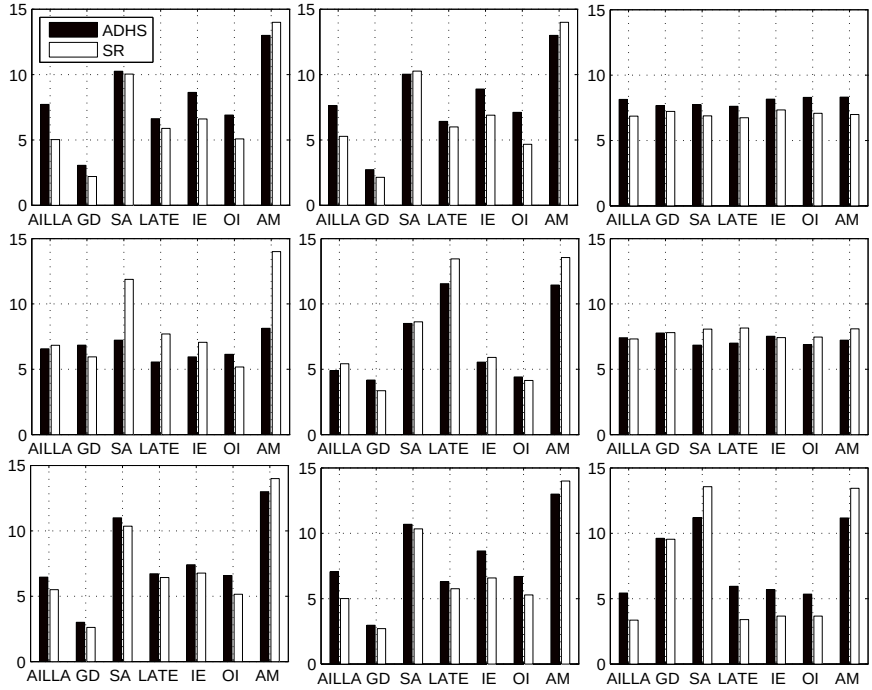

Figure 5.13: Average ranking of the hyper-heuristics on the NRP after 10 minutes. Each graph represents the results obtained with the heuristic set. They are ordered from left to right, top to bottom: $HS_1 \rightarrow HS_9$

particularly on the heuristic sets with $2n$ heuristics and 10 minutes computation time. The effect of ADHS can be seen more clearly when combined with the poor performing move acceptance criteria like SA and AM. In these cases, the heuristic selection mechanism is more influential than the move acceptance criteria. The explanation for the performance of ADHS on the aforementioned heuristic sets was the high speed differences between heuristics and the short execution time limit. However, on other heuristic sets, SR generally beats ADHS. After hourly experiments, the performance of SR was much better since ADHS could not provide sufficient diversification capabilities after finding high quality solutions in a short amount of time.

Figure 5.15 presents a box plot based on the rankings of the hyper-heuristics for each execution time limit. The performance changes of each hyper-heuristic due to these execution time limits can be seen. From 10 minutes to 1 hour, the performance of the hyper-heuristics with AILLA, GD, SA improve while those with LATE, IE and IT worsen.

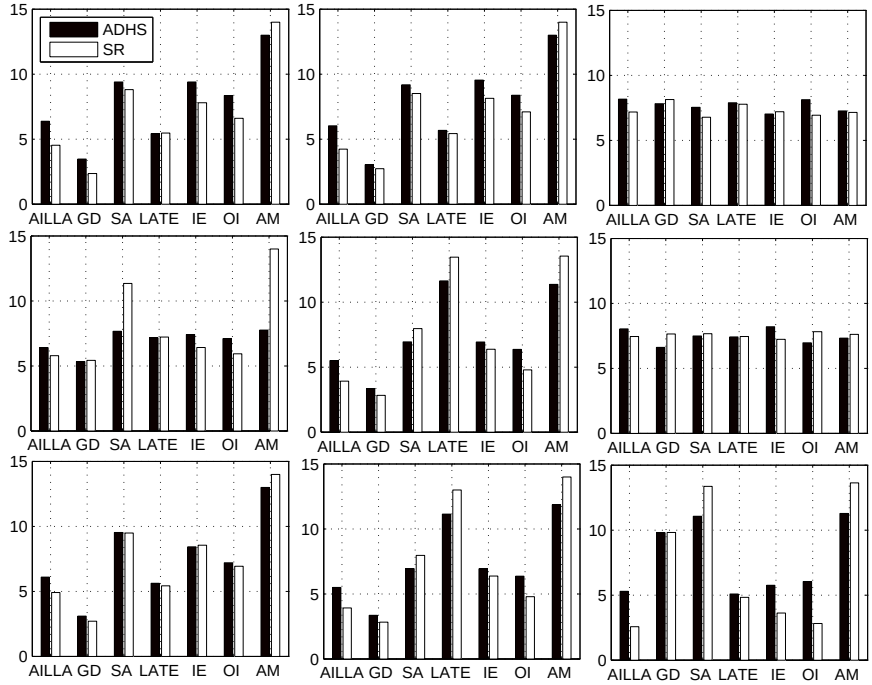

Figure 5.14: Average ranking of the hyper-heuristics on the NRP after 1 hour. Each graph represents the results obtained with the heuristic set. They are ordered from left to right, top to bottom: $HS_1 \rightarrow HS_9$

Best hyper-heuristic settings Table 5.4 presents the best results found by the hyper-heuristics. The best results were always found by the hyper-heuristics with GD. This demonstrates that the performance of GD for finding the best solutions was as good as its average performance. Additionally, in certain problem instances, multiple hyper-heuristics delivered solutions with the same quality.

Which Heuristic Set is Better? Table 5.5 presents the best heuristic sets for SR-GD after 10 minutes and 1 hour of execution. The results show that the tested execution time limits do not affect the performance of SR-GD. Thus, the best heuristic sets for the two cases do not change. The reasoning behind this outcome is that a good quality solution can be found very quickly. Running an algorithm longer that results in a limited amount of improvement on the quality of a solution is the other reason. For the heuristic set with n heuristics, the best sampling method was $BEST$. $FIRST\_IMPROVING$ also showed

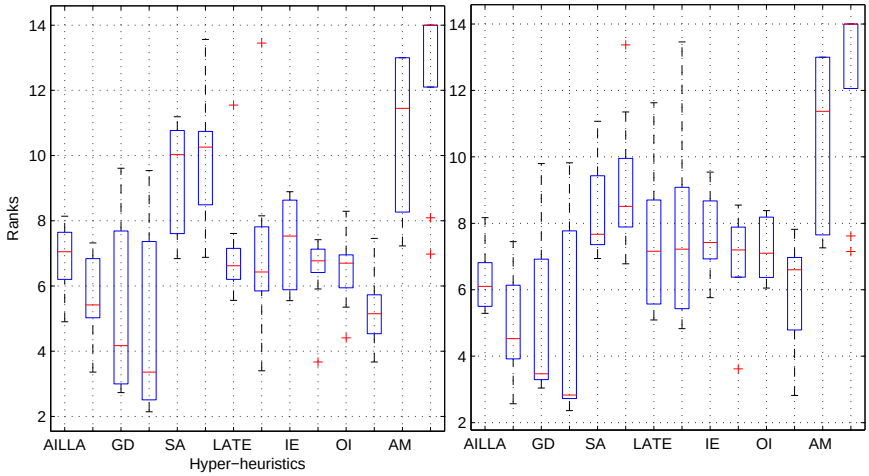

Figure 5.15: Box plot for average ranks of the hyper-heuristics on the nurse rostering problem after 10 minutes (left) and 1 hour (right). For each acceptance, two consecutive boxes denote ADHS and SR

a similar performance. However, there is a significant performance disparity between the results determined with $HILL\_CLIMBER$ and the other two sampling rules. The second type of heuristic set using $2n$ heuristics can be better managed with $FIRST\_IMPROVING$ as the sampling criterion. In this case, $BEST$ performed worse than $FIRST\_IMPROVING$, yet better than $HILL\_CLIMBER$. A similar performance was achieved with the last heuristic set type, $4n$ heuristics, when $FIRST\_IMPROVING$ and $BEST$ were used. Again, $HILL\_CLIMBER$ was the worst one among the three. However, when we look at the same performance measure for the worst hyper-heuristic among the 14 tested hyper-heuristics, SR-AM, the best results by far were found when the sampling rule $HILL\_CLIMBER$ was used. $BEST$ and $FIRST\_IMPROVING$ follow it and $BEST$ was also significantly better than $FIRST\_IMPROVING$. This performance difference is due to the AM acceptance criterion. Since it accepts all solutions without any check, it proves useful to have heuristics with internal solution quality control. This shows the strong correlation between the behaviour of the heuristics and the characteristics of the hyper-heuristic performing search over them.

Inst.	**Best known**	**Our best**	**Hyper-heuristic (Heuristic Set)**
long_late01	235	241	ADHS-GD(HS7), SR-AILLA(HS7), SR-GD(HS1,HS2)
long_late02	229	269	SR-GD(HS1)
long_late03	220	232	SR-GD(HS2)
long_late04	221	252	SR-GD(HS1)
long_late05	83	85	SR-GD(HS1)
long_hidden01	**346**	349	SR-GD(HS8)
long_hidden02	**89**	**89**	#(HS1,HS2,HS5,HS7,HS8)
long_hidden03	**38**	**38**	SR-GD(HS7), SR-IE(HS1,HS8), SR-OI(HS2)
long_hidden04	**22**	**22**	#(HS1,HS2,HS5,HS7,HS8)
long_hidden05	41	49	ADHS-GD(HS2,HS7,HS8), ADHS-LATE(HS7), SR-GD(HS1,HS2,HS7,HS8), SR-LATE(HS2)

Table 5.4: The best results on the NRP instances. The best known results are from [216] and each of these best known solutions was found after around 10-14 hours of a run. # means that several hyper-heuristics found the solutions with the given quality

10'	**Sampling rules**
n	$BEST \approx FIRST\_IMPROVING \gg HILL\_CLIMBER$
$2n$	$FIRST\_IMPROVING \gg BEST \gg HILL\_CLIMBER$
$4n$	$FIRST\_IMPROVING \approx BEST \gg HILL\_CLIMBER$
1h	
n	$BEST \approx FIRST\_IMPROVING \gg HILL\_CLIMBER$
$2n$	$FIRST\_IMPROVING \gg BEST \gg HILL\_CLIMBER$
$4n$	$FIRST\_IMPROVING \approx BEST \gg HILL\_CLIMBER$

Table 5.5: The performance of different neighbourhood sampling rules using SR-GD on the NRP. $\approx$ denotes similar performance, $\gg$ refers to significantly better performance

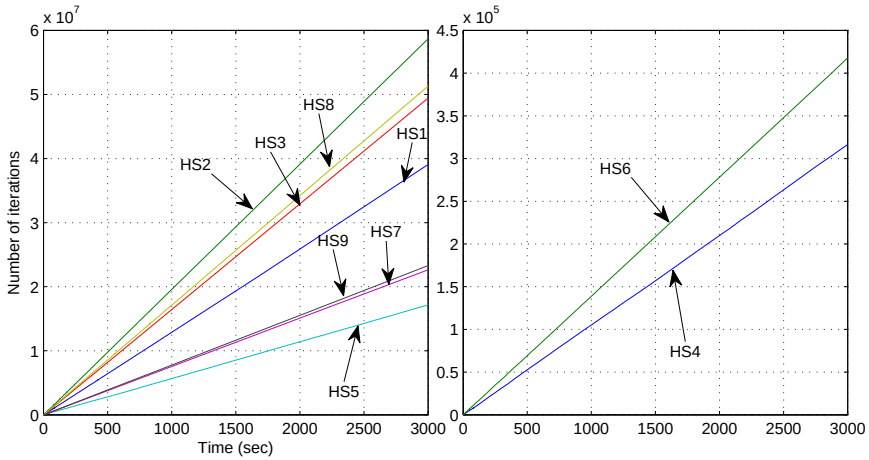

Figure 5.16: The number of iterations spent on each heuristic set for the PASP by choosing heuristics in a uniformly random manner

5.3.3 The patient admission scheduling problem

Problem instances

The PASP instances used in the experiments are available in [46] [1]. The seven PASP instances are as follows: *Inst0*, *Inst1*, *Inst2*, *Inst3*, *Inst4*, *Inst5*, *Inst6*. The instances' details are given in Chapter 3, Section 3.4. The heuristic set includes the 11 low-level heuristics introduced in the same section.

Speed of the heuristic sets Figure 5.16 illustrates the speed of the heuristic set. It was generated while performing moves by choosing heuristics with equal chance. A similar speed difference as that which occurred on the other problem domains is also visible here.

Evolvability of the heuristic sets Figure 5.17 presents the number of new best solutions visited during 50 minutes execution time on each heuristic set by ADHS-AILLA. The number of new best solutions varies between 400 and 2000 for the different heuristic sets.

[1] `http://allserv.kahosl.be/~peter/pas/`

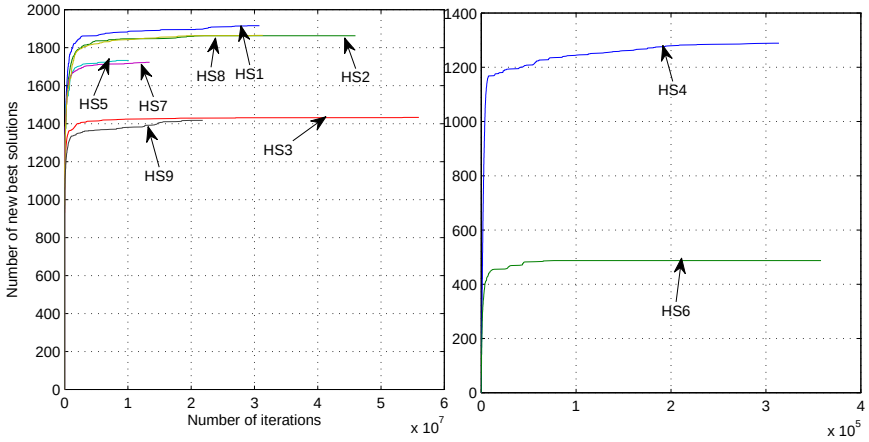

Figure 5.17: Number of new best solutions per spent iterations charts on each heuristic set for the PASP. ADHS-AILLA on $Inst3$, 50 minutes test results were used

Computational Results

Figure 5.18 presents the best performing hyper-heuristics on each heuristic set after 10 minutes and 50 minutes respectively. The results are significant according to the Wilcoxon test with a 95% confidence interval. The empirical results after 10 minutes indicate that different hyper-heuristics show significantly better performance on distinct heuristic sets. Among these hyper-heuristics, ADHS-AILLA delivered the best results on 8 out of 9 heuristic sets. ADHS-LATE showed significantly better performance on 7 out of 9 heuristic sets. The same situation occurs for SR-AILLA, SR-LATE and ADHS-IE on 6 heuristic sets, ADHS-GD and SR-IE on 5 heuristic sets, ADHS-SA on 4 heuristic sets, SR-GD and SR-SA on 3 heuristic sets, ADHS-AM on 2 heuristic sets, ADHS-OI, SR-OI and SR-AM on one heuristic set. This means that each tested hyper-heuristic found significantly best solutions on at least one heuristic set. Particularly, on the set $HS6$, 11 out of 14 hyper-heuristics found significantly best results. The reason behind this outcome is the lack of highly diversifying heuristics since all the heuristics in $HS6$ were considered hill-climbers. Additionally, 10 hyper-heuristics showed the significantly best performance on $HS2$.

The performance difference between the hyper-heuristics becomes more evident after 50 minutes. Here, ADHS-GD performed best on 7 heuristic sets, SR-GD was superior on 6 heuristic sets and SR-AILLA on 5 heuristic sets. Together with the rest of the hyper-heuristics, each showed high performance on at

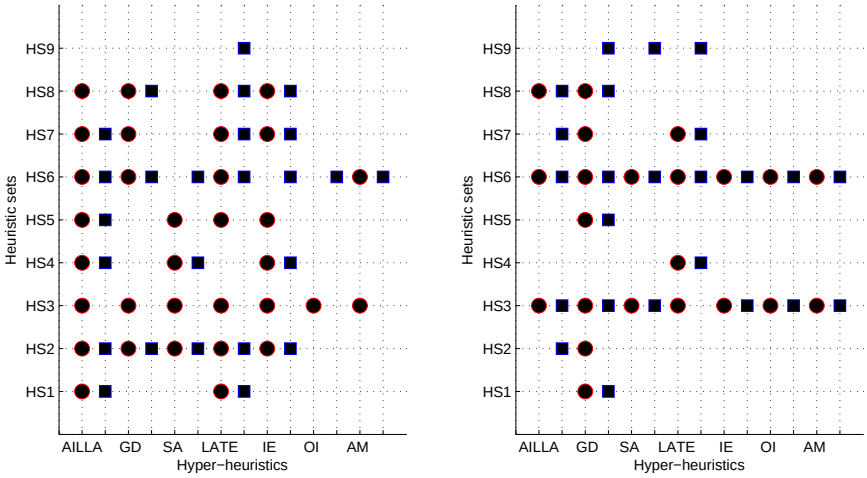

Figure 5.18: The significantly best hyper-heuristics on each heuristic set for the PASP after 10 minutes (left) and 50 minutes (right). Consecutive elements on the x axis refer to ADHS and SR respectively

least 2 heuristic sets. The most common heuristic sets for significantly better performance were two heuristic sets with the hill climbing sampling rule, $HS3$ and $HS6$. Thus, using only one of these heuristic sets in order to compare a set of hyper-heuristics on PASP would not reveal much.

Figure 5.19 and 5.20 present the hyper-heuristics' ranking on each heuristic set after 10 minutes and 50 minutes. Generally, ADHS delivered superior performance to SR on each heuristic set except $HS9$ after 10 minutes. However, this performance difference decreases after 50 minutes, since choosing the incorrect heuristics is not as important as in the 10 minutes experiment. Furthermore, the solutions found by the hyper-heuristics with ADHS were generally poorer than SR on $HS3$ and $HS6$, which is clearly different from the 10 minutes experiments.

Figure 5.21 provides two boxplots for the 10 and 50 minutes experiments based on the average ranks of the tested hyper-heuristics within each heuristic set. The given graphs also illustrate how the hyper-heuristics with AILLA and LATE are more effective than the other hyper-heuristics. In addition, the performance increment of GD after 50 minutes can be clearly seen. Moreover, the hyper-heuristics with SA and IE perform slightly worse after 50 minutes compared with their performance after 10 minutes.

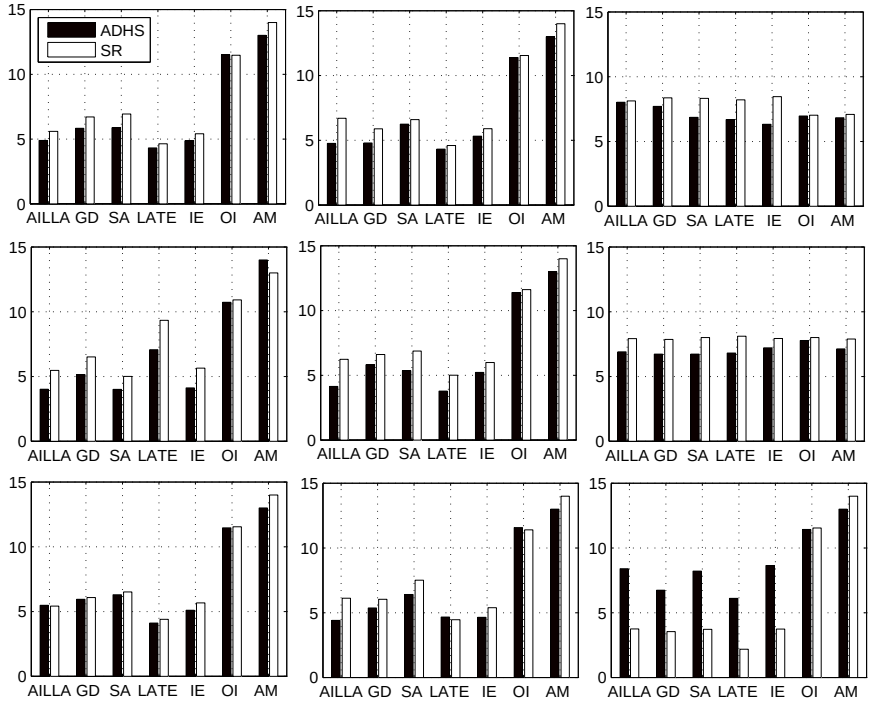

Figure 5.19: Average ranking of the hyper-heuristics on the PASP after 10 minutes. Each graph represents the results obtained on a heuristic set. They are ordered from left to right, top to bottom: $HS_1 \rightarrow HS_9$

Best hyper-heuristic settings Table 5.6 presents the best results found by the tested hyper-heuristics. ADHS-GD found the best solutions on three problem instances whereas SR-GD, ADHS-AILLA, SR-AILLA and ADHS-SA explored the best solution on only one instance.

Which Heuristic Set is Better? Table 5.7 presents the performance difference between the results achieved by ADHS-AILLA on different heuristic sets. In the 10 minutes experiments, the performance of ADHS-AILLA on the heuristic sets with 11 heuristics (HS_1,HS_2,HS_3) and 44 heuristics (HS_7,HS_8,HS_9) was better than the heuristic sets with 22 heuristics. Also, one of the heuristic sets involving 22 heuristics (HS_5) provided a similar performance opportunity. It contains fast heuristics for which the $FIRST\_IMPROVING$ sampling criterion is very suitable. Additionally, the quality of the results on this set differed statistically from the $BEST$ sampling method. The other outcome was the significantly

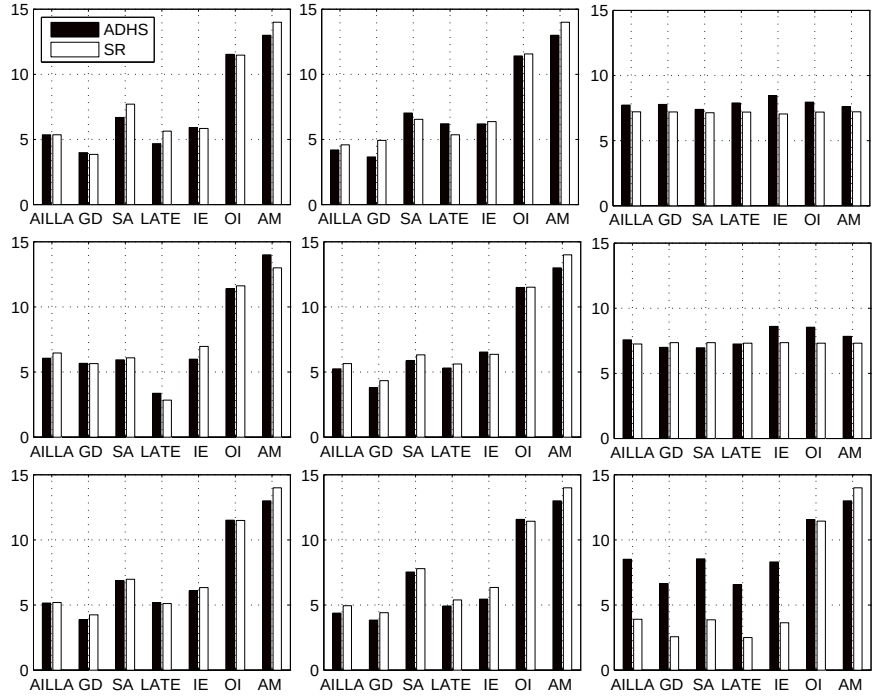

Figure 5.20: Average ranking of the hyper-heuristics on the PASP after 50 minutes. Each graph represents the results obtained on a heuristic set. They are ordered from left to right, top to bottom: $HS_1 \rightarrow HS_9$

worse performer on certain hyper-heuristics on the heuristic sets with hill climbers. As aforementioned, these heuristic sets limit the diversification process. Among these heuristic sets, the results on the heuristic set with 44 heuristics (HS_9) were the best. The reason was that 11 of these heuristics only check one neighbouring solution and thus helping diversify the search process. However, still it is not possible to say that any hyper-heuristic will fail on the heuristic sets with only or mostly hill climbers. A hyper-heuristic with an additional diversification tool such as restarting might also yield high quality results comparable with the other tested heuristic sets.

For the 50 minutes experiments, $FIRST\_IMPROVING$ provided significant performance improvement compared to $BEST$ for 22 and 44 heuristic sets. The heuristic sets with $HILL\_CLIMBER$ still performed worse than these two types.

All these results show that the performance of a hyper-heuristic can be affected

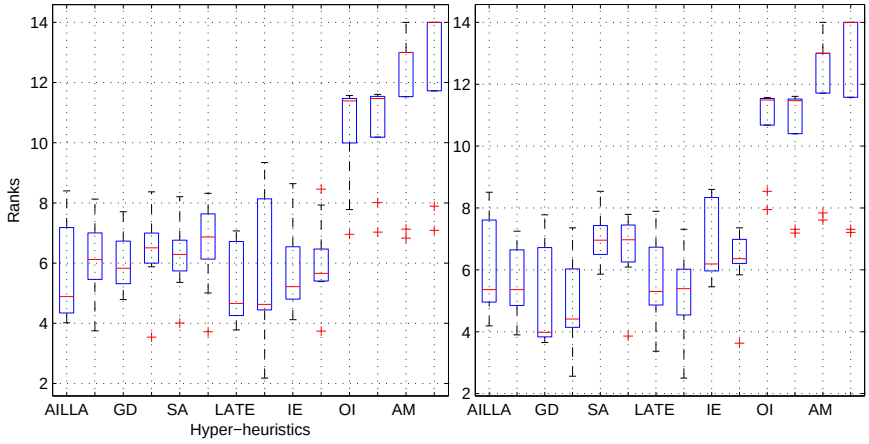

Figure 5.21: Box plot for average ranks of the hyper-heuristics on the PASP after 10 minutes (left) and 50 minutes (right). For each acceptance, two consecutive boxes belong to ADHS and SR

Inst.	**Best known**	**Best known HH**	**Our best**	**Hyper-heuristic (Heuristic Set)**
Inst0	815,2	815,2	816,8	ADHS-SA(HS1)
Inst1	659,2	672,8	672,8	SR-GD(HS7)
Inst2	1143,6	1174,2	1182	ADHS-AILLA(HS2)
Inst3	776,6	797,8	801	ADHS-GD(HS8)
Inst4	1176	1227,8	1246,4	ADHS-GD(HS7)
Inst5	625,6	634,28	633,6	SR-AILLA(HS2)
Inst6	801,2	818,6	821,4	ADHS-GD(HS7)

Table 5.6: The best known results and the best known hyper-heuristic results on the PASP instances are from [94] and [46]

by different factors associated with heuristic sets. Therefore, these factors should be accounted for and certain adaptive devices should be developed in order to achieve a higher generality level.

5.4 Conclusion

The present chapter aimed at investigating the performance of several hyper-heuristics on different problem domains with various heuristic sets for distinct execution time limits. Our main motivation is the lack of studies that elaborate

10'	**Sampling rules**
n	$BEST \approx FIRST\_IMPROVING \gg HILL\_CLIMBER$
$2n$	$FIRST\_IMPROVING \gg BEST \gg HILL\_CLIMBER$
$4n$	$FIRST\_IMPROVING \approx BEST \gg HILL\_CLIMBER$
50'	
n	$FIRST\_IMPROVING \approx BEST \gg HILL\_CLIMBER$
$2n$	$FIRST\_IMPROVING \gg BEST \gg HILL\_CLIMBER$
$4n$	$FIRST\_IMPROVING \gg BEST \gg HILL\_CLIMBER$

Table 5.7: The performance of different neighbourhood sampling rules using ADHS-AILLA on the PASP. $\approx$ denotes similar performance, $\gg$ refers to significantly better performance

on the generality of hyper-heuristics even though hyper-heuristics are defined as high-level approaches strengthening the level of generality. Differentiating the experimental conditions is useful to see the real performance of a hyper-heuristic from a broader perspective. For this purpose, solving different problems helps to see the generality level of a hyper-heuristic in a traditional sense. Furthermore, employing heuristics with distinct improvement characteristics is beneficial for examining the effect of different types of fitness landscapes generated by the heuristic sets. Moreover, conducting an empirical analysis using distinct experimental limits such as a time limitation is also practical in order to realise the time-based behavioural analysis of hyper-heuristics.

These aforementioned generality elements were considered to present a comprehensive performance analysis concerning the generality level of hyper-heuristics. Three problems, home care scheduling, nurse rostering and patient admission scheduling, were chosen for the first generality element. Nine heuristic sets with varying speed and size features were generated by using a set of parametric heuristics for each problem domain. Then, 14 hyper-heuristics built using 2 heuristic selection mechanisms and 7 move acceptance criteria from the literature were employed. Each hyper-heuristic was tested for short and long running time limits to monitor for the purpose of the time-based analysis.

The computational results indicated that changing the heuristic sets affect the performance of the hyper-heuristics. This performance change was mainly caused by the improvement characteristics of the heuristics alongside the distribution of different types of heuristics in the heuristic sets. Additionally, changing the execution time affects the behaviour of the hyper-heuristics. In particular, a hyper-heuristic with high quality results after short execution may perform worse than other hyper-heuristics after running it for longer, and vice versa. Moreover, it was shown that such experimental limitations may change the

contribution of a hyper-heuristic's sub-mechanisms to its performance. Hence, it is important to combine different hyper-heuristic components that can work together in harmony rather than just combining different sub-mechanisms. For instance, if a move acceptance criterion has effective diversification capabilities, it is not useful to combine it with selection mechanism that always chooses the best heuristic. As a counter example, if a move acceptance criterion has no control on diversification such as AM, it is better to combine it with such a selection mechanism.

5.4.1 Effect of heuristic selection

The effect of a heuristic selection mechanism depends on various elements including the characteristics of the other hyper-heuristic sub-mechanisms and experimental limitations. By way of illustration, if a poor performing move acceptance criterion is utilised, one expects to see a higher influence imparted by the heuristic selection part, particularly when using a learning based selection strategy. Regarding the experimental limitations, if the given execution time or the number of iterations is limited, the time to explore the search space will also be limited. As a consequence, spending time with wrong selection decisions will be very critical. In such cases, it is highly probable to recognise the high performance of an intelligent selection approach. On the contrary, a naive selection mechanism can deliver comparable performance to a highly adaptive method. Therefore, time related limitations should be taken into account in the design of heuristic selection mechanisms.

5.4.2 Effect of move acceptance

The move acceptance mechanisms can be very influential on the performance of the selection hyper-heuristics. Although a very effective and intelligent selection mechanism is employed, it may prove difficult to use its strength if the move acceptance strategy cannot be appropriately combined. Additionally, the nature of the heuristics in the heuristic set can affect the performance of the acceptance method. For instance, if the heuristics are highly perturbative and destructive then a move acceptance strategy with effective diversification capabilities is a vital requirement. Also, a very naive acceptance mechanism can still deliver high quality results if the low-level heuristics have effective improvement capabilities. This is an important observation to take into consideration for the performance analysis of different hyper-heuristics.

The hyper-heuristic sub-mechanisms discussed in Chapter 4 and the generality factors determined in this chapter were used to design a generic intelligent hyper-heuristic (GIHH) framework. The details of GIHH are explained in Chapter 6.

Chapter 6

An Intelligent Selection Hyper-heuristic

Generality is considered the key concept and underlying motivation behind hyper-heuristics. The idea is to design a high-level approach that is intelligent enough to determine the strengths and weaknesses of a set of low-level heuristics [61] so that it can be successfully applied to several problems with different heuristic sets. In other words, a selection hyper-heuristic's job is to identify the characteristics of the heuristics and to match them with the problem states. A very naive hyper-heuristic could say that a specific heuristic should be utilised during the first half of the search and after that, another heuristic should take the lead to discover better solutions for a problem instance. Based on this example, the main question for the hyper-heuristic research is how far a hyper-heuristic can be made more specific in relation to heuristic-problem state matchings. A perfect selection hyper-heuristic would systematically select the best heuristic in each application. The primary limitation to this perfect case is the problem-independent nature of the hyper-heuristics. This means that, it is disallowed to provide certain problem specific information to a hyper-heuristic. For this reason, a hyper-heuristic should consist of a number of learning devices to explore the search space of heuristics and gather the required knowledge to lift its generality level.

There is no study examining the generality potential of hyper-heuristics in detail. Chapter 5 discusses the foremost generality factors that need to be taken into account for a new hyper-heuristic aiming generality. Subsequently, a complete hyper-heuristic that can provide a high generality level does not exist.

This chapter presents particular components to build an evolutionary hyper-heuristic for the purpose of generality. First of all, the most relevant features reflecting the performance of a low-level heuristic are identified. Then, a number of components is developed relying on these features and possible changes that can be encountered during a run on the search space. Among these components, a heuristic subset selection approach is introduced. The subset selection operation is handled by an adaptive dynamic heuristic set (ADHS) strategy that explores the best heuristic subsets for different parts of the search. The selection process from these heuristics is guided by a probability vector that is updated based on the improvement capabilities and the speed of the heuristics. For more effective application of the heuristic set, a pairwise heuristic hybridisation approach is proposed. In addition, an adaptive threshold accepting strategy, i.e. adaptive iteration limited list-based threshold accepting (AILLA), is devised as the move acceptance mechanism. In the case of discovering good quality solutions immediately, the solution re-initialisation is triggered by the move acceptance mechanism to reach higher quality solutions faster. Figure 6.1 shows the whole working process of the designed hyper-heuristic, i.e. *generalised intelligent hyper-heuristic (GIHH)*[1].

For empirically examining the generality level of GIHH, a series of experiments over the instances from six problem domains provided by a high-level search library, HyFlex[255], is conducted. Prior to these experiments, GIHH competed against 19 other high-level, problem-independent algorithms in the first International Cross Domain Heuristic Search Challenge (CHeSC 2011). GIHH beats all these competing algorithms with respect to their overall performance across six problem domains by a large score difference. Furthermore, the experiments indicate that GIHH outperforms different hyper-heuristics from the literature for different running time limits.

The remainder of the chapter first treats the design principles in Section 6.1. Then, the hyper-heuristic framework for the generality purpose with all the algorithmic details is demonstrated in Section 6.2, 6.3 and 6.4. After that, the HyFlex software environment used for the experiments is explained in Section 6.5. Section 6.6 provides computational results as well as an analysis of the performance and the behaviour of GIHH. Section 6.7 concludes the chapter.

The details of this chapter were exhibited in the following papers:

> M. Mısır, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, Design and Analysis of an Evolutionary Selection Hyper-heuristic Framework, *under review.*

[1] Also referred as ADHS-AILLA

M. Mısır, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, An Intelligent Hyper-heuristic Framework for CHeSC 2011, in *Proceedings of the 6th Learning and Intelligent OptimizatioN Conference (LION'12), vol. 7219 of LNCS*, Paris, France, 2012.

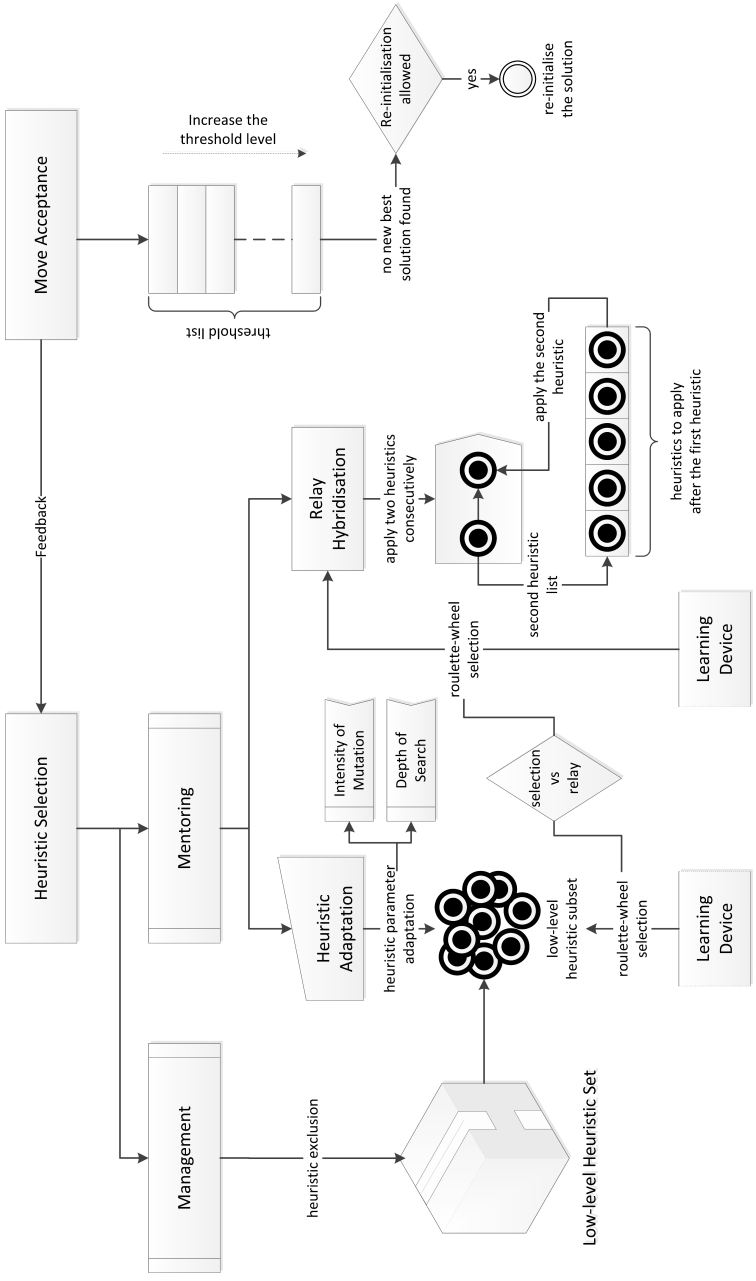

Figure 6.1: Generalised intelligent hyper-heuristic (GIHH)

6.1 Design principles

In most of the hyper-heuristic studies, validation proceeds on a set of instances from one problem domain with one heuristic set and a specific execution time limit. The hyper-heuristics in these studies have been designed and tuned to solve the target instances of only one problem. The aim is to deliver high quality solutions compared to state of the art methods. This is not in line with the main motivation of hyper-heuristics, which is to provide generality. A hyper-heuristic should be capable of working under different circumstances. Distinct heuristic sets with varying size, different instances from different problem domains and varying execution times should be accommodated.

The following list shows the main questions to be addressed by hyper-heuristic designers.

- which heuristic should be applied at each decision step?
- how to manage relatively large heuristic sets?
- what are the effects of different heuristic sets?
- how can a heuristic set be used to get the best performance?
- what should be the quality of a solution for accepting it?
- how should the search be diversified?

These questions can be answered before starting to solve a problem instance (offline) or while solving it (online). Offline learning methods need to be trained on a group of instances. Thus, the performance of a hyper-heuristic with offline learning is limited by these instances. Making decisions while solving a problem instance based on changing characteristics of a search space is always useful for better hyper-heuristics. In some cases, a very naive approach can deliver similar or even better quality results compared to a hyper-heuristic with sophisticated learning. The execution time limit is a major factor. For instance, if the number of improving or new best solutions is small with respect to the total number of iterations spent and the given execution time is relatively high, then choosing poor heuristics often will not cost much to the hyper-heuristic. On the other hand, in the case of a limited execution time, each decision is very valuable. Therefore a learning mechanism may be extremely useful in which case, generality requires a hyper-heuristic to have some learning components available.

In hyper-heuristic studies, the number of applied heuristics is generally small. Utilising as many heuristics as possible might bring increased performance. This

requires additional components for the management of such large sets. Therefore, a hyper-heuristic should accommodate certain features to handle heuristic sets of different sizes. Heuristic hybridisation opportunities and parameter adaptation methods for parametric heuristics should be investigated.

The decision to accept or reject a solution generated by the selected heuristics has a major effect on the overall performance of a hyper-heuristic. If the heuristic set consists of only a few heuristics, its effect is more significant. Move acceptance generally employs a method with diversification capabilities. The search process is diversified by accepting worsening solutions using threshold values under certain conditions. The evolvability characteristic of the search space is the most important criterion to decide on this threshold level. The number of neighbouring solutions around a current solution should be taken into account for the diversification strategy. Restarting and re-initilisation methods should be devised to support this diversification.

6.2 Management

The present tendency toward solving combinatorial search and optimisation problems is to combine multiple mechanisms to arrive at more competent algorithms. Aggregating the strength of multiple search mechanisms is the primary motivation underlying selection hyper-heuristics. Various components exist for achieving this management process favourably. A considerable number of these approaches involve learning devices to facilitate the selection of these mechanisms. For this purpose, the characteristics and performance of these search mechanisms are monitored *offline* or *online*. In this study, an online learning strategy, i.e. adaptive dynamic heuristic set, is employed to identify effective heuristic subsets for specific regions of the search space while addressing a target problem instance.

6.2.1 Adaptive dynamic heuristic set strategy

The availability of a large set of low-level search mechanisms can be considered an advantage because of its potential to explore large parts of the search space. Such a situation can turn into a disadvantage due to the hardness of managing the high number of low-level heuristics. In particular, this happens when these heuristics have different characteristics and a diverse speed range. Therefore we introduce the adaptive dynamic heuristic set (ADHS) strategy [228, 233]. This method assesses the performance of each heuristic at the end of a number of iterations to keep the best performing heuristics in the

set whilst excluding the other ones. The contribution of this method as a heuristic exclusion process to the whole framework is shown in Figure 6.1. The number of iterations required to determine such an elite heuristic subset refers to a phase. For the exclusion process, a performance metric identifying the most obvious performance elements is used. These elements are related to the improvement capabilities of the heuristics as well as to their speed. The details of this metric are shown in Equation 6.1. $C_{p,best}(i)$ denotes the number of new best solutions found. $f_{imp}(i)$ and $f_{wrs}(i)$ correspond to the total improvement and worsening. $f_{p,imp}(i)$ and $f_{p,wrs}(i)$ refer to the same measurement but only during a single phase. t_{remain} refers to the remaining time to finish the whole search process. $t_{spent}(i)$ and $t_{p,spent}(i)$ are the time spent by heuristic i until that moment, the same measurement during a phase respectively. Each w_i denotes the weight of its corresponding performance element. The weights are set as $\{w_1 >> w_2 >> w_3 >> w_4 >> w_5\}$ to provide a strict priority between the given performance elements.

$$
\begin{aligned}
p_i \;=\; & w_1\Big[\Big(C_{p,best}(i)+1\Big)^2\Big(t_{remain}/t_{p,spent}(i)\Big)\Big]\times b + \\
& w_2\Big(f_{p,imp}(i)/t_{p,spent}(i)\Big) - w_3\Big(f_{p,wrs}(i)/t_{p,spent}(i)\Big) + \\
& w_4\Big(f_{imp}(i)/t_{spent}(i)\Big) - w_5\Big(f_{wrs}(i)/t_{spent}(i)\Big) \qquad (6.1)
\end{aligned}
$$

$$
b \;=\; \begin{cases} 1, & \sum_{i=0}^{n} C_{p,best}(i) > 0 \\ 0, & otw. \end{cases} \qquad (6.2)
$$

The performance of the heuristics is measured with the performance metric in Equation 6.1 at the end of each phase. A phase refers to a number of iterations. The resulting p_i values appear to be noisy. Therefore, these values are converted into a quality index ($QI \in [1, n]$) value as a simple scoring mechanism. The heuristic with the lowest p_i gets $QI = 1$, the other heuristics get one unit more based on their order. For instance, for a heuristic set with 5 heuristics, the heuristics' QI values are set to {1,2,3,4,5} from the worst heuristic to the best performing heuristic. The average (avg) of these QI values is calculated to specify the heuristics that will be excluded. The heuristics that were already excluded in the previous phases, i.e. tabu heuristics, have $QI = 1$. All the heuristics with a QI lower than avg are excluded for a number of phases. The

duration of the exclusion in terms of number of phases is referred to as tabu duration (d). For decreasing user dependency, the tabu duration and the number of iterations for one phase are calculated based on the number of heuristics (n) available in the elite heuristic subset. The tabu duration is set to $d = \lfloor\sqrt{2n}\rfloor$. The phase length ($pl$) is defined as the product of the tabu duration and a constant value ($ph_{factor} = 500$). Both d and pl are updated at the end of each phase.

$$avg = \left\lfloor \left(\sum_{i}^{n} QI_i \right) / n \right\rfloor \tag{6.3}$$

Phase length adaptation

The phase length (pl) is the number of iterations required to check the performance of the heuristics within the ADHS strategy. Considering pl for different heuristic sets may be unfair in case of speed differences between heuristic subsets. Thus, pl is updated with respect to the average time (t_{subset}) required per move of the heuristic subset. As a reference point, we use a constant parameter denoting the number of phases requested ($ph_{requested}$) during the whole run to determine a possible phase duration. In order to effectively use ADHS, $ph_{requested}$ is set to an arbitrarily chosen, large enough value, i.e. 100. pl is calculated depending on this information as shown in Equation 6.4. $ph_{duration}$ shows the execution time allowed to complete a phase and it is calculated using the total execution time given (t_{total}) and $ph_{requested}$. pl is kept within certain bounds ($pl \in [d \times ph_{base}, d \times ph_{factor}]$) if the calculated pl is too low or too high. For determining the bounds calculated using the default tabu duration (d), ph_{base} is set to 50 for the lower bound and ph_{factor} is set to 500 for the upper bound.

$$\begin{aligned} pl &= ph_{duration} / t_{subset} \\ ph_{duration} &= t_{total} / ph_{requested} \end{aligned} \tag{6.4}$$

Tabu duration adaptation

We propose a way to adapt tabu duration of each heuristic. The tabu duration of each heuristic is the same at the beginning of the search and is updated during

the run. If an excluded heuristic enters the active heuristic set and is excluded immediately after this phase again, its tabu duration is incremented by 1. If this heuristic stays in the elite heuristic subset at the end of the phase, its tabu duration is set to its initial value ($d = \lfloor\sqrt{2n}\rfloor$). This incrementation continues until the tabu duration reaches its upper limit, resulting in a permanent exclusion of this heuristic.

Extreme heuristic exclusion

An additional heuristic exclusion procedure is used at the end of each phase to fasten the search process by immediately eliminating slow and ineffective heuristics. The metric employed for this purpose is shown in Equation 6.4. $t_{perMove}(i)$ denotes the average time spent for applying heuristic i once and $t_{perMove}(fastest)$ refers to the same measure for the fastest heuristic. $exc(i)$ indicates how many times heuristic i is slower than the fastest heuristic. The extreme exclusion process takes place under certain conditions presented in formula 6.5. The standard deviation (σ) and the average (ϖ) of the $exc(i)$ values are used to determine the heuristics that should be excluded. First, there should be large speed differences between the heuristics. This is valid if $\sigma > 2.0$. In addition, $exc(i)$ should be large enough ($exc(i) > 2\varpi$) meaning that heuristic i is very slow compared to the other heuristics. Finally, there always should be a number of heuristics that found new best solutions. In particular, if the heuristics residing in the current heuristic subset did not generate any new best solution (the number of new best solutions found: $nb = 0$) or only one heuristic found a new best solution ($nb = 1$) during the past iterations, no heuristics are excluded. Our aim here is to keep at least one heuristic that produced new best solutions before in the heuristic set.

$$exc(i) = \left\lfloor t_{perMove}(i)/t_{perMove}(fastest) \right\rfloor \tag{6.4}$$

$$\sigma > 2.0 \; ; \; exc(i) > 2\varpi \; ; \; nb > 1 \tag{6.5}$$

Heuristic selection

Most of the available heuristic selection mechanisms consider the improvement capabilities of the heuristics. The general trend is to choose effective heuristics subject to their performance changes. In GIHH, the selection operation is based on a probability vector determined and updated by the number of new best solutions and speed for each heuristic. Equation 6.6 shows the rule to specify

heuristic i's probability of being selected. The probability values should be normalised to keep the total equal to 1. The heuristic selection probabilities are calculated based on the number of new best solution found ($C_{best}(i)$) and the time spent (t_{spent}) by each heuristic. In order to provide large changes on the probabilities in the beginning of a search and reduce this effect during the later iterations, $3tf^3$ is used. tf is a parameter that is linearly decreasing from 1 to 0.

$$pr_i = \left((C_{best}(i) + 1)/t_{spent}\right)^{(1+3tf^3)} \tag{6.6}$$

6.3 Mentoring

In a traditional selection hyper-heuristic, one heuristic is selected at each decision step and applied as it is. It might be useful to additionally direct heuristics besides selecting them. In the mentoring part, possible performance enhancement opportunities are investigated based on this idea. The first approach consists in exploring whether heuristic hybridisations or heuristic pairs result in a better performance than when the heuristics are applied alone. Manipulating heuristics through playing with their parameters can also be considered a mentoring strategy. The following sub-sections explain how these two mentoring approaches work.

6.3.1 Relay hybridisation

Relay hybridisation [316] refers to consecutively applying meta-heuristics where each meta-heuristic uses the solution generated by the preceding meta-heuristic. In this study, we introduce a relay hybridisation technique that consecutively applies two low-level heuristics for finding new best solutions during a search. We first accommodate a heuristic list with the heuristics that found new best solutions when applying them as follow-up heuristics. The application of this list, which has been built for each heuristic, is illustrated in Figure 6.1. At each iteration, it is decided whether to use relay hybridisation as presented in Algorithm 5. C_{phase} is a counter showing the number of iterations finished within the current phase. $C_{best,s}$ and $C_{best,r}$ refer to the number of new best solutions found by the regular, single heuristic selection mechanism and by the relay hybridisation respectively. A randomly generated value ($p \in [0, 1]$) is employed for supporting this decision. p is compared to $(C_{phase}/pl)^{\gamma}$. The value in the parentheses linearly changes from 0 to 1. This means that it is

preferred to use single heuristics over relay hybridisation during the earlier iterations of a phase. γ is responsible for determining how to effectively use relay hybridisation. It respects bounds that were set based on some preliminary experiments, $\{(1/R, R)$ for $R = 50\}$. If more new best solutions are generated by the relay hybridisation than single heuristics, γ is lower than 1. In such a case, the relay hybridisation will have a high selection probability than single heuristics. If γ is higher than 1, single heuristics are preferred. For example, the phase length is set as $pl = 1000$ and the half of the phase is passed, $C_{phase} = 500$. Then, $(C_{phase}/pl) = 0.5$. If the number of new best solutions found by single heuristics and by relay hybridisation are the same, the relay hybridisation is performed with a 50% probability. If the number of new best solutions found by single heuristics is higher, e.g. $C_{best,s} = 1000$ and $C_{best,r} = 10 \rightarrow \gamma = 91$, the relay hybridisation is less preferable ($(C_{phase}/pl)^{\gamma}$=4.03e-28). If the number of new best solutions found by the relay hybridisation is higher, e.g. $C_{best,s} = 10$ and $C_{best,r} = 1000 \rightarrow \gamma \approx 0.011$, the relay hybridisation is preferred more ($(C_{phase}/pl)^{\gamma} \approx 0.99$).

Algorithm 5: Relay hybridisation

Input: heuristic list size $list_{size} = 10$; randomly generated values $p, p' \in [0, 1]$

```
1 γ = (C_best,s + 1)/(C_best,r + 1)
2 if p ≤ (C_phase/pl)^γ then
3 |   select LLH using a LA and apply to S → S'
4 |   if size(list_i) > 0 and p' <= 0.25 then
5 |   |   select a LLH from list_i and apply to S' → S''
6 |   else
7 |   |   select a LLH and apply to S' → S''
  |   end
  end
```

In addition, the tabu approach used in ADHS is employed to disable relay hybridisation if no best solution was found during a phase. If this is repeated during consecutive phases, the tabu duration is incremented by 1 like in ADHS.

A learning automaton maintaining the selection probabilities of the heuristics was employed in the relay hybridisation. We previously proposed a learning automaton to choose single heuristics, presented in Chapter 4. For the update operations, a *linear reward-inaction* update scheme was used. This scheme rewards heuristics discovering new best solutions. No other type of outcome was used to update the probabilities. The corresponding learning rate λ_1 was set to 0.5.

After choosing the first heuristic for the relay hybridisation, a second heuristic

is chosen for applying it to the solution generated by the first heuristic. If the corresponding heuristic list is not empty and a randomly generated value ($p' \in [0, 1]$) is less than or equal to 0.25, the second heuristic is randomly selected from the list. Otherwise, a heuristic is selected from the current heuristic subset used for single heuristics.

6.3.2 Parameter adaptation of the parametric heuristics

In the literature, there are different types of heuristics such as mutational heuristics and hill climbers, that can be used in a heuristic set. For the purpose of generality, it is required to have a hyper-heuristic that can manage any type of heuristics. Hence, we define a number of generic heuristic types, i.e. *ImprovingOrEqual*, *ImprovingMore*, *WorseningMore*, *WorseningOrEqual*, *OnlyEqual*, to cover all possible heuristics. These generic types are essentially used to adapt the parameters of the parametric heuristics such as the mutation rate of a mutational heuristic. Such parameters are adapted by considering the heuristics' types. These types are always checked to reflect any change. For instance, one heuristic can be *ImprovingMore* during early iterations, then it can turn into a *WorseningMore* type. In such cases, changing reward and penalty functions may be very effective. The details of the parameter (val_i) update operations are shown in Algorithm 6. The θ values are the update rates for the given environmental feedbacks as presented in Table 6.1. This adaptation process was handled by setting some predetermined values. These values were set based on a simple logic. Both improvement capabilities and speed changes determined the best parameter values. Thus, the parameter of a heuristic with high improvement performance should not be consistently increased and at the same time, the parameter of a heuristic with high worsening performance should not be immediately decreased. All these parameters should be set in such a way that their strengths are in balance.

Usually heuristics used in the hyper-heuristic studies require one or two solutions. For the heuristics requiring two solutions, a population of five solutions including previously explored new best solutions is accommodated. They are applied using the current solution and a randomly selected one from these five solutions. Each time a new best solution is found, a randomly chosen solution from these solutions is replaced by the new solution.

Oscillating parameters

Figure 6.1 shows a feedback transmission from the move acceptance mechanism through the heuristic selection. This feedback warns the selection mechanism if

	Feedback	Value
θ_1	new best solution	0.01
θ_2	better solution	0.001
θ_3	worse solution	0.0005
θ_4	equal quality solution	0.0001

Table 6.1: Learning rates used for the adaptation of the heuristics' parameters (Algorithm 6)

the system fails to generate improvements. This warning is sent if re-initialisation is disabled due to the fact that re-initialisation is employed for the same purpose. It means that this issue could not be handled by the move acceptance mechanism, i.e. the threshold level reaches l. Whenever these conditions occur, the parameters of the parametric heuristics oscillate. For the *OnlyImproving* heuristics, val_i is linearly updated between 0.5 and 1.0. For the other heuristics, val_i is changed between 0.2 and 0.5. At each 5000th iteration, the corresponding parameters are updated between their limits.

6.4 Move acceptance

6.4.1 Adaptive iteration limited list-based threshold accepting

Adaptive iteration limited list-based threshold accepting (AILLA) is a new acceptance mechanism developed during the course of this research. AILLA determines the threshold level in an adaptive manner using the fitness values of the previously visited new best solutions [233]. Its simplest version, i.e. iteration limited threshold accepting (ILTA) [241], requires two parameters. The first parameter determines the number of consecutively found worsening solutions. If this number reaches a predetermined iteration limit, then a worsening solution is accepted provided that meets a threshold value. This threshold value is determined based on the current best solution and a constant range factor. That is, the threshold value changes whenever a new best solution is found. Adaptive ILTA (AILTA) [232] adds another iteration limit for increasing the threshold value if required. Each time the number of consecutively encountered worsening solutions reaches this value, the range value is updated by another constant factor. This operation provides a larger search region to get out of the point the search process converged to. The value of the range parameter is allowed to be incremented to an upper bound. On the other hand, AILLA determines threshold values for accepting worsening solutions based on previously found

Algorithm 6: Parameter adaptation for the heuristics

$u = +1$; $p \in rand(0,1)$; $val_i \in [0.2, 1]$

$$\theta = \begin{cases} \theta_1, & \text{if } f(S') < f(S_b) \\ \theta_2, & \text{if } f(S') < f(S) \\ -\theta_3, & \text{if } f(S') > f(S) \\ -\theta_4, & \text{otherwise} \end{cases}$$

if $heuristic\_type = ImprovingOrEqual$ **then**
  if $(f(S') < f(S)$ *and* $p < 0.5)$ *or* $(f(S') = f(S)$ *and* $0.25 \leq p < 0.5)$ **then**
    $u = 0$
  else if $f(S') = f(S)$ *and* $p < 0.25$ **then**
    $u = -1$
  end
else if $heuristic\_type = ImprovingMore$ **then**
  if $(f(S') < f(S)$ *and* $0.25 \leq p < 0.5)$ *or* $(f(S') \geq f(S)$ *and* $p < 0.5)$ **then**
    $u = 0$
  else if $f(S') < f(S)$ *and* $p < 0.25$ **then**
    $u = -1$
  end
else if $heuristic\_type = WorseningMore$ **then**
  if $f(S') < f(S_b)$ *and* $p < 0.5$ **then**
    $u = 0$
  else if $(f(S') < f(S)$ *and* $p < 0.5)$ *or* $f(S') = f(S)$ **then**
    $u = -1$
  end
else
  if $f(S') = f(S)$ **then**
    $u = -1$
  end
end
$val_i = val_i + \theta \times u$

new best solutions constituting a fitness list as displayed in Figure 6.1. It accepts improving and equal quality solutions. A worse solution is accepted if no new best solutions were found after k consecutive worsening solutions and if the fitness value of the new solution is lower than a threshold value from the fitness list. Additonally, it involves certain adaptive components to bring high performance with less user dependency. The details of AILLA are presented in Algorithm 7.

Algorithm 7: AILLA move acceptance

Input: threshold index $i = 1$, iteration limit $k \geq 0$, iteration limit to increase the threshold level $K \geq k$, list length $l > 0$

for *j=0* **to** *l-1* **do** $best_{list}(j) = f(S_{initial})$
1 **if** $adapt\_iterations \geq K$ **then**
2 **if** $i < l - 1$ **then**
3 $i++$
  end
  end
4 **if** $f(S') < f(S)$ **then**
5 $S \leftarrow S'$
6 $w\_iterations = 0$
7 **if** $f(S') < f(S_b)$ **then**
8 $i = 1$
9 $S_b \leftarrow S'$
10 $w\_iterations = adapt\_iterations = 0$
11 $best_{list}.remove(last)$
12 $best_{list}.add(0, f(S_b))$
13 $\boldsymbol{update(k, K)}$
  end
14 **else if** $f(S') = f(S)$ **then**
15 $S \leftarrow S'$
16 **else**
17 $w\_iterations++$
18 $adapt\_iterations++$
19 **if** $w\_iterations \geq k$ *and* $f(S') \leq best_{list}(i)$ **then**
20 $S \leftarrow S'$ and $w\_iterations = 0$
  end
  end

Adaptive iteration limit

The iteration limit ($k \in [5, \infty)$) is an major parameter for AILLA. The effectiveness of a specific k-value may depend on the problem instance. Thus, this value is updated by a simple parameter control strategy as explained in Equation 6.7. The update operation is triggered when a new best solution is found as shown in Line 13, Algorithm 7. $iter_{elapsed}$ refers to the number of iterations elapsed, $t_{elapsed}$ shows the spent execution time, t_{exec} means the total given execution time. cw indicates the number of times k iterations spent without a new best solution. If cw is zero meaning that a new best solution

is found before k iterations spent, k is decreased. Otherwise, it is increased. While incrementing k, it is considered not to make large changes on k by using 0.5^i. The reason behind this approach is to take into account the possibility of spending much time due to searching across the same search region and visiting the same solutions. Additionally, it is known that it gets harder to deliver improving solutions during further iterations. This means that the number of iterations spent to find a new best solution does not reflect the required iterations to find a new best solution as much as during earlier iterations. Hence, the effect of the number of iterations spent during further iterations is reduced over time using a linearly decreasing time factor (tf).

$$
\begin{aligned}
k &= \begin{cases} \left\lfloor ((l-1) \times k + iter_{elapsed})/l \right\rfloor, & \text{if } cw = 0 \\ \left\lfloor ((l-1) \times k + \sum_{i=0}^{cw} k \times 0.5^i \times tf)/l \right\rfloor, & \text{otherwise} \end{cases} \\
& tf = (t_{exec} - t_{elapsed})/t_{exec} \\
& cw = \left\lfloor iter_{elapsed}/k \right\rfloor
\end{aligned} \tag{6.7}
$$

The other iteration limit (K) used to increase the threshold level is set based on k. Each time k is updated, K is set to $k \times 5$.

Decreasing list length

The list length (l) of AILLA is decreased over time in order to reduce the level of diversification towards the end of the search process. The implemented update method is shown in Equation 6.6. In this study, the parameters were set as follows: $l_{base} = 5$ and $l_{initial} = 10$. This means the l decreases from 10 to 5 over time. The aforementioned time factor (tf) is used in a way that l exponentially decreases.

$$
l = l_{base} + (l_{initial} - l_{base} + 1) \times tf^3 \tag{6.6}
$$

Re-initialisation

Due to the often stochastic nature of heuristic methods, it is impossible to determine the exact execution time required to find a solution at a specific

quality level. Thus, the same algorithm running multiple times may produce different results. Re-initialisation methods and restarting mechanisms may be useful to take advantage of such circumstances. These approaches should be considered together with the other mechanisms diversifying the search. The acceptance mechanism is already responsible for diversification. The level of diversification provided by the acceptance depends on its list length. Finding a good value for the list length can be problematic. An adaptive parameter based on the correlation between the solutions belonging to a specific sub-region may be effective. However, the requirement of exploitation in a limited time should be handled using a more effective method. For that purpose, whenever the threshold level reaches l, with additional limitations a new initial solution is generated as depicted in Figure 6.1. Based on the remaining execution time, the cost of re-initialisation and the possibility of finding a new best solution after re-initialisation, it is decided whether the re-initialisation should be disabled or not. Each time it is disabled, the best solution found after re-initialisations is used as the current solution for further improvement. For consistency with the acceptance mechanism, the list constituted for this specific solution is also employed.

6.5 HyFlex

HyFlex is a software environment providing a number of instances from different problem domains [255]. The purpose is to design and apply a selection hyper-heuristic to show its level of generality across the given instances. As one of the main traits of hyper-heuristics, the problem-independency is guaranteed by this environment. Therefore, a specific heuristic set for each problem domain is also provided. For the sake of heuristic diversity, four types of heuristics were implemented in each heuristic set. These types are *mutational heuristics*, *ruin-recreate heuristics*, *local search* and *crossovers* as mentioned in the previous section. Here, local search heuristics (hill climbers) guarantee to find improved or equal quality solutions with respect to the solutions they start from. The rest of the heuristics may also generate worsening solutions. Some of these heuristics can be manipulated by changing their parameters. A group of heuristics uses parameters called "*intensity of mutation*" denoting the impact of the perturbation. Each local search heuristic have a parameter called "*depth of search*" that refers to the number of steps to check for improvement.

Six problems from different domains are available in HyFlex. The problems are the 1D bin packing, max SAT, permutation flowshop scheduling, personnel scheduling, travelling salesman and vehicle routing problems. More details about the problems and instances are presented in Chapter 3, Section 3.6.

6.6 Computational results

The hyper-heuristic developed, ADHS-AILLA, was applied 10 times on each problem instance using Pentium Core 2 Duo 3GHz PCs with 3.23 GB memory on Windows XP. There are 12 instances for the max SAT, bin packing, permutation flowshop scheduling and personnel scheduling problems. 10 instances are available for the travelling salesman and vehicle routing problems. GIHH was tested for 10 minutes and 1 hour execution time on all these instances.

	Inst.	ADHS-AILLA		ADHS-GD		ADHS-LATE		ADHS-SA		ADHS-IE		SR-AILLA	
		min	avg	min	avg	min	avg	min	avg	min	avg	min	avg
Max SAT	0	2	2,7	4	11,6	4	8,4	2	3	7	12	5	8,7
	1	20	23,8	23	29,6	23	25,6	20	26,8	22	28,2	21	24,5
	2	15	18,1	21	25,7	18	22,3	19	21,5	19	24,4	14	19,2
	3	1	3,5	5	12	7	14,4	4	9,1	15	19,6	5	9,3
	4	1	2,7	2	17,6	2	22	2	15,6	13	28,4	6	10
	5	3	13,2	18	38,9	19	39,3	7	26,9	24	42	16	28,6
	6	5	5,3	7	8,1	5	9,3	5	7,9	5	10,2	5	6,8
	7	5	5,4	7	8,3	7	9,6	5	7,5	8	10,4	5	6,1
	8	5	7,1	9	11,9	9	11,4	9	10,1	9	13,6	8	8,8
	9	209	212	213	216,2	213	217,2	209	214,3	215	218,4	211	212,9
	10	1	2,2	11	17,7	9	14,5	1	7,3	15	18,8	11	14,4
	11	7	8	10	11,5	9	12,1	7	9,7	12	13,6	7	9,1
Bin Packing	0	0,00276	0,00565	0,00637	0,00686	0,00264	0,00544	0,00595	0,00673	0,00244	0,00449	0,00659	0,01024
	1	0,00288	0,00442	0,00671	0,00716	0,00281	0,00433	0,00670	0,00706	0,00335	0,00483	0,00751	0,00940
	2	0,02030	0,02115	0,01871	0,01942	0,02030	0,02128	0,01840	0,01928	0,01971	0,02160	0,02106	0,02260
	3	0,01970	0,02089	0,01872	0,01911	0,01970	0,02185	0,01872	0,01902	0,01970	0,02143	0,02039	0,02236
	4	0,00034	0,00250	0,00501	0,00542	0,00034	0,00440	0,00541	0,00566	0,00034	0,00422	0,00034	0,00375
	5	0,00306	0,00348	0,00374	0,00403	0,00306	0,00358	0,00360	0,00545	0,00306	0,00343	0,00306	0,00335
	6	0,01082	0,01522	0,00572	0,01310	0,00561	0,01549	0,00563	0,01013	0,01069	0,01477	0,01107	0,01569
	7	0,01340	0,01524	0,02149	0,02623	0,01300	0,01533	0,01964	0,03199	0,01358	0,01530	0,05635	0,05989
	8	0,04081	0,04263	0,03701	0,03820	0,03733	0,04080	0,04243	0,04419	0,03997	0,04215	0,11697	0,12356
	9	0,00141	0,00363	0,00153	0,00354	0,00049	0,00273	0,00152	0,00301	0,00146	0,00303	0,03402	0,03641
	10	0,10828	0,10828	0,10828	0,10828	0,10828	0,10828	0,10834	0,10840	0,10828	0,10828	0,11051	0,11089
	11	0,00234	0,00378	0,00428	0,01164	0,00352	0,00601	0,00556	0,01101	0,00222	0,00335	0,05932	0,06371
Perm. Flowshop	0	6251	6271,9	6266	6300	6258	6300,7	6266	6297,8	6265	6304,7	6326	6345,2
	1	6225	6238	6232	6257	6244	6262,9	6231	6259,8	6243	6262,5	6261	6297,2
	2	6309	6316,1	6313	6341,5	6324	6343	6307	6339,9	6313	6336,1	6342	6364,5
	3	6318	6340,6	6306	6339,9	6310	6355,1	6305	6342,8	6323	6342,7	6331	6363,6
	4	6364	6386,8	6361	6384,8	6368	6391,3	6357	6387	6362	6386,8	6394	6418,8
	5	10493	10497,3	10497	10502,2	10497	10501,5	10494	10505,1	10494	10503,4	10497	10514
	6	10922	10925	10922	10923,3	10922	10922,9	10923	10923	10922	10925,3	10922	10927,2
	7	26245	26296	26247	26299,3	26366	26399,3	26275	26303,3	26220	26297,5	26328	26391
	8	26778	26812,7	26771	26809,6	26804	26883,3	26704	26786	26779	26815,7	26787	26829,6
	9	26614	26678,3	26578	26654,2	26646	26710,1	26558	26662,9	26593	26649,4	26599	26690,3
	10	11319	11364,6	11333	11360,6	11336	11398,5	11344	11379,5	11319	11375,7	11397	11436,8
	11	26585	26630,1	26514	26611,1	26640	26680,9	26574	26644,8	26577	26626,9	26595	26659,9
Pers. Scheduling	0	3300	3332,3	3321	3335,4	3310	3336,3	3296	3325,5	3311	3338,1	3323	3347,6
	1	2203	2346,1	1975	2280,4	2230	2357	2025	2443,9	2050	2348,5	2228	2475,7
	2	335	373	355	389,5	345	386	330	379,5	315	390,5	385	1178,1
	3	16	22,1	20	25	18	23,9	20	29,3	20	27,8	13	24,1
	4	20	25,6	17	24	22	26,6	25	32,1	21	27,2	21	25,3
	5	18	25,9	23	25,7	19	28,8	25	31	29	34	11	27,3
	6	1108	1169,2	1128	1152,1	1110	1147,2	1121	1229,6	1106	1251,2	1108	1234,9
	7	2189	2282,7	2221	2304,8	2210	2265	2182	2279,9	2174	2267	2184	2273
	8	3173	3258,9	3170	3257	3267	3324,4	3177	3349,6	3154	3355,9	3172	3311,7
	9	9406	9614,8	9619	10016,9	10181	11463,7	9440	9663,4	9390	9614,2	9522	9718,6
	10	1535	1670,8	1620	1723	1489	1652,9	1655	1839,2	1560	1750,1	1689	1995,9
	11	315	335,5	330	353,1	310	345,5	290	318	305	466,5	355	943,5
TSP	0	48194,92	48195,44	48194,92	48213,29	48194,92	48213,29	48194,92	48222,47	48194,92	48250,02	48689,30	49109,89
	1	107509,62	109023,22	108207,20	109058,58	108121,24	109295,50	109212,79	109485,25	108300,28	109113,91	110437,02	111682,08
	2	6797,74	6809,59	6799,68	6810,81	6799,67	6816,14	6795,97	6809,00	6799,30	6808,64	6944,03	6979,64
	3	41965,50	42025,67	41951,46	42022,91	41972,26	42048,51	41945,83	42038,86	41970,75	42027,68	42702,84	43038,84
	4	8857,43	8875,78	8861,95	8881,52	8863,07	8886,16	8842,99	8887,62	8867,05	8885,60	9010,54	9112,40
	5	57305,54	57473,51	57201,15	57506,05	57246,11	57451,09	57261,60	57453,98	57177,79	57442,20	59593,48	60063,03
	6	52513,79	53097,43	52870,50	53993,95	52587,69	53519,85	52028,89	53437,29	52965,01	53661,89	53873,99	54773,29
	7	66219,91	66700,34	66531,26	66925,67	66316,43	66674,21	66154,66	66674,01	66360,95	66761,00	68816,69	69137,90
	8	20688884,44	20791227,71	20713409,15	20804743,56	20766756,65	20857643,74	20703118,69	20798709,26	20688847,10	20815142,59	21203912,18	21335269,51
	9	667481,20	669868,77	667899,50	671185,01	669271,95	670847,38	668223,98	670190,82	668466,98	669942,61	675137,61	676934,66
VRP	0	5093,77	5123,63	4158,04	4775,20	5057,15	5122,85	5102,44	5149,55	5135,82	5201,24	5100,96	5140,79
	1	20654,06	20758,71	20652,47	20755,40	20660,03	21265,34	21650,18	21774,53	20695,61	21989,32	20651,26	20656,30
	2	13286,60	13340,39	12330,09	13152,21	13361,03	13787,77	13312,37	13873,27	14384,55	14521,91	13319,40	13335,66
	3	5301,64	5330,35	5317,38	5355,76	5313,71	5338,24	5331,32	5552,44	5349,24	5664,67	5310,36	5345,73
	4	14264,83	14289,90	13277,95	14074,31	14277,80	14424,07	14282,24	15022,00	14310,28	14930,38	14257,83	14283,32
	5	145390,19	148755,39	142479,18	144776,16	142490,60	146715,69	145424,21	149171,17	145436,10	149344,45	147491,90	152626,07
	6	58170,85	61050,39	56757,46	59768,08	57152,31	60290,38	58782,04	60672,66	58788,27	60177,41	59234,69	59932,63
	7	159715,50	162125,26	159266,06	160807,34	159383,38	160535,57	159587,27	161230,18	160448,29	161754,46	161810,90	162411,09
	8	148316,30	152533,05	145129,59	148821,77	148903,27	151465,09	151092,14	154398,56	153854,25	157180,94	155569,62	158679,76
	9	145998,40	148724,85	145598,32	147328,19	145118,16	146872,12	145824,15	147828,00	146885,89	148502,40	146922,79	148065,62

Table 6.2: The experimental results with 10 minutes of execution time (Re-initialisation is available only for ADHS-AILLA)

	Inst.	ADHS-AILLA		ADHS-GD		ADHS-IE		ADHS-LATE		ADHS-SA	
		min	avg	min	avg	min	avg	min	avg	min	avg
Max SAT	0	2	2,5	4	8,1	5	12,6	2	7,4	2	2,5
	1	20	21,8	24	27,4	21	25,5	23	25,9	21	24,1
	2	15	16,7	16	23,5	19	22,6	18	21,6	17	20,3
	3	1	2,2	4	9,2	7	15,9	4	10,6	2	4,8
	4	0	1,7	1	10,7	11	24,9	5	22,2	2	7,7
	5	1	3,6	9	31,8	20	40,7	9	28,7	1	22
	6	5	5,1	5	7,9	8	10,1	6	8,9	5	6,9
	7	5	5	7	8,3	8	10,5	6	8,9	5	6,6
	8	5	6,6	8	11	11	12,7	10	11,8	7	9,1
	9	209	210,4	211	214,8	213	218,5	213	216,9	211	215,4
	10	1	1,3	11	14	16	18,2	2	10,3	1	5,5
	11	7	7,9	9	12	11	14,2	8	11,3	9	9,8
Bin Packing	0	0,00214	0,00324	0,00593	0,00607	0,00220	0,00294	0,00229	0,00402	0,00594	0,00612
	1	0,00277	0,00329	0,00358	0,00547	0,00254	0,00294	0,00273	0,00374	0,00365	0,00595
	2	0,01923	0,02024	0,01837	0,01894	0,02009	0,02124	0,01986	0,02113	0,01818	0,01881
	3	0,01873	0,02010	0,01872	0,01875	0,01937	0,02042	0,01900	0,02159	0,01871	0,01886
	4	0,00034	0,00161	0,00456	0,00484	0,00034	0,00249	0,00034	0,00395	0,00501	0,00529
	5	0,00306	0,00328	0,00347	0,00376	0,00306	0,00314	0,00306	0,00337	0,00387	0,00402
	6	0,01001	0,01428	0,00496	0,00543	0,00994	0,01384	0,01055	0,01397	0,00485	0,00528
	7	0,01062	0,01220	0,00809	0,01234	0,01017	0,01224	0,01014	0,01281	0,00829	0,01326
	8	0,03975	0,04116	0,03266	0,03318	0,03733	0,04138	0,03188	0,03421	0,03696	0,04025
	9	0,00042	0,00207	0,00051	0,00136	0,00042	0,00174	0,00042	0,00137	0,00054	0,00202
	10	0,10828	0,10828	0,10828	0,10828	0,10828	0,10828	0,10828	0,10828	0,10830	0,10834
	11	0,00126	0,00213	0,00427	0,00542	0,00036	0,00177	0,00033	0,00218	0,00461	0,00858
Perm. Flowshop	0	6245	6257,4	6257	6281,9	6245	6285,2	6263	6290,1	6272	6286,7
	1	6212	6223,5	6232	6247,2	6227	6247,8	6235	6251,4	6217	6244,3
	2	6284	6307	6315	6332,2	6300	6325,2	6302	6320,5	6299	6327,1
	3	6287	6307,4	6303	6322,3	6303	6336,3	6303	6340,6	6303	6337,6
	4	6343	6361,2	6352	6369,5	6342	6365,2	6350	6367,8	6344	6364,4
	5	10485	10492,9	10487	10495,4	10494	10496,4	10494	10496,6	10485	10495,2
	6	10922	10922,2	10922	10922,7	10922	10922,8	10922	10922,6	10922	10922,6
	7	26197	26231,5	26199	26250,7	26193	26227,6	26191	26241	26215	26241,3
	8	26674	26732,1	26721	26749,5	26687	26758,2	26694	26738,3	26689	26732,3
	9	26553	26584	26549	26596,1	26546	26595,5	26549	26596,6	26549	26594,9
	10	11308	11317,6	11319	11345,7	11308	11336,5	11319	11344,3	11313	11343,6
	11	26464	26507,8	26537	26586	26529	26578,3	26527	26582,2	26486	26564
Pers. Scheduling	0	3294	3308,4	3290	3301,9	3291	3311,1	3292	3315,2	3292	3311,2
	1	1998	2177,7	1978	2101,7	2120	2267,3	1995	2117,8	2078	2308,6
	2	300	342	335	356	280	313	300	328	290	315
	3	11	16,8	14	18	18	25,3	12	18,4	12	19,3
	4	15	18,6	15	19,5	23	27,1	14	18,5	21	25,5
	5	16	18,6	16	20	19	26,3	17	21	19	23,9
	6	1093	1115,6	1101	1114,1	1102	1169,9	1100	1105,9	1098	1139,1
	7	2149	2221,5	2191	2219,7	2155	2264,9	2186	2227,2	2157	2249,5
	8	3133	3181,3	3139	3204,1	3121	3196	3138	3169,8	3135	3215,7
	9	8978	9337,9	9503	9658,1	9194	9385,5	9524	9721,9	9253	9343,1
	10	1470	1559,7	1400	1476,4	1530	1840,2	1515	1590,3	1523	1716,4
	11	280	303	300	323,5	280	298,5	300	312	280	298,5
TSP	0	48194,92	48194,92	48194,92	48222,47	48194,92	48222,47	48194,92	48213,29	48194,92	48204,10
	1	107294,71	108541,14	107215,30	108997,10	108121,24	109024,87	107215,30	109119,62	107215,30	108842,12
	2	6795,97	6801,87	6795,97	6807,32	6795,97	6802,83	6795,97	6803,26	6795,97	6805,95
	3	41925,50	41970,03	41935,03	41987,48	41934,72	41995,07	41918,65	41989,03	41912,61	41985,74
	4	8846,81	8863,45	8852,68	8875,22	8855,73	8869,37	8848,67	8870,53	8848,67	8865,17
	5	56938,34	57136,71	57161,45	57322,03	56971,70	57353,65	56991,95	57315,07	57142,49	57322,33
	6	52582,83	52862,89	51928,01	53005,08	52684,92	53293,96	52497,38	53236,37	52389,69	53154,93
	7	65651,15	66234,31	66039,03	66315,11	65562,45	66153,53	65800,85	66153,77	65730,81	66078,90
	8	20587684,00	20628368,50	20642078,05	20708254,72	20602094,23	20692600,71	20579110,06	20667170,61	20556131,96	20654062,46
	9	664479,75	667026,10	666798,86	668122,40	666567,89	667311,87	667666,43	668426,51	665877,60	667219,99
VRP	0	4164,31	4918,88	4152,99	4754,00	5092,29	5148,17	5075,22	5147,68	5082,57	5129,20
	1	20652,47	20654,19	20650,80	20750,46	20676,70	21372,07	20656,49	21267,62	20672,64	21778,58
	2	12361,88	13040,32	12289,92	12730,22	13316,18	14094,39	13300,80	13778,65	13342,02	13867,59
	3	5265,99	5300,69	5288,31	5324,26	5341,89	5561,12	5315,48	5355,97	5297,97	5435,10
	4	13287,09	14079,33	13267,47	13787,66	14299,57	14928,77	14266,88	14612,32	14263,55	14489,12
	5	145321,58	146507,32	142479,08	144035,49	146775,65	148934,00	144153,69	146942,49	144062,78	149416,32
	6	59555,53	65282,61	56950,10	58007,13	58457,22	60492,64	57327,48	60245,43	58045,27	60131,58
	7	159878,05	160253,58	158404,97	159828,78	160645,44	162229,53	159321,12	160608,18	160735,40	161714,55
	8	146523,89	152184,37	142726,85	144540,10	151237,31	155370,15	146106,56	151197,18	152788,51	155142,48
	9	146331,27	148737,45	144534,23	145606,71	146805,26	147785,55	145243,45	146852,47	146614,73	147457,57

Table 6.3: The experimental results with 1 hour of execution time (Reinitialisation is available only for ADHS-AILLA)

	Inst.	SR-AILLA		SR-GD		SR-IE		SR-LATE		SR-SA	
		min	avg	min	avg	min	avg	min	avg	min	avg
Max SAT	0	4	6,6	118	182,8	8	14	6	9,3	2	4,3
	1	20	21,9	183	227,9	19	26,5	22	24,2	18	20,5
	2	14	17,4	166	229	15	20	15	17,9	15	16,5
	3	5	8,9	61	78,4	20	29	7	10,2	2	5,9
	4	5	7,1	60	79,8	36	41,1	5	8,5	1	3,9
	5	10	14,8	152	186,3	34	53,9	8	12,7	5	10,8
	6	5	5,6	13	17,1	7	9,9	8	9,7	5	5,6
	7	5	6,1	14	16,1	7	9,5	6	7,8	5	6
	8	5	7,5	15	21,8	8	11,4	9	10,3	5	7,7
	9	209	211,2	231	243	216	218,6	211	215,4	209	212
	10	3	11,1	105	134,7	24	28,8	6	11,9	1	2
	11	7	7,7	20	23,8	11	12,9	8	10,9	7	8,4
Bin Packing	0	0,00282	0,00558	0,03183	0,03512	0,00281	0,00574	0,00280	0,00551	0,00778	0,01055
	1	0,00329	0,00412	0,03213	0,03518	0,00346	0,00485	0,00335	0,00483	0,00766	0,00823
	2	0,01926	0,02074	0,02577	0,02665	0,02253	0,02372	0,02188	0,02320	0,01849	0,01889
	3	0,01937	0,02034	0,02844	0,02934	0,02349	0,02534	0,02155	0,02525	0,01871	0,01872
	4	0,00034	0,00034	0,01923	0,01991	0,00034	0,00119	0,00456	0,00463	0,01193	0,01263
	5	0,00306	0,00306	0,01821	0,01852	0,00314	0,00332	0,00358	0,00376	0,00966	0,01081
	6	0,01005	0,01185	0,09091	0,09738	0,00958	0,01232	0,00548	0,00937	0,00999	0,01250
	7	0,01329	0,01464	0,09546	0,10201	0,01334	0,01525	0,01412	0,01595	0,01437	0,02011
	8	0,05466	0,05719	0,07386	0,07619	0,05009	0,05288	0,05833	0,06008	0,03956	0,04078
	9	0,01159	0,01408	0,02458	0,02550	0,01247	0,01427	0,01461	0,01523	0,01127	0,01250
	10	0,10836	0,10839	0,11845	0,11980	0,10841	0,10844	0,10869	0,10875	0,11022	0,11062
	11	0,02093	0,02427	0,04205	0,04363	0,02077	0,02572	0,02991	0,03206	0,01979	0,02153
Perm. Flowshop	0	6301	6331,5	6364	6376,3	6319	6338,6	6298	6324,5	6290	6326,8
	1	6266	6286,5	6293	6322,3	6271	6308,1	6248	6294,5	6289	6303,6
	2	6328	6351,4	6384	6398,7	6318	6357,7	6338	6354,1	6336	6354,8
	3	6323	6347,8	6361	6378,3	6323	6354,5	6326	6350	6327	6354,9
	4	6377	6399,5	6447	6460	6377	6400,5	6377	6400,1	6367	6396,3
	5	10499	10515,6	10515	10529,4	10501	10528,3	10528	10539,9	10507	10534,8
	6	10922	10922,7	10947	10963,2	10922	10927,5	10922	10927,5	10923	10935
	7	26272	26328,8	26380	26430,6	26360	26451,4	26327	26384,9	26389	26474,4
	8	26764	26814,8	26869	26914,2	26873	26932,8	26787	26834,8	26841	26915,2
	9	26612	26647	26713	26732,2	26697	26754,9	26665	26697,7	26674	26748,3
	10	11377	11412,5	11454	11491,9	11389	11464,2	11432	11461,3	11426	11481,1
	11	26574	26629,4	26662	26703,8	26718	26753,9	26608	26663,4	26649	26721,8
Pers. Scheduling	0	3296	3326,6	3313	3343,3	3295	3336,9	3294	3317,3	3295	3328,9
	1	2120	2377,4	2310	2410,4	2260	2399,2	2040	2159,9	2130	2409,3
	2	350	404	360	387	350	1004,5	340	364	485	1210
	3	14	18,9	18	28,5	17	24,8	11	16,8	18	23,4
	4	14	20,7	31	33,6	17	26,5	17	19	22	27,5
	5	15	20,1	24	32,1	22	27	18	19,9	20	27
	6	1095	1123,8	1155	1241,3	1099	1106,3	1097	1115,7	1092	1125,3
	7	2142	2203,4	2321	2437	2140	2234	2166	2217,9	2156	2235,1
	8	3121	3224,7	3354	3645,6	3135	3248,4	3157	3188,2	3132	3202,3
	9	9225	9418,5	11879	15854,7	9325	9371,8	9600	9710,6	9204	9370,6
	10	1545	1690,1	1635	1832	1680	2003,7	1480	1606,7	1430	1830
	11	350	364,5	340	352	355	1348	320	341,5	370	798,5
TSP	0	48783,64	49156,77	48599,79	48915,61	49668,09	50946,67	49107,00	49833,44	49259,48	50297,25
	1	110564,87	111419,31	110713,99	111896,49	112419,39	113973,87	112963,60	114513,31	112212,38	114839,34
	2	6926,94	6959,94	6952,33	6970,01	7011,57	7083,83	6979,99	7016,49	7011,57	7048,73
	3	42715,80	43049,30	42835,63	43068,60	43408,08	43744,32	43223,11	43403,01	43385,76	43738,36
	4	9067,84	9110,53	9055,54	9087,11	9186,73	9248,54	9091,63	9156,40	9171,94	9207,92
	5	59412,81	59735,21	59295,64	59634,71	60087,77	60807,46	59571,67	60087,70	59996,40	60635,19
	6	53557,30	54039,64	54840,37	55783,46	53722,65	54905,25	54695,22	55473,46	54434,51	54979,89
	7	68471,97	68999,59	69383,13	69711,01	68790,31	69962,01	69629,79	70178,41	68881,81	69314,48
	8	21119457,88	21316901,87	21153503,04	21262559,87	21255472,60	21401367,92	21145364,26	21322066,84	21268106,35	21448099,47
	9	674371,48	675931,05	675165,29	676503,02	676435,22	678735,34	674723,33	676058,97	675560,00	678768,49
VRP	0	4183,81	4846,15	5216,65	5274,65	5178,67	5236,05	5147,14	5180,91	5126,35	5186,24
	1	20651,10	20652,89	20667,04	21476,93	20663,28	21176,92	20661,69	21170,60	20656,64	21364,02
	2	12289,05	12925,35	13342,67	14528,52	13358,84	14107,57	12402,50	13267,71	13331,80	13584,72
	3	5309,32	5333,93	5360,17	5507,90	5383,55	5596,26	5373,71	5408,55	5316,05	5388,25
	4	13308,27	14076,91	15325,53	15658,31	14311,70	15147,16	13320,87	14211,03	14280,54	14411,75
	5	143908,77	146723,57	261278,71	267206,18	142509,88	147480,42	143871,49	145783,48	143904,85	145716,25
	6	58849,50	59651,23	107161,68	113411,06	58342,60	59458,97	60844,93	61663,48	57397,60	59271,45
	7	159753,92	161198,40	197189,41	204495,39	159959,74	161332,56	159476,14	160247,01	160781,49	161358,38
	8	150175,78	153348,54	237084,51	241517,40	148801,51	153461,12	145646,27	148316,12	150398,65	152841,70
	9	145941,03	147019,25	183843,11	189242,81	145716,09	146924,70	144650,00	145780,46	144781,65	146554,35

Table 6.4: The experimental results by SR hyper-heuristics with 1 hour of execution time

Table 6.5 shows the scores of the different hyper-heuristics after 10 minutes of execution time based on the CHeSC scoring system. The overall scores show that ADHS-AILLA is clearly the best approach for the given problem instances. The rest of the hyper-heuristics are ranked as ADHS-GD, ADHS-LATE, AHDS-SA, AHDS-IE, SR-AILLA, SR-LATE, SR-SA, SR-GD from better to worse overall. ADHS-AILLA is the best approach for the max SAT, permutation flowshop, personnel scheduling and travelling salesman domains. It comes second for the bin packing problem and third for the vehicle routing problem. AILLA performs best among the hyper-heuristics with SR. This indicates that the proposed acceptance strategy is capable of effectively working on different problem domains. ADHS compared to SR for all the acceptance mechanisms provides a high performance improvement. In particular, the score of GD goes from 35 to 418 by using ADHS. Table 6.6 presents the scores after running the hyper-heuristics for 1 hour. ADHS-AILLA is still the best performing method. There are slight differences in the hyper-heuristics' ranking based on their overall performance. SR-AILLA has a higher score than ADHS-IE and ADHS-SA is slightly better than ADHS-LATE.

A Wilcoxon test with a confidence interval of 95% determined the significance of the performance difference between the best hyper-heuristic and the rest after 1 hour of execution for each problem instance. The hyper-heuristics delivering similar performance for a specific instance compared to the best approach for the corresponding problem are coloured in Table 6.3 and 6.4. In the max SAT case, ADHS-AILLA outperforms the tested hyper-heuristics. When dealing with the bin packing problem, ADHS-IE is the best approach, but its performance is not significantly better than ADHS-AILLA. It also performs similarly to ADHS-LATE for all the instances except two. ADHS-AILLA provides the highest average fitness for almost all the permutation flowshop problem instances. For five or six instances, there is no significant performance difference between ADHS-AILLA and the other hyper-heuristics using ADHS. Moreover, the ADHS hyper-heuristics are significantly better than the hyper-heuristics with SR for this problem domain. In consideration of the personnel scheduling problem, the hyper-heuristics with ADHS perform statistically similar. In addition, ADHS-AILLA could not achieve significantly better solutions compared to SR-AILLA and SR-LATE for most of the instances. Regarding the travelling salesman problem, ADHS-AILLA performs similarly to the other hyper-heuristics with ADHS. The ADHS hyper-heuristics perform significantly better than the SR hyper-heuristics. In view of the vehicle routing problem, ADHS-GD provides significantly better results for most of the instances compared to the other hyper-heuristics. However, for five instances, there is no significant performance difference with the AILLA hyper-heuristics.

Hyper-heuristic	Max SAT	Bin Packing	Perm. Flowshop	Pers. Scheduling	TSP	VRP	OverAll
ADHS-AILLA	116	77	95	89	85	56	518
ADHS-GD	29	72	93	75	60	89	418
ADHS-IE	8	81	76	44	65	28	302
ADHS-LATE	29,5	74	53	75	56	69	356,5
ADHS-SA	59	68	73	54,5	64	36	354,5
SR-AILLA	73	33	38	41	25	59	269
SR-GD	0	2	7	6	20	0	35
SR-IE	12,5	40	4	30	2	1	103,5
SR-LATE	58	17	24	26	13	20	158
SR-SA	83	4	5	27,5	0	18	137,5

Table 6.5: The scores of the tested hyper-heuristics after 10 minutes of execution based on the CHeSC scoring system (the highest possible score is 680). The results of SR hyper-heuristics except SR-AILLA were taken from [236]

Hyper-heuristic	Max SAT	Bin Packing	Perm. Flowshop	Pers. Scheduling	TSP	VRP	OverAll
ADHS-AILLA	113	70	118	87	95	52	535
ADHS-GD	31	67	62,5	72	51,5	94	378
ADHS-IE	10	76	71	41	54,5	10	262,5
ADHS-LATE	31	62	62	72	55	39	321
ADHS-SA	63,5	53	81	49	74	20	340,5
SR-AILLA	76	53	34,5	40	26	59	288,5
SR-GD	0	0	5	5	20	1	31
SR-IE	10	31	5,5	20,5	3	21	91
SR-LATE	44,5	24	21,5	62	8	51	211
SR-SA	89	32	7	19,5	3	43	193,5

Table 6.6: The scores of the tested hyper-heuristics after 1 hour of execution based on the CHeSC scoring system (the highest possible score is 680)

Table 6.7 shows the percentual difference between the best known solution and the best solution found by ADHS-AILLA after 1 hour. Only the best solutions for the first four bin packing instances are available. ADHS-AILLA finds solutions with the same number of bins for these instances. The proposed approach finds a solution with only one unsatisfiable clause for max SAT instance 3. The hyper-heuristic found the same quality solutions as the best known solutions for the instances 4 and 11. Moreover, it explored new best solutions for instances 6, 7 and 8. The best solutions for the other instances are not known, therefore it is not possible to make a comparison for them. The hyper-heuristic found the best known solution for the permutation flowshop problem instance 6. The percentual performance difference is usually lower than 1% except for instance 10 (1.14%). The best results on the personnel scheduling problem instances 6 and 9 are new best solutions. For the instances 2 and 10, the difference increases above 10%. The percentual difference on the instances 3, 4 and 5 is very high since the fitness values of the best known solutions are less than 10 (e.g. best known fitness= 3 $\rightarrow$ found fitness= 16, percentual difference= 433.33%). The difference between the best solutions found by ADHS-AILLA and the optimal solutions is smaller than 1% for the first six travelling salesman instances. This difference increases up to 3.51% for larger instances.

The best solutions found by ADHS-AILLA for the first five VRP instances utilise the same number of vehicles and the travelled distance is very near to the best known solutions. For the last five instances, the proposed approach finds the same number of vehicles only for instance 7 and the difference is relatively high. For the rest of the problem instances, the number of vehicles is different. Therefore it is hard to make a direct comparison between them.

6.6.1 Improvement provided based on execution time

Table 6.8 shows the performance difference between running ADHS-AILLA for 10 minutes and for 1 hour. The performance difference is limited considering the permutation flowshop scheduling and travelling salesman problems. For both problems, finding a good quality solution appears to be very easy after short amount of time and it becomes hard to improve. The relative improvement provided after 1 hour is quite large for the max SAT problem compared to 10 minutes, however the objective values are small in values. This also shows that for the corresponding SAT instances, the proposed approach finds high quality solutions at the very beginning of the search. A similar case is valid for some personnel scheduling problem instances. The gap is relatively large due to the particular fitness function used to evaluate the bin packing problem solutions. This indicates that there is large room for improvement after 10

Inst.	MSAT	BP	FS	PS	TSP	VRP
0	N/A	0	0.69	0.43	0.01	9.72
1	N/A	0	0.47	2.30	0.07	0.41
2	N/A	0	0.21	11.11	0.34	7.94
3	100	1.67	0.29	450	0.04	1.09
4	0	N/A	0.46	480	0.46	2.80
5	N/A	N/A	0.05	433.33	0.08	N/A
6	-16.67	N/A	0	-0,18	3.51	N/A
7	-16.67	N/A	0.53	0.33	2.18	11.08
8	-37.50	N/A	0.58	0.38	3.03	N/A
9	N/A	N/A	0.37	-0.43	2.98	N/A
10	N/A	N/A	1.14	13.60	-	-
11	0	N/A	0.35	3.70	-	-

Table 6.7: the % difference between the best known solutions and the best solutions found by ADHS-AILLA after 1 hour

minutes running time. The improvement provided after 10 minutes is small for most of the vehicle routing problem instances. For instance 6, the average fitness after 1 hour of execution appears to be worse than the 10 minutes case. The underlying reason is that the hyper-heuristic prematurely converged during certain runs.

Inst.	MSAT	BP	FS	PS	TSP	VRP
0	35,56	7,41	0,23	0,72	0,00	7,05
1	26,04	8,40	0,23	7,18	0,44	0,51
2	4,37	7,73	0,14	8,31	0,11	0,12
3	5,73	37,14	0,52	23,98	0,13	0,61
4	35,60	37,04	0,40	27,34	0,14	0,75
5	7,43	72,73	0,04	28,19	0,59	2,02
6	15,78	3,77	0,03	4,58	0,44	-9,12
7	22,53	7,41	0,25	2,68	0,70	0,63
8	1,87	7,04	0,30	2,38	0,78	1,03
9	52,90	0,75	0,35	2,88	0,42	0,63
10	0,00	40,91	0,41	6,65	-	-
11	30,06	1,25	0,46	9,69	-	-

Table 6.8: The % performance difference between 10 minutes and 1 hour cases based on their average results by ADHS-AILLA

6.6.2 Adaptive dynamic heuristic set strategy

Figure 6.2 shows for each problem domain how many times each heuristic was applied during a 10 minutes run. There is a trend regarding the number of calls for the bin packing problem. A hill climber (LLH_6) is called more frequently than the others. It is followed by the ruin-recreate heuristics (LLH_2, LLH_1) and a mutational heuristic (LLH_0). The number of calls for different heuristics change over time for the max SAT problem. The frequently selected heuristics are listed from most to least frequent as follows: LLH_4, LLH_3 and LLH_5 that are hill climbers. A crossover operator is selected most frequently after these hill climbers. For the permutation flowshop scheduling problem, heuristics can be considered divided into groups. After 400 seconds, one crossover operator (LLH_{13}) was applied most often. Other crossover operators are also called very frequently after LLH_{13}. Only a ruin-recreate heuristic (LLH_5) is called more than 1000 iterations for the personnel scheduling problem. In the case of the travelling salesman problem, two crossover operators (LLH_9, LLH_{12}) are chosen more often than others. Also for the vehicle routing problem, two heuristics take the lead, i.e. two mutational heuristics (LLH_0, LLH_1).

Heuristic set size

Figure 6.3 depicts the changes of the heuristic set size for each problem domain. The fluctuations of the heuristic set size are very frequent for the bin packing, max SAT and vehicle routing problems. Changes mostly take place due to the presence of fast heuristics in the heuristic subset. The set size changes are slower for the other problem domains. A set of four heuristics is most of the time sufficient when dealing with the bin packing problem. This value is generally around 5 for the max SAT problem. The heuristic set size decreases to 5 during the search by excluding 10 heuristics at the same time in the case of the permutation flowshop scheduling problem. The size of the heuristic set associated with the personnel scheduling problem oscillates around 5. A set with 5 or 6 heuristics is often enough for the travelling salesman problem. For the vehicle routing problem, the size of the heuristic set is generally 4 and for some cases, the set size even reduced to 2.

These results indicate that there is no need to use the whole heuristic set during the entire search process. The subset selection or heuristic exclusion strategy effectively determines good heuristic subsets and this reduces the complexity of the heuristic selection operation. However, it should be noted that there is not one heuristic subset which always works well. Therefore, it is important to appoint different heuristic subsets for different parts of the search.

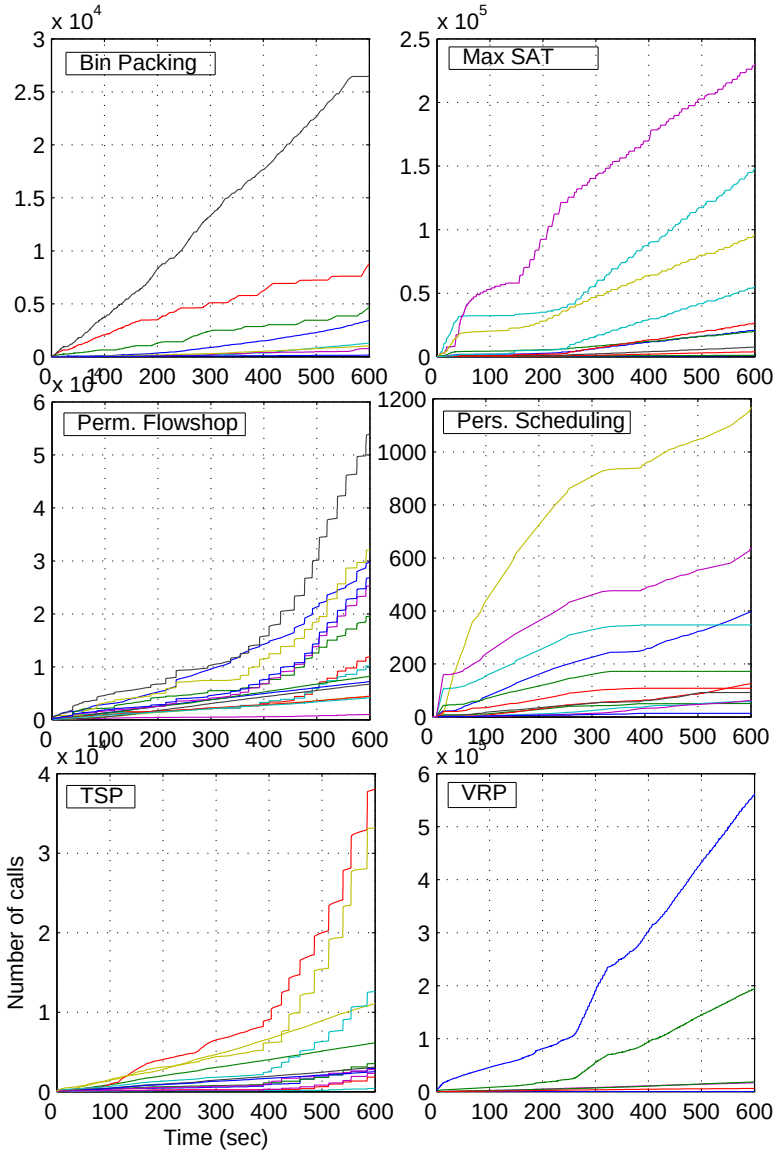

Figure 6.2: The number of calls for each heuristic on each problem domain by ADHS-AILLA with 10 minutes of execution time (a run on one sample instance represents each domain)

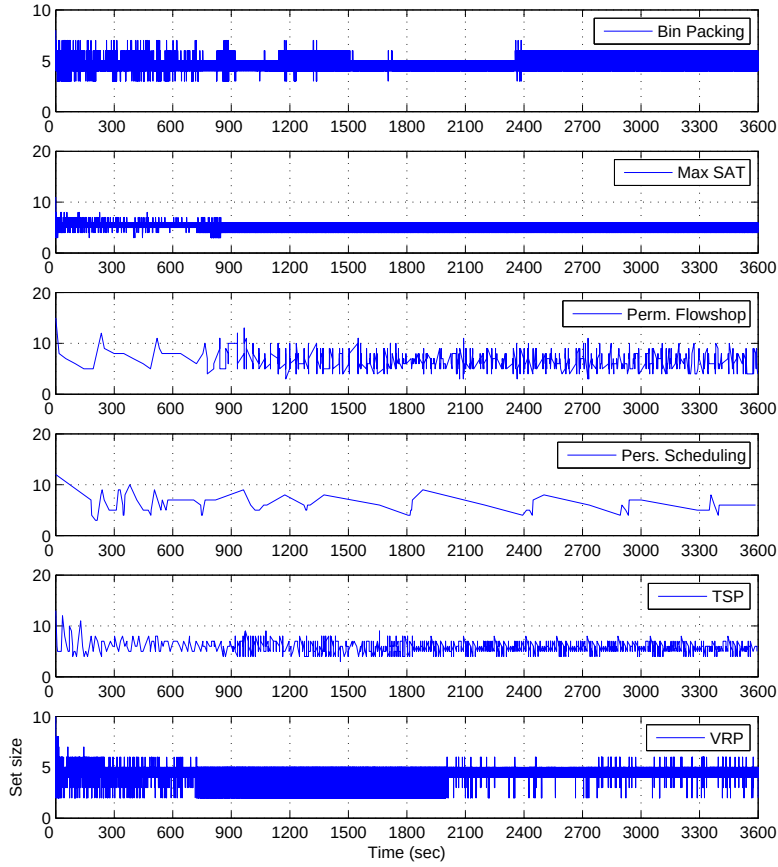

Figure 6.3: The heuristic set size changes on each problem domain by ADHS-AILLA with 1 hour of execution time (a run on one sample instance representing each domain was used)

QI changes

Figure 6.4 presents the QI values for each bin packing heuristic during a 10 minutes run. Among them, one ruin-recreate heuristic (LLH_2), one mutational heuristic (LLH_3) and only crossover operator (LLH_7) have high QI values during almost the whole run. One ruin-recreate heuristic (LLH_1) and two hill-climbers (LLH_4, LLH_6) have relatively worse QI values. Two mutational heuristics (LLH_0, LLH_5) generally obtain the lowest QI values. It is apparent that all the QI values tend to change, so high QI values may turn into low ones for some heuristics during the course of the search, and vice versa. This illustrates that why an effective hyper-heuristic is required to have an online learning component for the heuristic selection process.

6.6.3 Relay hybridisation

Figure 6.5 presents heuristic pairs that explored new best solutions during 1 hour. In the light of the bin packing problem, such heuristic pairs were rarely identified during the first 10 minutes. After that time, almost no pair contributed to the optimisation. During the first half of the search, hybridisation frequently found new best solutions while solving the max SAT problem instances. From the given heuristic set, a mutational heuristic (LLH_6) was used as the first heuristic to apply. Two mutational heuristics (LLH_0, LLH_4), and one hill climber (LLH_8) were utilised as the second heuristics. All the hill climbers (LLH_7,LLH_8,LLH_9,LLH_{10}) given to solve the permutation flowshop scheduling problem discovered new best solutions as the second heuristics. Except for three mutational heuristics (LLH_2,LLH_3,LLH_4), all other heuristics were used as the first ones of the pairs. For the personnel scheduling problem, two hill climbers (LLH_3,LLH_4) were used as the second heuristics. The remaining heuristics, except a crossover operator (LLH_9), were used to change the solution before the second heuristics were applied. These heuristics found new best solutions only during the first half of the execution time. The only ruin-recreate heuristic (LLH_5) was mostly used as the first heuristic with regard to the travelling salesman problem. All the available hill climbers (LLH_6,LLH_7,LLH_8) were used as the second heuristic. Among them, especially LLH_8 successfully found many new best solutions via relay hybridisation. A crossover operator (LLH_6) was utilised as the first heuristic for most of the cases on the subject of the vehicle routing problem. A mutational heuristic (LLH_0) and two hill climbers (LLH_4, LLH_8) were generally applied as the second heuristics.

Figure 6.6 demonstrates the ratio between the number of iterations spent by relay hybridisation and by the single heuristic selection. This ratio changes over time for all the problem domains. For the permutation flowshop scheduling

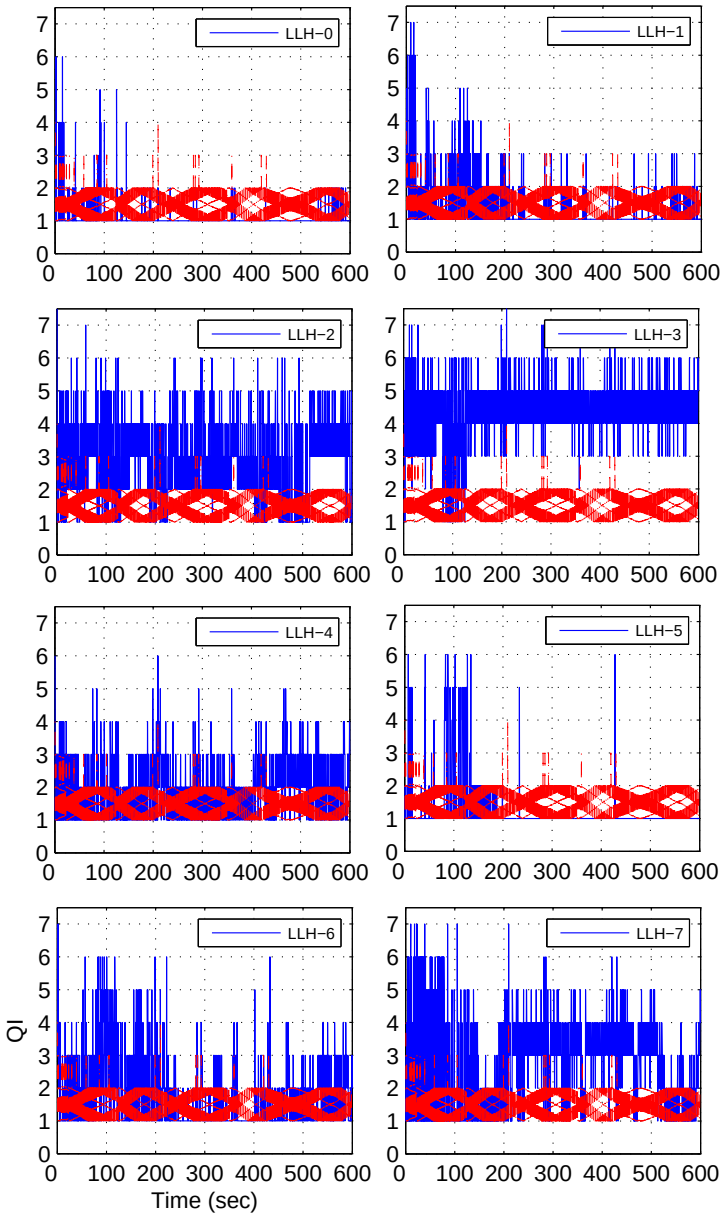

Figure 6.4: QI changes of the bin packing heuristics by ADHS-AILLA with 10 minutes of execution time (the red dotted lines show avg)

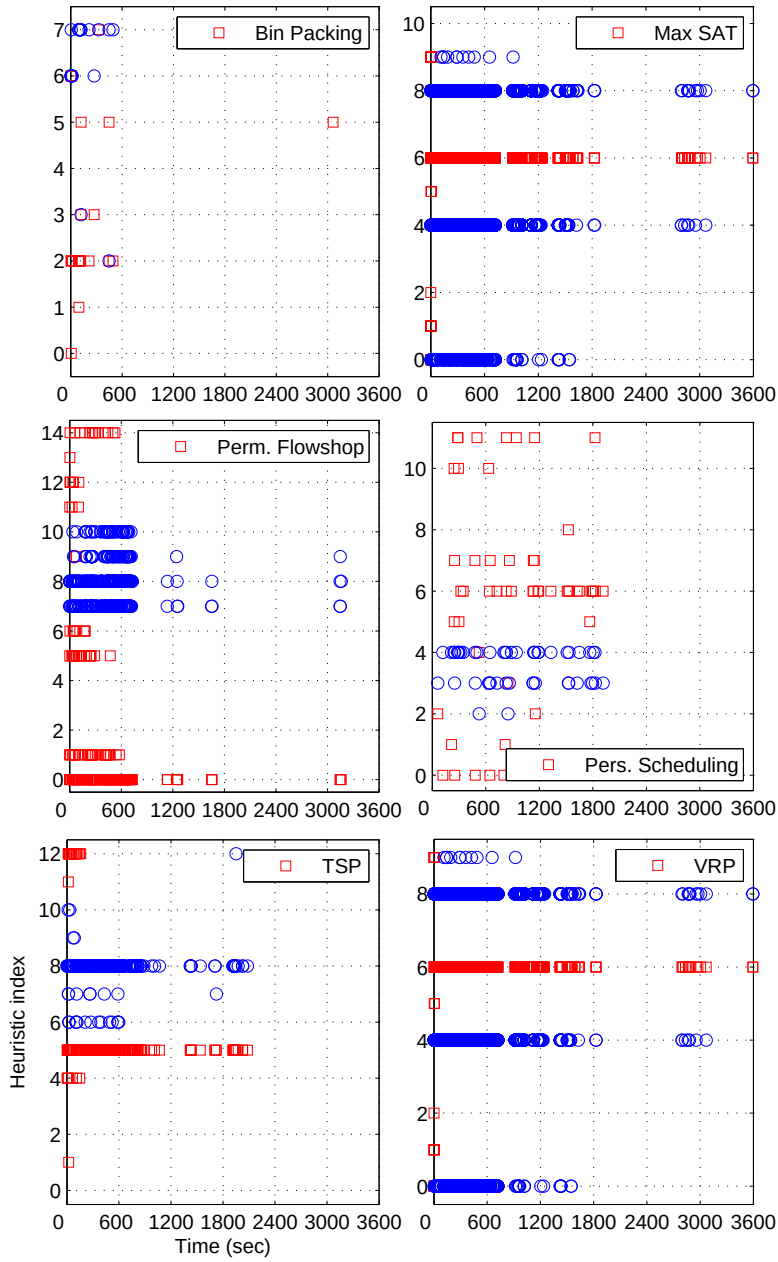

Figure 6.5: Heuristic pairs yielding new best solutions by relay hybridisation using ADHS-AILLA for 1 hour of execution time (squares show the first applied heuristics, circles indicate the heuristics applied afterwards)

problem, the ratio increases during the first 10 minutes and it is higher than 1. This shows that for this problem domain, relay hybridsation is preferred over single heuristic selection. Afterwards, it decreases and around 1000 seconds later, its value goes under 1 since it is more profitable to apply single heuristics. With the travelling salesman problem, a similar trend occurs. This ratio decreases to a very low value due to the fast heuristics with very high improvement capabilities for the bin packing problem. The ratio slightly increases for the max SAT problem, but it decreases towards the end of the search. The increment is relatively larger while solving the vehicle routing problem, but it also starts to decrease after almost 10 minutes. The ratio constantly increases for the personnel scheduling problem to about 0.2.

For all the tested problem domains, various heuristic pairs that can explore new best solutions were detected. Hence, this approach can be considered a method to increase the quality of a heuristic for the mentoring task. However, adding new heuristics makes the heuristic selection operation more complex. The decision rule provided in the relay hybridisation solves this problem by deciding whether to apply only one heuristic or heuristic pairs. For instance, ADHS-AILLA prefers heuristic pairs over single heuristics for the permutation flowshop scheduling problem. Conversely, ADHS-AILLA applies single heuristics more often for the bin packing problem, even though it found very effective heuristic hybridisations for the bin packing problem. This means that heuristics should be selected depending on both their improvement and speed traits.

From the competition point of view, success of the competing methods combining mutational and ruin-recreate heuristics with hill climbers can be deduced from our relay hybridisation results. As expected, for most of the cases, the detected heuristic pairs that find new best solutions are in the form of perturbing a solution first, then improving the new solution using a hill climber. However, in certain cases e.g. in case of the bin packing problem, hybridisation slows down the hyper-heuristic and makes it miss fast and effective single heuristics. As discussed in Chapter 4, it is important to determine effective heuristics according to their improvement characteristics. However, it is even more important to consider these abilities along with their speed for better judgement.

6.6.4 Adaptive iteration limited list-based threshold accepting

Figure 6.7 shows how the iteration limit (k) changes. The value of k increases over time with some fluctuations in certain time limits while solving the bin packing problem instances. The hyper-heuristic waits about 500 iterations before diversifying the search during later iterations. For the max SAT problem, k constantly changes during the first 100 seconds. After that it is almost stable

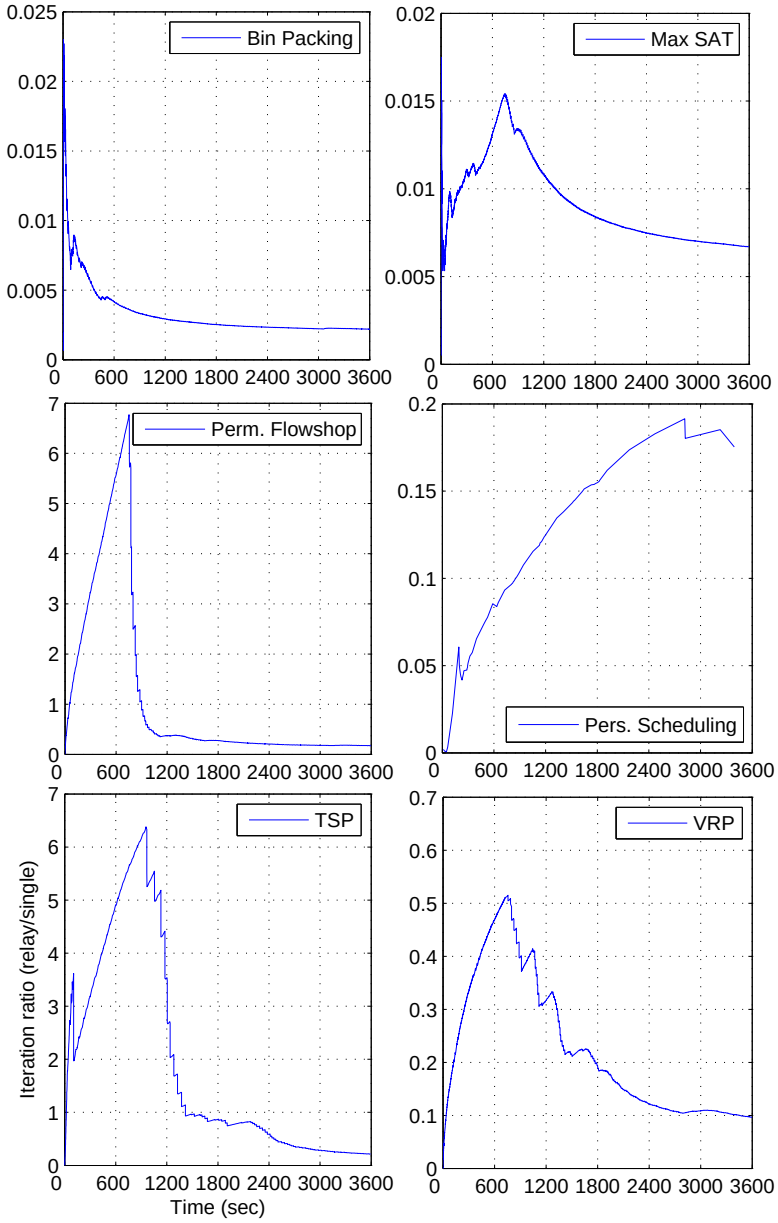

Figure 6.6: Number of iterations ratio between relay hybridisation and single heuristic selection (relay/single)

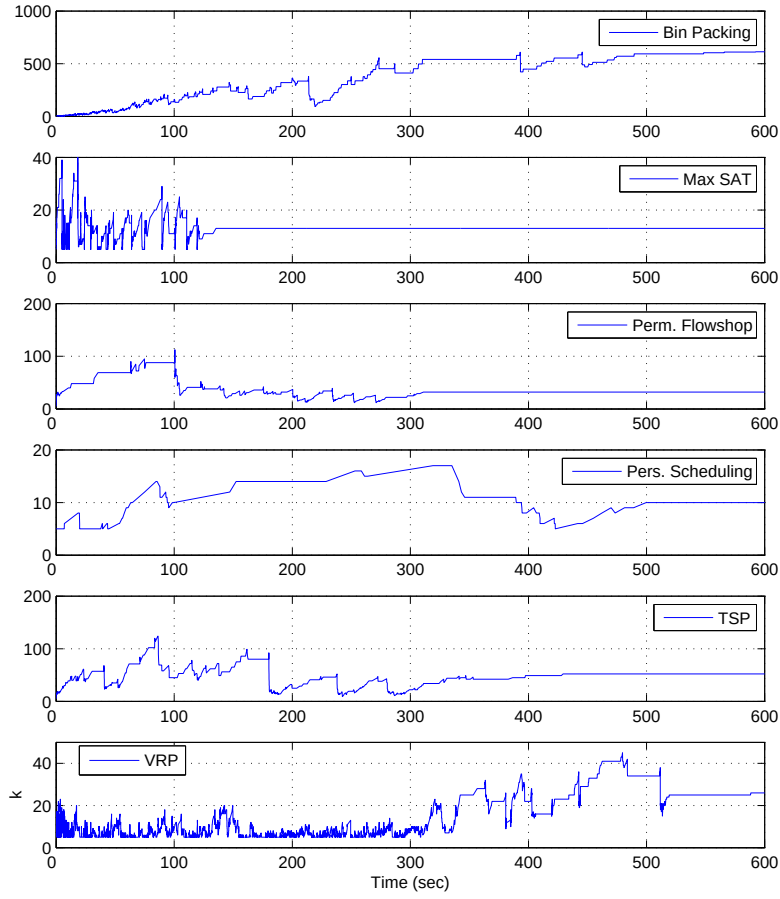

Figure 6.7: Iteration limit (k) changes on each problem domain (a run on one sample instance representing each domain was used)

due to the fast improvement realised by the hyper-heuristic. In case of the permutation flowshop problem, changes on k are limited to the first half of the running time. Due to the limited number of improvements and slow heuristics, the value of k changes over the whole period for the personnel scheduling problem. Considering the travelling salesman problem, k reaches 100 after several fluctuations and stabilizes after 400 seconds. Even though the variation of k is limited for the vehicle routing problem, changes are nevertheless frequent. This shows that improving solutions for 300 seconds is relatively easy and that this is mostly due to the re-initialisation procedure. After that time, the number of improvements decreases.

6.6.5 Re-initialisation

When good quality solutions are detected early and while there is still time to make improvements, ADHS-AILLA randomly re-inisialises the solution. Figure 6.8 presents the changes to the best fitness with re-initialisations by ADHS-AILLA. The frequency of re-initialisation is very high for the max SAT problem. After almost 100 seconds, re-initialisation stops and the hyper-heuristic starts working to improve the overall best solution found after all performed re-initialisations. The vehicle routing problem was subject to several re-initialisations but fewer than the max SAT domain. Re-initialisation is much less frequent for the remaining problems.

These results disclose that even such a naive approach should be adaptive for different heuristics sets being associated with different problems. A few major points can be derived from these results. The first point relates to the difficulty of finding a good quality solution for a particular problem instance. The other point is related to the continuity of improvement. For instance, the re-initialisation activities for the max SAT problem are very frequent while the frequency of generating new initial solutions is limited in case of the bin packing problem. ADHS-AILLA immediately finds good quality solutions due to the capabilities of the given heuristics and the structure of the solution space. Then, improving these quickly found good solutions is very hard. The fitness landscape(s) associated with the bin packing problem domain allows finding solutions fast. However, the improvement process is persistent. Therefore, there is no need to call the re-initialisation method as frequently as in the max SAT case. This re-initialisation mechanism can deal with such differences efficiently.

6.6.6 Cross-domain heuristic search challenge (CHeSC 2011)

The CHeSC 2011 organisers provided four problem domains along with heuristic sets as a testbed for selection hyper-heuristics. The problem domains given prior to the competition were max SAT, bin packing, permutation flowshop scheduling and personnel scheduling problems. The competitors performed a set of experiments on these domains to examine the performance of their hyper-heuristics. For the actual competition, the testbed was extended with two hidden problem domains, namely the travelling salesman and vehicle routing problems. Three available instances were randomly selected from the problem domains announced before the competition. In addition, two hidden instances were introduced. As a consequence, 5 instances from 6 problem domains were used as the test-bed. In this setting, the highest possible score for an algorithm is 300 based on the formula 1 scoring system.

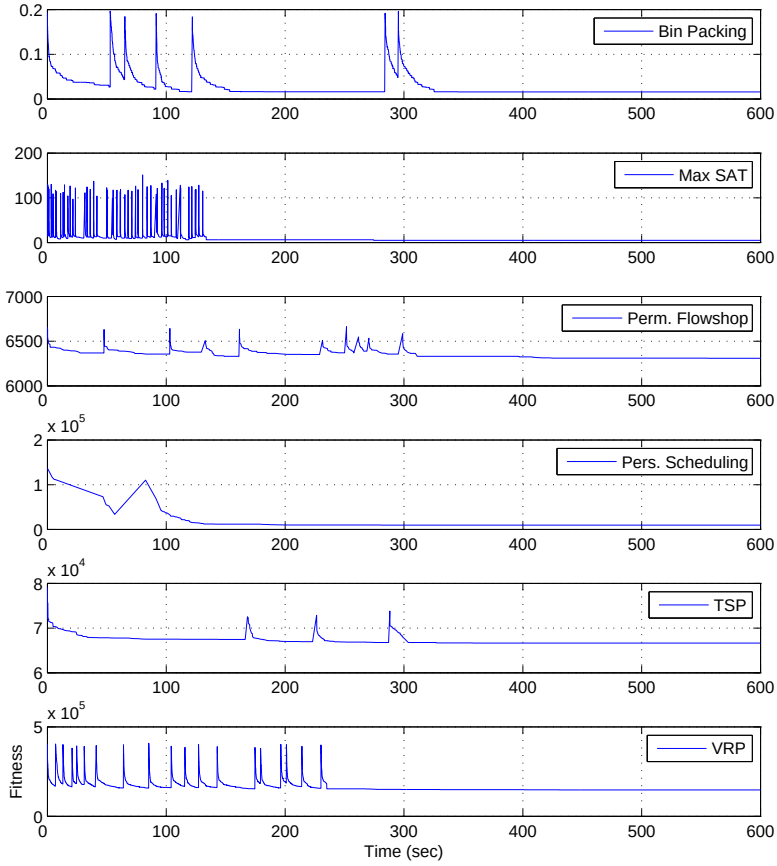

Figure 6.8: Changes on the best fitness found by ADHS-AILLA on each problem domain

Table 6.9 presents the ranking and scores of the 20 competing algorithms. Each algorithm runs 31 times for 10 minutes. These results reveal a clear performance difference between our approach, GIHH (ADHS-AILLA), and the other competing algorithms. This is a vital indicator that GIHH indeed meets a high generality level. More detailed results are available at the competition website[2]. Problem-wise, the proposed approach outperforms the other algorithms for max SAT, 1D bin packing and travelling salesman. It comes second for permutation flowshop scheduling, 5th for vehicle routing and 10th for the personnel scheduling problem. Among the unseen problem domains, the travelling salesman problem is a good example to show the adaptive capabilities

[2] `http://www.asap.cs.nott.ac.uk/external/chesc2011/`

of this hyper-heuristic. On the other hand, the results on vehicle routing problems indicate that there is still room for improvement on its generality level because a few other competitors obtained better scores. Its relatively worse performance for the personnel scheduling is caused by the limited time available to the learning process given the speed of the heuristics.

Algorithm	Score	MSAT	BP	FS	PS	TSP	VRP
GIHH	181	34.75	45.0	37.0	9.0	40.25	15.0
VNS-TW	134	34.25	3.0	34.0	39.5	17.25	6.0
ML	131.5	14.5	12.0	39.0	31.0	13.0	22.0
PHUNTER	93.25	10.5	3.0	9.0	11.5	26.25	33.0
EPH	89.75	0.0	10.0	21.0	10.5	36.25	12.0
HAHA	75.75	32.75	0.0	3.5	25.5	0.0	14.0
NAHH	75	14.0	19.0	22.0	2.0	12.0	6.0
ISEA	71	6.0	30.0	3.5	14.5	12.0	5.0
KSATS-HH	66.5	24.0	11.0	0.0	9.5	0.0	22.0
HAEA	53.5	0.5	3.0	10.0	2.0	11.0	27.0
ACO-HH	39	0.0	20.0	9.0	0.0	8.0	2.0
GenHive	36.5	0.0	14.0	7.0	6.5	3.0	6.0
DynILS	27	0.0	13.0	0.0	0.0	13.0	1.0
SA-ILS	24.25	0.75	0.0	0.0	19.5	0.0	4.0
XCJ	22.5	5.5	12.0	0.0	0.0	0.0	5.0
AVEG-Nep	21	12.0	0.0	0.0	0.0	0.0	9.0
GISS	16.75	0.75	0.0	0.0	10.0	0.0	6.0
SelfSearch	7	0.0	0.0	0.0	4.0	3.0	0.0
MCHH-S	4.75	4.75	0.0	0.0	0.0	0.0	0.0
Ant-Q	0	0.0	0.0	0.0	0.0	0.0	0.0

Table 6.9: CHeSC 2011 competition scores (Algorithms are sorted from the best to the worst based on their overall scores)

6.7 Conclusion

A selection hyper-heuristic, i.e. generic intelligent hyper-heuristic (GIHH), accommodating various evolvable mechanisms has been developed in the present chapter. The goal was addressing possible generality related factors using new hyper-heuristic sub-mechanisms. Accordingly, combining these sub-mechanisms by using some adaptive coordination and decision strategies was the next phase to deliver a complete hyper-heuristic design. The empirical results clearly demonstrated the effectiveness of GIHH in terms of generality. GIHH's superior performance to the hyper-heuristics using some sub-mechanisms from

the literature constitute the first part of the results. In addition, the success of GIHH as the winner of the first cross-domain heuristic search challenge (CHeSC) 2011 illustrates its generic characteristics while solving different problems using distinct heuristic sets. Thus, GIHH can be considered currently the best hyper-heuristic in the literature. It should be used as the benchmark for the new hyper-heuristics claiming generality.

Most of the competing algorithms in the CHeSC 2011 used the information about heuristic types to devise memetic algorithm or iterated local search kind of approaches. GIHH ignores this information and defines a set of heuristic types to generalise the hyper-heuristic by definition. These types were specified with respect to heuristics' improvement skills and are valid for all possible heuristics. Hence, our approach helps to generalise the applicability of the hyper-heuristic across various heuristic sets involving different heuristic types.

Adaptiveness and coordination are the primary concepts that need to be considered for the purpose of generality in the hyper-heuristic research. These concepts materialised within the hyper-heuristic sub-mechanisms. In that sense, certain components to decide about the sub-mechanisms' lifetime, frequency of usage, status regarding whether they are active or passive are required. Furthermore, the corresponding parameters must be adapted during a run. As a consequence, the hyper-heuristic developed should be able to properly change its behaviour for different problem sets and heuristic sets via various adaptation and decision mechanisms.

The extensive empirical results presented in the chapter indicated the effectiveness of the proposed hyper-heuristic for determining the requirements for solving a problem instance. Therefore, Chapter 7 uses the hyper-heuristic as an analysis tool. The analysis considers the contribution of low-level heuristics used while solving three structured problems with common characteristics. Additionally, it provides new design perspectives for solving the tested problems.

Chapter 7

A Selection Hyper-heuristic as Analysis Tool

Various algorithms have been applied to different search and optimisation problems. Thus, it is likely to find several algorithms from the literature for solving a particular problem. The other option is designing a new algorithm if a highly specialised solution is required. Some of these algorithms can be applicable to a particular class of problems if the same solution representation is used. For instance, graph colouring heuristics can be used for a class of timetabling problems [249]. In such a case, it can be useful to determine whether there exist performance variations when one algorithm is applied to a number of problems sharing similar characteristics. Unlike the traditional hyper-heuristic studies, hyper-heuristics can be used to specify these performance variations rather than aiming to solve the problems.

The present chapter investigates the performance and contribution of low-level heuristics while simultaneously solving different problems with routing and rostering characteristics. Three interrelated problems are investigated:

- The home care scheduling problem (HCSP)
- The routing and rostering of security guards (SPRR)
- The maintenance personnel scheduling problem (MPSP)

The details on these problems are presented in Chapter 3, Section 3.2.

We developed a set of low-level heuristics for solving these problems and a selection hyper-heuristic, the CHeSC 2011 winner hyper-heuristic GIHH, presented in Chapter 6, was used to analyse these heuristics. The motivation behind hyper-heuristics in this context is to use multiple generalised heuristics and show which heuristics are useful for solving which combined routing and rostering instances. The novelty of this study is in applying a selection hyper-heuristic as an analysis tool for a set of generalised low-level heuristics instead of focussing on solving the target problems. Note that, in the present chapter, the focus is not on solving the problems, but rather on analysing a set of general low-level heuristics for a structured problem by means of a selection hyper-heuristic. This is a more generic contribution that goes beyond the solutions to individual problems. The empirical outcomes can be used to design effective high-level approaches as well as dedicated solvers for the problems on the subject of this study.

The experiments across the aforementioned problem domains revealed that different low-level heuristics perform better on different problems. Besides that, their performance variations during a search process was shown. The body of this paper details the required features for embedding in an algorithm to solve problems particularly with a vehicle routing component.

The contribution of this chapter resulted in the following paper:

> M. Mısır, P. Smet, G. Vanden Berghe, An Analysis of Generalised Heuristics for Vehicle Routing and Personnel Rostering Problems, *under review*.

7.1 Low-level heuristics

The following 12 low-level heuristics, based on move and swap operations, were used for the analysis hyper-heuristic.

- $\boldsymbol{LLH_0}$: Swap two randomly selected visits between two randomly selected routes.
- $\boldsymbol{LLH_1}$: Swap two randomly selected visits within a randomly selected route.
- $\boldsymbol{LLH_2}$: Swap two groups of consecutive visits from two randomly selected routes.
- $\boldsymbol{LLH_3}$: Move a randomly selected visit in a randomly selected route to another randomly selected route.

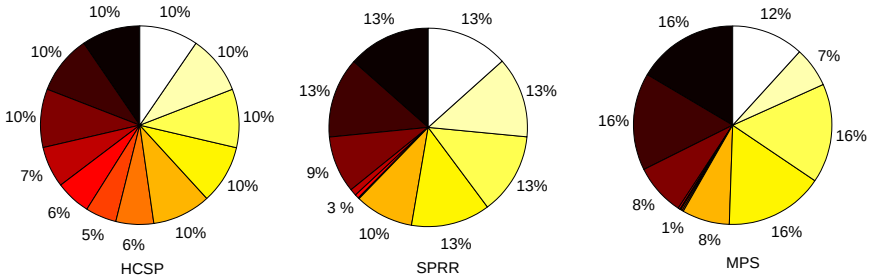

Figure 7.1: Comparison between the average speed of the low-level heuristics on each problem domain (from black to white in counter-clockwise direction: $LLH_0 \rightarrow LLH_{11}$)

- $\boldsymbol{LLH_4}$: Move the most conflicting visit in a randomly selected route to another randomly selected route.
- $\boldsymbol{LLH_5}$: Move the most conflicting visit in a randomly selected route to the least conflicting route.
- $\boldsymbol{LLH_6}$: Move the most conflicting visit in a randomly selected route to a route with the highest idle time.
- $\boldsymbol{LLH_7}$: Move a group of randomly selected consecutive visits to a another randomly selected route.
- $\boldsymbol{LLH_8}$: Move a randomly selected visit to another location in a randomly selected route. This heuristic can be considered an Or-opt-1 [278] move.
- $\boldsymbol{LLH_9}$: Move a group of randomly selected consecutive visits to another location in a randomly selected route. This heuristic can be considered an Or-opt [278, 54] move.
- $\boldsymbol{LLH_{10}}$: Reverse all the visits between two randomly selected visits in a randomly selected route. This heuristic can be considered a 2-opt [278, 54] move.
- $\boldsymbol{LLH_{11}}$: Scramble all the visits between two randomly selected visits in a randomly selected route.

These heuristics are responsible for changing the order of visits within a route or among routes. Therefore, after applying each heuristic, a separate function determines the best and earliest possible time assignments for each visit.

Figure 7.1 presents the speed of these heuristics with respect to each other across the three aforementioned problem domains. The HCSP experiments

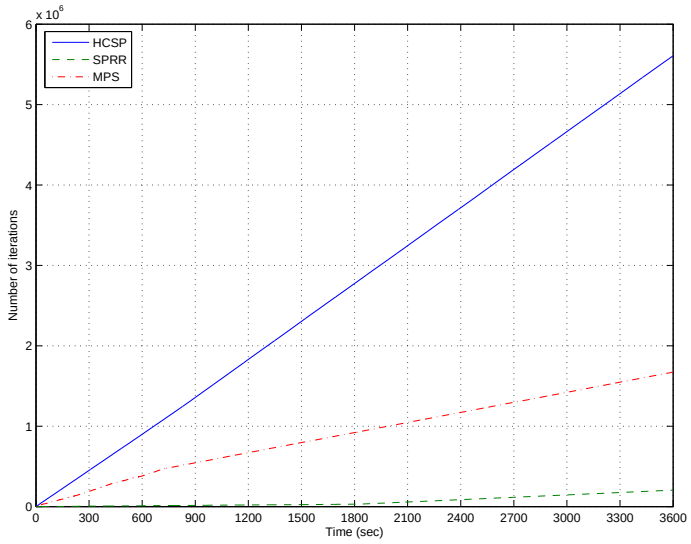

Figure 7.2: Average operator speed for each problem domain

revealed that the speed of the all heuristics is similar, but the speed of certain move operators is almost half when compared to the rest. The speed difference is greater in the case of the SPRR and MPSP and this occurs mainly due to the required time to perform these moves when working with larger data sets. As shown in Figure 7.2, the average number of iterations performed by choosing heuristics in a uniformly random manner also illustrates these differences. The problems, with respect to their speed on executing an optimisation step, can be ordered from fastest to slowest as: HCSP, MPSP, SPRR.

7.2 Computational results

The ADHS-AILLA hyper-heuristic was applied to each instance 10 times with one hour execution time limits. The experiments were carried out on Pentium Core 2 Duo 3 GHz PCs with 3.23 GB memory using Windows XP.

Figure 7.3 provides information on the structure of the search space created by the hyper-heuristic together with the low-level heuristics on each problem. Performing a move can result in major changes in the quality of a solution for the HCSP. This means that, after a high number of improving moves, it is possible to visit a worse solution than a randomly constructed initial solution. Therefore, it is important to develop or use a controlled diversification strategy while solving

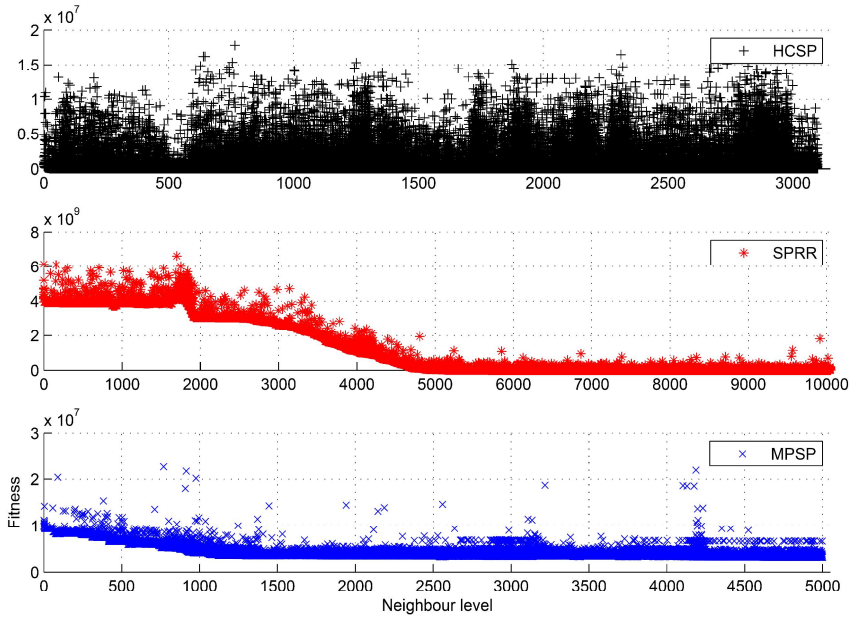

Figure 7.3: Solutions visited in the course of the search process for each problem

the HCSP instances. Considering the MPSP, an atomic move in general cannot make a large quality difference with respect to the visited solutions. During the later iterations, worse solutions can be visited by performing a move, but this does not occur as often as in the HCSP. The fundamental reason explaining this is the size of the corresponding problem instances. As just mentioned, the HCSP is a daily problem and the number of tasks requiring handling is small compared to the other two problems. The MPSP instances are larger than for the HCSP in terms of the planning horizon, number activities and resources. These elements are even larger for the SPRR case, thus applying a move provides less changes on the quality of a solution with respect to the quality range between the best and worst solutions visited. This means that, it is unlikely to find a solution with the quality near to an initially generated relatively worse solution or the best solution that can be found after a large number of iterations. In conclusion, the size of a problem instance for the problems under investigation is an important element on the performance of a search and optimisation algorithm.

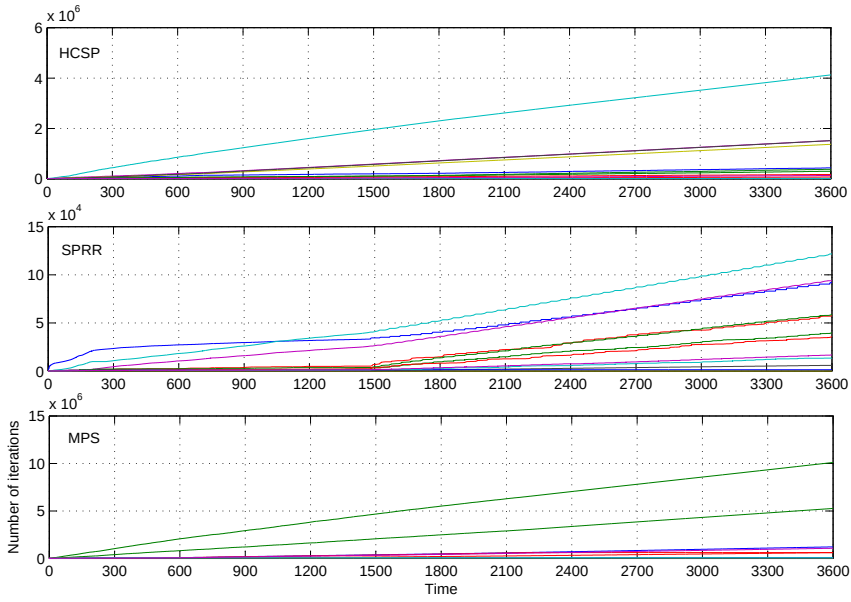

Figure 7.4: Number of iterations spent per time for each heuristic on each problem domain (each graph belongs to one problem domain ordered as HCSP, SPRR, MPSP from top to bottom)

7.2.1 Performance of the low-level heuristics

Each low-level heuristic has certain strengths and weaknesses. It is important to evaluate these features and to perform the heuristic selection process based on them.

Figure 7.4 illustrates the number of iterations spent by each heuristic throughout the search on the given problems. In the case of HCSP, LLH_3 is the most frequently chosen heuristic by ADHS among those available. LLH_5 and LLH_6 are called less than half as much as LLH_3. The hyper-heuristic prefers the other heuristics less than these three. For the SPRR, LLH_3 is again preferred for most of the iterations even though LLH_0 is chosen more often at the beginning of the search process. LLH_0 and LLH_4 are also frequently used. The speed of the heuristics has a major effect on the selection process, as shown in Figure 7.1 and 7.2. The heuristic primarily applied for the MPSP is LLH_1 due to its improvement capabilities. LLH_8 is the other heuristic expected to contribute to the search most.

The number of improving solutions and the number of new best solutions

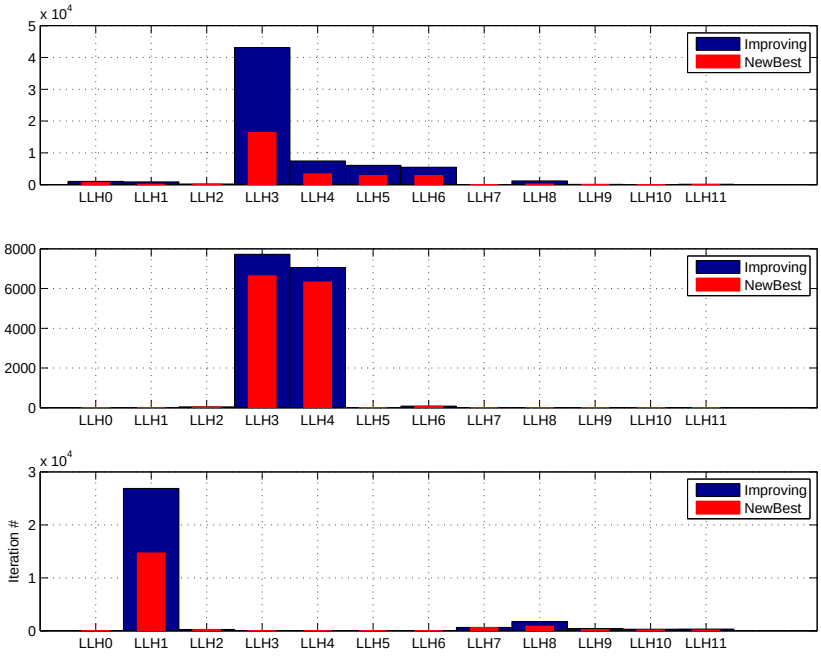

Figure 7.5: Number of new best and improving solutions found by each heurstic for each problem domain (each graph belongs to one problem domain ordered as HCSP, SPRR, MPSP from top to bottom)

visited by each heuristic are depicted in Figure 7.5. They show that moving visits from one route to another is the most profitable approach to explore new best solutions of the HCSP and SPRR. However, this is not the case for the MPSP. In this problem, an inner move swapping two visits in a single route delivers a high number of new best solutions when compared with the rest of the heuristics. This is an important indicator for showing the effect and performance of different heuristics on different problem domains that have common characteristics. Another outcome from this figure is that there are certain heuristics that could not find any new best solution at all. For the HCSP, LLH_{10} exhibits this trait. However, three heuristics, namely LLH_0, LLH_{10} and LLH_{11}, could not succeed in discovering a new best solution for the SPRR. The same situation occurs for LLH_0 while solving the MPSP.

7.2.2 Exploring new heuristics

The performance of a hyper-heuristic depends on a given heuristic set. Hence, its performance is limited to the best performance that can be derived from the heuristic set. Therefore, more sophisticated strategies can be employed for more efficient usage of the heuristic set rather than just choosing one heuristic and applying it. One possibility is finding effective heuristic hybridisations as shown using relay hybridisation presented in Chapter 6. The employed hyper-heuristic in this study consists of a mechanism to detect good heuristic pairs that can find new best solutions on the run.

Figure 7.6 depicts the heuristic pairs that found new best solutions for each problem. The majority of these pairs were effective during the first 10 minutes. These heuristic pairs could not deliver new best solutions during the rest of the search process due to the performance of the single heuristics and the evolvability of the search spaces. LLH_6 performed effectively as the secondly applied heuristic in the heuristic pairs for the HCSP. LLH_4, LLH_6 and LLH_8 found solutions that can be easily improved by the second heuristics. The heuristic pairs detected for this problem are $LLH_4 \rightsquigarrow LLH_3$, $LLH_4 \rightsquigarrow LLH_6$, $LLH_6 \rightsquigarrow LLH_3$, $LLH_6 \rightsquigarrow LLH_4$, $LLH_6 \rightsquigarrow LLH_5$, $LLH_6 \rightsquigarrow LLH_6$ and $LLH_8 \rightsquigarrow LLH_6$. All the heuristics responsible for moving visits between routes showed good performance for the heuristic hybridisations in case of the HCSP. Move operators also contributed to the hybridisation process for the SPRR. In this case, LLH_4 is used mostly as the second heuristic of a pair. Effective heuristic pairs discovered by the hyper-heuristic are $LLH_3 \rightsquigarrow LLH_3$, $LLH_3 \rightsquigarrow LLH_4$, $LLH_4 \rightsquigarrow LLH_3$, $LLH_5 \rightsquigarrow LLH_4$ and $LLH_6 \rightsquigarrow LLH_4$. The number of hybridised heuristics found appears to be greater for the MPSP. LLH_1, LLH_2, LLH_7 and LLH_8 showed effective performance as second heuristics of the pairs. Differently to the other two problems, swap heuristics performed well in hybridisations. It is possible to see various heuristic combinations with all the heuristics except for when LLH_2 and LLH_5 are the first heuristics for this domain.

These results show that it is possible to find or generate new heuristics based on the given general heuristics for the tested problems. In addition, the heuristics that are not powerful enough alone can be used in combination with other heuristics and can deliver good results. This underlines the importance of using the potential of the different heuristics. More complex hybridisation strategies that can find heuristic combinations of more than two heuristics can be useful for attaining higher performance.

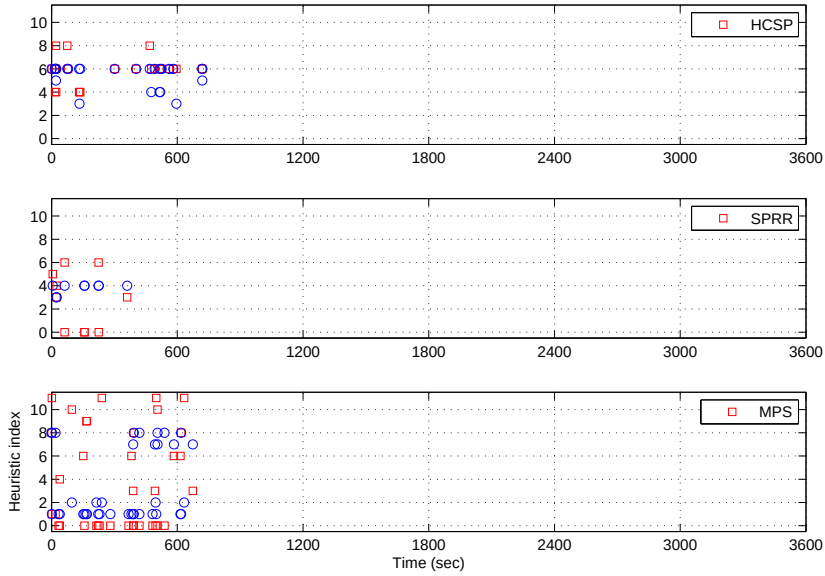

Figure 7.6: Heuristic pairs found new best solutions (squares and circles represent the consecutively applied heuristics respectively)

7.3 Conclusion

The purpose of this chapter is to show the requirements for solving different problems from routing and rostering domains. Accordingly, it is aimed at addressing how to incorporate multiple generalised heuristics in the context of hyper-heuristics. In answering this question, the first international cross-domain heuristic search challenge (CHeSC) 2011 winner hyper-heuristic, GIHH, was employed. It was applied to three problems incorporating some common objectives as well as the same constraint set. Each of the problems has a particular planning horizon, e.g. one day, one month or one week. A set of heuristics that can be used in solving these problems in both a routing and rostering sense were implemented. These heuristics involve certain operators that move and swap visits between different routes and make inner route changes.

The experimental results demonstrated that, although the problems have several common features, the heuristics that contribute most to the best resulting solution differ. One major reason is connected with certain problem characteristics, namely the planning horizon, the number of tasks and the available personnel to handle these tasks. Another outcome is the opportunity

of using a selection hyper-heuristic as an analysis tool to study the correlation between algorithms and problems. The proposed methodology can be used to develop new dedicated approaches using the gathered knowledge about available heuristics for solving instances from any target problem domain.

Chapter 8

Conclusion

The primary goal of research focusing on algorithms for combinatorial optimisation problems is to deliver the best results over a set of problem instances. The development period for designing such an algorithm is usually long. The development process includes examining the problem instances, designing an algorithm, testing and tuning it and finalising. It is expected that such an approach would successfully find high quality solutions for the target problem instances. Although the development is time consuming, it is not usually possible to apply such algorithms to other problems. It is even required to adapt these specifically designed algorithms in order to enable solving other instances of the same problem. Particularly meta-heuristics sometimes suffer from their problem-dependent implementations. Hyper-heuristics, search heuristic spaces rather than solution spaces. No problem-dependent information is required when applying a hyper-heuristic. Therefore, hyper-heuristics are easily applicable to any problem requiring search. The literature survey on hyper-heuristics presented in Chapter 2 provides details about hyper-heuristics and their application domains.

The objective of the dissertation was to design a hyper-heuristic that can effectively solve any search and optimisation problem. This objective can be considered in the context of ***generality***. Although generality is frequently mentioned in hyper-heuristic studies, there is no clear generality definition. It is possible to define it from different perspectives. The most comprehensive definition of generality can be stated as efficiently solving any problem requiring search under different conditions. The no free lunch theorem [339] is the theoretical limitation to achieve this generality level. This theorem indicates that the performance of all search algorithms is the same when they are applied

to all possible problems. Hyper-heuristics need not suffer from this limitation if the target problems are not closed under permutation [277]. Although this is the case in most applications, it is still a challenging task to provide generality over a number of problems.

We investigated how performance of a hyper-heuristic changes when the target problem, heuristic set or the execution time limit vary. In other words, this investigation was subject to the determining generality elements. The hyper-heuristic that we designed, i.e. generic intelligent hyper-heuristic (GIHH), based on the examined generality elements proved its effectiveness by winning the first cross-domain heuristic search challenge (CHeSC) 2011. Therefore, GIHH, the best hyper-heuristic in the literature in terms of generality, can be used as a benchmark hyper-heuristic to compare with new hyper-heuristic designs.

8.1 Discussion

Limitations of GIHH: GIHH can be used to solve any search and optimisation problem. There is no limitation on the type of low-level heuristics employed for solving a problem, although this might be an issue for certain meta-heuristic based hyper-heuristics. For instance, a hyper-heuristic based on iterated local search requires at least one mutational or perturbative heuristic and one local search operator. The only limitation of GIHH is that it requires a single objective function for solving a problem. Multi-objective optimisation problems can be addressed by, e.g. combining the objectives in a weighted sum. This means that GIHH is still applicable to the problems with multiple objectives, however it sacrifices the problem-independent information regarding separate objectives.

How to design an effective hyper-heuristic: While designing a hyper-heuristic, the first step should be determining the required generality level. For instance, generality for solving exam timetabling problems or generality for a specifically designed heuristic set. Followingly, the factors affecting the target generality level should be determined. Appropriate hyper-heuristic sub-mechanisms should be designed or existing mechanisms from the literature should be employed to address these requirements. The main issue in this approach is the design complexity of the hyper-heuristic. Usually the number of sub-mechanisms increase proportionally to the generality factors. In this case, combining the designed components efficiently is the next step to complete the hyper-heuristic design. For combining the components, a number of decision mechanisms or coordination methods is required to decide on when a particular sub-mechanism needs to be used.

Importance of online learning and adaptation: In order to raise the level of generality for the aforementioned factors, it is important to employ learning devices and use adaptive hyper-heuristic components. These learning and adaptation processes can take place *offline*, before starting to solve a problem instance, or *online*, while solving a problem instance. Offline methods are very useful. However, they usually need a set of representative test instances for training. This takes extra time before the actual search process can start. On the other hand, performing learning and adaptation processes online is practical to address the changing search requirements. It is known that the performance of low-level heuristics changes over time. Thus, an online approach has more potential than an offline strategy to capture these changes.

This online and offline learning comparison can additionally degrade to setting a hyper-heuristic's parameters offline and online. It is important to use multiple mechanisms for addressing various generality issues. However, it should be noted that adding a mechanism means adding at least one parameter or rule to the hyper-heuristic. Therefore, it is important to consider the offline learning and online learning comparison for parameter adaptation. These parameter adaptation methods were categorised in [137] as *parameter tuning* and *parameter control*. Parameter tuning refers to the methods adapting parameters offline while parameter control takes place online.

The significance of parameter control in comparison with parameter tuning can be explained by the following example. What type of a parameter setting approach should be used for the threshold level parameter of a threshold accepting mechanism [132]? Here, the acceptance mechanism accepts a solution if its fitness value is lower than a given threshold value. It is obvious that setting this parameter beforehand is likely to fail due to the varying search requirements. However, a large number of parameters makes the parameter control process hard and complex. Therefore, parameter tuning strategies can be considered to set a hyper-heuristic's parameters if there is no acceptable strategy to adapt the available parameters. Careful selection of training instances is vital. One also needs to investigate how the parameters limit the generality level of a hyper-heuristic. If these generality related factors are not considered, it turns out that the hyper-heuristic should be adapted each time it will be applied to other problems or instances of the same problem. Offline learning methods can be preferred if online learning is extremely hard or online learning can be supported by offline learning for setting the initial parameter values.

Measuring the performance of hyper-heuristics: Although hyper-heuristics are not intended to beat state of the art algorithms for a particular problem, they can deliver superior solutions. The two possible reasons for such a successful

hyper-heuristic application include: *1)* tailored hyper-heuristic design, *2)* a very effective heuristic set. The first option occurs when a hyper-heuristic is designed for a generality level limited to a set of target problem instances. In this case, it is reasonable to expect that the designed hyper-heuristic performs effectively through the strengths of a set of low-level heuristics. In addition, if a heuristic set is well defined with respect to the low-level heuristics' *intensification*, *diversification* and *speed* characteristics, a hyper-heuristic can reach competitive solutions to the dedicated algorithms. However, if the quality of the heuristic set it not good enough, it is likely that solutions are worse than those determined with an effective heuristic set. Therefore, the quality of a solution returned by a hyper-heuristic is not directly correlated with those determined with the hyper-heuristic's performance. The latter should be measured with respect to the quality and potential of the employed heuristic set.

8.2 Future research

As a part of my future research agenda, I intend to focus on the following research topics in the context of hyper-heuristics. All these research topics can also be considered for anyone who wants to provide new perspectives to the hyper-heuristic research.

A fully automated hyper-heuristic: Selection hyper-heuristics are easy-to-use methods for application to any problem requiring a search process. However, as shown in Chapter 5, the performance of a hyper-heuristic depends on the available heuristic set. Therefore, although a hyper-heuristic approach is effective to manage a given heuristic set, it is almost impossible to be competitive with problem specific algorithms. Including effective low-level heuristics designed for a specific problem into the heuristic set can provide an opportunity to generate better quality results. Thus, automatic heuristic generation strategies, i.e. generation hyper-heuristics, may help to construct effective heuristics for a specific problem. A combination of a generation hyper-heuristic and a selection hyper-heuristic can then be used as a fully automated hyper-heuristic tool for solving problems more effectively.

A hyper-heuristic analysis tool for offline learning/tuning: Hyper-heuristics were initially described as high-level, problem-independent search and optimisation strategies for finding "*good enough - soon enough - cheap enough*" [62] solutions for any target problem domain. When some problem-dependent information is available, it is always possible to design a superior algorithm than

a hyper-heuristic. As explained in Chapter 6, using selection hyper-heuristics as heuristic analysis tools can lead to new, efficient dedicated algorithms for problems. The work will be extended to automatise the approach as an offline learning or tuning tool.

Theoretical backgrounds of hyper-heuristics: The superiority of using several mutation operators compared to using a single mutation operator in evolutionary algorithms was theoretically discussed in a recent paper [172]. Although this research on evolutionary algorithms provides insights about the effectiveness of hyper-heuristics, there is no published theoretical work dedicated to hyper-heuristics other than an abstract [211]. Almost all the hyper-heuristic studies can be considered empirical work. The only option available is to rely on experience with the existing hyper-heuristic studies and to make educated guesses. Although this type of research path is also valid and acceptable, investigating the underlying behaviour of the hyper-heuristic methods from a theoretical perspective may yield new principles for more powerful hyper-heuristic designs.

Multi-objective optimisation: All the problems in the present research were modelled on single-objective optimisation problems like most hyper-heuristic studies indeed do. They often consider single objectives by combining multiple objectives into one weighted sum objective function. However, considering each objective separately means using more information about the solution space. Therefore, we assume that it is possible to use the power of hyper-heuristics more effectively even though multiple objectives requires more complex hyper-heuristics. It is also possible to have multi-objective optimisation problems by counting constraints as objectives. Overall, this topic promises a wide range of research opportunities to provide new perspectives for the hyper-heuristic field.

Fitness landscape analysis: Fitness landscapes are used to investigate the relationship between solutions in a search space. A fitness landscape is composed of a solution representation, a fitness function and a neighbourhood. The neighbourhood part changes by each low-level heuristic. Thus, each low-level heuristic itself provides a separate landscape in a hyper-heuristic setting. Hence, the effectiveness of selection hyper-heuristics also depends on accommodating multiple landscapes. If a solution is represented on each landscape provided by a low-level heuristic, selecting different heuristics during a search can help to find improving solutions easier. The literature covers a few studies [256, 257, 218] focusing on fitness landscapes for hyper-heuristics. The information gathered by existing fitness landscape analysis techniques would be useful for the selection

process. In addition, information related to the hardness of the landscapes would help to adapt the move acceptance criteria together with other hyper-heuristic sub-mechanisms. This particular open research question offers plenty opportunities to develop ideas on the fitness landscape analysis for hyper-heuristics.

The number of researchers working on algorithms for search and optimisation problems is large. Accordingly, finding an effective algorithm gets harder. Therefore, *hybridisation* is one of the current and future research subjects for algorithm design. The motivation to use hybridisation is to combine the power of different strategies for superior performance on any application domain. However, such designs are likely to require a more time consuming design process. This leads researchers to design more *automated* and *adaptive* systems. Hyper-heuristics already behave on the idea of hybridisation and collaboration. The hybridisation comes with employing a number of hyper-heuristic sub-mechanisms and using more than one heuristic. Moreover, hyper-heuristics aiming at generality are automated approaches by definition. Consequently, these advantages make the hyper-heuristic research more attractive for researchers and the aforementioned open research questions concerning future research provide a large room of promising long-running research directions.

Bibliography

[1] Aamodt, A., and Plaza, E. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications 7*, 1 (1994), 39–52. pages 22

[2] Abdul-Rahman, S., Burke, E., Bargiela, A., McCollum, B., and Özcan, E. A constructive approach to examination timetabling based on adaptive decomposition and ordering. *Annals of Operations Research* (2011). pages 43

[3] Abednego, L., and Hendratmo, D. Genetic programming hyper-heuristic for solving dynamic production scheduling problem. In *Proceedings of the International Conference on Electrical Engineering and Informatics (ICEEI'11)* (Bandung, Indonesia, July 17–19 2011). pages 13, 25, 41

[4] Abramson, D., and Dang, H. *Applied Simulated Annealing, LNEMS*. Springer, 1993, ch. School timetables: A case study in simulated annealing, pp. 103–124. pages 43

[5] Ahmed, A., Mazhar, A., Sajid, A., and Bukhari, A. Particle swarm based hyper-heuristic for tackling real world examinations scheduling problem. *Australian Journal of Basic and Applied Sciences 5*, 10 (2011), 1406–1413. pages 13, 43

[6] Ahmed, A., Shaikh, A., Mazhar, A., and Bukhari, A. Hyper-heuristic approach for solving scheduling problem: A case study. *Australian Journal of Basic and Applied Sciences 5*, 9 (2011), 190–199. pages 43

[7] Ahmed, A., and Zhoujun, L. A biphasic approach for university timetabling problem. In *Proceedings of the 2nd International Conference on Computer Engineering and Technology (ICCET'10)* (2010), vol. 1, IEEE, pp. 192–197. pages 43

[8] Aickelin, U., and Dowsland, K. Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *Journal of Scheduling 3*, 3 (2000), 139–153. pages 21

[9] Aickelin, U., and Dowsland, K. An indirect genetic algorithm for a nurse-scheduling problem. *Computers and Operations Research 31*, 5 (2004), 761–778. pages 29

[10] Akjiratikarl, C., Yenradee, P., and Drake, P. PSO-based algorithm for home care worker scheduling in the UK. *Computers and Industrial Engineering 53*, 4 (2007), 559–583. pages 52

[11] Alinia Ahandani, M., Vakil Baghmisheh, M., Badamchi Zadeh, M., and Ghaemi, S. Hybrid particle swarm optimization transplanted into a hyper-heuristic structure for solving examination timetabling problem. *Swarm and Evolutionary Computation* (to appear). pages 13, 43

[12] Allen, S., Burke, E., Hyde, M., and Kendall, G. Evolving reusable 3D packing heuristics with genetic programming. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO'09)* (Montreal, Quebec, Canada, 2009), F. Rothlauf, Ed., ACM, pp. 931–938. pages 13, 25

[13] Alperovits, E., and Shamir, U. Design of optimal water distribution systems. *Water Resources Research 13*, 6 (1977), 885–900. pages 42

[14] Anagnostopoulos, A., Michel, L., Van Hentenryck, P., and Vergados, Y. A simulated annealing approach to the traveling tournament problem. *Journal of Scheduling 9*, 2 (2006), 177–193. pages 13, 14, 28, 40, 42, 90, 91

[15] Anagnostopoulos, K., and Koulinas, G. A simulated annealing hyperheuristic for construction resource levelling. *Construction Management & Economics 28*, 2 (2010), 163–175. pages 13

[16] Anagnostopoulos, K., and Koulinas, G. Resource-constrained critical path scheduling by a GRASP based hyperheuristic. *Journal of Computing in Civil Engineering 26*, 2 (2012), 204–213. pages 13

[17] Araya, I., Neveu, B., and Riff, M. An efficient hyperheuristic for strip-packing problems. In *Adaptive and Multilevel Metaheuristics*, C. Cotta, M. Sevaux, and K. Sörensen, Eds., vol. 136 of *Studies in Computational Intelligence*. Springer, 2008, pp. 61–76. pages 45

[18] Asbach, L., Dorndorf, U., and Pesch, E. Analysis, modeling and solution of the concrete delivery problem. *European Journal of Operational Research 193*, 3 (2009), 820–835. pages 60

[19] Ayob, M., and Kendall, G. A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine. In *Proceedings of the International Conference on Intelligent Technologies (InTech'03)* (Chiang Mai, Thailand, December 17–19 2003), pp. 132–141. pages 14, 40, 112, 127

[20] Aziz, Z., and Kendall, G. An investigation of an ant-based hyperheuristic for the capacitated vehicle routing problem. In *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA'09)* (Dublin, Ireland, August 10–12 2009), J. Blazewicz, M. Drozdowski, G. Kendall, and B. McCollum, Eds., pp. 823–826. pages 13

[21] Bader-El-Den, M., and Poli, R. A GP-based hyper-heuristic framework for evolving 3-SAT heuristics. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO'07)* (New York, NY, USA, 2007), H. Lipson, Ed., ACM, pp. 1749–1749. pages 25, 46

[22] Bader-El-Den, M., and Poli, R. Inc*: an incremental approach for improving local search heuristics. In *Proceedings of the 8th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP'08)* (2008), J. van Hemert and C. Cotta, Eds., vol. 4972 of *LNCS*, Springer, pp. 194–205. pages 13, 25

[23] Bader-El-Den, M., and Poli, R. *Genetic Programming Theory and Practice VI*. Springer, 2009, ch. Evolving Effective Incremental Solvers for SAT with a Hyper-heuristic Framework based on Genetic Programming, pp. 163–178. pages 13, 25, 46

[24] Bader-El-Den, M., and Poli, R. Grammar-based genetic programming for timetabling. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'09)* (2009), IEEE, pp. 2532–2539. pages 25

[25] Bader-El-Den, M., Poli, R., and Fatima, S. Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. *Memetic Computing 1*, 3 (2009), 205–219. pages 3, 13, 25, 43

[26] Bai, R., Blazewicz, J., Burke, E., Kendall, G., and McCollum, B. A simulated annealing hyper-heuristic methodology for flexible decision

support. *4OR: A Quarterly Journal of Operations Research 10*, 1 (2012), 43–66. pages 13, 15, 28, 40, 41, 43, 45

[27] Bai, R., Burke, E., Gendreau, M., and Kendall, G. A simulated annealing hyper-heuristic: Adaptive heuristic selection for different vehicle routing problems. In *Proceedings of the 3rd Multidisciplinary Intternational Conference on Scheduling: Theory and Applications (MISTA'07)* (2007), pp. 67–70. pages 13

[28] Bai, R., Burke, E., Gendreau, M., Kendall, G., and McCollum, B. Memory length in hyper-heuristics: An empirical study. In *Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling (SCIS'07)* (2007), IEEE, pp. 173–178. pages 15, 43

[29] Bai, R., Burke, E., and Kendall, G. Heuristic, meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation. *Journal of the Operational Research Society 59*, 10 (2008), 1387–1397. pages 13, 15, 29, 40, 41

[30] Bai, R., Burke, E., Kendall, G., Li, J., and McCollum, B. A hybrid evolutionary approach to the nurse Rostering problem. *IEEE Transactions on Evolutionary Computation 14*, 4 (2010), 580–590. pages 40

[31] Bai, R., and Kendall, G. An investigation of automated planograms using a simulated annealing based hyper-heuristics. In *Meta-heuristics: Progress as Real Problem Solvers, Selected Papers from the 5th Metaheuristics International Conference (MIC'03)* (2005), T. Ibaraki, K. Nonobe, and M. Yagiura, Eds., Springer, pp. 87–108. pages 13, 14, 15, 28, 29, 40, 41

[32] Bai, R., and Kendall, G. A model for fresh produce shelf space allocation and inventory management with freshness condition dependent demand. *INFORMS Journal on Computing 20*, 1 (2008), 78–85. pages 29, 41

[33] Beasley, J. Algorithms for unconstrained two-dimensional guillotine cutting. *Journal of the Operational Research Society 36*, 4 (1985), 297–306. pages 45

[34] Beasley, J. An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research 33*, 1 (1985), 49–64. pages 45

[35] Beasley, J. A note on solving large p-median problems. *European Journal of Operational Research 21*, 2 (1985), 270–273. pages 42

[36] Begur, S., Miller, D., and Weaver, J. An integrated spatial DSS for scheduling and routing home-health-care nurses. *Interfaces 27*, 4 (1997), 35–48. pages 52

[37] Bellmore, M., and Nemhauser, G. The traveling salesman problem: a survey. *Operations Research 16*, 3 (1968), 538–558. pages 75

[38] Bengtsson, B. Packing rectangular pieces - a heuristic approach. *The Computer Journal 25*, 3 (1982), 353–357. pages 45

[39] Benoist, T., Laburthe, F., and Rottembourg, B. Lagrange relaxation and constraint programming collaborative schemes for travelling tournament problems. In *Proceedings of the 3rd International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'01)* (2001), pp. 15–26. pages 90

[40] Berberoğlu, A., and Uyar, A. A hyper-heuristic approach for the unit commitment problem. In *Proceedings of the 10th European Conference on the Applications of Evolutionary Computation (EvoApplications'10)* (2010), C. Di Chio, A. Brabazon, G. Di Caro, M. Ebner, M. Farooq, A. Fink, J. Grahl, G. Greenfield, P. Machado, M. O'Neill, E. Tarantino, and N. Urquhart, Eds., vol. 6025 of *LNCS*, Springer, pp. 121–130. pages 25, 35, 40, 41

[41] Berberoğlu, A., and Uyar, A. Experimental comparison of selection hyper-heuristics for the short-term electrical power generation scheduling problem. In *Proceedings of the 11th European Conference on the Applications of Evolutionary Computation (EvoApplications'11)*, C. Di Chio, A. Brabazon, G. Di Caro, R. Drechsler, M. Farooq, J. Grahl, G. Greenfield, C. Prins, J. Romero, G. Squillero, E. Tarantino, A. Tettamanzi, N. Urquhart, and A. Uyar, Eds., vol. 6625 of *LNCS*. Springer, 2011, pp. 444–453. pages 14, 15, 36, 40, 41

[42] Bertels, S., and Fahle, T. A hybrid setup for a hybrid scenario: combining heuristics for the home health care problem. *Computers & Operations Research 33*, 10 (2006), 2866–2890. pages 52

[43] Bhanu, S., and Gopalan, N. A hyper-heuristic approach for efficient resource scheduling in grid. *International Journal of Computers, Communication and Control 3*, 3 (2008), 249–258. pages 30, 40, 41

[44] Bilgin, B., De Causmaecker, P., Rossie, B., and Vanden Berghe, G. Local search neighbourhoods to deal with a novel nurse rostering model. In *Proceedings of the 7th International Conference on Practice and Theory of Automated Timetabling (PATAT'08)* (2008). pages 41

[45] Bilgin, B., De Causmaecker, P., and Vanden Berghe, G. A hyperheuristic approach to belgian nurse rostering problems. In *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA'09)* (Dublin, Ireland, August 10–12 2009), J. Blazewicz, M. Drozdowski, G. Kendall, and B. McCollum, Eds., pp. 683–689. pages 14, 29, 40, 41

[46] Bilgin, B., Demeester, P., Misir, M., Vancroonenburg, W., and Vanden Berghe, G. One hyperheuristic approach to two timetabling problems in health care. *Journal of Heuristics 18*, 3 (2012), 401–434. pages xx, 6, 7, 13, 15, 37, 40, 41, 65, 67, 68, 124, 127, 143, 148

[47] Bilgin, B., Demeester, P., Misir, M., W., V., Vanden Berghe, G., and Wauters, T. A Hyper-heuristic Combined with a Greedy Shuffle Approach to the Nurse Rostering Competition. In *the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT'10)* (Belfast, Northern Ireland, August 10–13 2010). pages 7, 41

[48] Bilgin, B., Demeester, P., and Vanden Berghe, G. A hyperheuristic approach to the patient admission scheduling problem. Technical report, KaHo Sint-Lieven, 2008. pages 41, 92, 94, 95, 96

[49] Bilgin, B., Özcan, E., and Korkmaz, E. An experimental study on hyper-heuristics and final exam scheduling. In *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT'06)* (Brono, Czech Republic, August 31–1 September 2006), pp. 123–140. pages 14, 15, 25, 31, 34, 40, 43

[50] Bittle, S., and Fox, M. Learning and using hyper-heuristics for variable and value ordering in constraint satisfaction problems. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO'09): Late Breaking Papers* (2009), F. Rothlauf, Ed., ACM, pp. 2209–2212. pages 41, 42

[51] Blazewicz, J., Burke, E., Kendall, G., Mruczkiewicz, W., Oguz, C., and Swiercz, A. A hyper-heuristic approach to sequencing by hybridization of DNA sequences. *Annals of Operations Research* (to appear). pages 15, 36

[52] Blum, C., and Blesa, M. Probabilistic beam search for the longest common subsequence problem. In *Proceedings of the International Workshop on Engineering Stochastic Local Search Algorithms (SLS'07): Designing, Implementing and Analyzing Effective Heuristics* (2007), T. Stützle, M. Birattari, and H. H. Hoos, Eds., vol. 4638 of *LNCS*, Springer-Verlag, pp. 150–161. pages 42

[53] Branke, J. Memory-enhanced evolutionary algorithms for dynamic optimization problems. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'99)* (1999), vol. 3, pp. 1875–1882. pages 36

[54] Bräysy, O., and Gendreau, M. Vehicle routing problem with time windows, Part I: Route construction and local search algorithms. *Transportation science 39*, 1 (2005), 104–118. pages 195

[55] Burke, E., and Bykov, Y. Solving exam timetabling problems with the flex-deluge algorithm. In *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling (PATAT'06)* (Brono, Czech Republic, August 30–September 1 2006), E. K. Burke and H. Rudova, Eds., pp. 351–358. pages 30, 40

[56] Burke, E., and Bykov, Y. A late acceptance strategy in hill-climbing for exam timetabling problems. In *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT'08)* (Montreal, Canada, August 18–22 2008). pages 31, 40, 95

[57] Burke, E., Curtois, T., Hyde, M., Kendall, G., Ochoa, G., Petrovic, S., Vázquez-Rodriguez, J., and Gendreau, M. Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'10)* (Barcelona, Spain, July 18–23 2010), pp. 3073–3080. pages 13, 25, 40, 41, 125

[58] Burke, E., De Causmaecker, P., Vanden Berghe, G., and Van Landeghem, H. The state of the art of nurse rostering. *Journal of scheduling 7*, 6 (2004), 441–499. pages 52

[59] Burke, E., Dror, M., Kendall, G., O'Brien, R., Redrup, D., and Soubeiga, E. A pheromone-based look-ahead hyper-heuristic for timetabling problems. In *Proceedings of the 1st Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA'03)* (Nottingham, UK, August 13–16 2003). pages 13

[60] Burke, E., Dror, M., Petrovic, S., and Qu, R. Hybrid graph heuristics within a hyper-heuristic approach to exam timetabling problems. In *The Next Wave in Computing, Optimization, and Decision Technologies*, B. Golden, S. Raghavan, and E. Wasil, Eds., vol. 29 of *Operations Research Computer Science Interfaces Series*. Springer, 2005, pp. 79–91. pages 43

[61] Burke, E., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Qu, R. Hyper-heuristics: A survey of the state of the

art. *Journal of the Operational Research Society* (to appear). pages 5, 11, 153

[62] Burke, E., Hart, E., Kendall, G., Newall, J., Ross, P., and Schulenburg, S. *Handbook of Meta-Heuristics*. Kluwer Academic Publishers, 2003, ch. Hyper-Heuristics: An Emerging Direction in Modern Search Technology, pp. 457–474. pages 2, 121, 206

[63] Burke, E., Hyde, M., and Kendall, G. Evolving bin packing heuristics with genetic programming. In *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN'06)* (Reykjavik, Iceland, September 9–13 2006), T. Runarsson, H-G.B., E. Burke, J. Merelo-Guervos, L. Whitley, and X. Yao, Eds., vol. 4193 of *LNCS*, Springer-Verlag, pp. 860–869. pages 3, 13, 24, 25, 45

[64] Burke, E., Hyde, M., and Kendall, G. Providing a memory mechanism to enhance the evolutionary design of heuristics. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'10)* (Barcelona, Spain, July 18–23 2010), pp. 3883–3890. pages 45

[65] Burke, E., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. Exploring hyper-heuristic methodologies with genetic programming. In *Computational Intelligence*, C. L. Mumford and L. C. Jain, Eds., vol. 1 of *Intelligent Systems Reference Library*. Springer Berlin Heidelberg, 2009, pp. 177–201. pages 13

[66] Burke, E., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. A classification of hyper-heuristic approaches. In *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds., vol. 146 of *International Series in Operations Research & Management Science*. Springer, 2010, pp. 449–468. pages 2, 77

[67] Burke, E., Hyde, M., Kendall, G., and Woodward, J. Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a master of one. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO'07)* (London, England, July 12–16 2007), H. Lipson, Ed., pp. 1559–1565. pages 3, 13, 25

[68] Burke, E., Hyde, M., Kendall, G., and Woodward, J. A genetic programming hyper-heuristic approach for evolving 2-D strip packing heuristics. *IEEE Transactions on Evolutionary Computation 14*, 6 (2010), 942–958. pages 3, 13, 25

[69] Burke, E., Kendall, G., L. Silva, D., O'Brien, R., and Soubeiga, E. An ant algorithm hyperheuristic for the project presentation

scheduling problem. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'05)* (2005), vol. 3, pp. 2263–2270. pages 3, 13, 15, 18, 40, 41

[70] Burke, E., Kendall, G., Misir, M., and Özcan, E. A study of simulated annealing hyperheuristics. In *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT'08)* (Montreal, Canada, August 18–22 2008). pages 13

[71] Burke, E., Kendall, G., Misir, M., and Özcan, E. Monte carlo hyper-heuristics for examination timetabling. *Annals of Operations Research 196*, 1 (2012), 73–90. pages 3, 40, 43

[72] Burke, E., Kendall, G., OBrien, R., Redrup, D., and Soubeiga, E. An ant algorithm hyper-heuristic. In *Proceedings of the 5th Metaheuristics International Conference (MIC'03)* (Kyoto, Japan, August 25–28 2003), pp. 25–28. pages 13

[73] Burke, E., Kendall, G., and Soubeiga, E. A tabu-search hyper-heuristic for timetabling and rostering. *Journal of Heuristics 9*, 3 (2003), 451–470. pages 3, 13, 15, 21, 22, 29, 35, 40, 41, 43, 67

[74] Burke, E., Li, J., and Qu, R. A hybrid model of integer programming and variable neighbourhood search for highly-constrained nurse rostering problems. *European Journal of Operational Research 203*, 2 (2010), 484–493. pages 67

[75] Burke, E., MacCarthy, B., Petrovic, S., and Qu, R. Knowledge discovery in a hyper-heuristic for course timetabling using case-based reasoning. In *Selected Revised Papers from the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT'02)* (Gent, Belgium, August 21–23 2002), E. K. Burke and P. De Causmaecker, Eds., vol. 2740 of *LNCS*, pp. 276–287. pages 15, 22, 43

[76] Burke, E., McCollum, B., Meisels, A., Petrovic, S., and Qu, R. A graph-based hyper heuristic for educational timetabling problems. *European Journal of Operational Research 176* (2007), 177–192. pages 13, 15, 19, 23, 29, 43

[77] Burke, E., Misir, M., Ochoa, G., and Özcan, E. Learning heuristic selection in hyperheuristics for examination timetabling. In *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling (PATAT'08)* (Montreal, Canada, August 18–22 2008). pages 14, 86

[78] Burke, E., Petrovic, S., and Qu, R. Case based heuristic selection for timetabling problems. *Journal of Scheduling 9*, 2 (2006), 115–132. pages 2, 15, 22, 43

[79] Burke, E., Pham, N., and Qu, R. An investigation of population-based hyper-heuristics for graph colouring. In *Proceedings of the 7th Metaheuristics International Conference (MIC'07)* (2007). pages 13

[80] Burke, E., Qu, R., and Soghier, A. Adaptive selection of heuristics for improving constructed exam timetables. In *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT'10)* (Belfast, Northern Ireland, August 10–13 2010), pp. 136–152. pages 43

[81] Burke, E., Qu, R., and Soghier, A. Adaptive selection of heuristics for improving exam timetables. *Annals of Operations Research* (to appear). pages 43

[82] Burke, E., and Silva, J. L. The influence of the fitness evaluation method on the performance of multiobjective search algorithms. *European Journal of Operational Research 169*, 3 (2003), 875–897. pages 22

[83] Burke, E., Silva, J. L., and Soubeiga, E. *Meta-heuristics: Progress as Real Problem Solvers*. Springer, 2005, ch. Multi-objective Hyper-heuristic Approaches for Space Allocation and Timetabling, pp. 129–158. pages 13, 15, 21, 41, 43, 46

[84] Burke, E., and Soubeiga, E. Scheduling nurses using a tabu-search hyperheuristic. In *Proceedings of the 1st Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA'03)* (Nottingham, UK, 2003), pp. 180–197. pages 13, 15

[85] Burke, E. K., Hyde, M. R., and Kendall, G. Grammatical evolution of local search heuristics. *IEEE Transactions on Evolutionary Computation 16*, 3 (2012), 406–417. pages 26, 40, 45

[86] Burke, E. K., Hyde, M. R., Kendall, G., and Woodward, J. Automating the packing heuristic design process with genetic programming. *Evolutionary Computation 20*, 1 (2012), 63–89. pages 25, 26, 45

[87] Burke, E. K., Silva, J. L., and Soubeiga, E. Hyperheuristic approaches for multiobjective optimisation. In *Proceedings of the 5th Metaheuristics International Conference (MIC'03)* (Kyoto, Japan, August 25–28 2003). pages 15, 41, 46

[88] Calvo, R. W., and Cordone, R. A heuristic approach to the overnight security service problem. *Computers & Operations Research 30*, 9 (2003), 1269–1287. pages 52

[89] Cano-Belmán, J., Ríos-Mercado, R., and Bautista, J. A scatter search based hyper-heuristic for sequencing amixed-model assembly line. *Journal of Heuristics 16*, 6 (2010), 749–770. pages 13

[90] Cardemil, A. Optimizacion de fixtures deportivos: Estado del arte y un algoritmo tabu search para el traveling tournament problem. Master's thesis, Departamento de Computacion Facultad de Ciencias Exactus y Naturales, Universidad de Buenes Aires, 2002. pages 90

[91] Carter, M., Laporte, G., and Lee, S. Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society 47*, 3 (1996), 373–383. pages 43

[92] Castrillón, O., Sarache, W., and Giraldo, J. Design of a hyperheuristic for production scheduling in job shop environments. *Ingeniare. Revista Chilena de Ingeniería 18*, 2 (2010), 203–214. pages 41

[93] Ceschia, S., and Schaerf, A. Multi-neighborhood local search for the patient admission problem. In *Proceedings of the 6th International Workshop on Hybrid Metaheuristics (HM'09)* (2009), M. J. Blesa, C. Blum, L. Di Gaspero, A. Roli, M. Sampels, and A. Schaerf, Eds., vol. 5818 of *LNCS*, Springer, pp. 156–170. pages 65

[94] Ceschia, S., and Schaerf, A. Local search and lower bounds for the patient admission scheduling problem. *Computers & Operations Research 38*, 10 (2011), 1452–1463. pages xx, 65, 148

[95] Chakhlevitch, K., and Cowling, P. Choosing the fittest subset of low level heuristics in a hyperheuristic framework. In *Proceedings of the 5th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP'05)* (2005), G. Raidl and J. Gottlieb, Eds., vol. 3448 of *LNCS*, Springer, pp. 23–33. pages 35

[96] Chakraborty, G. An efficient heuristic algorithm for channel assignment problem in cellular radio networks. *IEEE Transactions on Vehicular Technology 50*, 6 (2001), 1528–1539. pages 42

[97] Chao-Hsien, P. A study of integer programming formulations for scheduling problems. *International Journal of Systems Science 28*, 1 (1997), 33–41. pages 73

[98] Chen, P.-C., Kendall, G., and Vanden Berghe, G. An ant based hyper-heuristic for the travelling tournament problem. In *Proceedings of IEEE Symposium of Computational Intelligence in Scheduling (CISched'07)* (Hawaii, USA, April 1–5 2007), pp. 19–26. pages xix, 13, 15, 19, 42, 90, 91

[99] Christofides, N., and Beasley, J. The period routing problem. *Networks 14*, 2 (1984), 237–256. pages 44

[100] Christofides, N., and Whitlock, C. An algorithm for two-dimensional cutting problems. *Operations Research 25*, 1 (1977), 30–44. pages 45

[101] Cordeau, J.-F., Desaulniers, G., Desrosiers, J., Solomon, M., and Soumis, F. *The Vehicle Routing Problem*, vol. 9 of *SIAM Monographs on Discrete Mathematics and Applications*. 2002, ch. The VRP with Time Windows, pp. 157–193. pages 52

[102] Cowling, P., Colledge, N., Dahal, K., and Remde, S. The trade off between diversity and quality for multi-objective workforce scheduling. In *Proceedings of the 6th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP'06)* (2006), J. Gottlieb and G. R. Raidl, Eds., vol. 3906 of *LNCS*, Springer, pp. 13–24. pages 34, 41

[103] Cowling, P., Kendall, G., and Han, L. An adaptive length chromosome hyperheuristic genetic algorithm for a trainer scheduling problem. In *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution And Learning (SEAL'02)* (Orchid Country Club, Singapore, November 18–22 2002), pp. 267–271. pages 13, 14, 15, 41

[104] Cowling, P., Kendall, G., and Han, L. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'02)* (May 12–17 2002), pp. 1185–1190. pages 12, 13, 15, 41

[105] Cowling, P., Kendall, G., and Soubeiga, E. A hyperheuristic approach to scheduling a sales summit. In *Selected Papers from the 3rd International Conference on Practice and Theory of Automated Timetabling (PATAT'00)* (London, UK, 2001), E. K. Burke and W. Erben, Eds., vol. 2079 of *LNCS*, Springer-Verlag, pp. 176–190. pages 2, 14, 15, 33, 35, 40, 41, 95, 110, 127, 128

[106] Cowling, P., Kendall, G., and Soubeiga, E. A parameter-free hyperheuristic for scheduling a sales summit. In *Proceedings of 4th Metahuristics International Conference (MIC'01)* (Porto, Portugal, July 16–20 2001), pp. 127–131. pages 15, 40, 41

[107] Cowling, P., Kendall, G., and Soubeiga, E. Hyperheuristics: A robust optimisation method applied to nurse scheduling. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature (PPSN'02)* (London, UK, 2002), J. Merelo Guervos, P. Adamidis, H.-G. Beyer, J. Fernandez-Villacanas Martin, and H.-P. Schwefel, Eds., vol. 2439 of *LNCS*, Springer-Verlag, pp. 851–860. pages 15, 41

[108] Cowling, P., Kendall, G., and Soubeiga, E. Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In *Proceedings of the Applications of Evolutionary Computing (EvoWorkshops'02)* (London, UK, 2002), S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. R. Raidl, Eds., vol. 2279 of *LNCS*, Springer-Verlag, pp. 1–10. pages 14, 15, 16, 41

[109] Cowling, P., and Soubeiga, E. Neighborhood structures for personnel scheduling: A summit meeting scheduling problem. In *Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling (PATAT'00)* (Constance, Germany, August 16–18 2000). pages 15

[110] Crama, Y., Moonen, L. S., Spieksma, F. C., and Talloen, E. The tool switching problem revisited. *European Journal of Operational Research 182*, 2 (October 2007), 952–957. pages 52

[111] Crauwels, H., and Oudheusden, D. V. Ant colony optimization and local improvement. In *The 3rd EU/ME Workshop on Real-Life Applications of Metaheuristics* (2003). pages 90

[112] Crawford, B., Montecinos, M., Castro, C., and Monfroy, E. A hyperheuristic approach to select enumeration strategies in constraint programming. In *Proceedings of the International Conference on Advances in Computing, Control, & Telecommunication Technologies (ACT'09)* (2009), IEEE, pp. 265–267. pages 15

[113] Crawford, B., Soto, R., Castro, C., and Monfroy, E. A hyperheuristic approach for dynamic enumeration strategy selection in constraint satisfaction. In *Proceedings of the 4th International Work-conference on the Interplay Between Natural and Artificial Computation: New Challenges on Bioinspired Applications (IWINAC'11)* (2011), J. Ferrandez, J. Alvarez Shez, F. de la Paz, and F. Toledo, Eds., vol. 6687 of *LNCS*, Springer, pp. 295–304. pages 15

[114] Cuesta-Cañada, A., Garrido, L., and Terashima-Marín, H. Building hyper-heuristics through ant colony optimization for the 2d bin packing problem. In *Proceedings of the 9th International Conference*

on Knowledge-Based Intelligent Information and Engineering Systems (KES'11) (2005), R. Khosla, R. J. Howlett, and L. C. Jain, Eds., vol. 3684 of *LNCS*, Springer, pp. 654–660. pages 13, 15, 19, 20

[115] Curtois, T., Ochoa, G., Hyde, M., and Vázquez-Rodríguez, J. A. A HyFlex module for the personnel scheduling problem. CS Technical Report, University of Nottingham, 2010. pages 74

[116] de Armas, J., Miranda, G., and León, C. Hyperheuristic encoding scheme for multi-objective guillotine cutting problems. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO'11)* (2011), N. Krasnogor and P. L. Lanzi, Eds., ACM, pp. 1683–1690. pages 45, 46

[117] De Givry, S., Larrosa, J., Meseguer, P., and Schiex, T. Solving Max-SAT as weighted CSP. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP'03)* (2003), vol. 2833 of *LNCS*, Springer, pp. 363–376. pages 72

[118] Deb, K., Thiele, L., Laumanns, M., and Zitzler, E. Scalable test problems for evolutionary multiobjective optimization. In *Evolutionary Multiobjective Optimization*, A. Abraham, L. Jain, R. Goldberg, L. C. Jain, and X. Wu, Eds., Advanced Information and Knowledge Processing. Springer Berlin Heidelberg, 2005, pp. 105–145. pages 46

[119] Demeester, P., Bilgin, B., De Causmaecker, P., and Vanden Berghe, G. A hyperheuristic approach to examination timetabling problems: benchmarks and a new problem from practice. *Journal of Scheduling 15*, 1 (2012), 83–103. pages 37, 40, 43, 112

[120] Demeester, P., Causmaecker, P. D., and Vanden Berghe, G. A general approach for exam timetabling: a real-world and a benchmark case. In *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT'10)* (Belfast, Northern Ireland, August 10–13 2010), pp. 486–489. pages 112, 127

[121] Demeester, P., Souffriau, W., De Causmaecker, P., and Vanden Berghe, G. A hybrid tabu search algorithm for automatically assigning patients to beds. *Artificial Intelligence in Medicine 48*, 1 (2010), 61–70. pages 65

[122] Di Gaspero, L., and Urli, T. A reinforcement learning approach for the cross-domain heuristic search challenge. In *Proceedings of the 9th Metaheuristics International Conference (MIC'11)* (2011). pages 42, 45

[123] Di Gaspero, L., and Urli, T. Evaluation of a family of reinforcement learning cross-domain optimization heuristics. In *Proceedings of the 6th Learning and Intelligent OptimizatioN Conference (LION'12)* (2012), Y. Hamadi and M. Schoenauer, Eds., vol. 7219 of *LNCS*. pages 15, 41

[124] Dorigo, M. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992. pages 2, 18

[125] Dorigo, M., and Stützle, T. *Ant colony optimization*. the MIT Press, 2004. pages 18

[126] Dowsland, K. Some experiments with simulated annealing techniques for packing problems. *European Journal of Operational Research 68*, 3 (1993), 389–399. pages 29

[127] Dowsland, K., Soubeiga, E., and Burke, E. A simulated annealing hyper-heuristic for determining shipper sizes. *European Journal of Operational Research 179*, 3 (2007), 759–774. pages 3, 13, 15, 29, 40, 45

[128] Drake, J., Özcan, E., and Burke, E. Controlling crossover in a selection hyper-heuristic framework. CS technical report no: NOTTCS-TR-SUB-1104181638-4244, University of Nottingham, 2011. pages 15, 37, 45

[129] Drake, J., Özcan, E., and Burke, E. An improved choice function heuristic selection for cross domain heuristic search. In *Proceedings of the 12th International Conference on Parallel Problem Solving from Nature (PPSN'12)* (2012), C. A. Coello Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, Eds., vol. 7492 of *LNCS*, Springer, pp. 307–316. pages 15, 41, 42, 44, 45

[130] Drake, J. H., Hyde, M., Ibrahim, K., and Özcan, E. A genetic programming hyper-heuristic for the multidimensional knapsack problem. In *Proceedings of the 11th IEEE International Conference on Cybernetic Intelligent Systems (CIS'12)* (2012), pp. 76–80. pages 13, 25, 45

[131] Dueck, G. New optimization heuristics: The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics 104*, 1 (1993), 86–92. pages 30, 31

[132] Dueck, G., and Scheuer, T. Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. *Journal of Computational Physics 90*, 1 (1990), 161–175. pages 205

[133] Easton, K., Nemhauser, G., and Trick, M. The traveling tournament problem description and benchmarks. In *Proceedings of*

the 7th International Conference on Principles and Practice of Constraint Programming (CP'01) (2001), T. Walsh, Ed., vol. 2239 of *LNCS*, Springer, pp. 580–584. pages 48, 90

[134] Easton, K., Nemhauser, G., and Trick, M. The traveling tournament problem description and benchmarks. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming (CP'01)* (2001), T. Walsh, Ed., vol. 2239 of *LNCS*, Springer, pp. 580–584. pages 52

[135] Easton, T., and Singireddy, A. A large neighborhood search heuristic for the longest common subsequence problem. *Journal of Heuristics 14*, 3 (2008), 271–283. pages 42

[136] Egeblad, J., and Pisinger, D. Heuristic approaches for the two-and three-dimensional knapsack packing problem. *Computers & Operations Research 36*, 4 (2009), 1026–1049. pages 45

[137] Eiben, A., Hinterding, R., and Michalewicz, Z. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation 3*, 2 (1999), 124–141. pages 205

[138] Ergin, F., Uyar, A., and Yayimli, A. Investigation of hyper-heuristics for designing survivable virtual topologies in optical wdm networks. In *Proceedings of the 10th European Conference on the Applications of Evolutionary Computation (EvoApplications'10)*, C. Di Chio, A. Brabazon, G. Di Caro, R. Drechsler, M. Farooq, J. Grahl, G. Greenfield, C. Prins, J. Romero, G. Squillero, E. Tarantino, A. Tettamanzi, N. Urquhart, and A. Uyar, Eds., vol. 6625 of *LNCS*. Springer, 2011, pp. 1–10. pages 13, 20, 44

[139] Ergin, F., Yayimli, A., and Uyar, A. Survivable cross-layer virtual topology design using a hyper-heuristic approach. In *Proceedings of the 13th International Conference on Transparent Optical Networks (ICTON)* (2011), IEEE, pp. 1–4. pages 13, 20, 44

[140] Ernst, A., Jiang, H., Krishnamoorthy, M., and Sier, D. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research 153*, 1 (2004), 3–27. pages 52

[141] Eveborn, P., Flisberg, P., and Ronnqvist, M. Laps care–an operational system for staff planning of home care. *European Journal of Operational Research 171*, 3 (June 2006), 962–976. pages 52

[142] Falkenauer, E. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics 2*, 1 (1996), 5–30. pages 45

[143] Fang, H., Ross, P., and Corne, D. A promising hybrid GA/heuristic approach for open-shop scheduling problems. In *Proceedings of the 11th European Conference on Artificial Intelligence (ECAI'94)* (1994), A. Cohn, Ed., John Wiley & Sons, Ltd, pp. 590–594. pages 12, 13, 41

[144] Fatima, S., and Bader-El-Den, M. Co-evolutionary hyper-heuristic method for auction based scheduling. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'10)* (Barcelona, Spain, July 18–23 2010), pp. 1489–1496. pages 13, 43

[145] Fekete, S., and Schepers, J. On more-dimensional packing iii: Exact algorithms. Technical paper zpr97-290, Mathematisches Institut, Universitat zu Koln, 1997. pages 45

[146] Feng, C., Cheng, T., and Wu, H. Optimizing the schedule of dispatching RMC trucks through genetic algorithms. *Automation in Construction 13*, 3 (2004), 327–340. pages 60

[147] Finn, R., Tate, J., Mistry, J., Coggill, P., Sammut, S., Hotz, H., Ceric, G., Forslund, K., Eddy, S., Sonnhammer, E., and Bateman, A. The pfam protein families database. *Nucleic acids research 36*, suppl 1 (2008), D281–D288. pages 42

[148] Fisher, M. Optimal solution of vehicle routing problems using minimun k-trees. *Operations Research 42*, 2 (1994), 626–642. pages 44

[149] Fleurent, C., and Ferland, J. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research 63*, 3 (1996), 437–461. pages 42

[150] Fratta, L., Gerla, M., and Kleinrock, L. The flow deviation method: An approach to store-and-forward communication network design. *Networks 3*, 2 (1973), 97–133. pages 20

[151] Fujita, K., Akagi, S., and Hirokawa, N. Hybrid approach for optimal nesting using a genetic algorithm and a local minimization algorithm. In *Proceedings of the 19th Annual ASME Design Automation Conference* (1993), vol. 65, pp. 477–484. pages 45

[152] Fujiwara, O., and Ba Khang, D. A two-phase decomposition method for optimal design of looped water distribution networks. *Water resources research 26*, 4 (1990), 539–549. pages 42

[153] Fukunaga, A. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation 16*, 1 (2008), 31–61. pages 3, 13, 26, 46

[154] Furtuna, R., Curteanu, S., and Leon, F. Multi-objective optimization of a stacked neural network using an evolutionary hyper-heuristic. *Applied Soft Computing 12*, 1 (2012), 133–144. pages 46

[155] Garrido, P., and Castro, C. Stable solving of cvrps using hyperheuristics. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation* (2009), ACM, pp. 255–262. pages 44

[156] Garrido, P., and Riff, M. Collaboration between hyperheuristics to solve strip-packing problems. In *Proceedings of the 12th International Fuzzy Systems Association World Congress (IFSA'07): Foundations of Fuzzy Logic and Soft Computing* (2007), P. Melin, O. Castillo, L. T. Aguilar, J. Kacprzyk, and W. Pedrycz, Eds., vol. 4529 of *LNCS*, Springer, pp. 698–707. pages 45

[157] Garrido, P., and Riff, M. DVRP: a hard dynamic combinatorial optimisation problem tackled byanevolutionary hyper-heuristic. *Journal of Heuristics 16*, 6 (2010), 795–834. pages 18, 44

[158] Gaspero, L., and Schaerf, A. A composite-neighborhood tabu search approach to the traveling tournament problem. *Journal of Heuristics 13*, 2 (2007), 189–207. pages 90

[159] Gibbs, J., Kendall, G., and Özcan, E. Scheduling english football fixtures over the holiday period using hyper-heuristics. In *Proceedings of the 9th Parallel Problem Solving from Nature (PPSN'10)* (Krakow, Poland, September 11–15 2010), R. Schaefer, C. Cotta, J. Kolodziej, and G. Rudolph, Eds., vol. 6238 of *LNCS*, Springer, pp. 496–505. pages 15, 35, 40, 42

[160] Glover, F. Tabu search – part i. *ORSA Journal on Computing 1*, 3 (1989), 190–206. pages 2, 21

[161] Glover, F. Tabu search – part ii. *ORSA Journal on Computing 2*, 1 (1990), 4–32. pages 2, 21

[162] Glover, F., and Laguna, M. *Modern heuristic techniques for combinatorial problems*. Springer, 1993, ch. Tabu Search, pp. 70–150. pages 40

[163] Gomez, J., and Terashima-Marín, H. Approximating multi-objective hyper-heuristics for solving 2d irregular cutting stock problems. In *Advances in Soft Computing*, G. Sidorov, A. Hernandez Aguirre, and C. Reyes Garcia, Eds., vol. 6438 of *LNCS*. Springer Berlin / Heidelberg, 2010, pp. 349–360. pages 45, 46

[164] Gottlieb, J., Marchiori, E., and Rossi, C. Evolutionary algorithms for the satisfiability problem. *Evolutionary Computation 10*, 1 (2002), 35–50. pages 46

[165] Graham, L., Forbes, D., and Smith, S. Modeling the ready mixed concrete delivery system with neural networks. *Automation in Construction 15*, 5 (2006), 656–663. pages 60

[166] Grobler, J., Engelbrecht, A., Kendall, G., and Yadavalli, S. Alternative hyper-heuristic strategies for multi-method global optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'10)* (Barcelona, Spain, July 18–23 2010), pp. 826–833. pages 14

[167] Han, L., and Kendall, G. Guided operators for a hyper-heuristic genetic algorithm. In *Proceedings of AI-2003: Advances in Artificial Intelligence. The 16th Australian Conference on Artificial Intelligence (AI'03), Tamás D Gedeon and Lance Chun Che Fung (eds.)* (Perth, Australia, December 3–5 2003), vol. 2903, pp. 807–820. pages 13, 15, 16, 41

[168] Han, L., and Kendall, G. An investigation of a tabu assisted hyper-heuristic genetic algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'03)* (2003), vol. 3, pp. 2230–2237. pages 3, 13, 15, 41

[169] Hansen, P., and Mladenović, N. Variable neighborhood search: Principles and applications. *European journal of operational research 130*, 3 (2001), 449–467. pages 34

[170] Haspeslagh, S., De Causmaecker, P., Schaerf, A., and Stølevik, M. The first international nurse rostering competition 2010. *Annals of Operations Research* (to appear). pages 47, 67, 68, 136

[171] Hauptman, A., Elyasaf, A., and Sipper, M. Evolving hyper heuristic-based solvers for rush hour and freecell. In *Proceedings of the 3rd Annual Symposium on Combinatorial Search (SOCS'10)* (2010), pp. 149–150. pages 13, 25, 26

[172] He, J., He, F., and Dong, H. Pure strategy or mixed strategy? - an initial comparison of their asymptotic convergence rate and asymptotic hitting time. In *Proceedings of the 12th European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP'12)* (2012), J.-K. Hao and M. Middendorf, Eds., vol. 7245 of *LNCS*, pp. 218–229. pages 207

[173] Hentenryck, P. V., and Vergados, Y. Population-based simulated annealing for traveling tournaments. In *Proceedings of the 21th AAAI Conference on Artificial Intelligence* (2007), vol. 22, pp. 267–272. pages 90, 91

[174] Hernández, P., Gómez, C., Cruz, L., Ochoa, A., Castillo, N., and Rivera, G. Hyperheuristic for the parameter tuning of a bio-inspired algorithm of query routing in p2p networks. In *the 10th Mexican International Conference on Artificial Intelligence (MICAI'11) - Advances in Soft Computing* (2011), I. Batyrshin and G. Sidorov, Eds., vol. 7095 of *LNCS*, Springer Berlin / Heidelberg, pp. 119–130. pages 13, 15, 44

[175] Hopper, E., and Turton, B. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research 128*, 1 (2001), 34–57. pages 45

[176] Hsiao, P., Chiang, T., and Fu, L. A VNS-based hyper-heuristic with adaptive computational budget of local search. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'12)* (2012), IEEE, pp. 1–8. pages 13, 41, 42, 45

[177] Hussin, N. *Tabu Search Based Hyper-Heuristic Approaches to Examination Timetabling*. PhD thesis, University of Nottingham, 2005. pages 13, 15

[178] Hyde, M., and Ochoa, G. Hyflex competition instance summary. Tech. rep., University of Nottingham, 2011. pages xviii, 71, 72, 73, 74, 75

[179] Hyde, M., Ochoa, G., Curtois, T., and Vázquez-Rodríguez, J. A. A HyFlex module for the one dimensional bin packing problem. CS Technical Report, University of Nottingham, 2010. pages 72

[180] Hyde, M., Özcan, E., and Burke, E. Multilevel search for evolving the acceptance criteria of a hyper-heuristic. In *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA'09)* (2009), J. Blazewicz, M. Drozdowski, G. Kendall, and B. McCollum, Eds., pp. 798–801. pages 45, 46

[181] Jiang, H., Qiu, J., and Xuan, J. A Hyper-Heuristic Using GRASP with Path-Relinking: A Case Study of the Nurse Rostering Problem. *Journal of Information Technology Research 4*, 2 (2011), 31–42. pages 13

[182] Jiang, H., Zhang, S., Xuan, J., and Wu, Y. Frequency distribution based hyper-heuristic for the bin-packing problem. In *Proceedings of the 11th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP'11)* (2011), P. Merz and J.-K. Hao, Eds., vol. 6622 of *LNCS*, Springer Berlin / Heidelberg, pp. 118–129. pages 40, 45

[183] Justesen, T., and Rasmussen, M. The home care crew scheduling problem. Master's thesis, .U. Denmark & U. of Copenhagen, 2008. pages xviii, 47, 53, 54

[184] Kalender, M., Kheiri, A., Özcan, E., and Burke, E. A greedy gradient-simulated annealing hyper-heuristic for a curriculum-based course timetabling problem. In *Proceedings of the 12th UK Workshop on Computational Intelligence (UKCI'12)* (Edinburgh, Scotland, 2012). pages 14, 15, 40, 41, 42, 43, 44, 45

[185] Kazarlis, S., Bakirtzis, A., and Petridis, V. A genetic algorithm solution to the unit commitment problem. *IEEE Transactions on Power Systems 11*, 1 (1996), 83–92. pages 36, 41

[186] Keleş, A., Uyar, A., and Yayimli, A. Solving the physical impairment aware routing and wavelength assignment problem in optical wdm networks using a tabu search based hyper-heuristic approach. In *Proceedings of the 10th European Conference on the Applications of Evolutionary Computation (EvoApplications'10)*, C. Di Chio, A. Brabazon, G. Di Caro, M. Ebner, M. Farooq, A. Fink, J. Grahl, G. Greenfield, P. Machado, M. ONeill, E. Tarantino, and N. Urquhart, Eds., vol. 6025 of *LNCS*. Springer Berlin / Heidelberg, 2010, pp. 81–90. pages 13, 15

[187] Keleş, A., Yayimli, A., and Uyar, A. Ant based hyper heuristic for physical impairment aware routing and wavelength assignment. In *Proceedings of the 33rd IEEE conference on Sarnoff* (2010), IEEE Press, pp. 90–94. pages 13, 15, 19, 44

[188] Keller, R., and Poli, R. Cost-benefit investigation of a genetic-programming hyperheuristic. In *Revised Selected Papers from the 8th International Conference on Artificial Evolution (EA'07)* (2008), N. Monmarche, E.-G. Talbi, P. Collet, M. Schoenauer, and E. Lutton, Eds., vol. 4926 of *LNCS*, Springer, pp. 13–24. pages 13

[189] Keller, R., and Poli, R. Self-adaptive hyperheuristic and greedy search. In *Proceeedings of the IEEE Congress on Evolutionary Computation (CEC'08)* (2008), IEEE, pp. 3801–3808. pages 13, 44

[190] Keller, R., and Poli, R. Subheuristic search and scalability in a hyperheuristic. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO'08)* (2008), C. Ryan and M. Keijzer, Eds., ACM, pp. 609–610. pages 13, 44

[191] Kendall, G., and Hussin, N. A tabu search hyper-heuristic approach to the examination timetabling problem at the mara university of technology.

In *Proceedings of the 5th Practice and Theory of Automated Timetabling (PATAT'04)* (2005), E. Burke and M. Trick, Eds., vol. 3616 of *LNCS*, Springer, pp. 270–293. pages 13, 15, 43

[192] Kendall, G., and Li, J. Competitive travelling salesmen problem: A hyper-heuristic approach. *Journal of the Operational Research Society* (to appear). pages 44

[193] Kendall, G., and Mohamad, M. Channel assignment in cellular communication using a great deluge hyper-heuristic. In *Proceedings of the 12th IEEE International Conference on Network (ICON'04)* (2004), pp. 769–773. pages 14, 30, 40, 42

[194] Kendall, G., and Mohamad, M. Channel assignment optimisation using a hyper-heuristic. In *Proceedings of the 2004 IEEE Conference on Cybernetics and Intelligent Systems (CIS'04)* (Singapore, December 1–3 2004), pp. 790–795. pages 14, 31, 40, 42

[195] Khamassi, I. Ant-Q hyper heuristic approach applied to the cross-domain heuristic search challenge problems. CHeSC'2011 report, 2011. pages 13

[196] Khamassi, I., Hammami, M., and Ghédira, K. Ant-Q hyper-heuristic approach for solving 2-dimensional cutting stock problem. In *Proceedings of the IEEE Symposium on Swarm Intelligence (SIS'11)* (2011), IEEE, pp. 199–205. pages 13, 15, 20

[197] Kilby, P., Prosser, P., and Shaw, P. Dynamic VRPs: A study of scenarios. Report APES-06, University of Strathclyde, 1998. pages 44

[198] Kiraz, B., and Topçuoğlu, H. Hyper-heuristic approaches for the dynamic generalized assignment problem. In *Proceedings of the 10th International Conference on Intelligent Systems Design and Applications (ISDA'10)* (2010), pp. 1487–1492. pages 14, 15, 35, 40

[199] Kiraz, B., Uyar, A., and Özcan, E. An investigation of selection hyper-heuristics in dynamic environments. In *Proceedings of the 11th European Conference on the Applications of Evolutionary Computation (EvoApplications'11)* (2011), C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ek, A. Esparcia-Alcr, J. Merelo, F. Neri, M. Preuss, H. Richter, J. Togelius, and G. Yannakakis, Eds., vol. 6624 of *LNCS*, Springer, pp. 314–323. pages 15

[200] Kirkpatrick, S., Gelatt, C., and Vecchi, M. Optimization by simulated annealing. *Science 220* (1983), 671–680. pages 2, 28

[201] Kohl, N., and Madsen, O. An optimization algorithm for the vehicle routing problem with time windows based on lagrangian relaxation. *Operations Research 45*, 3 (1997), 395–406. pages 75

[202] Koohestani, B., and Poli, R. A hyper-heuristic approach to evolving algorithms for bandwidth reduction based on genetic programming. In *Proceedings of the 31st SGAI International Conference (AI'11)* (Cambridge, UK, December 13–15 2011). pages 13, 42

[203] Kovacs, A. A., Parragh, S., Doerner, K. F., and Hartl, R. F. Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of Scheduling 15*, 5 (2012), 579–600. pages 53

[204] Krumke, S. O., Rambau, J., and Torres, L. M. Online-dispatching of automobile service units. Tech. Rep. 02-44, ZIB, Takustr.7, 14195 Berlin, 2002. pages 53

[205] Kubalík, J. Hyper-heuristic based on iterated local search driven by evolutionary algorithm. In *European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP'12)*, J.-K. Hao and M. Middendorf, Eds., vol. 7245 of *LNCS*. Springer Berlin / Heidelberg, 2012, pp. 148–159. pages 41, 42, 44, 45

[206] Kumar, R., and Rockett, P. Improved sampling of the pareto-front in multiobjective genetic optimizations by steady-state evolution: a pareto converging genetic algorithm. *Evolutionary Computation 10*, 3 (2002), 283–314. pages 27

[207] Langford. In challenging travelling tournament instances http://mat.gsia.cmu.edu/tourn/. pages 90

[208] Leake, D. *CBR in Context: The Present and Future*. AAAI Press/MIT Press, California, 1996. pages 22

[209] Lee, D., Vassiliadis, V., and Park, J. List-based threshold-accepting algorithm for zero-wait scheduling of multiproduct batch plants. *Industrial & Engineering Chemistry Research 41*, 25 (2002), 6579–6588. pages 110

[210] Lee, J., Lee, Y., and Lee, Y. Mathematical modeling and tabu search heuristic for the traveling tournament problem. In *Proceedings of the 6th International Conference on Computational Science and Its Applications (ICCSA'06)* (2006), vol. 3982 of *LNCS*, Springer, pp. 875–884. pages 48

[211] Lehre, P. K., and Özcan, E. A time-complexity analysis of hyper-heuristics. In *Proceedings of the the 25th Conference of European Chapter*

on Combinatorial Optimization (ECCO'12) (Antalya, Turkey, 2012). pages 207

[212] León, C., Miranda, G., and Segura, C. A memetic algorithm and a parallel hyperheuristic island-based model for a 2D packing problem. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO'09)* (2009), F. Rothlauf, Ed., ACM, pp. 1371–1378. pages 17, 45

[213] Li, J., Burke, E., and Qu, R. Integrating neural networks and logistic regression to underpin hyper-heuristic search. *Knowledge-Based Systems 24*, 2 (2010), 322–330. pages 43

[214] Lim, A., Rodrigues, B., and Zhang, X. A simulated annealing and hill-climbing algorithm for the traveling tournament problem. *European Journal of Operational Research 174*, 3 (2006), 1459–1478. pages 90

[215] López-Camacho, E., Terashima-Marín, H., and Ross, P. A hyper-heuristic for solving one and two-dimensional bin packing problems. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO'11)* (New York, NY, USA, 2011), N. Krasnogor and P. Lanzi, Eds., ACM, pp. 257–258. pages 45

[216] Lu, Z., and Hao, J.-K. Adaptive neighborhood search for nurse rostering. *European Journal of Operational Research 218*, 3 (2012), 865–876. pages xx, 142

[217] M. Bader-El-Den and R. Poli. Generating SAT local-search heuristics using a GP hyper-heuristic framework. In *Proceedings of the 8th International Conference on Artificial Evolution (EA'07)* (2008), N. Monmarche, E.-G. Talbi, P. Collet, M. Schoenauer, and E. Lutton, Eds., vol. 4926 of *LNCS*, Springer, pp. 37–49. pages 13, 25, 46

[218] Maden, I., Uyar, A., and Özcan, E. Landscape analysis of simple perturbative hyper-heuristics. In *Proceedings of the 15th International Conference on Soft Computing (Mendel)* (2009). pages 14, 40, 207

[219] Marín-Blázquez, J., and Schulenburg, S. A hyper-heuristic framework with XCS: Learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients. In *Selected Papers from the International Workshops on Learning Classifier Systems (IWLCS'03-05)* (2007), T. Kovacs, X. Llor. Takadama, P. Lanzi, W. Stolzmann, and S. Wilson, Eds., vol. 4399 of *LNCS*, Springer, pp. 193–218. pages 2, 15

[220] Matsatsinis, N. Towards a decision support system for the ready concrete distribution system: A case of a Greek company. *European Journal of Operational Research 152*, 2 (2004), 487–499. pages 60

[221] McClymont, K., and Keedwell, E. Markov chain hyper-heuristic (MCHH): an online selective hyper-heuristic for multi-objective continuous problems. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation (GECCO'11)* (2011), N. Krasnogor and P. Lanzi, Eds., pp. 2003–2010. pages 46

[222] Mcmullan, P. An extended implementation of the great deluge algorithm for course timetabling. In *Proceedings of the 7th International Conference on Computational Science (ICCS'07)* (2007), Y. Shi, G. Van Albada, J. Dongarra, and P. Sloot, Eds., vol. 4487 of *LNCS*, Springer, pp. 538–545. pages 30, 40

[223] Meignan, D., Koukam, A., and Créput, J.-C. Coalition-based metaheuristic: a self-adaptive metaheuristic using reinforcement learning and mimetism. *Journal of Heuristics 16*, 6 (2010), 859–879. pages 44

[224] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. Equation of state calculations by fast computing machines. *Journal of Chemical Physics 21*, 6 (1953), 1087–1092. pages 28

[225] Misir, M. Group decision making for move acceptance in hyperheuristics. Master's thesis, Yeditepe University, 2008. pages 40, 43

[226] Misir, M., Bilgin, B., Demeester, P., Verbeeck, K., De Causmaecker, P., and Vanden Berghe, G. A hyperheuristic approach to the patient admission scheduling problem. In *Proceedings of the 35th International Conference of Operational Research Applied to Health Services (ORAHS'09)* (Leuven, Belgium, July 12–17 2009). pages 6, 124

[227] Misir, M., Smet, P., and Vanden Berghe, G. An analysis of generalised heuristics for vehicle routing and personnel rostering problems. *under review*. pages xiii, 3, 6, 7

[228] Misir, M., Smet, P., Verbeeck, K., and Vanden Berghe, G. Security personnel routing and rostering: a hyper-heuristic approach. In *Proceedings of the 3rd International Conference on Applied Operational Research (ICAOR'11)* (Istanbul, Turkey, August 24–26 2011), vol. 3 of *LNMS*, pp. 193–205. pages 6, 7, 56, 158

[229] Misir, M., Vancroonenburg, W., and Vanden Berghe, G. A selection hyper-heuristic for scheduling deliveries of ready-mixed concrete. In *Proceedings of the 9th Metaheuristic International Conference (MIC'11)* (Udine, Italy, July 25–28 2011). pages 6, 7

[230] Misir, M., Verbeeck, K., De Causmaecker, P., and Vanden Berghe, G. Design and analysis of an evolutionary selection hyper-heuristic framework. *under review*. pages 6, 7

[231] Misir, M., Verbeeck, K., De Causmaecker, P., and Vanden Berghe, G. An investigation on the generality level of selection hyper-heuristics under different empirical conditions. *under review*. pages 7

[232] Misir, M., Verbeeck, K., De Causmaecker, P., and Vanden Berghe, G. Hyper-heuristics with a dynamic heuristic set for the home care scheduling problem. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'10)* (Barcelona, Spain, July 18–23 2010), pp. 2875–2882. pages xx, 6, 7, 52, 111, 123, 135, 165

[233] Misir, M., Verbeeck, K., De Causmaecker, P., and Vanden Berghe, G. A new hyper-heuristic implementation in HyFlex: a study on generality. In *Proceedings of the 5th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA'11)* (Phoenix/Arizona, USA, August 10–12 2011), J. Fowler, G. Kendall, and B. McCollum, Eds., pp. 374–393. pages 6, 7, 124, 125, 126, 127, 158, 165

[234] Misir, M., Verbeeck, K., De Causmaecker, P., and Vanden Berghe, G. Design principles and performance analysis of a selection hyper-heuristic across multiple problem domains. In *Proceedings of the 26th Belgian Conference on Operations Research (ORBEL'12)* (2012). pages 7

[235] Misir, M., Verbeeck, K., De Causmaecker, P., and Vanden Berghe, G. An intelligent hyper-heuristic framework for CHeSC 2011. In *Proceedings of the 6th Learning and Intelligent OptimizatioN Conference (LION'12)* (2012), Y. Hamadi and M. Schoenauer, Eds., vol. 7219 of *LNCS*, Springer. pages 6, 7, 8, 124, 125, 126

[236] Misir, M., Verbeeck, K., De Causmaecker, P., and Vanden Berghe, G. A new hyper-heuristic as a general problem solver: an implementation in HyFlex. *Journal of Scheduling* (to appear). pages xxi, 6, 7, 175

[237] Misir, M., Verbeeck, K., Vanden Berghe, G., and De Causmaecker, P. A hyper-heuristic approach to the home care scheduling problem. In *Proceedings of the 14th Belgian-French-German Conference on Optimization (BFG'09)* (2009), p. 175. pages 6, 7

[238] Misir, M., Verbeeck, K., Vanden Berghe, G., and De Causmaecker, P. A hyper-heuristic approach to the home care scheduling problem. Tech. rep., KaHo Sint-Lieven, 2009. pages 104

[239] Misir, M., Verbeeck, K., Vanden Berghe, G., and De Causmaecker, P. Design of a generic selection hyper-heuristic. In *Proceedings of the 25th Belgian Conference on Operations Research (ORBEL'11)* (2011). pages 7

[240] Misir, M., Wauters, T., Verbeeck, K., and Vanden Berghe, G. A new learning hyper-heuristic for the traveling tournament problem. In *Proceedings of the 8th Metaheuristic International Conference (MIC'09)* (Hamburg, Germany, July 13–16 2009). pages 6, 7, 111

[241] Misir, M., Wauters, T., Verbeeck, K., and Vanden Berghe, G. A hyper-heuristic with learning automata for the traveling tournament problem. In *Metaheuristics: Intelligent Decision Making, the 8th Metaheuristics International Conference - Post Conference Volume*. to appear. pages 6, 7, 165

[242] Mladenovic, N., and Hansen, P. Variable neighborhood search. *Computers & OR 24*, 11 (1997), 1097–1100. pages 23

[243] Moscato, P. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Report 826, Caltech Concurrent Computation Program, 1989. pages 12

[244] Narendra, K., and Thathachar, M. *Learning automata: an introduction*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989. pages 79

[245] Nareyek, A. *Metaheuristics: Computer Decision-Making*. Kluwer Academic Publishers, 2003, ch. Choosing search heuristics by non-stationary reinforcement learning, pp. 523–544. pages 2, 15, 23, 35

[246] Naso, D., Surico, M., Turchiano, B., and Kaymak, U. Just-in-time production and delivery in supply chains: a hybrid evolutionary approach. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC'05)* (2005), vol. 2, IEEE, pp. 1932–1937. pages 60

[247] Naso, D., Surico, M., Turchiano, B., and Kaymak, U. Genetic algorithms for supply-chain scheduling: A case study in the distribution of ready-mixed concrete. *European Journal of Operational Research 177*, 3 (2007), 2069–2099. pages 60

[248] Nawaz, M., Enscore Jr, E., and Ham, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega 11*, 1 (1983), 91–95. pages 32

[249] Neufeld, G., and Tartar, J. Graph coloring conditions for the existence of solutions to the timetable problem. *Communications of the ACM 17*, 8 (1974), 450–453. pages 193

[250] Nguyen, S., Zhang, M., and Johnston, M. A genetic programming based hyper-heuristic approach for combinatorial optimisation. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO'11)* (New York, USA, 2011), N. Krasnogor and P. Lanzi, Eds., ACM, pp. 1299–1306. pages 13, 15, 25, 41, 42, 45

[251] Ning, K. Deposition and extension approach to find longest common subsequence for thousands of long sequences. *Computational Biology and Chemistry 34*, 3 (2010), 149–157. pages 42

[252] Nunez, J., and Ceballos, A. A general purpose hyper–heuristic based on ant colony optimization. CHeSC'2011 report, 2011. pages 13

[253] Obit, J., Landa-Silva, D., Sevaux, M., and Ouelhadj, D. Non-linear great deluge with reinforcement learning for university course timetabling. In *Metaheuristics: Intelligent Decision Making, the 8th Metaheuristic International Conference - Post Conference Volume*. to appear. pages 15, 30, 40, 43

[254] O'Brien, R. Ant algorithm hyperheuristic approaches for scheduling problems. Master's thesis, University of Nottingham, 2007. pages 13

[255] Ochoa, G., Hyde, M., Curtois, T., Vázquez-Rodríguez, J., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A., Petrovic, S., and Burke, E. Hyflex: A benchmark framework for cross-domain heuristic search. In *Proceedings of the 12th European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP'12)* (2012), J.-K. Hao and M. Middendorf, Eds., vol. 7245 of *LNCS*, Springer, pp. 136–147. pages 47, 71, 154, 169

[256] Ochoa, G., Qu, R., and Burke, E. Analyzing the landscape of a graph based hyper-heuristic for timetabling problems. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO'09)* (2009), F. Rothlauf, Ed., ACM, pp. 341–348. pages 207

[257] Ochoa, G., Vázquez-Rodríguez, J., Petrovic, S., and Burke, E. Dispatching rules for production scheduling: a hyper-heuristic landscape analysis. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'09)* (2009), pp. 1873–1880. pages 13, 18, 41, 207

[258] Ochoa, G., Walker, J., Hyde, M., and Curtois, T. Adaptive evolutionary algorithms and extensions to the HyFlex hyper-heuristic

framework. In *Proceedings of the 12th International Conference on Parallel Problem Solving from Nature (PPSN'12)* (2012), C. A. Coello Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, Eds., vol. 7492 of *LNCS*, Springer, pp. 418–427. pages 13, 15, 44

[259] Osman, I., and Kelly, J. *Meta-heuristics: theory & applications*. Springer, 1996. pages 1

[260] Ouelhadj, D., and Petrovic, S. A cooperative distributed hyper-heuristic framework for scheduling. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC'08)* (Singapore, October 12–15 2008), pp. 2560–2565. pages 14, 32, 40, 41

[261] Ouelhadj, D., and Petrovic, S. A cooperative hyper-heuristic search framework. *Journal of Heuristics 16*, 6 (2009), 835–857. pages 14, 32, 41

[262] Ouelhadj, D., Petrovic, S., and Özcan, E. A multi-level search framework for asynchronous cooperation of multiple hyper-heuristics. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference (GECCO'09): Late Breaking Papers* (2009), F. Rothlauf, Ed., ACM, pp. 2193–2196. pages 14, 33, 41

[263] Özcan, E., Bilgin, B., and Korkmaz, E. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis 12*, 1 (2008), 3–23. pages 14, 15, 24, 25, 35, 38, 40, 125

[264] Özcan, E., Bykov, Y., Birben, M., and Burke, E. Examination timetabling using late acceptance hyper-heuristics. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'09)* (2009), pp. 997–1004. pages 14, 15, 31, 40, 43, 110

[265] Özcan, E., and Kheiri, A. A hyper-heuristic based on random gradient, greedy and dominance. In *Proceedings of the 26th International Symposium on Computer and Information Sciences (ISCIS'11)* (2011), E. Gelenbe, R. Lent, and G. Sakellari, Eds., Computer and Information Sciences II, pp. 557–563. pages 14, 40, 41, 42, 45

[266] Özcan, E., Misir, M., and Burke, E. A self-organising hyper-heuristic framework. In *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA'09)* (Dublin, Ireland, August 10–12 2009), J. Blazewicz, M. Drozdowski, G. Kendall, and B. McCollum, Eds., pp. 784–787. pages 24, 25, 100

[267] Özcan, E., Misir, M., Ochoa, G., and Burke, E. A reinforcement learning - great-deluge hyper-heuristic for examination timetabling. *International Journal of Applied Metaheuristic Computing 1*, 1 (2010), 39–59. pages 2, 15, 23, 24, 40, 43

[268] Özcan, E., and Parkes, A. Policy matrix evolution for generation of heuristics. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO'11)* (2011), N. Krasnogor and P. Lanzi, Eds., ACM, pp. 2011–2018. pages 45

[269] Özcan, E., Uyar, Ş., and Burke, E. A greedy hyper-heuristic in dynamic environments. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference (GECCO'09): Late Breaking Papers* (Montreal, Canada, July 8–12 2009), F. Rothlauf, Ed., ACM, pp. 2201–2204. pages 25, 40

[270] Pérez, L., and Riff, M. New bounds for the relaxed traveling tournament problems using an artificial immune algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'11)* (2011), IEEE, pp. 873–879. pages 42

[271] Petrovic, S., and Qu, R. Case-based reasoning as a heuristic selector in a hyper-heuristic for course timetabling problems. In *Proceedings of the 6th International Conference on Knowledge-Based Intelligent Information & Engineering Systems (KES'02)* (Crema, Italy, September 2002), pp. 336–340. pages 15, 22, 43

[272] Pillay, N. An empirical study into the structure of heuristic combinations in an evolutionary algorithm hyper-heuristic for the examination timetabling problem. In *Proceedings of the 2010 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists* (2010), ACM, pp. 251–257. pages 25, 26, 43

[273] Pillay, N. A study into the use of hyper-heuristics to solve the school timetabling problem. In *Proceedings of the 2010 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists* (2010), ACM, pp. 258–264. pages 43

[274] Pillay, N. Evolving hyper-heuristics for the uncapacitated examination timetabling problem. *Journal of the Operational Research Society 63*, 1 (2012), 47–58. pages 43

[275] Pillay, N., and Banzhaf, W. A genetic programming approach to the generation of hyper-heuristics for the uncapacitated examination timetabling problem. In *Proceedings of the 13th Portuguese conference on Progress in Artificial Intelligence* (2007), Springer-Verlag, pp. 223–234. pages 27, 43

[276] Pillay, N., and Banzhaf, W. A study of heuristic combinations for hyper-heuristic systems for the uncapacitated examination timetabling

problem. *European Journal of Operational Research 197*, 2 (2009), 482–491. pages 43

[277] Poli, R., and Graff, M. There is a free lunch for hyper-heuristics, genetic programming and computer scientists. In *Proceedings of the 12th European Conference on Genetic Programming (EuroGP'09)* (2009), L. Vanneschi, S. Gustafson, A. Moraglio, I. De Falco, and M. Ebner, Eds., vol. 5481 of *LNCS*, Springer, pp. 195–207. pages 204

[278] Potvin, J.-Y., and Rousseau, J.-M. An exchange heuristic for routeing problems with time windows. *The Journal of the Operational Research Society 46*, 12 (1995), 1433–1446. pages 195

[279] Qu, R., and Burke, E. Hybrid variable neighbourhood hyperheuristics for exam timetabling problems. In *Proceedings of the 6th Metaheuristic International Conference (MIC'05)* (Vienna, Austria, August 22–26 2005). pages 13, 23, 43

[280] Qu, R., and Burke, E. Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems. *Journal of the Operational Research Society 60*, 9 (2009), 1273–1285. pages 13, 15, 43

[281] Qu, R., Burke, E., and McCollum, B. Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems. *European Journal of Operational Research 198*, 2 (2009), 392–404. pages 43

[282] Raad, D., Sinske, A., and van Vuuren, J. Multiobjective optimization for water distribution system design using a hyperheuristic. *Journal of Water Resources Planning and Management 136*, 5 (2010), 592–596. pages 42, 46

[283] Rattadilok, P., Gaw, A., and Kwan, R. A distributed hyper-heuristic for scheduling. Research report 2004.01, University of Leeds, School of Computing, 2004. pages 15, 32, 43

[284] Rattadilok, P., Gaw, A., and Kwan, R. Distributed choice function hyper-heuristics for timetabling and scheduling. In *Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling (PATAT'04)* (2005), E. Burke and M. Trick, Eds., vol. 3616 of *LNCS*, Springer-Verlag, pp. 51–67. pages 15, 32, 43

[285] Reinelt, G. Tspliba traveling salesman problem library. *ORSA Journal on Computing 3*, 4 (1991), 376–384. pages 42

[286] Remde, S., Cowling, P., Dahal, K., and Colledge, N. Exact/heuristic hybrids using rvns and hyperheuristics for workforce scheduling. In *Proceedings of the 7th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP'07)* (2007), C. Cotta and J. van Hemert, Eds., vol. 4446 of *LNCS*, Springer, pp. 188–197. pages 13, 14, 34, 35, 40, 41

[287] Remde, S., Cowling, P., Dahal, K., Colledge, N., and Selensky, E. An empirical study of hyperheuristics for managing very large sets of low level heuristics. *Journal of the Operational Research Society 63*, 3 (2012), 392–405. pages 15, 35, 40, 41

[288] Remde, S., Dahal, K., Cowling, P., and Colledge, N. Binary exponential back off for tabu tenure in hyperheuristics. In *Proceedings of the 9th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP'09)* (2009), C. Cotta and P. I. Cowling, Eds., vol. 5482 of *LNCS*, Springer, pp. 109–120. pages 13, 15, 34, 35, 41

[289] Ren, Z., Jiang, H., Xuan, J., and Luo, Z. Ant based hyper heuristics with space reduction: a case study of the p-median problem. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature (PPSN'11)* (2011), R. Schaefer, C. Cotta, J. Kolodziej, and G. Rudolph, Eds., vol. 6238 of *LNCS*, Springer, pp. 546–555. pages 13, 15, 19, 40, 42

[290] Rice, J. The algorithm selection problem. *Advances in computers 15* (1976), 65–118. pages 2

[291] Ross, P., Hart, E., Marín-Blázquez, J., and Schulenberg, S. Learning a procedure that can solve hard bin-packing problems: a new GA-based approach to hyperheuristics. In *Proceedings of the 5th Annual Conference on Genetic and Evolutionary Computation (GECCO'03)* (Chicago, Illinois, USA, July 12–16 2003), E. Cantu-Paz, J. Foster, K. Deb, L. Davis, R. Roy, U.-M. O'Reilly, H.-G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M. Potter, A. Schultz, K. Dowsland, N. Jonoska, and J. Miller, Eds., vol. 2724 of *LNCS*, pp. 1295–1306. pages 13, 15, 16, 17, 45

[292] Ross, P., Marín-Blázquez, J., and Hart, E. Hyper-heuristics applied to class and exam timetabling problems. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'04)* (Portland, Oregon, 2004), IEEE Press, pp. 1691–1698. pages 43

[293] Ross, P., Schulenburg, S., Marín-Blázquez, J., and Hart, E. Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation (GECCO'02)* (New York, USA, July 9–13 2002), W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, Eds., Morgan Kaufmann Publishers, pp. 942–948. pages 2, 15, 16, 45

[294] Sabar, N., Ayob, M., Qu, R., and Kendall, G. A graph coloring constructive hyper-heuristic for examination timetabling problems. *Applied Intelligence 37*, 1 (2012), 1–11. pages 43

[295] Schaake, J., and Lai, D. *Linear programming and dynamic programming application to water distribution network design.* Hydrodynamics Laboratory, Department of Civil Engineering, Massachusetts Institute of Technology, 1969. pages 42

[296] Schmid, V., Doerner, K., Hartl, R., and Salazar-González, J. Hybridization of very large neighborhood search for ready-mixed concrete delivery problems. *Computers & Operations Research 37*, 3 (2010), 559–574. pages 60

[297] Schmid, V., Doerner, K., Hartl, R., Savelsbergh, M., and Stoecher, W. A hybrid solution approach for ready-mixed concrete delivery. *Transportation Science 43*, 1 (2009), 70. pages 60

[298] Schoenfield, J. Fast, exact solution of open bin packing problems without linear programming, *Draft, US Army Space & Missile Defense Command*, 2002. pages 45

[299] Scholl, A., Klein, R., and Jürgens, C. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research 24*, 7 (1997), 627–645. pages 45

[300] Schulenburg, S., Ross, P., Marín-Blázquez, J., and Hart, E. A hyper-heuristic approach to single and multiple step environments in binpacking problems. In *Proceedings of the 5th International Workshop on Learning Classifier Systems (IWLCS'02)* (Granada, Spain, September 7–8 2003), P. Lanzi, W. Stolzmann, and S. Wilson, Eds., LNAI, Springer-Verlag, pp. 193–218. pages 16

[301] Shen, H., and Zhang, H. Greedy big steps as a meta-heuristic for combinatorial search. The university of Iowa AR reading group, 2004. pages 90

[302] Shyu, S., and Tsai, C. Finding the longest common subsequence for multiple biological sequences by ant colony optimization. *Computers & Operations Research 36*, 1 (2009), 73–91. pages 42

[303] Simpson, A., Dandy, G., and Murphy, L. Genetic algorithms compared to other techniques for pipe optimization. *Journal of Water Resources Planning and Management 120*, 4 (1994), 423–443. pages 42

[304] Sin, E. Reinforcement learning with egd based hyper heuristic system for exam timetabling problem. In *the IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS'2011)* (2011), pp. 462–466. pages 15, 30, 40, 43

[305] Sin, E., and Kham, N. Hyper heuristic based on great deluge and its variants for exam timetabling problem. *International Journal of Artificial Intelligence & Applications 3*, 1 (2012), 149–162. pages 15, 30, 40, 43

[306] Smet, P., Misir, M., and Vanden Berghe, G. An introduction to new application domains for the home care scheduling problem. In *Proceedings of the 25th Belgian Conference on Operations Research (ORBEL'11)* (2011). pages 7

[307] Smith, T., Husbands, P., Layzell, P., and O'Shea, M. Fitness landscapes and evolvability. *Evolutionary Computation 10*, 1 (2002), 1–34. pages 83

[308] Socha, K., Knowles, J., and Sampels, M. A MAX-MIN ant system for the university course timetabling problem. In *Proceedings of the 3rd International Workshop on Ant Algorithms (ANTS'02)* (2002), M. Dorigo, G. Di Caro, and M. Sampels, Eds., vol. 2463 of *LNCS*, Springer, pp. 1–13. pages 21, 29, 43

[309] Solomon, M. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research 35*, 2 (1987), 254–265. pages 44

[310] Soria-Alcaraz, J., Carpio-Valadez, J., and Terashima-Marín, H. Academic timetabling design using hyper-heuristics. In *Soft Computing for Intelligent Control and Mobile Robotics*, O. Castillo, J. Kacprzyk, and W. Pedrycz, Eds., vol. 318 of *Studies in Computational Intelligence*. Springer Berlin / Heidelberg, 2011, pp. 43–56. pages 43

[311] Sotelo-Figueroa, M., Soberanes, H., Carpio, J., Fraire Huacuja, H., Reyes, L., and Soria Alcaraz, J. Evolving bin packing heuristic using micro-differential evolution with indirect representation. In *Recent Advances on Hybrid Intelligent Systems*, O. Castillo, P. Melin,

and J. Kacprzyk, Eds., vol. 451 of *Studies in Computational Intelligence*. Springer Berlin / Heidelberg, 2013, pp. 349–359. pages 25, 45

[312] Swan, J., Özcan, E., and Kendall, G. Hyperion–a recursive hyper-heuristic framework. In *Proceedings of the 5th Learning and Intelligent OptimizatioN Conference (LION'11)* (2011), C. Coello Coello, Ed., vol. 6683 of *LNCS*, Springer, pp. 616–630. pages 46

[313] Tabataba, F., and Mousavi, S. A hyper-heuristic for the longest common subsequence problem. *Computational Biology and Chemistry 36* (2012), 42–54. pages 42

[314] Taillard, E. Benchmarks for basic scheduling problems. *European Journal of Operations Research 64*, 2 (1993), 278–285. pages 41

[315] Taillard, E. Parallel iterative search methods for vehicle routing problem. *Networks 23*, 8 (1993), 661–673. pages 44

[316] Talbi, E. A taxonomy of hybrid metaheuristics. *Journal of Heuristics 8*, 5 (2002), 541–564. pages 162

[317] Terashima-Marín, H., Farías-Zárate, C., P, R., and Valenzuela-Rendón, M. A GA-based method to produce generalized hyper-heuristics for the 2D-regular cutting stock problem. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO'06)* (New York, USA, 2007), M. Cattolico, Ed., ACM Press, pp. 591–598. pages 17

[318] Terashima-Marín, H., Moran-Saavedra, A., and Ross, P. Forming hyper-heuristics with GAs when solving 2D-regular cutting stock problems. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'05)* (2005), vol. 2, pp. 1104–1110. pages 17, 20

[319] Terashima-Marín, H., P, R., Farías-Zárate, C., López-Camacho, E., and Valenzuela-Rendón, M. Generalized hyper-heuristics for solving 2D regular and irregular bin packing problems. *Annals of Operations Research 179*, 1 (2010), 369–392. pages 45

[320] Terashima-Marín, H., Ross, P., and Valenzuela-Rendón, M. Evolution of constraint satisfaction strategies in examination timetabling. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation Conference (GECCO'99)* (1999), W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honavar, M. Jakiela, and R. Smith, Eds., pp. 635–642. pages 43

[321] Terrazas, G., Landa-Silva, D., and Krasnogor, N. Discovering beneficial cooperative structures for the automated construction of

heuristics. In *Nature Inspired Cooperative Strategies for Optimization (NICSO'10)*, J. R. Gonzalez, D. A. Pelta, C. Cruz, G. Terrazas, and N. Krasnogor, Eds., vol. 284 of *Studies in Computational Intelligence*. Springer, 2010, pp. 89–100. pages 44

[322] Terrazas, G., Landa-Silva, D., and Krasnogor, N. Towards the design of heuristics by means of self-assembly. In *Electronic Proceedings in Theoretical Computer Science (EPTCS'10), the Workshop on Developments in Computational Models (DCM'10)* (2010), vol. 26, pp. 135–146. pages 44

[323] Thathachar, M., and Sastry, P. *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Kluwer Academic Publishers, 2004. pages xiii, 80, 91

[324] Tolay, P., and Kumar, R. Evolution of hyperheuristics for the biobjective graph coloring problem using multiobjective genetic programming. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO'09)* (2009), F. Rothlauf, Ed., ACM, pp. 1939–1940. pages 13, 25, 27, 42, 46

[325] Vancroonenburg, W., Misir, M., Bilgin, B., Demeester, P., and Vanden Berghe, G. A hyper-heuristic approach for assigning patients to hospital rooms. In *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT'10)* (Belfast, Northern Ireland, August 10–13 2010), pp. 553–555. pages 6, 7, 41

[326] Vancroonenburg, W., Misir, M., and Vanden Berghe, G. A hyper-heuristic approach for the ready-mixed concrete delivery problem. In *Proceedings of the 25th Belgian Conference on Operations Research (ORBEL'11)* (2011). pages 7

[327] Vázquez-Rodríguez, J., and Ochoa, G. On the automatic discovery of variants of the NEH procedure for flow shop scheduling using genetic programming. *Journal of the Operational Research Society 62*, 2 (2011), 381–396. pages 41

[328] Vázquez-Rodríguez, J., and Petrovic, S. A new dispatching rule based genetic algorithm for the multi-objective job shop problem. *Journal of Heuristics 16*, 6 (2010), 771–793. pages 41, 46

[329] Vázquez-Rodríguez, J., and Salhi, A. *Evolutionary Scheduling*. Springer Berlin / Heidelberg, 2007, ch. A Robust Meta-Hyper-Heuristic Approach to Hybrid Flow-Shop Scheduling, pp. 125–142. pages 41

[330] Verstichel, J., and Vanden Berghe, G. A late acceptance algorithm for the lock scheduling problem. In *Logistik Management*, S. Voss, J. Pahl, and S. Schwarze, Eds. Physica-Verlag HD, 2009, pp. 457–478. pages 52

[331] Walker, J., Ochoa, G., Gendreau, M., and Burke, E. Vehicle routing and adaptive iterated local search within the HyFlex hyper-heuristic framework. In *Proceedings of the 6th Learning and Intelligent OptimizatioN Conference (LION'12)* (2012), Y. Hamadi and M. Schoenauer, Eds., vol. 7219 of *LNCS*, Springer. pages 13, 15, 44

[332] Wang, P. Two algorithms for constrained two-dimensional cutting stock problems. *Operations Research 31*, 3 (1983), 573–586. pages 45

[333] Wang, P., and Valenzela, C. Data set generation for rectangular placement problems. *European Journal of Operational Research 134*, 2 (2001), 378–391. pages 45

[334] Wang, Q., Pan, M., Shang, Y., and Korkin, D. A fast heuristic search algorithm for finding the longest common subsequence of multiple strings. In *Twenty-Fourth AAAI Conference on Artificial Intelligence* (2010). pages 42

[335] Wäscher, G., and Gau, T. Heuristics for the integer one-dimensional cutting stock problem: a computational study. *OR Spectrum 18*, 3 (1996), 131–144. pages 45

[336] Wauters, T., Vancroonenburg, W., and Vanden Berghe, G. A guide-and-observe hyper-heuristic approach to the eternity II puzzle. *Journal of Mathematical Modelling and Algorithms* (to appear). pages 14, 40

[337] Whitley, D. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the 3rd International Conference on Genetic Algorithms (ICGA'89)* (1989), vol. 1, pp. 116–123. pages 27

[338] Willemse, E. J., and Joubert, J. W. Applying min-max k postmen problems to the routing of security guards. *Journal of the Operational Research Society 63*, 2 (2012), 245–260. pages 52

[339] Wolpert, D., and Macready, W. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation 1*, 1 (1997), 67–82. pages 203

[340] Yang, Y., and Petrovic, S. A novel similarity measure for heuristic selection in examination timetabling. In *Revised Selected Papers from*

the 5th International Conference on Practice and Theory of Automated Timetabling (PATAT'04) (2005), E. Burke and M. Trick, Eds., vol. 3616 of *LNCS*, Springer Berlin / Heidelberg, pp. 247–269. pages 43

[341] Zhang. In challenging travelling tournament instances http://mat.gsia.cmu.edu/tourn/. pages 90

Publications List

Articles *submitted* to internationally reviewed academic journals

- M. Mısır, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, Design and Analysis of an Evolutionary Selection Hyper-heuristic Framework, *under review*.
- M. Mısır, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, An Investigation on the Generality Level of Selection Hyper-heuristics under Different Empirical Conditions, *under review*.
- M. Mısır, P. Smet, G. Vanden Berghe, An Analysis of Generalised Heuristics for Vehicle Routing and Personnel Rostering Problems, *under review*.

Articles in internationally reviewed academic journals

- M. Mısır, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, A New Hyper-heuristic as a General Problem Solver: an Implementation in HyFlex, *Journal of Scheduling. to appear*.
- B. Bilgin, P. Demeester, M. Mısır, W. Vancroonenburg, G. Vanden Berghe, One Hyperheuristic Approach to Two Timetabling Problems in Health Care, *Journal of Heuristics*, 18(3), p.401–434, 2012.
- E. Burke, G. Kendall, M. Mısır, E. Özcan, Monte Carlo Hyper-heuristics for Examination Timetabling, *Annals of Operations Research*, 196(1), p.73–90, 2012.

- E. Özcan, M. Mısır, G. Ochoa, E. Burke, A Reinforcement Learning - Great Deluge Hyperheuristic for Examination Timetabling, *International Journal of Applied Metaheuristic Computing*, 1(1), p.39–59, 2010.

Article in academic book, internationally recognised scientific publisher

- M. Mısır, T. Wauters, K. Verbeeck, G. Vanden Berghe, A Hyper-heuristic with Learning Automata for the Traveling Tournament Problem, *Metaheuristics: Intelligent Decision Making*, 2012, *to appear*.

Papers at international scientific conferences and symposia, published in full in proceedings

- M. Mısır, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, The Effect of the Set of Low-level Heuristics on the Performance of Selection Hyper-heuristics, in *Proceedings of the 12th International Conference on Parallel Problem Solving From Nature (PPSN'12), vol. 7492 of LNCS*, Taormina, Italy, 2012

- M. Mısır, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, An Intelligent Hyper-heuristic Framework for CHeSC 2011, in *Proceedings of the 6th Learning and Intelligent OptimizatioN Conference (LION'12), vol.7219 of LNCS*, Paris, France, 2012.

- M. Mısır, P. Smet, Katja Verbeeck, G. Vanden Berghe, Security Personnel Routing and Rostering: a Hyper-heuristic Approach, in *Proceedings of the 3rd International Conference on Applied Operational Research (ICAOR'11), vol. 3 of LNMS*, p.193–205, Istanbul, Turkey, 2011.

- M. Mısır,, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, A New Hyper-heuristic Implementation in HyFlex: a Study on Generality, in *Proceedings of the 5th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA'11)*, p.374–393, Phoenix/Arizona, USA, 2011.

- M. Mısır, W. Vancroonenburg, K. Verbeeck, G. Vanden Berghe, A Selection Hyper-heuristic for Scheduling Deliveries of Ready-Mixed Concrete, in *Proceedings of the 9th Metaheuristics International Conference (MIC'11)*, Udine, Italy, 2011.

- M. Mısır, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, Hyper-heuristics with a Dynamic Heuristic Set for the Home Care Scheduling Problem, in *Proceedings of the IEEE Congress on Evolutionary Computation (IEEE CEC'10)*, p.2875–2882, Barcelona, Spain, 2010.

Meeting abstracts, presented at international scientific conferences and symposia, published or not published in proceedings or journals

- M. Mısır, P. De Causmaecker, G. Vanden Berghe, K. Verbeeck, A New Hyper-heuristic Implementation in HyFlex: a Study on Generality, in *Proceedings of the 23rd Benelux Conference on Artificial Intelligence (BNAIC'11)*, Gent, Belgium, 2011.

- W. Vancroonenburg, M. Mısır, B. Bilgin, P. Demeester, G. Vanden Berghe, Hyper-heuristic Approach for Assigning Patients to Hospital Rooms, in *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT'10)*, Belfast, Northern Ireland, 2010.

- B. Bilgin, P. Demeester, M. Mısır, W. Vancroonenburg, G. Vanden Berghe, T. Wauters, A Hyper-heuristic Combined with a Greedy Shuffle Approach to the Nurse Rostering Competition, in *Proceedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT'10)*, Belfast, Northern Ireland, 2010.

- M. Mısır, K. Verbeeck, G. Vanden Berghe, P. De Causmaecker, A Hyper-heuristic Approach to the Home Care Scheduling Problem, in *Proceedings of the 14th Belgian-French-German Conference on Optimization (BFG'09)*, Leuven, Belgium, 2009.

- M. Mısır, B. Bilgin, P. Demeester, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, A Hyper-heuristic Approach to the Patient Admission Scheduling Problem, in *Proceedings of the 35th International Conference of Operational Research Applied to Health Services (ORAHS'09)*, Leuven, Belgium, 2009.

- M. Mısır, T. Wauters, K. Verbeeck, G. Vanden Berghe, A New Learning Hyper-heuristic for the Traveling Tournament Problem, in *Proceedings of the 8th Metaheuristics International Conference (MIC'09)*, Hamburg, Germany, 2009.

- E. Özcan, M. Mısır, E.K. Burke, A Self-organising Hyper-heuristic Framework, in *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA'09)*, Dublin, Ireland, 2009.
- E. Burke, M. Mısır, G. Ochoa, E. Özcan, Learning Heuristic Selection in Hyperheuristics for Examination Timetabling, in *Proceedings of the 7th International Conference of Practice and Theory of Automated Timetabling (PATAT'08)*, Montreal, Canada, 2008.
- E. Burke, G. Kendall, M. Mısır, E. Özcan, A Study of Simulated Annealing Hyperheuristics, in *Proceedings of the 7th International Conference of Practice and Theory of Automated Timetabling (PATAT'08)*, Montreal, Canada, 2008.

Meeting abstracts, presented at other scientific conferences and symposia, published or not published in proceedings or journals

- M. Mısır, K. Verbeeck, P. De Causmaecker, G. Vanden Berghe, Design Principles and Performance Analysis of a Selection Hyper-heuristic across Multiple Problem Domains, in *Proceedings of the 26th Belgian Conference on Operations Research (ORBEL'12)*, Brussels, Belgium, 2012.
- M. Mısır, P. De Causmaecker, G. Vanden Berghe, K. Verbeeck, An Adaptive Selection Hyper-heuristic for CHeSC 2011, in *Proceedings of OR53 Annual Conference*, Nottingham, UK, 2011.
- M. Mısır, K. Verbeeck, G. Vanden Berghe, P. De Causmaecker, Design of a Generic Selection Hyper-heuristic, in *Proceedings of the 25th Belgian Conference on Operations Research (ORBEL'11)*, Gent, Belgium, 2011.
- P. Smet, M. Mısır, G. Vanden Berghe, An Introduction to New Application Domains for the Home Care Scheduling Problem, in *Proceedings of the 25th Belgian Conference on Operations Research (ORBEL'11)*, Gent, Belgium, 2011.
- W. Vancroonenburg, M. Mısır, G. Vanden Berghe, A Hyper-heuristic Approach for the Ready-Mixed Concrete Delivery Problem, in *Proceedings of the 25th Belgian Conference on Operations Research (ORBEL'11)*, Gent, Belgium, 2011.

- M. Mısır, K. Verbeeck, G. Vanden Berghe, P. De Causmaecker, Hyper-heuristics Learning a Varying Set of Low-level Heuristics, *Proceedings of the 24th Belgian Conference on Operations Research (ORBEL'10)*, Liege, Belgium, 2010.
- M. Mısır, P. De Causmaecker, K. Verbeeck, G. Vanden Berghe, Hyper-heuristics: Raising the Level of Generality, *Proceedings of the 23rd Belgian Conference on Operations Research (ORBEL'09)*, Leuven, Belgium, 2009.

Biography

Mustafa Mısır received his BSc and MSc degrees in Computer Engineering from Yeditepe University in 2007 and 2008, respectively. He was awarded a student assistantship during his undergraduate study. He worked as a teaching and research assistant during his graduate study in the Artificial Intelligence research group at Yeditepe University. He started his PhD in the Department of Computer Science at KU Leuven in 2009. He conducted this research in the CODeS research group at KAHO Sint-Lieven. His research interests include hyper-heuristics, meta-heuristics, adaptive/reactive/autonomous search, machine learning, reinforcement learning, evolutionary algorithms, combinatorial optimisation and operations research.

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Science
Department of Computer Science
Combinatorial Optimisation and Decision Support Research Group
Etienne Sabbelaan 53
8500 Kortrijk