CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Identifying sources of global contention in constraint satisfaction search
Authors	Grimes, Diarmuid
Publication date	2012-07
Original Citation	Grimes, D., 2012. Identifying sources of global contention in constraint satisfaction search. PhD Thesis, University College Cork.
Type of publication	Doctoral thesis
Rights	© 2012, Diarmuid Grimes - http://creativecommons.org/licenses/by-nc-nd/3.0/
Download date	2026-10-28 12:45:47
Item downloaded from	https://hdl.handle.net/10468/646

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Identifying Sources of Global Contention in Constraint Satisfaction Search

Diarmuid Grimes

A Thesis Submitted to the National University of Ireland

in Fulfillment of the Requirements for the Degree of

Doctor of Philosophy.

July, 2012

Research Supervisor: Dr. Richard J. Wallace
Research Supervisor: Prof. Eugene C. Freuder
Head of Department: Prof. James Bowen

Department of Computer Science,
National University of Ireland, Cork.

Contents

Abstract **xv**

Declaration **xvii**

1 Introduction **1**

1.1 Background 1

1.2 Motivation 4

1.3 Overview of Dissertation 6

1.4 Summary of Contributions 8

2 Background **11**

2.1 Constraint Satisfaction Problems 11

2.1.1 Complexity Theory 13

2.2 Search and Inference 14

2.2.1 Filtering Methods 16

2.3 Variable and Value Ordering Heuristics 21

2.4 Restarting and Randomness 24

2.5 Learning From Failure 27

2.5.1 Constraint weighting methods 28

2.6 Weighted Degree Heuristics 30

2.7 Statistical Analysis 32

2.7.1 Hypothesis Testing 32

2.7.2 Distribution Analysis and Correlations 34

3 Probing for Failure **39**
3.1 Introduction 39
3.2 Motivation 42
3.3 Initial Methods 44
3.4 Experimentation 50
3.4.1 Problems with embedded insoluble cores 50
3.4.2 Analysis of various restart-cutoff combinations for RNDI and WTDI 56
3.4.3 Random Problems 64
3.4.4 Structured Problems 70
3.5 Analysis of Weight Changes Produced by Each Strategy 75
3.6 Discussion 84
3.6.1 Variable Convection 84
3.6.2 Blame Assignment 86
3.7 Chapter Summary 87

4 Exploration of alternative constraint weighting techniques **89**
4.1 Introduction 89
4.2 Sampling Different Forms of Contention 91
4.2.1 Alternative Forms of Contention Experiments 93
4.2.2 Analysis of Weight Distributions Produced by Unbiased Sampling 99
4.3 Sampling Based on Different Search Procedures 102
4.3.1 Analysis of Weight Profiles Produced by Different Sampling Strategies 107
4.4 Importance of Initial Choices 112
4.5 Further Analysis of the Nature of Unbiased Sampling 115
4.5.1 Local Versus Global Contention 115
4.5.2 Search after Random Probing: Policy Measures and Heuristic Actions 116
4.6 Related Work 119
4.7 Chapter Summary 123

5 Solving Dynamic CSPs Through Failure Reuse 125
5.1 Introduction 125
5.2 Background 127
5.3 Previous Techniques 129
5.4 Experimental Methods 131
5.5 Impact of small changes at the phase transition 132
5.6 New approach - Contention Reuse 138
5.6.1 Stability of points of contention 138
5.6.2 DCSP search with weighted degree heuristics 140
5.6.3 Insoluble Problems 143
5.6.4 Probing with successive changes 145
5.6.5 Alterations Affecting the Solubility of a Problem 147
5.7 Solution Stability 151
5.7.1 Performance of an Algorithm Based on Solution Reuse: Local Changes 152
5.7.2 Nearest Solution Analysis 154
5.7.3 Solution guidance for failure reuse 158
5.8 Chapter Summary 161

6 Alternative restarting strategies 163
6.1 Introduction 163
6.2 Solver Description and Techniques Tested 164
6.2.1 Restarting Parameters for weighted degree approaches . . 166
6.2.2 Randomized non-adaptive heuristic 167
6.2.3 Impact-Based Search 168
6.3 Full Empirical Study 169
6.3.1 Problem Sets 169
6.3.2 Experimental Setup 169
6.3.3 Experimental Results for Weighted Degree Approaches . . 170
6.3.4 Comparison with Impact-Based Search 177
6.4 Case Study I: Open Shop Scheduling Problems 180
6.4.1 Analysis of weight profiles 186
6.5 Case Study II: Radio Link Frequency Assignment Problems . . . 191

6.6 Chapter Summary 194

7 A Generic Approach for Disjunctive Scheduling Problems 197

7.1 Introduction 197
7.2 Background 199
7.2.1 Traditional CP approach ("Heavy Model") 202
7.3 Light Weighted Model (LW) 206
7.3.1 Variable Ordering 207
7.3.2 Value Ordering 208
7.3.3 Additional Features 209
7.3.4 Search Strategy 210
7.4 Open Shop Scheduling 212
7.4.1 Problem Description 212
7.4.2 State of the art 213
7.4.3 Implementation of our model 214
7.4.4 Experimental Evaluation 214
7.5 Job Shop Scheduling 220
7.5.1 Problem Description 220
7.5.2 State of the art 222
7.5.3 Implementation of our model 223
7.5.4 Experimental Evaluation 223
7.6 Job Shop Scheduling with Sequence Dependent Setup Times . . . 225
7.6.1 Problem Description 225
7.6.2 State of the art 227
7.6.3 Implementation of our model 227
7.6.4 Experimental Evaluation 228
7.7 Job Shop Scheduling with Maximal Time Lags 231
7.7.1 Problem Description 231
7.7.2 State of the art 232
7.7.3 Implementation of our model 233
7.7.4 Experimental Evaluation 233
7.8 No-Wait Job Shop Scheduling 240
7.8.1 State of the art 240

7.8.2 Implementation of our model 242
7.8.3 Experimental Evaluation 243
7.9 Job Shop Scheduling with Earliness/Tardiness Objective 249
7.9.1 Problem Description 249
7.9.2 State of the art 250
7.9.3 Implementation of our model 253
7.9.4 Experimental Evaluation 253
7.10 Analysis of different factors in model 258
7.10.1 Experimental Setup and Benchmarks 259
7.11 Weight analysis 263
7.11.1 Weight profiles for OSPs 263
7.11.2 Weight profiles for the JSP and its variants 266
7.12 Chapter Summary 268

8 Conclusion and Future Work 271
8.1 Contributions 272
8.1.1 Boosting the weighted degree heuristic through information gathering 272
8.1.2 Blame assignment issue for constraint weighting 273
8.1.3 Alternative preprocessing techniques 273
8.1.4 Dynamic CSPs at the phase transition: impact of small changes 274
8.1.5 Analysis of various restarting strategies in combination with conflict-directed heuristics 274
8.1.6 Complementary performance between weighted degree and impact-based search 275
8.1.7 Ability of constraint weighting to identify the critical variables in unary resource scheduling problems 275
8.2 Future Work 276

Bibliography 279

A Full Experimental Analysis of Restarting Strategies: Benchmarks 305

List of Figures

1.1 Sample CSP 2
1.2 Impact of Variable Ordering 3

2.1 Sample Sudoku Puzzle. 17
2.2 Gini coefficient example: distribution of income. 35

3.1 Standard deviation of RNDI experiments with different seeds. . . 64
3.2 Solution to a sample 4x4 OSP. 71
3.3 Weight profiles: qk-15-5-add 74
3.4 Weight profiles: ehi-85 and composed instance sample 76
3.5 Average fail depth for RNDI and WTDI 77
3.6 Weight profiles: sample random binary instance 78
3.7 Gini coefficients for weight distribution 78
3.8 Top-down correlations for weighted-degree rankings across restarts 80
3.9 Top-down correlation statistics for RNDI 80
3.10 Top-down correlations for RNDI, short versus long runs 81
3.11 Weight profiles: sample open shop scheduling instance 82
3.12 Variable convection: WTDI example 85

4.1 Alternative forms of contention: Gini coefficients 101
4.2 Alternative forms of contention: Ratio of discrimination 102
4.3 Sampling strategies, weight profiles: random binary instance . . . 108
4.4 Sampling strategies, weight profiles: insoluble OSP instance . . . 108
4.5 Sampling strategies, weight profiles: soluble OSP instance 109
4.6 Sampling strategies, weight profile: queens-knights instance . . . 111
4.7 Importance of initial choices: random binary instance 113

4.8 Importance of initial choices: OSP instances 114

5.1 Impact of small changes on search effort 134
5.2 Impact of small changes on stability of major points of contention 139
5.3 Stability of major points of contention across successive changes . 145
5.4 Contention reuse: $5ad$ condition 146
5.5 Contention reuse: $5r$ condition 147
5.6 Contention reuse: changes affecting solubility. 148
5.7 Nearest solution analysis: DCSPs of varying tightness 156
5.8 Nearest solution analysis: $5ad$ condition 157
5.9 Nearest solution analysis: Extreme cases 158
5.10 Nearest solution analysis: Contention and solution reuse approach 159
5.11 Search performance for nearest solution extreme cases 160

6.1 Sample predicate tree. 166
6.2 Results summary. 170
6.3 Best / uniquely best per problem set 172
6.4 Number of instances solved with different parameter settings. . . . 173
6.5 Boxplot comparison of WTDI and Geowtd 174
6.6 Weight increase per variable with RNDI/WTDI on sample Bibd. . 176
6.7 Scatter plot of runtimes for Geoimpact versus Geowtd. 179
6.8 Number of unsatisfiable instances solved using two different models. 186
6.9 Weight increase on sample OSP instance for different models . . . 187
6.10 Evolution of weights on task variables 189
6.11 Number of constraints weighted in insoluble OSP instance 190
6.12 Individual results for RLFAP modified Scen11 Instances. 193
6.13 RNDI weight spread for over-constrained RLFAP instances 194

7.1 Disjunctive Graph for sample 3×3 JSP. 202
7.2 Optimal solution to sample 3×3 OSP. 213
7.3 Average runtimes on Taillard instances. 216
7.4 Average runtimes on Gueret-Prins instances. 217
7.5 Average runtimes on Brucker instances. 218
7.6 Light vs heavy Choco models for OSP: search effort 218

7.7 Optimal solution to sample 3×3 JSP. 221
7.8 Optimal solution to sample 3×3 SDST-JSP 226
7.9 Optimal solution to sample 3×3 TL-JSP. 232
7.10 Comparison with Artigues method on TL-JSPs 237
7.11 Performance analysis of LW-JTL 238
7.12 Optimal solution to sample 3×3 NW-JSP. 241
7.13 Optimal solution to sample 3×3 ET-JSP. 251
7.14 Boxplot representation of Gini coefficients for sample OSPs . . . 264
7.15 Evolution of weights across restarts for sample OSPs 265
7.16 Task weight versus Boolean weight for sample JSP variants 267

List of Tables

2.1 Two-way fixed-effects ANOVA summary 34

3.1 Results For Instances of the Queens-Knights Problem 53
3.2 Results For Unsatisfiable Embedded Problems 55
3.3 WTDI: Analysis of restart and cutoff factors. 58
3.4 WTDI: Analysis of restart and cutoff factors, final run. 59
3.5 WTDI: Analysis of restart and cutoff factors, only instances solved on run to completion, final run. 60
3.6 RNDI: Analysis of restart and cutoff factors. 61
3.7 RNDI: Analysis of restart and cutoff factors, final run. 63
3.8 Analysis of Variance for RNDI, Solution Run 64
3.9 Results For Random Binary and k-Coloring Problems 66
3.10 Results For 50 Variable 6-Coloring Problem Set 67
3.11 Results For Random Binary and k-Coloring Problems, Frozen Weights 68
3.12 RNDI Comparison of (Small R, Large C) with (Large R, Small C) 69
3.13 Results For Open Shop Scheduling Problems 73

4.1 Search Efficiency with Different Sampling Strategies: Random Binary Problems 94
4.2 Alternative Measurements of Contention: $dom/wdeg$-$nores$. Open Shop Scheduling Problems, # Solved. 95
4.3 Alternative Measurements of Contention: $dom/wdeg$-$nores$. Open Shop Scheduling Problem, Average Nodes. 96

4.4 Alternative Measurements of Contention: RNDI. Open Shop Scheduling Problems, # Solved. 97
4.5 Alternative Measurements of Contention: RNDI. Open Shop Scheduling Problems, Average Nodes. 97
4.6 Alternative Measurements of Contention: WTDI. Open Shop Scheduling Problems, # Solved. 98
4.7 Alternative Measurements of Contention: WTDI. Open Shop Scheduling Problems, Average Nodes. 99
4.8 Top-down correlation coefficients for different weighting methods across ten runs 100
4.9 Search Efficiency with Information Gathered under Different Search Procedures 104
4.10 Average Nodes. Open Shop Scheduling Problems 105
4.11 Top-down correlation coefficients for different sampling methods across ten runs 107
4.12 Analysis of level of discrimination between successively ranked variables on sample of soluble and insoluble scheduling problems. 110
4.13 Adherence to Policy Assessments 117
4.14 Refutations per Depth in Search 118
4.15 Factor Analysis of Heuristic Search Performance 120

5.1 Impact of Small Changes on Search Performance: Random Binary Problem 133
5.2 Impact of Small Changes on Search Performance: Scheduling Problems 135
5.3 Search Performance Correlations for Different Forms of Alterations 136
5.4 Solution Counts for Sample Instances 137
5.5 Correlations of Heuristic Measurements 137
5.6 Search Performance of Weighted Degree Approaches 141
5.7 Search Effort Correlations for Weighted Degree Approaches . . . 142
5.8 Search Performance Correlations: Soluble versus Insoluble Random Binary DCSPs 143

5.9 Search Performance Correlations: Soluble versus Insoluble Scheduling DCSPs 144
5.10 Search Performance for Random Problems where Alterations Changed the Solubility 148
5.11 Search Performance for Scheduling Problems where Alterations Changed the Solubility 149
5.12 Search Performance on Scheduling Instances 150
5.13 Solution Reuse Search Performance: Random DCSPs 153
5.14 Solution Reuse Search Performance: Scheduling DCSPs 154
5.15 Solution Stability Comparison 161

6.1 Full Results By Runtime 171
6.2 Full Results By Problem Type 175
6.3 Number of Instances Solved: Impact-Based Search Comparison . 178
6.4 Taillard Open Shop Scheduling Problems (Aux-Task model) . . . 182
6.5 Gueret-Prins Open Shop Scheduling Problems (Aux-Task model) 183
6.6 Results For Open Shop Scheduling Problems (Task-Only model) . 185
6.7 Results For RLFAP Modified Scen11 Problems 192

7.1 Table of Notation. 200
7.2 Sample 3×3 Job Shop Instance. 202
7.3 Results (Time) For Hard Open Shop Scheduling Problems 219
7.4 APRDs on JSPs: LW-JSP versus SGMPCS 224
7.5 SDST-JSP: Comparison with state-of-the-art exact methods. . . . 229
7.6 SDST-JSP: Comparison with state-of-the-art incomplete techniques. 230
7.7 Sample 3×3 Job Shop Instance with maximum time lag constraints. 232
7.8 TL-JSP. APRD comparison with AHL11 and CLT08 on easy flow/job shop with time lag instances. 234
7.9 TL-JSP. PRD comparison with CLT and AHL on hard job shop with time lag instances. 236
7.10 TL-JSP, optimality and failure to improve on initial upper bound. 237
7.11 LW-JTL: Initialization of upper bound with greedy job insertion heuristic. 240

7.12 NW-JSP: Comparison of NW-JSP and TL-JSP models. 244
7.13 NW-JSP: Runtime comparison with vdB09 for proofs of optimality. 246
7.14 NW-JSP: APRD comparison with state of the art on "easy" instances. 247
7.15 NW-JSP: comparison with state of the art on "hard" instances. . . 248
7.16 NW-JSP: comparison with new model. 248
7.17 Sample 3×3 Job Shop Instance with due dates and earliness/tardiness penalties. 250
7.18 ET-JSP - Random Problems, Number Proven Optimal 255
7.19 ET-JSP - Random Problems, Upper Bound Sum 256
7.20 ET-JSP - GA Problems, Normalized upper bounds 257
7.21 Analysis of algorithm components - OSP, JSP, SDST-JSP, ET-JSP 260
7.22 Analysis of algorithm components - Time lag JSPs 262

Abstract

Much work has been done on learning from failure in search to boost solving of combinatorial problems, such as clause-learning and clause-weighting in boolean satisfiability (SAT), nogood and explanation-based learning, and constraint weighting in constraint satisfaction problems (CSPs). Many of the top solvers in SAT use clause learning to good effect. A similar approach (nogood learning) has not had as large an impact in CSPs. Constraint weighting is a less fine-grained approach where the information learnt gives an approximation as to which variables may be the sources of greatest contention.

In this work we present two methods for learning from search using restarts, in order to identify these critical variables prior to solving. Both methods are based on the conflict-directed heuristic (*weighted-degree heuristic*) introduced by Boussemart et al. and are aimed at producing a better-informed version of the heuristic by gathering information through restarting and *probing* of the search space prior to solving, while minimizing the overhead of these restarts.

We further examine the impact of different sampling strategies and different measurements of contention, and assess different restarting strategies for the heuristic. Finally, two applications for constraint weighting are considered in detail: dynamic constraint satisfaction problems and unary resource scheduling problems.

Declaration

This dissertation is submitted to University College Cork, in accordance with the requirements for the degree of Doctor of Philosophy in the Faculty of Science. The research and thesis presented in this dissertation are entirely my own work and have not been submitted to any other university or higher education institution, or for any other academic award in this university. Where use has been made of other people's work, it has been fully acknowledged and referenced. Parts of this work have appeared in the following publications which have been subject to peer review.

1. Diarmuid Grimes and Emmanuel Hebrard, "Models and Strategies for Variants of the Job Shop Scheduling Problem", *Principles and Practice of Constraint Programming (CP'11)*, pp. 356-372, 2011

2. Richard J. Wallace and Diarmuid Grimes, "Problem-Structure vs. Solution-Based Methods for Solving Dynamic Constraint Satisfaction Problems", *22nd International Conference on Tools with Artificial Intelligence (ICTAI'10)* ,2010

3. Richard J. Wallace and Diarmuid Grimes and Eugene C. Freuder, "Dynamic Constraint Satisfaction Problems: Relations between Search Strategies and Variability in Performance ", *Recent Advances in Constraints*, pp. 105-121,2010

4. Diarmuid Grimes and Emmanuel Hebrard, "Job Shop Scheduling with Setup Times and Maximal Time-Lags: A Simple Constraint Programming Approach", *Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming (CPAIOR'10)*, pp. 147-161, 2010

5. Diarmuid Grimes and Emmanuel Hebrard and Arnaud Malapert, "Closing the Open Shop: Contradicting Conventional Wisdom", *Principles and Practice of Constraint Programming (CP'09)*, pp. 400-408, 2009

6. Richard J. Wallace and Diarmuid Grimes and Eugene C. Freuder, "Solving Dynamic Constraint Satisfaction Problems by Identifying Stable Features", *21st International Joint Conference on Artificial Intelligence (IJCAI'09)*, pp. 621-627, 2009

7. Richard J. Wallace and Diarmuid Grimes, "Experimental Studies of Variable Selection Strategies Based on Constraint Weights", *Journal of Algorithms: Cognition, Informatics and Logic*, pp. 114-129, 2008

8. Diarmuid Grimes "A Study of Adaptive Restarting Strategies for Solving Constraint Satisfaction Problems", *Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science (AICS'08)*, pp. 33-42, 2008

9. Diarmuid Grimes and Richard J. Wallace "Sampling Strategies and Variable Selection in Weighted Degree Heuristics", *Principles and Practice of Constraint Programming (CP'07)*, pp. 831-838, 2007

10. Diarmuid Grimes and Richard J. Wallace, "Learning to Identify Global Bottlenecks in Constraint Satisfaction Search", *20th Conference of the Florida*

Artificial Intelligence Research Society (FLAIRS'07), pp. 592-597, 2007

11. Diarmuid Grimes and Richard J. Wallace, "Learning from Failure in Constraint Satisfaction Search", *Learning for Search: Papers from the 2006 AAAI Workshop, pp. 24-31, 2006*,

The contents of this dissertation extensively elaborate upon previously published work and mistakes (if any) are corrected.

Diarmuid Grimes
May 2012.

Dedication

This dissertation is dedicated to the memory of my father, Michael Grimes (1942-2011).

Cuireann sé díomá an domhain orm nach raibh sé seo chríochnaithe agus thú fós beo, ach tá fhios agam go bhfuil tú liom agus ag féachaint anuas orm go bródúil. Beidh tú i gcónaí i mo chroí.

Acknowledgements

There are many people whose help and support I would like to acknowledge. First and foremost, my supervisors Dr. Richard Wallace and Prof. Eugene Freuder, who have helped me develop as a researcher and provided guidance and support in the good and bad times. I'd also like to thank Prof. Barry O'Sullivan for always having an encouraging word, and Science Foundation Ireland for providing financial support to the research presented here.

There are too many colleagues to thank individually in 4C over the years, however for the good distractions and enlightening discussions over coffee, I'd like to say a special thanks to Alexandre Papadapoulos, Deepak Mehta, Emmanuel Hebrard, Tarik Hadzic and Mark Hennessy. I'd also like to express my gratitude to Caitríona, Eleanor and Linda in the admin for all their help.

Most importantly, I'd like to acknowledge the amazing support I've received from my family, particularly my parents Phil and Mick. This dissertation would not have been possible without your support. I must also acknowledge the support I've received from my brothers Ciarán and Michael, and my sisters Eimear and Cora. A special thanks to my nephews Caoimhín, Éanna, Conor, Ruairí, Lorcan, and nieces Ailbhe, Clodagh and Máire Cáit, for always managing to bring a smile to my face, intentionally or not.

Go raibh maith agaibh go léir,
Diarmuid

Chapter 1

Introduction

The central thesis defended in this dissertation is that:

Most constraint satisfaction problems contain globally difficult elements which can be identified through repeated involvement in failures during search. These elements can then be used to both solve problems efficiently and provide valuable feedback to the user.

In this chapter we firstly provide some background and motivation before introducing the research presented in this dissertation. We then outline the main contributions of the dissertation.

1.1 Background

For many real-world problems, the number of potential solutions is too large for the problem to be solved by brute-force search. An example of such a problem would be creating a timetable for a university. This involves scheduling a room and a lecturer for each course lecture over all courses. This is subject to constraints such as the capacity of the room must be greater than or equal to the number of students in the lecture, a room/lecturer cannot be assigned to two lectures at the one time, etc. Obviously, each lecture can start at any time and take place in any room of capacity greater than the number of students in the lecture, thus the number of potential solutions can be enormous.

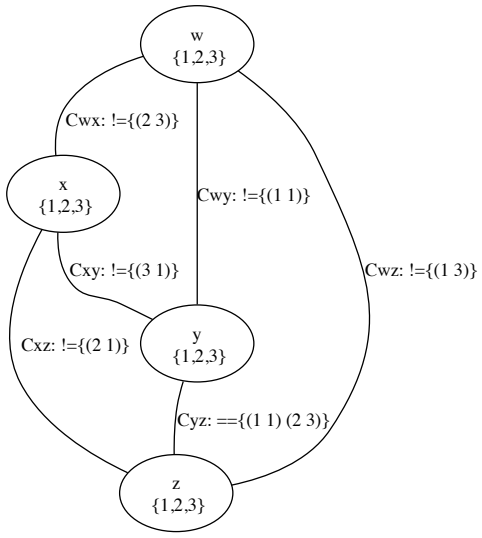


Figure 1.1: Sample CSP with extensional constraints:"==" for allowed tuples, "!=" for disallowed tuples

One method which has been used quite successfully to handle such problems is to model them as a constraint satisfaction problem (CSP) and use constraint programming techniques to solve them. A CSP consists of a set of variables $\mathscr{V} = \{V_1, \ldots, V_n\}$, having domains $\{D_1, \ldots, D_n\}$ containing the set of possible values that the variable may take in a solution; and a set of constraints $\mathscr{C} = \{C_1, \ldots, C_m\}$ over the variables defining what combinations of variable-value pairs are allowed. The search space is then defined by the combinations of all variables and values. A solution to the problem is a set of assignments $\{V_1 = a_1, \ldots, V_n = a_n\}$ where $a_i \in D_i$ and such that all constraints are satisfied.

One commonly used approach for solving CSPs is to use depth-first search, where the order in which the tree is explored is decided by search heuristics. A heuristic is a method for selecting the "best" option, where the best is defined in terms of some associated metric. Heuristics are used throughout constraint satisfaction, be it in choosing the next variable to assign, choosing a value to assign to the current variable, choosing which constraints to check for consistency, etc. The choice of heuristic can have a huge impact on the search effort required to solve a

problem. Most variable ordering heuristics aim to minimize the (probable) search space below the choice point, i.e. they try to simplify the underlying problem. These heuristics follow the Fail-First principle of Haralick and Eliot [97] which states that "to succeed, first try where you are most likely to fail". The variable which most constrains the underlying search space is expected to work best in general.

We illustrate the impact different variable orderings can have on search with a toy problem, depicted in Figure 1.1, consisting of four variables w, x, y, z, each with initial domain size 3. The constraints define what pairs of values are allowed (e.g. "== $\{(1\ 1)\}$"), or not allowed (e.g. "!= $\{(2\ 3)\}$"). For example the constraint between w and x states that all variable-value pairs are allowed except ($w = 2$, $x = 3$), while the constraint between y and z states that the only variable-value pairs allowed are ($y = 1$, $z = 1$) or ($y = 2$, $z = 3$). Figure 1.2 shows the search trees explored with chronological backtracking and depth first search for two different variable orderings: w, x, y, z and z, y, w, x. The latter ordering results in exploration of a much smaller search tree to find a solution.

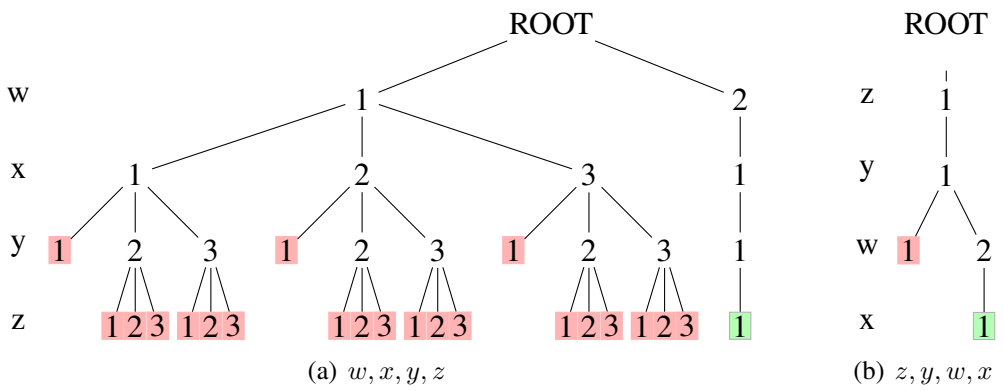


Figure 1.2: Impact of variable ordering. Depth-first search trees, chronological backtracking. Nodes in red are failed nodes, nodes in green are solution nodes.

1.2 Motivation

This work addresses two difficulties when using constraint programming to solve a problem. The first is that the user is often not an expert in the field, thus implementing a domain-specific heuristic for their problem may be beyond their capabilities. By creating an efficient generic heuristic, which is problem independent, we can remove some of the burden from the user.

The second bottleneck is with regard to search itself, in particular the impact of "thrashing" on the amount of search effort required to solve a problem. Thrashing refers to redundant search, where we repeatedly search over variables which are independent of the cause of failure. In the toy problem introduced earlier, there is no solution involving $w = 1$ due to conflicts with y and z. However, using the ordering w, x, y, z, in Figure 1.2(a), we repeatedly search over the variable x which is independent of the conflict. This increases the size of the refutation (i.e. the subtree searched) required to prove that the assignment ($w = 1$) does not occur in any solution to the problem. Now suppose instead of variable x, there are k variables of domain size 3 which are independent of the conflict between $w = 1$ and y and z. Then, searching over these variables before y and z would result in a refutation of size $3^{(k+2)}$, compared to the minimal refutation of size 9.

If the key variables can be identified and moved to the top of the search tree, this should reduce the amount of thrashing encountered, leading to a faster, more efficient method for solving the problem. Indeed, it has been shown that many real-world instances contain small sets of variables (known as *backdoors*) whose assignment results in a simplified problem which can be solved in polynomial time [227]. These variables are said to capture some hidden structure within the problem.

One method to improve search performance is to incorporate some form of learning, e.g. learn from the failures encountered during search. In the problem solving domain, learning from failure often involves avoiding certain areas of search when a condition is triggered recognizing the current state as a bad state from previous experience in solving. However, for constraint satisfaction problems, it can be somewhat different in that, rather than avoid an area of search which was a source of contention in past experience, often we would like to tackle

this area of contention as early as possible (Fail-First principle), with the hope that we can either prove the problem insoluble quickly or we can minimize the search space beneath the current choice point.

In this work, we combine two topics which have been the focus of much recent research in the area of CSP solving, both of which address the issue of thrashing. The first topic concerns search heuristics which use information from previous search states to guide subsequent search, e.g. the impact-based heuristics of Refalo [169] and the conflict-directed heuristics of Boussemart et al. [29]. We concentrate on the latter heuristic, which focuses search on areas of contention by weighting constraints that cause failure during constraint propagation, and choosing the variable with the largest sum of its constraints' weights.

Since these weights represent a likelihood of failure, choosing the variable with largest weight-sum is clearly following the Fail-First principle. Handling these contentious variables higher in the search tree increases the likelihood of detecting failures early in the subsequent search. In extreme cases the heuristic can identify insoluble cores in problems [105][91]. It has been shown to be one of the most effective general purpose heuristics [29].

The second topic of research concerns the use of restarting strategies combined with an element of randomization for problem solving. Such techniques generally randomize the search heuristics through random tie-breaking, e.g. rapid randomized restarts (RRR) of Gomes et al. [82]. In this case one maintains a degree of confidence in the selections made while still allowing for search diversification. Randomized restarting has been shown to be especially effective at dealing with problems which display a heavy-tailed runtime distribution [82]. However these methods do not learn from their failed search attempts. Furthermore, there is an implicit assumption that the heuristic being randomized is a good heuristic for the problem. If this is not the case, then randomly selecting from its top choices will not improve performance.

More recently, restarting approaches have been proposed which combine some form of learning with solving. For example, the SAT solver Chaff [155] combines restarting with clause weighting and clause learning. A similar approach has been proposed for combining nogood recording and conflict-directed heuristics with restarts in the CSP domain [135]. Refalo also proposed combining restarting with

the impact-based heuristics [169]. Gomes refers to these types of approach as ”deterministic randomization” [80] where the behaviour of search, although deterministic, is so complex as to appear random.

Both restarting and the weighted-degree heuristics are employed to reduce the amount of thrashing in search, where large unpromising parts of the search space are traversed systematically. Randomized restarting allows one to “jump out” of possibly unpromising subtrees, by restarting with a randomized heuristic which will result in search traversing a different part of the search space. The weighted-degree heuristic avoids thrashing by moving contentious variables up the ordering, i.e. selecting contentious variables, identified through their participation in constraint failures, higher in the search tree after backtracking.

Our approach works on the assumption that elements representing sources of global difficulty exist and can be identified. Using the terminology of Joslin and Clements [115], *globally* difficult elements are elements which are difficult across large parts of the search space while *locally* difficult elements are difficult only in the context of a particular state of search. These authors point out that identification of difficult elements through static analysis of the problem is sometimes possible but interactions between constraints can be quite complex. Often it is only through search that these intricate relations come to the fore. We use constraint weights to identify the difficult variables. Randomized (“deterministically”) restarting results in the traversal of different parts of the search space, and thus variables with a high weight after a number of restarts are likely to be sources of global contention.

1.3 Overview of Dissertation

In Chapter 2 we provide the general background for the work presented in the dissertation. We describe the constraint satisfaction problem, along with popular solving techniques such as lookahead algorithms and variable/value ordering heuristics before focusing on the two main areas related to this dissertation: randomized restarting and learning in search. With regard to the latter, we primarily focus on methods for learning from failure.

In Chapter 3, we introduce two novel approaches for identifying sources of

global contention in a problem. Our first method combines the benefits of restarting with the weighted degree heuristic, and is based on the results of Zhan [237] who found that most problems are either suited to rapid restarts or require intensive search exploration. The incorrect choice of algorithm (with/without restarting) can have disastrous consequences for the amount of search effort required to solve a problem.

Our second approach, *random probing*, uses the restarted search mainly for unbiased information gathering, regarding the critical variables in the problem. It performs a number of random probes of the search space, updating constraint weights. We show that both approaches can improve on the basic (non-restarting) use of the heuristic, and provide insight into the behaviour of search for both methods. In particular, we identify a key factor when combining the weighted degree heuristic with restarting, which we refer to as "*variable convection*".

Chapter 4 examines some alternative methods for determining contention with the aim of improving the quality of information learnt. We find that sampling contention directly related to failures was consistently better than sampling all instances of constraints removing values. However, this was less clear when combined directly with the heuristic, as opposed to the unbiased information gathering of random probing. We also found that incrementing weights based on the number of values removed by a constraint, generally improved the performance of the weighted-degree heuristic, but had a negative impact when combined with the random probing approach.

We next assess the quality of information produced by two other constraint weighting strategies: a local search technique and a weaker form of consistency checking in constructive search. The motivation is two fold. Firstly, passing failure information from local search to a constructive search algorithm has been previously proposed in a number of different ways (e.g. [60], [147]). Secondly, both alternatives are much faster than the method used in Chapter 3, thus if there was little fall-off in terms of quality of information learnt, our method could be improved by incorporating one of these alternatives. However, we found that the quality of information learnt by these alternatives was generally poorer for the problems tested.

In Chapter 5, we introduce a new method for solving dynamic constraint sat-

isfaction problems (DCSPs). Our method reuses information regarding the major bottlenecks of a problem to solve perturbed versions of the problem. We compare with a state-of-the-art method for solving DCSPs ("local changes" [209]), and investigate why our approach outperforms local changes for DCSPs at the phase transition. We also present analysis of the impact of problem alterations at the phase transition, and find that even relatively few alterations can have a drastic impact on a number of features of a problem, e.g. search effort, number of solutions, fail-firstness.

Chapter 6 compares the two approaches of Chapter 3 with alternative methods for combining the conflict-directed heuristics with restarts across a large sample of problems. We also compare these two techniques with other popular approaches for general purpose problem solving and provide insight into the performance of each approach. We present detailed analysis for a subset of problems, namely open shop scheduling problems [196] and radio link frequency allocation problems ("RLFAPs") [37].

Based on the findings of the previous chapter, a new approach for solving machine scheduling problems is introduced in Chapter 7. The algorithm combines a number of generic AI techniques such as constraint weighting and restarts, with simple propagation (bounds consistency). It uses no domain-specific heuristics or propagators. Although this has been one of the most heavily studied areas in CP [17], with many dedicated heuristics and constraint propagators, we find that our method outperforms many state-of-the-art, dedicated, techniques. We show how our method can be easily extended to handle side constraints and the alternative objective of minimizing earliness/tardiness penalties, which can cause difficulty for the standard CP techniques,.

The final chapter presents conclusions of the dissertation and future directions for research in this area.

1.4 Summary of Contributions

The main contributions of the dissertation are as follows:

- **Boosting conflict-directed heuristics.** We introduce two basic methods for improving the performance of weighted-degree heuristics. Both methods incorporate restarting, albeit with different purposes, and are developed with the overall goal of producing the best learning procedure for general problem-solving with conflict-directed heuristics. We empirically show the benefits of these methods, and provide insight into to the behavior of conflict directed search with both, identifying advantages and disadvantages.

- **Alternative methods of assessing contention.** We investigate different methods of extracting contention information, and we propose methods for improving the quality of the information extracted through better quantification of the information. However, we show that although all methods improve over a non-learning approach, methods based solely on failure information perform best.

- **Alternative methods of generating weights.** We assess the quality of information produced by different constraint-weighting procedures, in both local and complete search algorithms. We show that some methods are more susceptible to local sources of contention.

- **A contention-resuse approach for solving dynamic constraint satisfaction problems.** We introduce a new method for solving dynamic constraint satisfaction problems (DCSPs), based on identifying the major points of contention, and examine the impact of problem changes at the phase transition. We show that the contention based approach outperforms a traditional solution reuse method for problems at the phase transition.

- We investigate combining previously proposed universal restarting strategies with the heuristic, and provide insight as to the interaction of the heuristic with each restarting strategy. We also compare with other popular general purpose heuristics.

- **A generic approach for solving scheduling problems.** We introduce a new approach to solving scheduling problems which, rather than using domain-specific propagators and heuristics, combines a number of generic solving

techniques. We show that this can outperform state-of-the-art, dedicated, approaches on a number of scheduling problem types.

Chapter 2

Background

In this chapter we review the basic concepts used in constraint programming. We first provide a formal definition of the constraint satisfaction problem, followed by an overview of the standard techniques for solving these problems such as backtracking search, lookahead algorithms, and variable and value ordering heuristics. We then review the two main topics that motivate this dissertation: randomized restarting and learning from search.

2.1 Constraint Satisfaction Problems

Definition 2.1.1. *A finite* Constraint Satisfaction Problem (CSP) *is defined as a tuple of the form* $(\mathscr{V}, \mathscr{D}, \mathscr{C})$ *where:* $\mathscr{V} = \{V_1, \ldots, V_n\}$ *is a finite set of variables which must be assigned values;* $\mathscr{D} = \{D_1, \ldots, D_n\}$ *is the set of finite domains for those variables consisting of possible values which may be assigned to the variables; and* $\mathscr{C} = \{C_1, \ldots, C_m\}$ *is the finite set of constraints over subsets of the variables. Each constraint* C_k *expresses a relation* $Rel(C_k)$ *among domain values that that can be assigned to the variables in the scope of the constraint,* $Vars(C_k)$*, stating what combinations of variable-value pairs are allowed.*

An example of a CSP is that of post-enrolment university timetabling, where we require the assignment of a room and timeslot to each class (*event*) for a weekly period. In CSP terms, there are two variables r_i and t_i for each event i. The do-

main of r_i is the set of rooms, while the domain of t_i is the set of timeslots. The constraints on this problem are as follows:

- Two events with a common student/lecturer cannot be assigned the same timeslot.
- A room cannot be assigned two events in the same timeslot.
- The capacity of the room cannot be less than the number of students in the event.

In some cases we may have a metric for ranking the solutions and for defining the *optimal* solution. A *constraint optimization problem* (COP) is a CSP with an additional metric (objective function) for ranking the solutions. An optimal solution is one which minimizes/maximizes the objective function. In the timetabling example above, one objective function would be to minimize the number of consecutive events a student must attend, over all students.

A related problem to the CSP is that of *propositional satisfiability* (SAT). A SAT problem is typically defined in *conjunctive normal form* (CNF), where a CNF formula is a conjunction of clauses and a clause is a disjunction of literals (Boolean variables and their negations). The problem involves finding if there is an assignment to the variables that make the formula true.

Constraint Programming (CP) is a paradigm for solving CSPs and COPs. It is commonly viewed as a two-stage process composed of modeling and search. Modeling involves defining the problem in terms of variables, constraints and objectives. Search involves defining how the set of possible solutions to the problem is to be explored.

Constraints of a standard CSP can be defined in two ways. Extensional constraints state a list of allowed (or unallowed) tuples e.g. C_{ij} = {(1,2) (2,3) (3,1) (3,3)} which means that if variable V_i is assigned the value 1 then variable V_j must take the value 2 in a consistent assignment on this constraint. Intensional constraints are defined in terms of predicate function(s) on the variables in the scope of the constraint, e.g. $C_{ij} \equiv V_i = V_j$, which means that if variable V_i is assigned a value a, then variable V_j must also take the value a. Note that this con-

straint could also have been defined extensionally by listing all tuple pairs of the form (a,a).

An *assignment* is a set of tuples $\{(V_1, a), (V_2, b), \ldots, (V_k, h)\}$, each tuple consisting of a different instantiated variable and the value that is assigned to it. A *consistent assignment* is an assignment which does not violate any constraint covered by it. A *solution* to a problem is a consistent assignment to all variables $\{(V_1, a), (V_2, b), \ldots, (V_n, x)\}$, i.e. an assignment of a value to every variable such that no constraint is violated.

We define a *neighbor* of a variable V_i to be any variable V_j that shares a constraint with V_i, i.e. if $\exists\ C_k \in \mathscr{C}$ s.t. V_i, $V_j \in (Vars(C_k))$, then V_i and V_j are neighbors. Finally CSPs are often described in terms of the maximum *arity* over all constraints, where the arity of a constraint is the number of variables in its scope. In this work we focus mainly on binary CSPs, where the maximum arity over all constraints is 2. A non-binary CSP, or n-ary CSP, is a CSP that has at least one constraint of arity > 2.

2.1.1 Complexity Theory

A *decision problem* is a problem which is formulated as a yes or no question. An instance of the problem is a given input. A decision problem is said to be in the class **P** (*Polynomial time*), if there exists a deterministic Turing machine which can decide whether an instance of the problem is a yes or no instance in time polynomial in the size of the input.

A problem is said to be in the class **NP** (*Non-determinstic Polynomial*) if there is a non-deterministic Turing machine which can verify a yes instance in time polynomial in the input size. The non-deterministic Turing machine operates as a two step process. In the first step, a certificate is guessed and in the second stage it is verified in polynomial time whether, given the input and the certificate, the instance is a yes instance. A problem is said to be in **co-NP** if an instance can be verified by a non-deterministic Turing machine to be a no instance in time polynomial in the size of the input.

A *polynomial reduction* f of a problem X to a problem Y is a reduction such that for every instance x of X there is an instance $f(x)$ of Y which can be computed

in polynomial time, and such that x is a yes instance of X if and only if $f(x)$ is a yes instance of Y. A problem is said to be in the class ***NP**-hard* if all problems in **NP** can be polynomially reduced to it. A problem is said to be in the class ***NP**-complete* if it is both in **NP** and in **NP**-hard. This means that if there is a polynomial algorithm for an **NP**-hard problem, then there is also a polynomial algorithm for all **NP** problems. Similarly, if any problem in **NP** is proven to be intractable, then all **NP**-hard problems are also intractable.

A *pseudo-polynomial time algorithm* is one which can verify that an instance of a problem is a yes instance in time bounded by a polynomial function of the size of the input and the magnitude of the largest number appearing in the input. An **NP**-hard problem is said to be ***NP**-hard in the strong sense* if it cannot have a pseudo-polynomial time algorithm (unless **P=NP**). The CSP is in the complexity class **NP**-hard [141]. For further details on complexity theory of decision problems, the reader is pointed to (Garey and Johnson [69]).

2.2 Search and Inference

Search strategies for solving a CSP typically fall into one of two categories: *local* and *constructive* search algorithms. Local search algorithms start with a complete assignment to the variables in the problem, usually generated by a random assignment or a fast heuristic method. This initial assignment may violate a number of constraints, and so is iteratively improved by changing a variable assignment in each iteration. This process continues until either a solution is found or some iteration limit is reached.

Local search approaches are typically extremely fast, however the main drawback is that they are incomplete, i.e. there is no guarantee that a solution will be found. Furthermore they cannot prove infeasibility if no solution exists. Examples of local search methods are tabu search [79], simulated annealing [125], etc. For a more detail description of constraint-based local search techniques, see (Van Hentenryck and Michel [203]).

Constructive search algorithms start with the null assignment (no variable is assigned a value) and then systematically perform the following three steps:

1. Select a variable
2. Assign the variable a value
3. Checks if the assignment is consistent

Algorithm 1: CBT

```
Input  : A CSP in the form (𝒱, 𝒟, 𝒞)
Output: solution or insoluble
solution = Assign(𝒱, 𝒟, 𝒞, {})
if solution ≠ false then
    return solution
else return insoluble

function Assign(futureVars, 𝒟, 𝒞, assignment)
    if futureVars = ∅ then
        return assignment
    V_i = Select variable from futureVars
    futureVars ← futureVars − {V_i}
    result ← false
    while D_i ≠ ∅ do
        d_i = SelectValue(D_i)
        consistent ← Check consistency of {V_i ← d_i}
        if consistent = true then
            assignment ← assignment ∪ {V_i ← d_i}
            result ← Assign(futureVars, 𝒟, 𝒞, assignment)
            if result = false then
                assignment ← assignment − {V_i ← d_i}
                D_i ← D_i − {d_i}
            else
                return result
        else
            D_i ← D_i − {d_i}
        return result
```

These are repeated until either all variables have been assigned or an inconsistency has been detected. In the latter case, the value selected is removed from

the current domain of its variable and a new value is tried. If all the values in a variable's domain fail, it performs a *backtrack* i.e. it "backs" up to the preceding variable.

The simplest form of constructive search is chronological backtracking (CBT) with depth first search (Algorithm 1). The recursive function "Assign" (line 5) starts with an empty assignment and then extends this by repeatedly selecting an unassigned variable (line 8) and assigning it a value (line 12) until either it has found a complete consistent assignment (line 7), or a branch of search has failed (line 14).

2.2.1 Filtering Methods

Local consistencies

Local consistencies are consistencies defined over local parts of the CSP, i.e. defined typically over subsets of the variables and subsets of the constraints of the problem. These remove values which cannot be part of a solution to the problem (in its current state). In this work we are mainly concerned with arc-consistency (AC) [223, 141] and bounds-consistency (BC) [205]. Let P be a binary CSP and V_i,$V_j \in Vars(C_{ij})$.

Definition 2.2.1. *A value $d_i \in D_i$ is* supported *on C_{ij} iff $\exists\, d_j \in D_j$ s.t. $(d_i, d_j) \in Rel(C_{ij})$.*

Definition 2.2.2. *A (binary) CSP P is* arc-consistent *iff every value in every variables domain is supported over all constraints involving the variable.*

The definition of support can be extended to handle problems with n-ary constraints:

Definition 2.2.3. *For a constraint C_k with $Vars(C_k) = \{V_1, \ldots, V_r\}$, a value $d_i \in V_i$ is GAC-supported on C_k iff*

$\forall\, j \in \{1, \ldots, r\}$, $j \neq i$, $\exists\, d_j \in D_j$, *s.t.* $(d_1, \ldots, d_i, \ldots, d_r) \in Rel(C_k)$

A CSP P is *generalized arc-consistent* (GAC) on a problem iff every value in every variable is GAC-supported over all constraints involving the variable. Generalized arc-consistency is also referred to as as *hyper arc-consistency* or *domain consistency*.

4			8	1	3			9
	3		7		6		8	
		9		5		7		
8	9		2				4	7
		6				2		
2	4						9	3
		8		2		3		
	5		9		4		2	
3			6		8			5

Figure 2.1: Sample Sudoku Puzzle.

To illustrate the difference between AC and GAC, consider the well-known sudoku puzzle. This puzzle (in 9x9 format) involves assigning each blank square a number between 1 and 9 such that each number appears once in every row, every column and each highlighted 3x3 block. A sample sudoku puzzle is given in Figure 2.1.

In CSP terms, the squares are the variables (thus there are 81 variables in the 9x9 sudoku). We will refer to the variable in row i, column j as x_{ij}, where rows are numbered from top to bottom and columns are numbered from left to right. The domain of each blank square is $\{1,\ldots,9\}$, we will consider the non-blank squares as variables with singleton domains. This problem can be modeled as a binary CSP by adding an inequality constraint between every pair of variables on every row, every column and every highlighted block, requiring 972 constraints. Alternatively, one could model this as a non-binary CSP by adding an *alldifferent* constraint (Régin [170]) for the variables in each row, each column, and each highlighted block, requiring only 27 constraints.

Consider the 3x3 block with variables x_{ij}: $i = 4, 5, 6$; $j = 1, 2, 3$. Performing AC on the constraints involving variables in this block will result in the following reduced domains for the four non-singleton variables in the block:

- $x_{43} = \{1, 3, 5\}$
- $x_{51} = \{1, 5, 7\}$
- $x_{52} = \{1, 7\}$
- $x_{63} = \{1, 5, 7\}$

No further propagation will occur unless a neighbor of one of the variables has its domain reduced to a singleton. This is because if a variable has at least two values in its domain, then all values in each of its neighbors domains will have a support on the associated inequality constraint. On the other hand, performing GAC on the all-different constraint on these four variables would reduce the domains further. The values 1 and 5 in the domain of x_{43} are *not* GAC-supported, so these would be removed from the domain of x_{43}.

A number of algorithms have been proposed for enforcing arc consistency, with differing time and space complexity. The arc-consistency algorithm we will use in this dissertation is AC-3 (Mackworth [141]).

Bounds consistency (BC) is a weaker form of consistency than AC, i.e. AC will always remove *at least* as many values as BC and may remove more. BC is applied to variables with totally ordered domains. There are a number of definitions of bounds consistency in the literature, as discussed in Bessière [26] and Choi et al. [47]. The definition we use is referred to as *bounds*($\mathcal{Z}$) consistency in [26, 47] and *interval* consistency in Apt [5].

Let $min(V_i)$ (respectively $max(V_i)$) be the minimum (maximum *resp.*) value in the ordered domain of V_i, and $Vars(C_k) = \{V_1, \ldots, V_i, \ldots, V_r\}$.

- A variable V_i is said to be BC for a constraint C_k iff $\forall d_i \in \{min(V_i), max(V_i)\}$, $\forall\ j \in \{1, \ldots, r\}$, $j \neq i$, $\exists\ d_j$ where $min(V_j) \leq d_j \leq max(V_j)$, s.t. $(d_1, \ldots, d_i, \ldots, d_r) \in Rel(C_k)$
- A problem P is said to be BC iff $\forall\ V_i \in \mathscr{V}$, $\forall C_k \in \mathscr{C}$ s.t. $V_i \in Vars(C_k)$, V_i is BC on C_k

Singleton arc-consistency (Debruyne and Bessière [54]), on the other hand, is a stronger form of consistency than AC. A value a is singleton arc-consistent (SAC) for its variable V_i if the problem P with $V_i = a$ (denoted $P|_{V_i=a}$) can be made AC. Values that are not SAC are removed. As this can be expensive to calculate, a restricted form of SAC was proposed (RSAC) in [165], where each variable is only considered once.

Lookahead Algorithms

There have been many proposed improvements to chronological backtracking. These methods can be mainly partitioned under two headings: lookback algorithms and lookahead algorithms. Lookback algorithms (or "intelligent backtracking" algorithms) record information regarding which values are in conflict and use this information to *jump* back in search to the root of a failure thus avoiding some thrashing. Examples of lookback schemes are *conflict-directed backjumping* (Prosser [164]), and *dynamic backtracking* (Ginsberg [77]).

In this work we concentrate on lookahead algorithms (also referred to as *filtering/propagation* algorithms). These algorithms prune the underlying search tree after each variable assignment by enforcing some level of consistency (e.g. arc-consistency). To change the chronological backtracking search algorithm outlined above to a lookahead search algorithm, one would replace the "check-consistency" function with a lookahead function.

Forward-checking (FC) checks that the value assigned to the variable is consistent with the variable's unassigned neighbors, removing values from unassigned neighbors' domains which are not supported (Haralick and Elliott [97]). If all values from a neighbors domain are removed (causing a *domain wipeout*) then a new value is tried for the current variable, and search backtracks if all values have been tried unsuccessfully. The assignments never have to be checked against previous instantiations since they must be consistent otherwise they would have been removed when forward-checking was performed on the previous assignments. Forward-checking is also known as partial lookahead because it only removes inconsistent values from the neighbors of the variable assigned.

Maintaining arc-consistency (MAC), or full lookahead, enforces arc-consistency

after each value assignment (Sabin and Freuder [174]). This a stronger form of propagation than FC although there is a cost associated, namely the consistency checking of the extra constraints. We will refer to the constraints which may result in propagation as *active constraints* ($\mathscr{C}_A$). A constraint is active if at least two variables in its scope are unassigned.

To illustrate the stronger propagation of MAC relative to FC, let us reconsider the sudoku puzzle (Figure 2.1) and in particular the four variables discussed earlier with domains:

- $x_{43} = \{1, 3, 5\}$
- $x_{51} = \{1, 5, 7\}$
- $x_{52} = \{1, 7\}$
- $x_{63} = \{1, 5, 7\}$

Suppose x_{43} is selected and assigned the value 1. Forward checking will simply remove this value from the domains of the neighbors of x_{43} including the three unassigned variables in the block. Arc-consistency on this assignment will discover that it is inconsistent using the following inference:

$$(x_{43} = 1 \implies x_{52} = 7 \implies x_{51} = x_{63} = 5) \wedge (x_{51} \neq x_{63}) \implies \perp.$$

Finally we note that the branching strategy used in Algorithm 1 is *d-way branching*, for each variable V_i selected, there are $|d_i|$ possible branches to be explored, one for each value in its domain. An alternative branching scheme is known as *binary* or 2-*way branching*, where each variable choice point has two branches. Let V_i be the variable selected, and a the value selected for the variable from its domain. Then the constraint $V_i = a$ is posted on the left branch, while on the right branch the constraint $V_i \neq a$ is posted. Propagation of the constraint $V_i \neq a$ can potentially reduce the search space. Indeed, 2-way branching has been shown to be stronger than d-way branching Hwang and Mitchell [111].

2.3 Variable and Value Ordering Heuristics

The following discussion is based on depth-first chronological backtrack search with a lookahead algorithm, e.g. forward-checking or maintaining arc-consistency.

The ordering in which variables are chosen can have a large impact on the ability of search to find a solution or prove the problem insoluble quickly. At first glance this may seem somewhat counterintuitive for the soluble problem case, since, if there is to be a solution, then every variable will be part of it, but only one value per variable will be in any given solution.

Thus it would seem logical that value selection would be more important for reducing search. This is indeed true in the sense that if one had an infallible method for value selection then the variable ordering would be obsolete. However, if such a heuristic existed then the problem would be solvable in polynomial time. A general rule-of-thumb is to choose the variable most likely to fail [97], and for this variable choose the value most likely to succeed. There are a number of metrics one can use to decide which variable is most likely to fail, and which value is most likely to succeed.

Many general-purpose variable ordering heuristics have been proposed. They fall into two categories: static variable ordering heuristics (*SVO*s); and dynamic variable ordering heuristics (*DVO*s). For SVO heuristics the ordering is based on the initial state of search, for DVO heuristics the ordering is usually based on the current state of search. The advantage of using a static ordering is that the ordering only needs to be calculated once at the start of search. DVO heuristics are more costly because the heuristic has to calculate which variable is best at each choice point. However, even with this handicap, the latter generally outperform static orderings due to their ability to exploit features of the current search state.

Variables can be characterised by certain basic parameters. The *domain size* of a variable is the number of values in its current domain, and the *degree* of a variable is the number of edges connected to it (i.e. the number of constraints which have this variable in their scope). The *backward-degree* of a variable is the number of edges it shares with assigned variables, while the *forward-degree* or *fdeg* is the number of edges a variable shares with unassigned variables.

Haralick and Elliott [97] proposed selecting the variable with smallest domain

at each choice point, as this will minimize the expected branch depth. This DVO heuristic is known as the fail-first heuristic or simply as *dom*. The degree of the heuristic can be combined with the domain size either as a tie-breaker (Brélaz [31]) or by taking the ratio of the current domain to the forward-degree to create the *dom/fdeg* heuristic (Bessière and Régin [27]). The latter heuristic is often used as a baseline when comparing general purpose variable ordering heuristics. Russel and Norvig [173] describe *dom* as the most *constrained* variable, and *fdeg* as the most *constraining* variable.

A number of dual heuristics have recently been proposed which select the variable *and* value at each choice point. Zanarini [236] proposed *constraint-centered* heuristics based on solution counts for individual constraints. The solution counting algorithms are used to calculate the *solution density* (SD) of each variable-value pair (V_i, a), on each constraint. The solution density is defined as the number of solutions to the constraint involving (V_i, a) relative to the total number of solutions to the constraint. They proposed a number of heuristics based on this metric, and found the best to be simply choosing the variable-value pair with maximum SD over all constraints.

Refalo [169] proposed an adaptive search strategy known as *Impact-Based Search* (IBS). The strategy uses *impacts* to guide variable and value selection. The impact of a value is the proportional reduction in the search space after propagating the assignment of the value to the variable. More formally, let P_{before} be the Cartesian Product of the unassigned variables prior to propagation of the assignment of ($V_i = a$) and P_{after} be the Cartesian product after propagation. The impact of ($V_i = a$) is given by:

$$\text{Impact}(V_i, a) = 1 - \frac{P_{after}}{P_{before}}$$

This is quite similar to the *promise* heuristics proposed in Geelen [71], where for each variable-value pair, the promise is defined to be the product of the number of supports of the value in each unassigned neighbors domain. The promise of a variable was defined as the sum of the promise scores of the values in its domain. The heuristic selects the variable with minimum promise-sum, and assigns it the value with maximum promise.

Due to the expense in calculating the actual impact for each variable-value pair at each choice point, Refalo proposed maintaining a rolling average per value which is updated whenever the variable is assigned the value. The impact of a variable is defined, like Geelens promise heuristic, to be the sum of the average impact scores of the values in the variables domain. The heuristic then selects the variable with maximum impact-sum and assigns it the value with minimum impact.

In order to improve the initial decisions of the heuristic Refalo proposed a preprocessing step where the impacts are initialized by performing restricted singleton arc-consistency (RSAC) prior to search. For variables with large domains, one can calculate impacts for subsets of the domain to reduce the cost of preprocessing.

Correia and Barahona [49] proposed an extension of this heuristic where actual impacts are calculated at each choice point by maintaining RSAC. In order to reduce the cost of this step, they also proposed calculating actual impacts only for variables with domain size of 2, and using the impact as a tie-breaker. Cambazard proposed an augmented version of the impact heuristic which incorporates information from explanation based learning [38].

Michel and Van Hentenryck [151] proposed a similar framework for variable and value selection which they refer to as *Activity-Based Search* (ABS). Each variable has an associated *activity* counter, which is incremented whenever the domain of the variable is reduced at least once during the propagation phase of search. More formally, a CP solver applies a propagation algorithm F after a variable assignment. The enforcement of the required level of consistency by F produces a new domain store $D' \subseteq \mathscr{D}$. A set V' of affected variables are identified:

$$\begin{aligned} \forall V_i \in V' : &\quad D'(V_i) \subset D(V_i) \\ \forall V_i \in \mathscr{V} \backslash V' : &\quad D'(V_i) = D(V_i) \end{aligned}$$

The activity of variable V_i, denoted $A(V_i)$, is updated at each node of the search

tree (provided V_i is unassigned) by the following two rules:

$$\forall V_i \in \mathscr{V}\ s.t. |D(V_i)| > 1: \quad A(V_i) * \gamma$$
$$\forall V_i \in V': \quad A(V_i) \leftarrow A(V_i) + 1$$

where V' is the set of affected variables, and γ is an age decay parameter ($0 \leq \gamma \leq 1$). The latter ensures that more importance is attached to recent "activity".

They also defined the activity of an assignment $V_i = a$ as $|V'|$ after applying F to the assignment, i.e. the number of variables affected by the assignment. As with impact-based search, a rolling average is maintained for the activity scores for each value.

The heuristic chooses the variable that is most active with respect to its domain size, i.e. the variable with maximum $\frac{A(V_i)}{|D(V_i)}$. For this variable, the value with minimum average-activity is selected. Activities are initialized through a random sample of paths (a technique known as probing) from the root to a leaf node. The sample size is chosen so as to provide a good estimate of the mean activity per variable.

2.4 Restarting and Randomness

In devising general strategies for problem solving, robustness and adaptability are key. One will not necessarily know beforehand how difficult a problem set is, so a method that is equally adept at solving easy instances as hard instances is desirable. Furthermore it has been shown that even within a given problem set, there can be considerable variation in search effort required to solve different instances (Mammen and Hogg [143]).

Gomes et al. suggested that the existence of "outliers" (an event that is several standard deviations from the mean of distribution) could explain the *heavy tailed* (or *fat tailed*) distribution in the runtime of problem solving for some problem sets [80]. Outliers are not necessarily difficult problems in general, they may just be difficult in the context of the search algorithm used. One algorithm's outlier may be solved extremely quickly using a different algorithm.

The notion of fat-tailedness is based on the *kurtosis* of a distribution. The kur-

tosis is defined as $\mu_4/{\mu_2}^2$, i.e. the fourth central moment about the mean divided by the square of the variance. When the kurtosis is greater than 3 the distribution is said to be fat-tailed. Heavy-tailed distributions are non-standard distributions that have infinite moments (e.g. an infinite mean or variance). Examples of heavy-tail distributions can be found in Quasigroup Completion Problems and sports scheduling problems (Gomes [80]).

In order to avoid the heavy-tail to the right of the distribution and to take advantage of the tail to the left of the distribution, Gomes et al. proposed a form of "random" restarts for search [82]. The authors suggest that a randomized backtrack search procedure is most effective early in search. Thus a sequence of short runs may be more effective than one long run. In their "rapid randomized restarts" approach (RRR), search is repeatedly restarted after a fixed amount of time (long enough for the problem to fall into the left tail of the distribution).

Randomization is added through the tie-breaking of the heuristic's top choices. In order to ensure ties occur one may choose randomly from the top x choices, for some arbitrary x. Alternatively one may use the notion of heuristic equivalence to force ties [82], where all choices whose heuristic score is within $H\%$ of the top choices score are considered equivalent, e.g. all choices within 20% of the top choices score. The advantage of the former method is that it guarantees a tie will occur at each choice point, while the advantage of the latter is that the choice will always be one which is of good quality according to the heuristic metric. Other methods for randomizing the heuristic selection have been proposed which add a bias function to the previous two strategies, *Heuristic-Biased Stochastic Sampling* (Bresina [32]) and *Value-Biased Stochastic Samplimg* (Cicirello [48]) respectively.

The simplest method of combining randomization with restarting is the iterative sampling approach (*ISAMP*) of Langley [128]. Both variable and value selections are completely random, and a cutoff of 1 failure is used. This was shown to be extremely effective at solving job shop scheduling problems by Crawford and Baker [50] (albeit using a SAT version of ISAMP, where forward checking was replaced by unit propagation).

Although these scheduling problems contain many solutions, Crawford and Baker observed that they also contain large insoluble subspaces. An initial bad

guess by a heuristic can result in the solver being stuck in a virtually infinite search tree (on the relatively small job shop scheduling problems they tested, search trees of the order of 2^{70} nodes were found [50]). They refer to this as the *early mistake problem*. They further hypothesized that the reason for the better performance of ISAMP, compared to the local search solver *GSAT* (Selman et al. [184]), was the existence of a small number of "control" variables (those that define the problem) and a large number of "dependent" variables (those whose values are determined by the control variables).

A similar concept to the control variables has been introduced by Williams et al. [227] which they refer to as *backdoor variables*. A *backdoor* is a set of variables which, when assigned correctly, result in an underlying subproblem that a sub-solver can return a satisfying assignment to in polynomial time. A *strong backdoor* is a set of variables which, no matter their assignment, result in an underlying subproblem that a sub-solver can either return a satisfying assignment or prove infeasible in polynomial time. Williams et al. hypothesize that the existence of these backdoor variables can explain the superior performance of Gomes RRR approach on certain problem types. They found that many problems contain backdoors which constitute a small fraction of the total number of variables.

Gomes et al. formally proved that the underlying distribution of a restart strategy with a fixed cutoff eliminates heavy-tailed behaviour [83]. Luby et al. had earlier proven that when the runtime distribution is known the optimal restart strategy is a fixed cutoff strategy [140]. However, the runtime distribution of a problem set is often not known in advance. Hence Luby et al. proposed a *universal restarting strategy* for this scenario, which they proved can never be worse than a constant log factor of the optimal fixed cutoff strategy [140]. In practice, the Luby universal restarting strategy can be quite slow to converge [80]. Walsh proposed an alternative universal strategy, where the cutoff increases geometrically which was shown to outperform the Luby strategy on a number of problems [221].

Finally we note that a related area is that of *discrepancy based search* (e.g. Limited Discrepancy Search (LDS) [100] and Depth-Bounded Discrepancy Search (DDS) [220]. A discrepancy is defined as the assignment to a variable of a value which is not the "best" value according to the value ordering heuristic. In LDS, the solver first tries following the heuristic's advice, i.e. choosing the best selection

according to the heuristic (*discrepancy* 0). If this fails to produce a solution, the solver systematically attempts to find a solution path with discrepancy 1, and iteratively increments the discrepancy size allowed until a solution of that discrepancy is found or the search space has been exhausted.

For a more in-depth analysis of restarting and randomness the reader is referred to Gomes [80] and Gomes and Walsh [81],

2.5 Learning From Failure

Learning from failure has become a key component to many of the state of the art CSP (e.g. [132, 102]) and SAT solvers ([155, 192]). This form of learning can be done using both fine-grained and coarse-grained approaches. Examples of fine-grained learning are *nogood recording* [55, 123] / *explanation based learning* [119, 118] in CSPs; and *clause learning* in SAT [144]. These techniques involve learning new constraints / clauses during search which are dynamically added to the problem.

A *nogood* is a tuple (A, C'), where A is an assignment of values to a subset of the variables which cannot be extended to a solution to the problem; and C' is a subset of the constraints such that the constraints C' are sufficient to show that A cannot occur in any solution to a problem [180]. This definition has since been expanded by [123] to incorporate both positive and negative assignments, i.e. both $V_i = a$ and $V_i \neq a$.

Nogoods have been used by look-back schemes to reduce thrashing by identifying a *culprit variable* for the inconsistency which search can "jump" back to, e.g. *conflict-directed backjumping* (Prosser [163]) and *dynamic backtracking* (Ginsberg [77]), as opposed to simply backtracking to the previous variable selected. A less memory intensive technique to identify the culprit variable for backtracking is *last-conflict based reasoning* as proposed in Lecoutre et al. [134]. Rather than record an eliminating explanation for every value removal and use this information to identify the culprit variable upon backtracking, the algorithm instead selects the last variable selected prior to backtracking. This variable must partake in the conflict set for the inconsistency, and thus, by selecting this vari-

able after each sequential backtrack, search will quickly backtrack to the culprit variable.

In SAT, clause learning combined with restarting is used by most state-of-the-art solvers. In particular, it has been shown the addition of clause learning can guarantee completeness of a fixed restart strategy, provided all learnt clauses are kept (Richards and Richards [171]). However, storing all learnt clauses is impractical for many problems as the growth of the CNF formula is exponential in the worst-case (Baptista et al. [15]). To remedy this issue and to remove the possibility of revisiting search paths between restarts, Baptista et al. proposed only storing clauses learnt from the final search path when the cutoff was reached. A similar approach has been proposed for solving CSPs (Lecoutre et al. [135]).

Coarse-grained approaches use information from failure/conflict to guide subsequent search and can be incorporated into both local search strategies (e.g. Joslin and Clements [116], Müller et al. [156], Jussien and Lhomme [120]) and systematic algorithms (e.g. for variable ordering during search (Boussemart et al. [29], Karoui et al. [121]) and for revision ordering during consistency maintenance (Balafoutis and Stergiou [12])).

2.5.1 Constraint weighting methods

Constraint weighting was introduced by Morris [153] in his breakout algorithm for escaping local minima while using local search, and a similar technique was independently introduced by Selman [183] for solving large structured SAT problems. The breakout algorithm is an extension of the min-conflicts heuristic repair method (Minton et al. [152]), where the variable-value pair which minimizes the number of constraint violations is selected at each iteration in local search. This repair method can be combined with a variety of search strategies such as hill-climbing [152].

The breakout algorithm was proposed by Morris in order to avoid getting stuck in local minima when using a min-conflicts approach. The algorithm works as follows. All constraints are given an initial weight of 1. After a random initial assignment, a min-conflicts type heuristic is used which chooses the variable-value pair that minimizes the sum of the weights on violated constraints. If search becomes

stuck in a local minima, a *breakout* step is performed where the weight on all currently violated constraints are repeatedly incremented until the procedure *breaks out* of the local minimum, i.e. until a variable-value pair with a lower weighted sum than that of the current assignment is found.

Thornton investigated constraint weighting for local search algorithms in [198]. He found that constraint weighting methods worked best on structured problems which had distinct sub-groups of constraints, i.e. it is extremely effective when there is a subset of constraints which are much harder to satisfy than the rest of the constraints.

Hybrid methods have also been proposed which pass conflict information between local and systematic search strategies. Eisenberg and Faltings [60] proposed using Morris's breakout algorithm to identify hard and unsolvable subproblems. In their approach the breakout method was run for a fixed number of breakouts on a problem. If the problem was not solved within this breakout-limit, a complete search strategy was tried using the weights generated by the breakout method to guide variable ordering to insoluble subproblems. They further presented an extension which identifies minimally unsolvable subproblems, similar techniques to identify such subproblems have been proposed (e.g. Grégoire et al. [88], Hemery et al. [105]).

Vion also proposed a hybrid method combining the breakout algorithm with systematic search [212]. There are two steps to this method, in the first step the breakout method is run for a fixed number of iterations from a random initial assignment, and this is repeated a fixed number of times (x say). If the problem remains unsolved after running the breakout algorithm x times, the procedure moves to the second step involving systematic search with the weighted degree heuristic (Boussemart et al. [29]), where the weights are initialized to those learnt by the breakout method. Systematic search is run for a fixed number of backtracks, and if the problem is unsolved then the new weights are passed back to the breakout method along with other information such as nogoods learnt during systematic search, and the process repeats.

Mazure proposed interleaving systematic and local search for solving SAT problems. The main method is the systematic approach, however at each choice point local search is run on the subproblem to identify the hard elements. This

can be viewed as providing localized information, compared to the previous two strategies which used the weights on a global level.

In SAT, many state-of-the-art solvers combine restarts with both fine-grained and coarse grained failure information. For example, the systematic solvers Chaff (Moskewicz et al. [155]) and Minisat (Sörensson and Een [192]) both combine clause learning with restarts. They further use conflict information (in the form of clause weights) to guide search, the heuristics are referred to as *VSIDS* and *activity-based* in Chaff and Minisat respectively. The addition of conflict clauses through learning combined with the updated clause weights ensures that previously explored parts of the search space are not revisited upon restarting. Both solvers also employ short term memory techniques for their heuristics, where all clause weights are periodically divided by a constant.

The issue of whether clause weights, learnt during local search, are useful in a global or a local context is one which has received some attention in the SAT domain. Tompkins and Hoos [199] gave empirical evidence which suggested that clause weights generated during local search are not meaningful in themselves, but their effectiveness is primarily due to increasing diversification in the local search algorithm. However, as discussed in Ferreira and Thornton [62] the experimental setup of Tompkins and Hoos was inadequate for properly testing the hypothesis. Ferreira and Thornton provided contradictory empirical evidence that showed that performance improvement can be achieved by longer term memory, due to identifying clauses that are globally difficult to satisfy.

2.6 Weighted Degree Heuristics

Boussemart et al. [29] proposed using constraint weights during complete search as a metric for conflict-directed variable ordering heuristics. These "weighted-degree" ($wdeg$) heuristics are continuously updated during search by using information learnt from previous failures in order to guide search to the areas of most contention. The advantage that these heuristics have is that they use previous search states as guidance, while most other heuristics either use the initial state (static heuristics) or the current state. Thus the heuristic adapts to the problem.

The heuristic works as follows. All constraint weights are given an initial

value of 1. A constraints weight ($wt(C_k)$) is then incremented by 1 each time the constraint causes a domain wipeout during constraint propagation. The *weighted-degree* of a variable is the sum of the weights on the active constraints ($\mathscr{C}_A$) involving this variable, where a constraint is active if at least two variables in its scope are unassigned. The weighted degree heuristic ($wdeg$) chooses the variable with largest weighted degree.

$$wdeg(V_i) = \sum_{C_k \in \mathscr{C}_A, V_i \in Vars(C_k)} wt(C_k)$$

Current domain information about the variables can also be incorporated into the heuristic to produce the variant $dom/wdeg$ (similar to $dom/fdeg$ except it chooses the variable which minimizes the ratio of domain to weighted-degree as opposed to forward-degree). Heuristics based on $wdeg$ were shown to be extremely effective at improving search when compared to the popular fail-first heuristic $dom/fdeg$ (Boussemart et al. [29]) and when compared with backjump-based techniques (Lecoutre et al. [133]). Furthermore, Hulubei [110] has shown that combining $dom/wdeg$ with the $min\text{-}conflicts$ value heuristic can remove heavy-tails from some problem types.

Huguet et al. [109] propose an alternative to the weighted degree heuristic, which they referred to as *Wvar*. Here, the weight is stored directly with the variable, i.e. it is effectively the weighted *static* degree heuristic. The disadvantage to this heuristic is that weights due to failures of constraints which are currently entailed, are not removed as in *wdeg*. Since these constraints are entailed, they cannot cause a failure in the current search state, and thus the weighted degree method of ignoring their weight is the more logical. Nevertheless, they show that *Wvar* outperformed *wdeg* on some sets of random binary problems.

The weighted degree heuristic sits in a family of adaptive heuristic which use information from previous search states to guide subsequent search. The impact-based and activity based search heuristics also belong to this family. Although the weighted degree heuristic does not have a value ordering component, unlike those two approaches, it has been observed that the variable ordering component is much more critical for both IBS [†] and ABS [151]. Finally, we point the reader

[†]Personal correspondence with Philippe Refalo

to Chapters 9 and 10 of Lecoutre [131] for further details on both fine-grained and coarse-grained methods for learning from failure in CP.

2.7 Statistical Analysis

In this dissertation we use a number of statistical tools to analyze the results of our experiments and to test certain hypotheses. We briefly describe these below.

2.7.1 Hypothesis Testing

The *paired comparison t-test* is used to test the null hypothesis that the average of the differences between two paired samples is zero. For example suppose we solve the same set of sample instances from a population with two different methods, A and B, and we find that on average A performed better than B. The paired comparison t-test can be used to test whether the difference in the average search performance of the two methods is significant, i.e. is it likely that A will perform better than B on the rest of the instances in the population or is it more likely that the differences were due to spurious effects.

The null hypothesis for this example is that the two methods are equally good on average for the population, while the alternative hypothesis is that A is better than B on average for the population:

$$H_0\text{: } \mu_A = \mu_B$$
$$H_1\text{: } \mu_A < \mu_B$$

We then test the probability of getting the observed difference in average performance on the sample instances, given the null hypothesis. If this probability is very small we can reject the null hypothesis that both methods are equally good, and accept the alternative hypothesis that A is better than B for the population.

The t-test is useful when comparing the average performance of two methods. However, when testing whether the average performance of several methods are all equal, performing pairwise comparison t-tests for each pair of methods is time consuming and more importantly the probability of committing a “Type I”

error (where the null hypothesis is true but is incorrectly rejected) increases as the number of comparison methods increase.

In this case, *Analysis of Variance (ANOVA)* [101] is a statistical test which can be used to determine if the means of several "groups" are all equal. This can be viewed as a generalization of the t-test to more than two groups. The groups can be different levels for a set of factors.

Suppose that there are two factors A and B which we believe may influence the dependent variable. For each factor we test a (small) number of levels on the same set of n sample instances in a fully crossed design, i.e. we test each level of A in combination with all levels of B. There are R levels in factor A and C levels in factor B, and x_{irc} represents individual i of combination rc. We wish to assess whether each factor has a significant effect on the dependent variable and also whether the interaction of the two factors with each other has a significant effect on the dependent variable.

Performing a two-way fixed effects ANOVA will test the following three separate hypotheses:

- The population means for factor A are equal.
- The population means for factor B are equal.
- There is no interaction between the two factors.

There are a number of variations of the ANOVA, in this dissertation we employ both *two-way fixed effects* and *three-way mixed effects* ANOVA with *replication*. Replication means that for any combination of factors, there are at least two independent observations made under identical experimental circumstances [101].

The fixed/mixed effects refers to the type of factors under consideration. In particular, in the fixed effects model the levels of the factors are non-random, whereas in the mixed effects model some of the factors involve a random sample of a population. In the two-way (three-way) ANOVA we are not only interested in testing whether each factor has a significant effect on the dependent variable, but also whether there is a significant interaction between the two (*resp.* three) factors.

Table 2.1: Two-way fixed-effects ANOVA summary

Source	SS	df	MS	F
Rows	$nC\sum_{r=1}^{R}(\mu_r - \mu_G)^2$	$R-1$	$\frac{SS_{Rows}}{df_{Rows}}$	$\frac{MS_{Rows}}{MS_{Err.}}$
Columns	$nR\sum_{c=1}^{C}(\mu_c - \mu_G)^2$	$C-1$	$\frac{SS_{Col.}}{df_{Col.}}$	$\frac{MS_{Col.}}{MS_{Err.}}$
Interaction	$n\sum_{c=1}^{C}\sum_{r=1}^{R}(\mu_{rc} - \mu_r - \mu_c + \mu_G)^2$	$(R-1)(C-1)$	$\frac{SS_{Int.}}{df_{Int.}}$	$\frac{MS_{Int.}}{MS_{Err.}}$
Error	$\sum_{c=1}^{C}\sum_{r=1}^{R}\sum_{i=1}^{n}(x_{ijk} - \mu_{rc})^2$	$RC(n-1)$	$\frac{SS_{Err.}}{df_{Err.}}$	

Notes: $\mu_{r/c}$ is the mean of the data in row r / column c
μ_{rc} is the mean of the data in row r and column c. μ_G is the grand mean over all data.

For the two-way fixed effects ANOVA we have an orthogonal design, where we have a data table consisting of a row for each level of factor A and a column for each level of factor B. Each cell has n observations, each row has nC cases and each column has nR cases.The total sum of squares can be partitioned into sums of squares for four components:

$$SS_{total} = SS_{Rows} + SS_{Columns} + SS_{Interaction} + SS_{Error}$$

whose calculations are summarized in Table 2.1, along with calculations for the degrees of freedom, mean square and the F ratio for the four components. The F ratio along with the degrees of freedom of its two components is then used to accept/reject the associated null hypothesis using a table of the F distribution. For more information on analysis of variance, the reader is directed to [148, 101].

2.7.2 Distribution Analysis and Correlations

The *Gini* coefficient [75] is a measure of the inequality in the distribution of a variable. It is commonly used by economists to measure the inequality of income in a population. The Gini coefficient is based on the *Lorenz* curve, mapping the cumulated proportion of the total summation of values for the variable over the bottom $x\%$ of the population when ranked by their value. In economics this in-

volves mapping the cumulated proportion of income y earned by the bottom $x\%$ of the population (see figure 2.2).

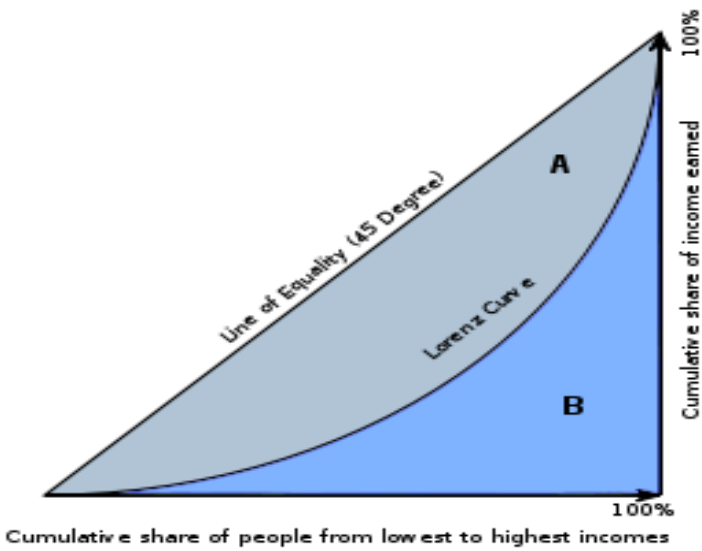


Figure 2.2: Gini coefficient example: distribution of income. (Source `www.wikipedia.org/wiki/Gini_coefficient`)

The Gini coefficient, G, is the ratio of the area lying between the Lorenz curve and $x = y$, over the total area below $x = y$, i.e. A/(A+B) in Figure 2.2. For data ordered in increasing size, this is given by the following equation:

$$G = \frac{\sum_{i=1}^{n}(2i - n - 1)x_i}{n^2\mu} \tag{2.1}$$

where n is the size of the population, μ is the mean of the population values, and x_i is the ith value in the ordered data.

When the distribution is perfectly fair (for income distribution this would mean each household earns exactly the same amount), the Lorenz curve is $x = y$ and so the Gini coefficient is 0. The other extreme is a theoretical maximum of 1, where in an infinite population all but one individuals are of size zero. In the income distribution example this would mean that all the income is earned by just one household.

Correlations

The *Pearson product-moment correlation coefficient*[148], r, is a measure of the correlation or linear dependence between two variables, X and Y. It is calculated for a sample for the two variables as follows:

$$r = \frac{1}{n-1} \sum_{i=1}^{n} \left(\frac{X_i - \bar{X}}{\sigma_X} \right) \left(\frac{Y_i - \bar{Y}}{\sigma_Y} \right) \tag{2.2}$$

where $\frac{X_i - \bar{X}}{\sigma_X}$ is known as the standard score, $\bar{X}$ is the sample mean and σ_X is the sample standard deviation of variable X.

The coefficient ranges from -1 to 1. Either of these extremes mean that there is a perfect (inverse for -1) correlation between the two variables, i.e. they are functionally related to each other and follow a linear rule. A value of 1 means that there is a linear equation for which X increases as Y increases. A value of -1 means that there is a linear equation for which X increases as Y decreases.

In certain cases, rather than the correlation of the actual scores for individuals of the sample (of size n) for the two variables, one wishes to assess the level of agreement between the *rankings* of the individuals. For example, one could rank the scores from 1 to n, with the highest score receiving the top rank and the lowest score receiving the bottom rank. The Spearman rank correlation coefficient (ρ) [148] can then be used to measure the correlation between the two rankings.

$$\rho = 1 - \frac{6 \sum_{i=1}^{n} D_i^2}{n(n^2 - 1)} \tag{2.3}$$

where D_i is the difference between the two ranks for individual i and n is the number of individuals in the sample. If there is a tie in the score for a set of individuals, then these are assigned the average of the ranks they would have received with lexico tie breaking.

Like r, the value of ρ ranges from -1 to 1, where a value of 1 means that the individuals were ranked identically, whilst a value of -1 means that the individuals had the opposite ranking for the two variables. A value of 0 means that there is no

correlation between the two rankings.

The Spearman rank correlation does not discriminate between agreement on top and bottom rankings. In our work we are more concerned with agreement amongst the top ranked choices. The *top-down rank correlation coefficient* [112] is a weighted correlation statistic which places more importance on agreement amongst top ranked individuals than on the bottom ranked individuals. The top-down correlation coefficient, r_T, is calculated similarly to ρ. However, rather than use the rank of the individuals, a score is computed for each.

The scoring system used is that of Savage [179], referred to as *savage* scores. The savage score, S_i, is computed as follows:

$$S_i = \sum_{j=i}^{n} 1/j \tag{2.4}$$

where i is the rank of the ith ordered statistic in a sample of size n. If individuals are tied, the average savage score is used. The Pearson product-moment correlation coefficient is then calculated on the savage scores of the individuals. If there are no ties, this formula reduces to the following:

$$r_T = \frac{\left(\sum_{i=1}^{n} S_{X_i} S_{Y_i} - n\right)}{(n - S_1)} \tag{2.5}$$

The maximum value that r_T can take is 1, which is the case where the rankings are identical. However, unlike Pearson's r and Spearman's ρ, the minimum value is not -1 (for $n \geq 3$), but increases away from -1 to approximately -0.645 as $n \to \infty$ [112].

Chapter 3

Probing for Failure

3.1 Introduction

In the book "Elements of Machine Learning" [129], Langley defines learning as "*the improvement of performance in some environment through the acquisition of knowledge from experience in that environment*". This is a broad definition which covers the methods we propose.

When devising learning strategies for problem solving there are some key questions that one should consider. Denzinger et al. [58] outlined 9 issues which they propose should be addressed when attempting to learn from previous proof experience. The first group of issues deals with the learning phase:

- Whom and what to learn from?
- What to learn?
- How to represent and store the learned knowledge?
- What learning method to use?

The second group of issues deals with the use of information learnt:

- How to detect applicable knowledge?
- How to apply knowledge?

- How to detect and deal with misleading knowledge?
- How to combine knowledge from different sources?

And the ninth, final issue they raise is which concepts of similarity are helpful. This is an important issue which extends to most machine learning paradigms. Normal machine learning approaches assume that there are certain characteristics common to all problems of a particular type and in turn that these characteristics can be exploited to improve performance. For example, in case-based reasoning [1] one assumes that an algorithm which works well on a specific type of problem, will work equally well on problems which share certain characteristics to this problem. Identifying these characteristics and their interaction with each other can be quite a challenge for general purpose problem solvers.

An adequate training set is vital for such learning procedures. However, due to the noted variation in problem difficulty within constraint satisfaction problem sets ([143], [68], [82]), it can be quite hard to learn information from a subset of problems which will work on all other problems in the set. Within-problem learning does not suffer from this issue as it adapts to each problem. It has the advantage that the "training set" is also the "testing set", i.e. the current problem, and thus more specific information can be learnt.

In this work we use within-problem learning to identify sources of contention in a problem. Boussemart et al. have proposed the *weighted degree* variable ordering heuristic [29]. The heuristic retains the feature of generality for problem solving while adapting to the specific problem it is being tested on. The purpose of the heuristic is not to identify characteristics of a problem which can be generalized to improve search performance on other problems, but to guide search to areas of the search space which are likely to be sources of contention within the context of the current state of search. Thus the ordering it places on variables is constantly evolving. The information learnt is obsolete when one moves on to a different problem, however the approach used in generating the information remains a viable guidance tool for adapting to a new environment.

For the work described here, the answers to the first group of issues given at the beginning of the section are as follows:

- The approach learns from failure, in particular failures during consistency checking, i.e. domain wipeouts.
- The aim is to learn which variables are sources of global contention in a problem.
- The learned knowledge is represented in the form of constraint weights. Each time propagation of a constraint causes a failure during consistency checking, the constraint's weight is incremented.
- The learning method used is constraint weighting combined with restarted search in order to generate a global weight profile for the problem.

We will go into more detail in the following sections regarding the issues outlined. The second group of issues will be dealt with in the discussion section of this chapter and in detail in Chapter 4. For the moment we will just address the issues of the application of the learned knowledge and the quality of this knowledge:

- The information learnt is used by a weighted degree heuristic to select the most contentious variables at the beginning of search.
- The expected quality of the information learnt in singularity is quite weak. A failure may occur because of a series of poor selections (*local* source of difficulty); or it may represent a more fundamental issue in the problem (*global* source of difficulty). However weights are only incremented by small amounts, it is through the buildup of these weights that import is attached to a variable, i.e. through a variable's constraints consistently causing failure.

The rest of the chapter is organized as follows. The next section provides the motivation for this work. Section 3.3 introduces two methods for improving the performance of the weighted degree heuristic, both of which employ a fixed number of restarts in an information gathering phase, followed by a run to completion. In section 3.4, we provide an empirical analysis on both structured and random problem instances. We also investigate the effects of varying the restart and cutoff parameters for both methods. Section 3.5 analyzes the quality of information

produced by the strategies during their respective information gathering phases, followed by a discussion of issues arising out of this research. The final section provides a summary of the chapter.

3.2 Motivation

The motivation for this work came from observations regarding two different topics which have been the focus of much recent research. The first topic is that of randomized restarting approaches [82, 93], and the second topic is that of adaptive heuristics which use information from previous search states to guide subsequent search [29, 169].

Randomized restarting approaches have been shown to be extremely effective when solving problems with a heavy-tailed distribution (Gomes et al. [82]) and problems with a small world topology (Walsh [221]). Randomization is usually added by randomizing the search ordering heuristics, where one randomly selects from a subset of the heuristic's top choices. In this case one maintains a degree of confidence in the selections made while still allowing for search diversification. These approaches continue to be widely studied, e.g. Guddeti and Choueiry [93], Wu and van Beek [232].

However randomized restarting techniques do not take advantage of having already sampled the search space. Furthermore, the approaches assume that the heuristic being randomized is a 'good' heuristic for the problem to be solved. If this is not the case then randomly choosing from the heuristic's top choices may not improve matters greatly. One would expect that these approaches could be improved by learning from their failed search attempts.

The second topic of research involves search heuristics that use information from previous search states to guide subsequent search. An example of this type of heuristic is the aforementioned weighted degree heuristic ($wdeg$) [29], which chooses variables based on their participation in failure during search. A variant $dom/wdeg$ of this heuristic, which incorporates domain information, has been shown to be extremely robust at solving both insoluble and soluble problems [29]. It can solve "easy" instances as quickly as $dom/fdeg$, while for harder instances it adapts to the search space it is in by prioritizing troublesome variables.

However, as Smith showed in her work on optimal static orderings [188], the first few variable selections can have a huge impact on the size of the search tree encountered when finding a solution. A bad choice at the top of the search tree can have disastrous consequences for search effort required to solve the problem. Furthermore, Harvey and Ginsberg [100] state that for many problems "heuristics are *least* reliable early in the search, before making decisions that reduce the problem to a size for which the heuristics become reliable".

Refalo outlined three principles for reducing search effort [169], the first two deal with variable ordering (choose the variable that maximally constrains the rest of the search space) and value ordering (choose the value that maximizes the number of possibilities for future assignment). The third, and final, principle concerns making good choices at the top of the search tree. Refalo suggests that applying this final principle means that some preprocessing must be done if we wish to identify the best starting point for search. This principle is of utmost importance to the work discussed in this dissertation.

Yet the weighted degree heuristic has the least information when making initial selections at the top of the search tree. In fact the heuristic has no weight information, other than the degrees of the variables, until at least one failure has occurred. It would seem logical to conclude that the heuristic can be improved upon by providing it with more information for those early choices.

In their work on Squeaky Wheel Optimization Joslin and Clements [116] found that, although identifying "difficult" elements through static analysis of the problem may be possible in some cases, interactions between constraints can be quite complex. Often it is only through search that these interactions can be identified. Joslin and Clements also found that *globally* difficult elements, i.e. those that are difficult across large parts of the search space, tend to be identified over time.

In Huang [108], a study of combining restarts with clause learning and clause weighting in SAT, the author comments that when the solver is run for a certain amount of time, it accumulates information in the form of learned clauses and clause weights. This information reflects the solver's current belief regarding the order in which future decisions should be made. However the solver is bound by the decisions made earlier so, unless it restarts, it will not be free to use the infor-

mation in the best manner. This applies equally to the weighted degree heuristic.

Both restarting and the weighted degree heuristics are employed to combat the effects of thrashing, where large insoluble subtrees are traversed systematically, repeatedly searching over variables which are independent of the cause of failure. Restarting, combined with some form of randomization, allows one to jump out of (possibly) unpromising search trees by restarting search with a different variable ordering. The weighted degree heuristic avoids thrashing by moving contentious variables up the ordering. The drawbacks to both the weighted degree heuristic and to randomized restarting approaches could be dealt with by combining constraint weighting with restarting. Here the constraint weights learnt are often sufficient to guarantee that the same search tree won't be explored upon restarting.

The above observations led us to the following two questions. Firstly, can we improve search by performing a minimal amount of preprocessing in order to learn about the search space, in particular which variables are the main search bottlenecks? Secondly, how should one go about gathering this information and at what cost? These are two questions which we aim to answer in this chapter.

3.3 Initial Methods

We devised two different approaches for boosting the weighted degree heuristic. Both approaches combine restarts with constraint weighting for information gathering. Search is run until either the problem has been solved or a fixed cutoff has been reached. This is repeated until either the problem has been solved or a fixed number of restarts has been reached. Constraint weights are carried along from one run to another. After reaching the fixed number of restarts (information gathering phase) the cutoff is removed (increased to ∞). A final search attempt is then made using the information learnt from the previous runs, and stops when a solution has been found or the problem has been proven insoluble (solving phase). Thus both methods are complete. More formally, for each cutoff t_i:

$$t_i = \begin{cases} t_{init} & \text{if} \quad i < R \\ \infty & \text{if} \quad i = R \end{cases}$$

where t_{init} is the fixed cutoff and R is the fixed number of restarts.

The first approach, which we call WTDI (for WeighTeD Information gathering), uses the weighted degree heuristic for selecting variables in both the information gathering phase and the solving phase. Combining a weighted degree heuristic with this form of restarting is expected to have the following results:

- Easy instances of the problem set are likely to be solved prior to the final run to completion (given a sufficient cutoff), as contentious variables are moved to the top of the ordering after restarting.
- Hard instances are expected to be solved more efficiently on the run to completion due to improved initial selections based on weights learnt in the previous runs.

With regard to the first expectation, upon each restart the weighted degree heuristic has more information available from which to guide search. Thus, in theory, each successive run should improve upon its predecessor. However this assumes that the quality of information learnt is uniform which is not necessarily the case.

The second approach, which we call RNDI (for RaNDom Information gathering), combines a form of *iterative sampling* [128] with information gathering. In its original form, iterative sampling involves selecting variables and values randomly during search until a failure occurs (so a cutoff of 1 failure), at which point the algorithm restarts. Iterative sampling is an incomplete approach, involving no learning, which is only applicable to problems which have a large number of solutions [50].

A number of recent approaches have incorporated a random probing phase for learning prior to search. Lombardi et al. [139] proposed using random probing (which they refer to as "diving") to extract information regarding the feasibility of individual variable-value assignments. For each variable-value assignment they store the average dive-depth involving the assignment, and the number of occurrences of the assignment in the feasible partial assignments stored for each dive.

Stamatatos and Stergiou [193] used a random probing technique to extract information regarding the activity of a constraint in reducing the search space. The

information recorded for each constraint during random probing (e.g. the number of times revision of the constraint reduced a variables domain versus the number of times the constraint was revised, the number of values removed by revision of the constraint, etc.) is then used to decide on the strength of the propagator to be applied to the constraint during search.

In our version (RNDI), during the information gathering phase variables are selected randomly (according to a uniform probability distribution over the unassigned variables). The cutoff is larger than one failure since the purpose of these "random probes" of the search space is to gather information regarding sources of contention in each individual search tree. Unlike iterative sampling, this random probing phase is mainly an information gathering step, albeit with the possibility of solving the problem in this phase. On the run to completion a weighted degree heuristic is used for solving the problem (with the weights learnt from these random probes guiding early selections).

As discussed earlier we believe that many problems contain elements which are *globally difficult*. A constraint with a high weight after many restarts is likely to have been a source of contention in different parts of the search space and thus should be more representative of global difficulty. Joslin and Clements suggested that globally difficult elements are likely to be identified over time [115].

Despite their similarities, the two approaches represent fundamentally different strategies for learning. WTDI continuously evolves as it learns, aiming to solve the problem before the final restart. RNDI, on the other hand, is intended to get as varied a sample of the search space as possible in order to provide a more 'global' representation of the spread of contention in the problem.

The first approach, WTDI, contains the benefits of restarting randomized search (i.e. intelligently traversing different areas of the search space) while learning from its failures in order to improve subsequent search. Gomes refers to this type of search (learning combined with restarting) as 'deterministic randomization' [80] in that the search is deterministic, but behaves in so complex a manner as to appear random.

In his work on randomized restarting strategies for SAT, Zhan found that most problems can either be solved very quickly (i.e. they are suited to rapid restarts) or require intensive search exploration [237], and the incorrect choice of algo-

rithm (with/without restarting) can have disastrous consequences for the amount of search effort required to solve a problem. WTDI attempts to strike a balance between intensive restarting and intensive search exploration.

The second approach, RNDI, has more in common with Refalo's impact-based heuristic work [169] in that the random runs are used mainly as a preprocessing step for gathering information from a diverse sample. It is not expected that the problem will be solved during preprocessing.

These approaches may seem to be two extremes, one using an informed search strategy, the other using an uninformed strategy. However if a portfolio of "good" heuristics were used for the information gathering phase, then the weights learnt would be biased. In particular it is of interest to consider the minimum and peak failure depths when assessing such learning strategies. Normally a few variables will be assigned before a failure occurs. Thus the first few variables selected will receive little if any weight. Most weight will be accrued on variables below the peak failure depth. If one were to use different 'good' heuristics for each probe, their top choices would rarely receive any weight.

Algorithms 2 and 3 are a general pseudo description of the restarting algorithms which we propose. Algorithm 2 outlines the overall $Solve$ function while Algorithm 3 is the $Search$ function. The algorithms given are for solving a binary constraint satisfaction problem while maintaining arc-consistency, (line 9 of Algorithm 3) with d-way branching (line 7 of Algorithm 3).

The $Solve$ function takes as input a CSP, a fixed cutoff-limit L in terms of nodes or failures, a fixed number of restarts R, and value and variable heuristics. In the algorithm given, the cutoff is in terms of nodes, however this can easily be altered for a cutoff in terms of failures. In this case a failure counter is added which is incremented in the consistency method whenever the function $Consistent$ returns false.

By fixing the cutoff in terms of nodes we ensure that the same proportion of the search space is explored in each run. If we were to use a cutoff that varied in size from run to run, the information learnt would be biased to runs which had a large cutoff. This may obscure sources of global contention compared with local sources of contention. Furthermore a time-based cutoff would not guarantee that learning occurred on each run, in which case our deterministic restarting approach

Algorithm 2: $Solve$ function for RNDI/WTDI

Input : A CSP in the form $\mathscr{V}$, $\mathscr{D}$, $\mathscr{C}$, cutoff-limit L, maximum number of restarts R, variable heuristic varH, value heuristic valH
Output: `solution` **or** `insoluble`

```
1  restarts ← 0
2  while restarts < R do
3      nodes ← 0
4      solution ← Search(𝒱, 𝒟, 𝒞, {}, L, varH, valH)
5      if solution ≠ Cutoff then
6          if solution = false then
7              return insoluble
8          else
9              return solution
10     else
11         restarts++
12 if restarts = R then
13     solution ← Search(𝒱, 𝒟, 𝒞, {}, ∞,
       SelectDomWdegVariable, valH)
14 if solution ≠ false then
15     return solution
16 else return insoluble
```

(WTDI) would merely repeat the previous run.

The only difference between WTDI and RNDI is in the variable heuristic used for probing (line 4 of Algorithm 2). In both our methods, there is a fixed maximum number of restarts, R (which gives a maximum number of runs $R + 1$). During each run, search continues until the problem has been solved or L nodes (failures) have been searched (*resp.* encountered), while updating weights after constraint violations.

On runs 1 through R, if no solution has been found after L nodes/failures, the function $Search$ is again called on the initial state of the problem with updated weights. On the $(R+1)$th run, the cutoff is removed (set to infinity) and search runs to completion (line 13 of Algorithm 2). On this run, a conflict-directed heuristic

Algorithm 3: *Search* function for RNDI/WTDI

Input : Set of unassigned variables `futureVars`, their current domains $\mathscr{D}$, set of constraints $\mathscr{C}$, node-limit L, variable heuristic *varH*, value heuristic *valH*

Output: Cutoff **or** `assignment` **or** *false*

```
1  if futureVars = ∅ then
2      return assignment
3  if nodes ≥ L then
4      return Cutoff
5  V_i ← Select variable from futureVars according to varH
6  futureVars ← ( futureVars − V_i )
7  while D_i ≠ ∅ do
8      d_i ← Select value from D_i according to valH
9      consistent ← AC (V_i ← d_i, assignment, C)
10     if consistent = true then
11         assignment ← ( assignment ∪ (V_i ← d_i) )
12         nodes++
13         result ← Search(futureVars, D, C, assignment, L,
           varH, valH)
14         if ¬result then
15             assignment ← assignment − {V_i ← d_i}
16             D_i ← D_i−{d_i}
17         else if result = Cutoff then
18             return Cutoff
19         else
20             return assignment
21     else
22         assignment ← assignment − {V_i ← d_i}
23         D_i ← D_i−{d_i}
24 return false
```

such as $dom/wdeg$ or $wdeg$ is used for variable selection.

3.4 Experimentation

In this section we provide experimental analysis of both WTDI and RNDI on two different types of problem. We compare these approaches with complete search approaches using $dom/fdeg$ and $dom/wdeg$. The first type of problem is one with clearly defined sources of global contention. The second type is one where the sources of difficulty are less obvious. We also analyse the impact of varying the restarts and cutoff factors for both approaches on a sample problem set.

In the following experiments the base algorithm used was a MAC-3 algorithm with d-way branching, value selection was lexical and no arc-ordering heuristics were used. To avoid confusion, we will refer to search using $dom/wdeg$ without restarts as "$dom/wdeg$-$nores$". For all experiments using the RNDI approach, the results reported are the averages of ten experiments. This was done to obtain adequate samples under randomization and to avoid spuriously good (or bad) effects due to random selections. All experiments, for which runtime results are given, were run on an Intel Xeon 2.66GHz machine with 12GB of RAM on Fedora 9.

3.4.1 Problems with embedded insoluble cores

We first provide results for some obvious cases where these two methods are appropriate, namely on problems with embedded insoluble subproblems. It should be noted that Hemery et al. have independently proposed using the weighted degree heuristic with restarts as part of a process for extracting minimial unsatisfiable cores from constraint networks [105]. Eisenberg and Faltings also used constraint weights to identify insoluble subproblems [60], their approach however used constraint weights produced by the breakout (local search) algorithm [153]. There has also been a focus in the SAT community on methods for identifying minimal unsatisfiable subformulas, see for example Grégoire et al. [88] and Liffiton et al. [138].

For these types of problem the smallest proof of insolubility occurs when the variables in the insoluble subproblem are selected first in search. However check-

ing whether a set of constraints form a minimal insoluble core is DP-complete in itself [160]. By restarting and collating information regarding failures in search, we hope to identify these globally difficult elements.

Case Study: Queens-Knights Problem

An academic example of such a problem is the queens-knights problem. This combines the problem of putting n queens on an n x n chessboard so that no two queens can attack each other, i.e. no two queens share a column, row or diagonal; and the problem of putting r knights on an n x n chessboard such that the knights form a cycle in knight moves. When the number of knights is odd the problem is insoluble, so the knights form an insoluble core of the queens-knights problem.

There are two formulations of the queens-knights problem. In the first the queens and the knights can share a cell (i.e. there are no constraints between the queens and the knights); and in the second they cannot share a cell. The first type is referred to as qk-n-r-add and the second qk-n-r-mul.

For the qk-n-r-add instances, each of the n queens has initial domain size n and has degree $(n-1)$, while each of the r knights has initial domain size n^2 and degree $(r-1)$. There is a predefined ordering on the knights, each knight has a constraint defining the knight move to its predecessor and a constraint defining the knight move to its successor in the order. (The other $(r-3)$ constraints are not equals constraints with the rest of the knights). The optimal search effort occurs when a knight is selected first, requiring exploration of n^2 search nodes, i.e. every value for that knight is tried and found invalid by MAC.

Selecting a queen first and then a knight increases the number of nodes to n^3 (n^2 failures for each value in the domain of the queen). The $dom/fdeg$ heuristic will perform an all-solutions search on the queens problem. Each time it finds a solution to the queens problem it will then select a knight, producing n^2 failures, before backtracking. This is an extreme case of thrashing.

The approach dom/$wdeg$-$nores$ will act identically to search with $dom/fdeg$ until the weighted degree on a knight is large enough to produce a smaller ratio of domain size to $wdeg$ than for any currently uninstantiated queen. It will then pick this knight and backtrack once all n^2 values have been tried. The knight(s) will

then be selected after each backtrack, moving up the ordering, until one has been tried with each of the remaining values for the first queen selected.

However there are two points which should be noted here. Firstly, when a knight is chosen the weight on *its* constraints may not be increased, instead a constraint between two other knights may repeatedly cause failure so their weighted degrees will each increase. This is because the constraints of the instantiated knight are propagated first, followed by the constraints of its reduced knight neighbors. Thus the next knight chosen may be a different knight.

This means that the weight will be spread between (some of) the knights, reducing the likelihood of a knight being selected ahead of a queen early in search. This effect can be seen in the results on knights instances (without queens) of Boussemart et al. [29]. Their results, using binary branching, show that $dom/fdeg$ solved the instances in the minimal refutation (because it kept choosing the same knight after each failure) while the weighted degree heuristics required more nodes to solve it. This was because the heuristic led search to jump between different knights upon failure.

A second point regarding the queens-knights problem is that, for each queen instantiated, the domains of all the uninstantiated queens are reduced further. This makes it less likely that a knight will be selected ahead of a queen deep in search. It is probable that search will have to backtrack almost to the top of the search tree before a knight will be selected.

The two methods we propose behave as follows. The random probing approach builds up the weight on the knights in the preprocessing phase (weight is rarely assigned to a queen) so that a knight will be selected first on the final run (provided sufficient weight is accrued). Similarly WTDI will keep restarting until sufficient weight has been attached to the constraints of a knight such that the knight is selected at the top of the search tree, provided R and C are large enough.

Results for a sample of queens-knights instances are given in Table 3.1, comparing $dom/wdeg\text{-}nores$ with our two approaches. All instances were taken from the benchmark website of the 2005 CP solver competition[†]. The same restart-cutoff (10 restarts, 200 node cutoff) combination was used for all RNDI experiments, a different restart-cutoff combination (10 restarts, 1000 node cutoff) was

[†]http://cpai.ucc.ie/05/Benchmarks.html

used for all WTDI experiments. Results for $dom/fdeg$ are not presented as it failed to solve any of these instances with a 2 million search node limit.

As one can see in Table 3.1, both restarting approaches achieve large gains in search performance over $dom/wdeg$-$nores$, with order of magnitude differences for large n across all three metrics of evaluation. Furthermore both approaches always solved the instance in the minimum refutation size (n^2) on the final run, i.e. the run when the instance was solved.

There are some points of note regarding cutoff values for WTDI. If the cutoff is so low that no wipeouts occur on the first run, then the same search tree will be explored on each subsequent run. Furthermore the cutoff must be large enough ($> n$) for the approach to reach the insoluble core. Finally, if one hopes to solve the problem prior to the run to completion, the cutoff must be greater than the optimal refutation for the problem set ($> n^2$) .

Table 3.1: Results For Instances of the Queens-Knights Problem

n-r		12-5		15-5		20-5		25-5	
type		add	mul	add	mul	add	mul	add	mul
	$dom/wdeg$	8.7K	20.3K	23.7K	30.0K	50.2K	55.6K	117.0K	112.6K
Nds	WTDI	1144	2144	2225	2225	2400	1400	2625	3625
	RNDI	744	744	2225	2225	2400	2400	2625	2625
	$dom/wdeg$	89M	307M	561M	1.0B	3.8B	4.5B	21.6B	21.5B
Cks	WTDI	12M	38M	50M	79M	149M	130M	557M	751M
	RNDI	6M	8M	45M	52M	140M	168M	309M	395M
	$dom/wdeg$	3.30	9.17	20.99	31.54	144.2	166.1	810.1	781.4
T(s)	WTDI	0.43	1.01	1.82	2.33	5.8	4.3	18.9	25.7
	RNDI	0.27	0.31	1.92	2.09	6.1	6.7	14.9	16.0

Notes: Total Search Nodes, Constraint Checks and Time.
Restarts and cutoff-limit → RNDI: 10R 200C; WTDI: 10R 1000C

Therefore WTDI needed a larger cutoff than RNDI for these instances. (Note that since RNDI uses random variable ordering, the probability of it weighting a variable in an insoluble core is quite large for a given probe provided the cutoff is within reason.) For example on the qk-25-5 instances, WTDI solved the problem in 2 restarts on qk-add and 3 restarts on qk-mul with a cutoff of 1000 nodes. For

the same instances RNDI identified the knights as the source of contention with 10 restarts and a cutoff of 200 nodes.

For the RNDI approach on the 12 queens instances, most of the experiments were solved during the random probing phase. This is because the minimal refutation for these two instances is 144 nodes, which is less than the cutoff. Thus when a knight was randomly selected first the instance was solved during the probe. The likelihood of a knight being selected first on a run is $r/(r+n)$, here 5/17.

Clearly we could have optimized the parameters for both WTDI and RNDI to each instance, however we were merely concerned with showing that large savings can be made with a weighted degree heuristic by performing a small amount of preprocessing.

Further results for problems with insoluble cores

The next two problem sets are also taken from the benchmark website of the 2005 CSP Solver Competition[†], (two sets of random 3-sat instances); the last problem set was generated using Richard Wallace's generator. The random 3-sat instances were originally proposed in SAT format by Bayardo [117]. They are 2 classes of easy random 3-sat instances, each with an embedded unsatisfiable subproblem which were thought to make them exceptionally hard instances (hence the problem name ehi-$*$).

These instances were converted to CSP format by Bacchus [9] using the dual encoding method [10]. The first class, *ehi-85*, has 100 instances, each containing 297 variables (85 in the original SAT instances). The second class, *ehi-90*, also has 100 instances, each containing 315 variables (90 in the original sat instances).

The last problem set reported in Table 3.2 is a set of 100 "composed" random instances which were generated by Richard Wallace. These consist of a main under-constrained component in the form $\langle n, d, m, t\rangle$ where n is the number of variables, d the uniform domain size, m the graph density of the component and t the uniform constraint tightness; and k satellite components also in this form attached by links of density m, tightness t. For these instances, the main component was $\langle$100,10,0.15,0.05$\rangle$, there were 5 satellites, all $\langle$20,10,0.25,0.5$\rangle$, and links

[†]http://cpai.ucc.ie/05/Benchmarks.html

were $\langle 0.012, 0.05 \rangle$.

Table 3.2: Results For Unsatisfiable Embedded Problems

		$dom - fdeg$	$dom - wdeg$	WTDI	RNDI
ehi-85-297	Solved	61%	100%	100%	100%
	Final Run Nodes	-	-	14.2	21.1
	Total Nodes	$>$ 63K	1160.3	224.2	170.2
	Total Checks	$>$ 101M	2.96M	0.67M	1.63M
	Total Time	-	3.30	0.65	1.27
ehi-90-315	Solved	58%	100%	100%	100%
	Final Run Nodes	-	-	30.0	23.5
	Total Nodes	$>$ 58K	1008.6	252.0	172.8
	Total Checks	$>$ 105M	2.9M	0.84M	1.72M
	Total Time	-	3.31	0.86	1.39
composed	Solved	0%	100%	100%	100%
	Final Run Nodes	-	-	45.96	133
	Total Nodes	-	13953	426	1133
	Total Checks	-	13M	0.5M	1.2M
	Total Time	-	14.57	0.41	1.03

Notes: Results are averages per instance. $dom/fdeg$ had a cutoff of 100,000 nodes in the above experiments. RNDI had restarting regimens of R=10, C=20 for ehi instances and R=20, C=50 for composed. WTDI had restarting parameters R=10, C=200 for all problem sets.

The results of Table 3.2 once again show that large gains can be made over $dom/wdeg$-$nores$ by performing a small amount of preprocessing. As expected, $dom/fdeg$ performed extremely poorly on these problem sets, failing to solve any of the composed instances within the cutoff. These problems are unsuited to $dom/fdeg$ as the variables in the insoluble core generally do not have large degrees nor small domains. Thus $dom/fdeg$ suffers from large amounts of thrashing without ever identifying the source of the thrashing, i.e. the problems in the insoluble core.

It is interesting to note that even though $dom/wdeg$-$nores$ was extremely efficient at solving the ehi problem sets (roughly 1000 nodes on average per instance), the two restarting approaches were still able to improve on it. WTDI solved most

instances prior to the run to completion. In fact it averaged less than two restarts per instance for all 3 problem sets.

RNDI solved many of the ehi instances during the random probing phase even with its short cutoff (nearly 40 instances in both sets on each run), averaging less than 9 restarts per instance for both ehi sets. However it failed to solve any of the composed instances prior to the run to completion.

Both RNDI and WTDI have advantages and disadvantages. For these problems, RNDI is likely to choose a variable in an insoluble core earlier than WTDI (since WTDI follows the search of $dom/fdeg$ up until at least one failure occurs). However, once WTDI discovers an insoluble core, the weight thereafter will be concentrated on variables in this core. On the other hand, the weight will be spread across all insoluble cores with RNDI. This may lead to "jumping around" between insoluble cores on the run to completion, which clearly would hinder search.

3.4.2 Analysis of various restart-cutoff combinations for RNDI and WTDI

We now provide detailed analysis of the effect on WTDI and RNDI of varying both the restart and cutoff factors on a given problem set. The problem set chosen is a set of random binary problems which take the form $\langle n, d, m, t\rangle$ as defined earlier. The problem parameters are based on one of the problem sets used by Boussemart et al. in their study of the weighted degree heuristic [29].

The parameters were $\langle$200,10,0.0153,0.45$\rangle$, the instances were generated by Richard Wallace. The only difference in the method of generation from that of Boussemart et al. is that each of the present instances consist of a single connected component. This was done by first connecting the nodes of the constraint graph successively to make a (random) spanning tree and then adding edges at random until the required number had been chosen.

The instances are of type Model B, as discussed in [81]. From the original set, 100 soluble and 100 insoluble instances were selected so that they could be tested separately. The purpose of this was to test if an approach was better suited to one class or the other.

For both approaches, the cutoff levels tested were 250, 500, 1000, and 2000

nodes. The number of restarts tested were 1, 5, 10 and 20. These *factors* were "fully crossed" in the experiments, so that all values of the cutoff factor were tested with all values of the restart factor. Thus, all combinations of factor-levels were tested. In the case of RNDI, each condition, i.e. combination of values for cutoff and restarts, was again tested ten times and the average of these ten experiments is reported.

Results were analysed with the analysis of variance (ANOVA), in some cases followed by comparisons of individual means using paired-comparison t-tests [101]. For experiments with WTDI (next sub-section), the ANOVA was based on a two-way fixed effects model. Experiments using RNDI (subsequent sub-section) were analysed with a three-way mixed effects model in which problems was a third factor (and each cell of the design had ten data points based on the ten tests for the instance), as well as a two-way ANOVA based on the mean for each instance across the ten tests.

To meet the statistical assumptions of the ANOVA, and to make computations tractable, the original data were transformed by taking logarithms prior to the analysis. (Since the test instances are in the critical complexity region, heavy-tail effects are not present.)

These factors were tested with the following expectations. Increasing the cutoff and increasing the number of restarts should result in a reduction in search effort compared to $dom/wdeg\text{-}nores$, as the amount of contention information grows. Furthermore increasing the cutoff and the number of restarts should increase the likelihood that the problem will be solved prior to the run to completion by WTDI and, to a lesser extent, by RNDI. Increasing the number of restarts should give a more diverse sample of the search space and, as such, should provide a better estimate of sources of global contention. However, as we show in the next section, this did not turn out to be the case for WTDI.

Results for WTDI

Summary results for the tests of WTDI are given in Table 3.3. The table shows mean nodes explored over the 100 instances in the two sets, for each condition. This measure includes nodes searched across all runs, including the final run to

Table 3.3: WTDI: Analysis of restart and cutoff factors.

		Soluble Problems Cutoff				Insoluble Problems Cutoff			
		250	**500**	**1000**	**2000**	**250**	**500**	**1000**	**2000**
Restarts	**1**	47,865	53,878	45,306	48,276	46,833	47,810	47,699	53,598
	5	46,032	44,239	43,486	51,221	45,754	46,553	49,730	51,581
	10	51,850	48,246	49,903	55,939	44,487	47,072	53,626	62,526
	20	**39,158**	53,155	55,062	59,692	48,607	54,022	64,805	80,899

Notes: $\langle$200,10,0.0153,0.55$\rangle$ problems. 100 instances per set. Mean nodes explored across all runs per instance. For reference, $dom/wdeg\text{-}nores$ averaged 42,468 nodes on the soluble instances and 41,200 nodes on the insoluble instances Figures in bold indicates improvement over $dom/wdeg\text{-}nores$.

completion if one was needed. In this and the following tables, cases where the average was better than that for $dom/wdeg\text{-}nores$ are marked in bold

For the soluble problem set, the analysis of variance (ANOVA) showed only a small statistically significant difference for the restarts factor ($F(3,1584) = 2.9752$, $p < 0.05$) and none for the cutoff factor, while for insoluble problems both restarts and cutoff, but not their interaction, were statistically significant ($p < 0.001$). For the latter, it is clear from the table that this significance is due to the average nodes *increasing* as restarts and cutoff increase. Indeed, WTDI improved on $dom/wdeg\text{-}nores$ in terms of nodes explored in only one case on the soluble problem set (20R250C) and no restart-cutoff combination resulted in improved performance on the insoluble problem set. This refutes the expectation that WTDI would result in a reduction in search effort over $dom/wdeg\text{-}nores$.

However the cost of preprocessing may obscure improvements. Thus it is of interest to consider means for the final run for each instance, to determine if there was any evidence of improvement that could be ascribed to learning. (By the final run we mean the run where either a solution was found or the instance was proven insoluble.) An improvement would imply that the weights learnt were meaningful and significant on a global scale. These results are given in Table 3.4.

There are no consistent trends for the insoluble instances, which is reflected in neither factor being statistically significant based on the ANOVA. Indeed, outside of the results with 1 restart, there is little difference between the averages for the other combinations (all in the range 42K – 45K). Furthermore, no combination resulted in improved performance which suggests that the weights learnt were not significant on a global scale.

For the soluble problem set on the other hand, the restarts factor was statistically significant ($p < 0.001$) as was the cutoff factor, although to a lesser degree ($p < 0.05$). Although there are no consistent trends in the averages, means tended to decrease as the number of restarts or the cutoff increased. Here, however, there are a number of cases where WTDI solved the instances on average in less nodes than $dom/wdeg\text{-}nores$ (e.g. for 3 of the cutoff factors when combined with 20 restarts).

Table 3.4: WTDI: Analysis of restart and cutoff factors, final run.

		Soluble Problems Cutoff				Insoluble Problems Cutoff			
		250	**500**	**1000**	**2000**	**250**	**500**	**1000**	**2000**
Restarts	**1**	47,615	53,388	44,336	46,416	46,583	47,310	46,709	51,678
	5	44,782	**41,889**	**38,856**	42,781	44,504	44,053	44,780	41,981
	10	49,353	43,686	**41,043**	**40,579**	41,987	42,072	43,726	43,326
	20	**34,191**	44,400	**38,292**	**31,472**	43,607	44,022	45,005	42,499

Notes. $\langle 200,10,0.0153,0.55 \rangle$ problems. 100 instances per set.
Mean nodes explored for run where instance was solved.
For reference, $dom/wdeg\text{-}nores$ averaged 42,468 nodes on the soluble instances and 41,200 nodes on the insoluble instances

One possible reason for the statistically significant results on the soluble problem set that may confound interpretation is that an instance was only tested until a solution was found. This means that whenever an instance was solved before the run to completion, it would necessarily have a low value for the number of nodes

in the final run. This would be more likely to happen for larger C and R, and is suggested by the overall trends noted above.

Low averages for certain conditions may be because the weights were effective at this point or because a degree of randomisation was added to a heuristic that is generally effective. Therefore, cases where the mean was better than the reference value (42,468) on the soluble problem set must be treated with caution.

To obviate this difficulty, instances were selected that were *never solved before the final run under any condition*. There were 54 such instances in the soluble set and 96 in the insoluble set. (Interestingly the 4 insoluble instances that were solved prior to the run to completion were solved on the first run, further emphasizing the notion that combining the heuristic with restarting is poor for these problems.)

Table 3.5: WTDI: Analysis of restart and cutoff factors, only instances solved on run to completion, final run.

		Soluble Problems Cutoff				Insoluble Problems Cutoff			
		250	**500**	**1000**	**2000**	**250**	**500**	**1000**	**2000**
Restarts	**1**	66,446	79,489	62,272	66,787	48,450	49,196	48,584	53,774
	5	62,636	58,875	**58,587**	69,877	46,291	45,827	46,586	43,673
	10	68,971	59,755	63,442	65,171	43,672	43,767	45,490	45,074
	20	**52,264**	64,430	63,570	**55,572**	45,365	45,803	46,829	44,212

Notes. $\langle 200,10,0.0153,0.55 \rangle$ problems. Only instances solved on the run to completion (54 instances in soluble set, 96 instances in insoluble set).
Mean nodes explored on run to completion.
For reference, $dom/wdeg\text{-}nores$ averaged 58,669 nodes on 54 soluble instances and 42,860 nodes on 96 insoluble instances

The means for these subsets under the present experimental regimen are given in Table 3.5. As one can see there are no consistent improvements as the number of restarts is increased nor as the cutoff is increased. Here the ANOVA gave *no* statistically significant results for either soluble or insoluble problems. This indicates that improvements over $dom/wdeg\text{-}nores$ in Table 3.4 and the statistically

significant results may be due to the factors suggested above.

There are at least two reasons why WTDI may not be effective here. The first reason is there may not have been anything to learn on a global scale for these problems. Since failure always occurs in a particular context in the form of a partial assignment, weights derived from such failures may not be relevant at the beginning of search when the contexts of those failures are no longer present.

Furthermore, variables with higher weights will tend to be chosen earlier after restart. These variables are less likely to have their weights increased in the subsequent search since wipeouts generally don't occur until after several variables have been assigned and only affect future variables when constraint propagation is used. This will lead to different variables being weighted each time. When these variables are chosen during subsequent restarts, still other variables may be weighted, and so forth.

Together, these effects may make it difficult to distinguish sources of "global" as opposed to "local" difficulty, i.e. difficulty that is basic to the problem and which will, therefore, be encountered throughout the search space, versus difficulty restricted to a specific part of the search space. We will return to this issue in section 3.5.

Table 3.6: RNDI: Analysis of restart and cutoff factors.

		Soluble Problems Cutoff				Insoluble Problems Cutoff			
		250	**500**	**1000**	**2000**	**250**	**500**	**1000**	**2000**
Restarts	**1**	50,008	46,882	50,246	45,897	46,363	47,153	46,726	47,502
	5	**41,107**	**38,505**	43,995	45,737	**39,551**	**39,323**	**40,409**	44,250
	10	**37,683**	**41,006**	43,673	51,697	**38,073**	**38,684**	42,800	51,186
	20	**39,616**	**42,284**	49,172	68,368	**38,501**	41,937	49,874	68,383

Notes. $\langle 200,10,0.0153,0.55 \rangle$ problems. 100 instances per set.
Mean nodes explored across all runs for each instance. Average of ten tests per instance. For reference, $dom/wdeg\text{-}nores$ averaged 42,468 nodes on the soluble instances and 41,200 nodes on the insoluble instances

Results for RNDI

Results for fully crossed restart-cutoff experiments using RNDI are given in Table 3.6 with mean total search nodes (including nodes explored during preprocessing). In contrast to those tests with WTDI, there was a more consistent improvement in total nodes for both sets compared to $dom/wdeg\text{-}nores$ (e.g. cutoff 250 for restarts 5,10 and 20). With regard to trends, increasing the cutoff generally led to an increase in average nodes (outside of the 1R condition), while there was no consistent trend for increasing number of restarts. It is also interesting to note the similarity between the behavior of RNDI on the soluble and insoluble problem sets.

Table 3.7 gives results for mean search nodes on the final run, i.e. the run where the instance was solved. The first point to note is that, outside of the 1R condition, every restart-cutoff combination improved on $dom/wdeg\text{-}nores$ for both sets.

RNDI failed to solved any instance during preprocessing. Thus every instance was solved on the final run using $dom/wdeg$ with the weights learnt in preprocessing guiding its early choices. This was expected as search with random variable ordering using d-way branching can be inefficient. Hence, the results in Tables 3.6 and 3.7 are not affected by finding a solution prior to the run to completion.

For comparisons of means in Table 3.7, we performed a two-way ANOVA with only the restarts and cutoff factors, using the means for the ten tests for each problem and condition. The restarts factor was highly significant ($F(3, 1584)$ = 15.8423 for soluble, $F(3, 1584)$ = 8.2953 for insoluble, $p \ll$ 0.001), while neither the cutoff factor nor the interaction of cutoff and restarts had any statistical significance.

In Table 3.8, we also provide the 3-way mixed effects ANOVA for the soluble set, containing problems as a third factor. This gave a highly significant result for the restarts factor ($p \ll 0.001$); in contrast, the result for the cutoff factor was much less significant ($p < 0.01$). Not surprisingly, the problems factor was highly significant, showing that there were stable differences in problem difficulty despite the randomisation of variable selection. The most significant interaction was between problems and number of restarts ($p \ll 0.001$), while there was a less significant interaction between the number of restarts and the cutoff ($p < 0.05$).

Table 3.7: RNDI: Analysis of restart and cutoff factors, final run.

		Soluble Problems Cutoff				Insoluble Problems Cutoff			
		250	**500**	**1000**	**2000**	**250**	**500**	**1000**	**2000**
Restarts	**1**	49,758	46,382	49,246	43,897	46,113	46,653	45,726	45,502
	5	**39,857**	**36,005**	**38,995**	**35,737**	**38,301**	**36,823**	**35,409**	**34,250**
	10	**35,183**	**36,006**	**33,673**	**31,697**	**35,573**	**33,684**	**32,800**	**31,186**
	20	**34,616**	**32,284**	**29,172**	**28,368**	**33,501**	**31,937**	**29,874**	**28,383**

Notes. $\langle 200,10,0.0153,0.55 \rangle$ problems. 100 instances per set.
Mean nodes explored in run where instance was solved. Average of 10 experiments.
For reference, $dom/wdeg\text{-}nores$ averaged 42,468 nodes on the soluble instances and 41,200 nodes on the insoluble instances

The consistent significance of the restarts factor is logical for RNDI, more restarts means a more diverse sample of the search space. A large cutoff, on the other hand, will just give greater weight increments in each individual search tree. This could lead to overweighting local sources of contention. This implies that a short cutoff (C_i) with many restarts (R_i) should learn better information than a large cutoff (C_j) with few restarts (R_j), where $C_iR_i = C_jR_j$, i.e. where the total number of search nodes explored during probing is the same (e.g. 10R500C versus 20R250C). These expectations are largely borne out by the experiments using RNDI, although differences are generally small and were rarely statistically significant when compared using paired comparison t-tests. We will investigate this hypothesis further in the following section.

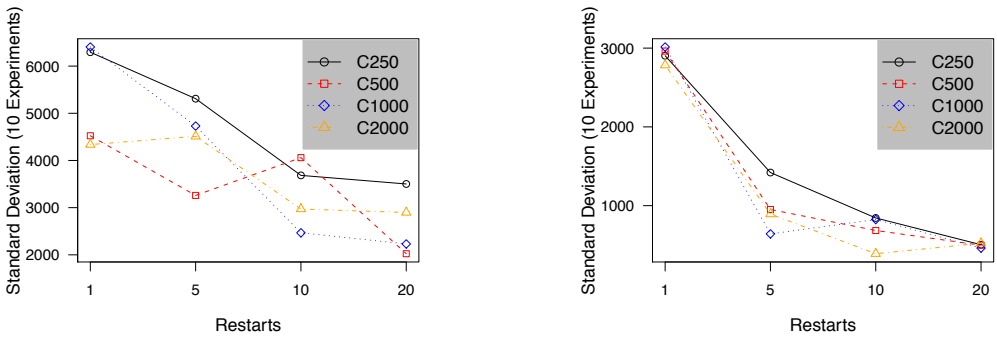
To assess the variation in performance of random probing across the ten experiments with different seeds, we calculated the standard deviation of the means of the 100 instances, which we plot in Figure 3.1. Increasing the number of restarts generally resulted in a reduction in variation (in 10/12 comparisons between sequential restart parameters for both soluble and insoluble instances). On the other hand, increasing the cutoff only resulted in a decrease in variation on roughly 50% of paired comparisons between sequential cutoff parameters. This further supports

Table 3.8: Analysis of Variance for RNDI, Solution Run

factor	Df	Sum Sq	Mean Sq	F value	Pr($>F$)
Problems	99	22001.6	222.238	297.9671	$<$ 2.20E-16
Restart	3	370.4	123.463	165.5341	$<$ 2.20E-16
Cutoff	3	11.6	3.854	5.1669	0.00144
P*R	297	866	2.916	3.9096	$<$ 2.20E-16
P*C	297	227.3	0.765	1.0262	0.366977
R*C	9	12.6	1.403	1.8811	0.049913
P*R*C	891	651.6	0.731	0.9805	0.65068
Residuals	14400	10740.2	0.746		

Notes. Based on data of Soluble Problem Set in Table 3.7. Column headers are standard abbreviations for ANOVA terms: "Df" = Degrees of freedom, "Sum Sq" = sum of squares, "Mean Sq" = mean sum of squares (Sum Sq / Df), "F" is the test statistic obtained by dividing the Mean Sq in the row by the Mean Sq Residual. "Pr($>F$)"is the probability of obtaining the value of F under the null hypothesis of no effects, where the expectation of F=1.

the hypothesis that the number of restarts is the key element to identifying globally contentious variables with random probing.



(a) Soluble Problems (b) Insoluble Problems

Figure 3.1: Standard deviation of RNDI experiments with different seeds.

3.4.3 Random Problems

The experiments in this section were carried out on random binary CSPs, which take the form $\langle n, d, m, t\rangle$ as defined earlier; and on random k-coloring problems.

The problem parameters for the random binary CSPs are, again, based on the problem sets used by Boussemart et al. in their study of the weighted degree heuristic [29].

The parameters for the random binary problems are $\langle 80,10,0.1042,0.65 \rangle$. The random k-coloring problems were generated using Richard Wallace's generator. There were 300 variables, with 3 colors and density 0.009. Here, instances were also separated into soluble and insoluble sets, each set containing 100 instances. A further set of soluble coloring instances (as studied in [92]) was also tested, there were 100 instances each with 50 variables, 6 colors and density 0.27.

The main results are shown in Table 3.9. Somewhat surprisingly, WTDI only improved on $dom/wdeg\text{-}nores$ in one problem set (the 3-Coloring Soluble set), and in fact was sometimes appreciably worse. If one considers nodes for just the final run, it also improved on the 6-coloring problem set, although here it solved 64 of the instances prior to the run to completion.

RNDI, on the other hand, nearly always led to improved performance although the differences were not always large. The largest improvement was 40% on the soluble 3-coloring problem set. The performance on the Rand-80 set, where there was most room for improvement, was surprising as it was expected that real gains could be made by preprocessing. Although it did reduce the average search nodes on the final run by over 5% this did not offset the cost of probing.

Interestingly, RNDI performed poorly on the relatively easy 6-coloring instances. Although there was not much room for improvement here (indeed the cost of preprocessing ($\approx$2400 nodes) was more than twice the average of $dom/wdeg\text{-}nores$ and $dom/fdeg$ for these instances), the average nodes explored on the final run was over three times greater than that of $dom/wdeg\text{-}nores$.

There are two possible reasons for this: the first is simply that the weights learnt by random probing were poor for these instances, i.e. random probing was unable to identify globally difficult elements; the second possibility is that the interaction of the weights with the domain factor was counter-productive. We tested whether the latter was the case by combining RNDI with two alternative heuristics, $wdeg$, and a variation of the $brelaz$ heuristic [31], where variables are chosen by their domain size and ties are broken by the variables' *weighted degrees*. We refer to this heuristic as *brelazwdeg*. The results are given in Table3.10.

Table 3.9: Results For Random Binary and k-Coloring Problems

		dom – $fdeg$	dom – $wdeg$	WTDI	RNDI
Rand-80 (54/100 sol)	Total Nodes	242.6K	186.9K	229.3K	**184.4K**
	Final Run Nodes	-	-	221.3K	176.4K
	Total Checks	342M	280M	352M	**276M**
	Total Time	125.44	**105.41**	131.00	105.50
	#PreSol	-	-	2	0
3-Coloring Soluble	Total Nodes	42.7K	22.1K	20.3K	**14.1K**
	Final Run Nodes	-	-	11.5K	10.1K
	Total Checks	3.5M	2.3M	2.0M	**1.5M**
	Total Time	39.71	25.15	21.77	**17.09**
	#PreSol	-	-	21	0
3-Coloring Insoluble	Total Nodes	145.5K	54K	65.6K	**50.6K**
	Final Run Nodes	-	-	55.8K	46.6K
	Total Checks	12M	5.6M	6.9M	**5.3M**
	Total Time	137.56	60.63	73.42	**58.08**
	#PreSol	-	-	2	0
6-Coloring Soluble	Total Nodes	1285.1	**1028.9**	2030.5	5796.5
	Final Run Nodes	-	-	714.1	3428.4
	Total Checks	101.9K	**93.0K**	185.8K	488.1K
	Total Time	**0.08**	**0.08**	0.15	0.40
	#PreSol	-	-	64	2.3

Notes: Results are averages per instance.
"#PreSol" refers to the number of instances solved prior to the run to completion.
For Rand-80 problem set, R = 40 and C = 200 for both RNDI and WTDI.
For 3-coloring problem sets, R = 40 and C = 100 for RNDI,
R = 20 and C = 500 for WTDI.
For 6-coloring problem set, R = 40 and C = 60 for both RNDI and WTDI.

Table 3.10: Results For 50 Variable 6-Coloring Problem Set

		$fdeg$	$wdeg$	WTDI $wdeg$	RNDI $wdeg$
$dom/*$	Total Nodes	1285.1	**1028.9**	2030.5	5796.5
	Final Run Nodes	-	-	714.1	3428.4
	Total Checks	101.9K	**93.0K**	185.8K	488.1K
	Total Time	**0.08**	**0.08**	0.15	0.40
	#PreSol	-	-	64	2.3
$*$	Total Nodes	347.3K	16.9K	17.6K	**15.3K**
	Final Run Nodes	-	-	15.5K	12.9K
	Total Checks	22.8M	2.0M	2.2M	**1.7M**
	Total Time	23.45	1.45	1.51	**1.25**
	#PreSol	-	-	28	2.3
$brelaz*$	Total Nodes	909.2	**805.5**	2263.6	2979.6
	Final Run Nodes	-	-	712.0	611.5
	Total Checks	64.7K	**59.2K**	202.4K	179.3K
	Total Time	**0.05**	**0.05**	0.15	0.17
	#PreSol	-	-	50	2.3

Notes: Results are averages per instance. "$*$" refers to either fdeg or wdeg in the top row. "#PreSol" refers to the number of instances solved during the information gathering phase.
R = 40 and C = 60 for both RNDI and WTDI.

When the domain factor is removed entirely (i.e. the heuristic was either $fdeg$ or $wdeg$), we see that random probing performs best. This supports our hypothesis that the reason for the poor performance of random probing followed by $dom/wdeg$ on these instances was due to an adverse interaction with the domain factor, and not due to the quality of information learnt during probing.

The $brelaz$ heuristic was specifically developed for graph coloring problems and, as the results illustrate, it is extremely effective. Interestingly, we find that the heuristic can still be improved upon by breaking ties with $wdeg$ rather than $fdeg$. The cost of probing again outweighs the benefits for both RNDI and WTDI, however we see that on the final run both improve upon $brelazwdeg$-$nores$. In the case of RNDI, which rarely solved an instance during the probing phase, this confirms that the weights learnt were indeed beneficial.

Analysis of information quality

In order to better analyze the quality of information gathered by the two restarting approaches we froze the weights after preprocessing, i.e. on the run to completion. Although this does go against the purpose of the methods (i.e. providing the heuristic with information from which to make better *initial* decisions), this was done to obtain information about the effectiveness of weight increments during the initial runs without contamination from updates during the final run of search. Note that results for WTDI are not completely representative of freezing the weights as they include instances which WTDI solved prior to the run to completion.

Table 3.11: Results For Random Binary and k-Coloring Problems, Frozen Weights

		dom – $wdeg$	WTDI	Frozen WTDI	RNDI	Frozen RNDI
Rand-200 Soluble Lex	Final Run Nodes	-	43.7K	61.5	36.0K	37.0K
	Total Nodes	42.5K	48.2K	66.1K	41.0K	42.0K
	Total Checks	107M	122M	168M	98M	104M
Rand-200 Insoluble	Final Run Nodes	-	42.1K	54.1K	33.7K	34.8K
	Total Nodes	41.2K	47.1K	59.1K	38.7K	39.8K
	Total Checks	115M	136M	168M	103M	109M
Rand-80 (54/100 sol)	Final Run Nodes	-	221.3K	268.4K	176.4K	142.0K
	Total Nodes	186.9K	229.3K	276.3K	184.4K	150.4K
	Total Checks	280M	352M	418M	276M	237M
3-Coloring Soluble	Final Run Nodes	-	11.5K	18.2K	10.1K	9.6K
	Total Nodes	22.1K	20.3K	27.1K	14.1K	13.6K
	Total Checks	2.3M	2.0M	2.7M	1.5M	1.4M
3-Coloring Insoluble	Final Run Nodes	-	55.8K	83.2K	46.6K	34.1K
	Total Nodes	54K	65.6K	93.0K	50.6K	38.1K
	Total Checks	5.6M	6.9M	9.7M	5.3M	4.2M

Notes: Results are averages per instance.
For Rand-200 problem sets, R = 10 and C = 500
For Rand-80 problem set, R = 40 and C = 200.
For coloring problem sets, R = 20 and C = 500
for WTDI, R = 40 and C = 100 for RNDI.

The results provide more compelling evidence concerning the quality of infor-

mation learnt by both approaches (Table 3.11). As expected, freezing the weights consistently lead to a large degradation in performance for WTDI. A puzzling aspect of these results is that learning during the final run can actually hinder search in the RNDI approach (Rand-80 and 3-Coloring insoluble problem sets). This could be ascribed to the interaction between the weights learnt during preprocessing and the weights learnt on the final run. Since we are learning from a relatively small sample in our preprocessing (a maximum of only 8000 nodes), the weights learnt can be quickly subsumed by the weights learnt on the final run, often changing the order of the variables (except for the first variable selected).

Table 3.12: RNDI Comparison of (Small R, Large C) with (Large R, Small C)

		R200 Sol	R200 Insol	R80	3-Col Sol	3-Col Insol
$dom/wdeg$ $-nores$	Nodes	42.5K	41.2K	186.9K	22.1K	54.0K
	Runtime	92.5	93.0	105.4	25.1	60.6
Frozen RNDI (10R Large C)	Total Nodes	42.0K	39.8K	166.2K	16.0K	48.3K
	Runtime	94.7	94.3	105.6	21.3	61.1
Frozen RNDI (100R Small C)	Total Nodes	**33.7K**	**32.3K**	**144.0K**	**13.4K**	**37.2K**
	Runtime	78.0	66.8	91.2	16.4	43.4

Notes: Average nodes/runtime per instance.
For Rand-200 problems ("R200"), Large C = 500 and Small C = 50.
For Rand-80 problems ("R80"), Large C = 800 and Small C = 80.
For 3-Col problems, Large C = 400 and Small C = 40.

In the previous section, we noticed a trend for RNDI where, for $R_i > R_j$, $C_i < C_j$, and R_iC_i = R_jC_j, the approach with a greater number of restarts and a shorter cutoff generally performed better. We tested this further by comparing $R = 100$ with small C versus $R = 10$ with large C, as shown in Table 3.12. Weights were again frozen to directly assess the quality of information learnt. (The values of C were based on the parameters used in Table 3.9, so as to explore the same number of nodes in preprocessing.)

The results clearly show that the information learnt with the 100R approach is of better quality, with improvements over the 10R approach ranging from 10-30% in terms of both average nodes explored and mean runtime. Similar improvements

were found over $dom/wdeg$-$nores$. These results were further corroborated using paired t-tests comparing the mean nodes of the two RNDI methods over 10 runs for each instance, the differences were statistically significant for all problem sets ($p \ll 0.001$).

3.4.4 Structured Problems

In this section, we look at problems with a more defined structure, in particular open shop scheduling problems ($OSPs$). An OSP involves a set of n Jobs and a set of m Machines, where each job consists of m tasks. Each task, t_i has an associated duration (p_i, its processing time) and machine; no two tasks of the same job share a machine. (Taillard found that only instances generated with n = m were non-trivial [196] so all "os-taillard-n" instances have n = m.)

The problem involves finding a schedule of all tasks such that no pair of tasks of a job overlap in their processing time, and no pair of tasks sharing a machine overlap in their processing time. In CSP terms: the variables are the tasks, the domain of a variable is the set of possible starting times for the variable. The constraints between pairs of tasks (t_i, t_j) of the same job/machine are as follows:

$$(t_i + p_i \ \leq \ t_j) \ \bigvee \ (t_j + dur_j \leq t_i)$$

i.e. either t_i finishes before t_j starts, or vice versa. When formulated as an optimization problem, the goal is generally to minimize the *makespan*, i.e. the latest finishing time over all tasks minus the earliest starting time over all tasks. A sample solution to an OSP with four jobs and four machines is given in Figure 3.2, where the makespan is given by (t_{end} - t_{start}).

We tested on OSPs from the CSP solver competition [†]. The instances are derived from the Taillard optimization instances [196]. Each optimization instance was converted to three satisfaction instances by fixing the latest allowed finishing time of all tasks based on the best known makespan (*BKM*) found by Taillard for the instance.

The latest allowed finishing times of tasks in the "os-taillard-n-100" instances is set to the *BKM* (i.e. the domain of each task t_i is (0...(*BKM* - p_i))), the latest allowed finishing time of tasks in the "os-taillard-n-105" instances is set to

[†] http://www.cril.univ-artois.fr/ lecoutre/benchmarks/ benchmarks.html

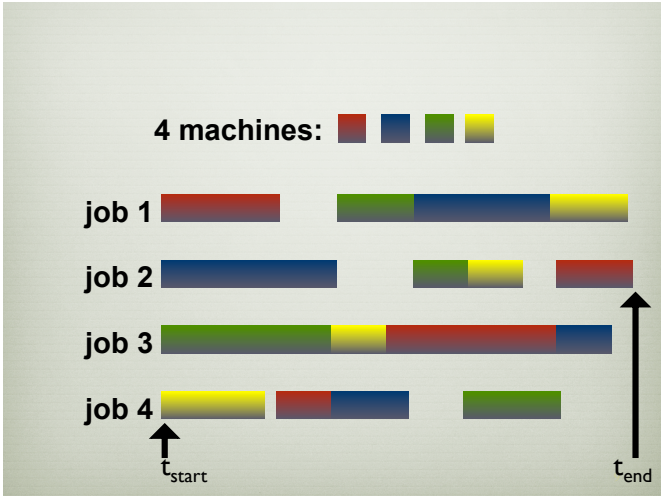


Figure 3.2: Solution to a sample 4x4 OSP.

105%(*BKM*), while the latest allowed finishing time of tasks in the "os-taillard-n-95" instances is set to 95%(*BKM*). All instances in the first two sets (*-100 and *-105) are soluble, whilst instances of the latter set are generally insoluble (unless *BKM* was not the optimal solution, which does not apply to the problem sets studied here).

In the previous section we showed that combining the weighted degree heuristic directly with a restarting strategy performed quite poorly on the random problem sets studied. However, there are reasons to believe that WTDI may perform better on more structured problems such as the open shop scheduling instances. Firstly, Thornton found that constraint weighting methods in local search performed better on structured (as opposed to random) CSPs where it was able to distinguish between harder and easier sets of constraints [198].

Secondly, systematic search methods for these types of scheduling problems are known to suffer from the "early mistake problem", where a poor decision at the top of the search tree may result in the exploration of an exponentially large insoluble subtree which cannot be recovered from in a feasible amount of time. This occurs because, although the instances often contain many solutions, there are also large parts of the search space without a solution [50].

Crawford and Baker further hypothesized that the job shop scheduling problems contain a small number of "control" variables (which define the schedule) and the rest are "dependent" variables (whose values are determined by the control variables). (Note that, as mentioned earlier, this is obviously closely related to the concept of backdoor variables as introduced by Williams et al. [227].) Restarting search, combined with some form of randomization, can reduce the impact of the early mistake problem on search effort by trying different orderings at the top of the search tree until the control variables are selected. Indeed, it is likely that these control variables are bottlenecks which can be identified through their constraint weights.

Experimentation

We tested on 5 sets of Taillard instances: two sets of instances with n=4 (so 16 tasks to be scheduled); and three sets of instances with n=5 (25 tasks to be scheduled). Each set contains ten instances. Domain sizes range from 120 to 270 for ost-4-100 instances and 180 to 330 for ost-5-100 instances.

We ran the same experimental setup as for the random problems, with a restarting regimen for both RNDI and WTDI of 50 restarts with a cutoff of 40 nodes per run. However, even though the instances tested were relatively small, search was limited to an overall cutoff of one million nodes due to the issues discussed with regard to the early mistake problem. Thus we also present our results in terms of number of instances solved (for RNDI this is the average number solved across 10 experiments).

The value ordering was a form of lexico, the ordering alternated between the lower and upper bound on the variable's domain. In other words, when a variable is first selected it is assigned its lower bound. If search backtracks to the variable it will remove the last value assigned and assign it the opposite bound of what was chosen the previous time.

Table 3.13 presents the results on the five sets. Both restarting strategies improve on $dom/wdeg$-$nores$ on all but the first set. The biggest differences can be seen for the larger sets (containing 25 tasks per instance). Interestingly, random probing solved roughly the same number of instances during the probing phase

Table 3.13: Results For Open Shop Scheduling Problems

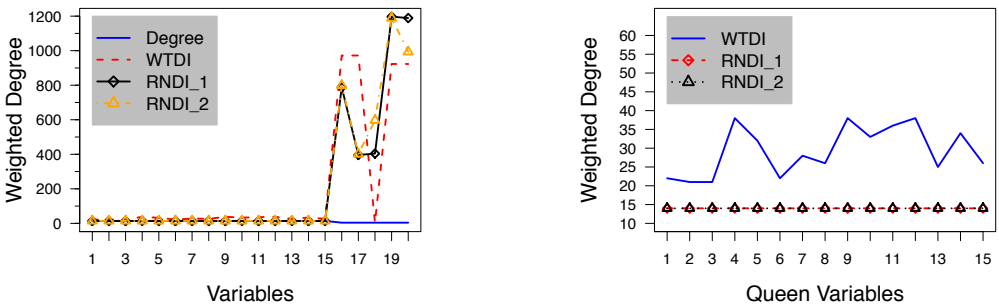
		$dom - fdeg$	$dom - wdeg$	WTDI	RNDI
ost-4-95	#Sol	10	10	10	10
	#PreSol	-	-	1	1
	Total Nodes	56.3K	3.7K	3.8K	**3.3K**
	Final Run Nodes	-	-	2.0K	1.5K
	Total Time	1.41	**0.10**	0.15	0.15
ost-4-100	#Sol	9	10	10	10
	#PreSol	-	-	8	9.8
	Total Nodes	246.1K	17.2K	3.3K	**0.4K**
	Final Run Nodes	-	-	2.8K	0.03K
	Total Time	7.77	0.59	0.12	**0.02**
ost-5-95	#Sol	5	6	8	**9.4**
	#PreSol	-	-	0	0
	Total Nodes	595.9K	410.5K	251.6K	**121.3K**
	Final Run Nodes	-	-	249.6K	119.3K
	Total Time	29.32	20.55	13.21	**6.64**
ost-5-100	#Sol	2	2	**6**	4.1
	#PreSol	-	-	2	0.9
	Total Nodes	895.2K	830.0K	**417.0K**	660.4K
	Final Run Nodes	-	-	415.3K	658.5K
	Total Time	36.08	39.26	**20.92**	33.58
ost-5-105	#Sol	6	9	10	10
	#PreSol	-	-	10	9.5
	Total Nodes	406.0K	134.3K	**0.3K**	0.5K
	Final Run Nodes	-	-	0.03K	0.04K
	Total Time	13.52	6.67	**0.02**	0.03
	Total Sol	32	37	**44**	43.5

Notes: Results are averages per instance, including cases where the overall cutoff was reached. "#PreSol" refers to the number of instances solved during the information gathering phase.
R = 50 and C = 40 for both RNDI and WTDI for all sets.

as WTDI. A similar result was found by Crawford and Baker [50] on job shop scheduling problems (JSP) using iterative sampling where decisions were made randomly and search restarted after the first failure.

According to Crawford and Baker [50], iterative sampling was effective at solving JSPs for the following reasons. Firstly, these types of problem often contain many solutions and their perceived difficulty is due to the "early mistake problem", repeatedly restarting will avoid the early mistake problem and eventually find one of the solution paths. They further hypothesized that the JSPs contain a small number of *control* variables which determine the values for the other (*dependent*) variables.

Overall, we see that the two restarting strategies offer large improvements over $dom/wdeg$-$nores$. We also tested freezing the weights for these instances, but this resulted in a deterioration in performance for both RNDI and WTDI. Thus, it is important to continue updating weights on these problems.



(a) Queens and Knights

(b) Queens Alone

Figure 3.3: Weight profiles on qk-15-5-add for WTDI and two runs of RNDI with different seeds (RNDI_1, RNDI_2).

3.5 Analysis of Weight Changes Produced by Each Strategy

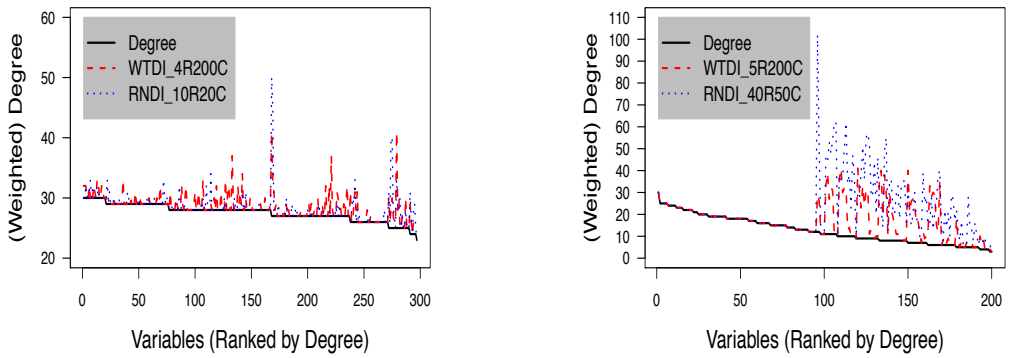
In this section we analyse the weights produced by RNDI and WTDI to gain a deeper insight into the behavior of the approaches during search. We refer to the weighted degrees of the variables in an instance as the *weight profile*. We first look at weight profiles after our preprocessing methods on instances with insoluble cores. Figures 3.3(a) and 3.3(b) illustrate weight profiles on the qk-15-5-add instance. Weights for RNDI are averaged over ten experiments.

The first figure shows the degrees, and weighted degrees after probing, of the queens and the knights variables. The first 15 variables are the queen variables while the last 5 variables are the knight variables. If we compare the weights on the knights with those on the queens we see the queens receive very little weight while most of the knights receive a large amount of weight as expected.

Figure 3.3(b) shows the weights for the queens alone. RNDI rarely failed on a queen constraint while WTDI resulted in a weight increase on a number of the queens. However the weight gain by queens variables are dwarfed by the weight increases on the knight variables. This shows the ability of both approaches to focus on global sources of contention and avoid weighting local sources of contention.

Sample weights for the first ehi-85-297 instance and the first of the composed instances are given in figures 3.4(a) and 3.4(b) respectively. The variables are ranked by their degrees in decreasing order. It is clear from the figures that the reason these problems are difficult for normal degree-based heuristics is that the variables that are part of the insoluble core(s), the variables with largest weighted degrees, do not also have the largest degrees. Furthermore the domain sizes of all variables are nearly identical at the start of solving so domain-based heuristics (with lexical tie-breaking) will suffer from the same thrashing effects which hinder degree-based heuristics on these problems.

To better understand the differences between the two strategies on random binary instances used in the previous experiments, we generated weight profiles on 5 randomly selected instances of the random binary 200-variable soluble set. For these tests, we used a cutoff of 500 nodes with 100 restarts. The five instances



(a) ehi-85-P1 (b) Composed-P1

Figure 3.4: Weight profiles for WTDI and RNDI on sample ehi-85 and composed instances. *_*i*R*j*C refers to *i* restarts and cutoff *j*.

chosen for these tests were sufficiently difficult that they were not solved within the specified cutoff by either approach. Each variable's weighted degree was saved after every restart.

After 100 restarts the sum of weights across all variables was 80% greater for random probing than for WTDI on average. This means that when using random variable selection there were 80% more failures than when using $dom/wdeg$ in preprocessing.

This is a slightly surprising result. One would expect that search with random variable ordering would not discover failures till a greater depth than search with $dom/wdeg$. Since variables are selected randomly it is less likely to build up the contention at the top of the search tree to the same degree as WTDI, and thus one may expect that WTDI would fail more often within the same cutoff.

To test this, we plot in Figure 3.5 the average number of failures at each search depth per probe, for both RNDI and WTDI after 100 restarts with a 500 node cutoff, for a sample instance from the Rand-200 soluble set. As we can see, the peak failure depth for random probing is indeed deeper than WTDI, but interestingly there is a much larger number of failures at this peak. This can be explained by differences in thrashing, the weighted degree heuristic moves contentious variables up the ordering during WTDI, while RNDI will repeatedly fail without identifying

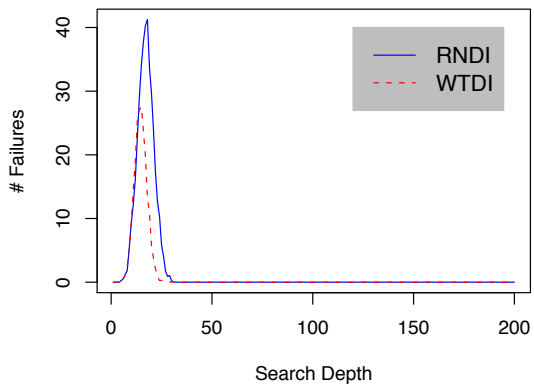
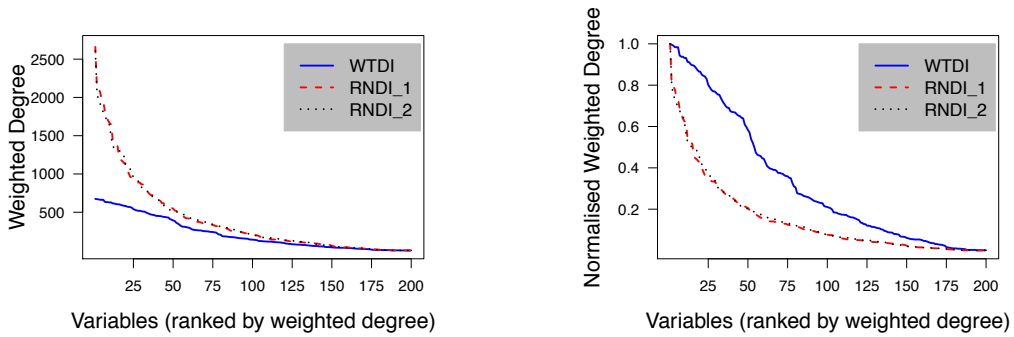


Figure 3.5: Average number of failures at each search depth per probe for RNDI and WTDI, 100R500C. Sample Rand-200 soluble instance.

the cause.

One reasonable explanation for the greater effectiveness of RNDI is that it provides better discrimination among variables, especially those with the largest weights. Figure 3.6(a) shows a typical weight plot for a single instance after 100 restarts, for a WTDI experiment and for two RNDI experiments which had different random seeds. In all three cases the variables were ranked according to their weight after the 100th restart. Figure 3.6(b) is the same three weight profiles where each weighted degree was normalised with respect to the maximum weighted degree in the profile.

The slope of the line indicates the level of discrimination between successive variables in the ranked order. Note that, for both RNDI runs, the slope is very steep for the top 50 variables and maintains a better level of discrimination over the next 50 variables than WTDI. The weight plot for WTDI has a much more gradual inclination, which indicates that even after 100 restarts there are no variables (or even subsets of variables) standing out as clearly contentious. This can be seen more clearly when the weights were normalised. Also, since the differences in weighted degrees between successively ranked variables are small in WTDI, the domain factor of $dom/wdeg$ will have more influence than for RNDI.



(a) Wdeg Profiles (b) Normalised Wdeg Profiles

Figure 3.6: Weight profiles and normalised weight profiles after preprocessing with 100R 500C for WTDI and two independent runs of RNDI.

An alternative method to assess the level of discrimination in the weight profiles is to look at their *Gini* coefficient, which is a measure of the inequality of a distribution of a variable. In our case we are mapping the proportion of weight on the bottom $x\%$ of variables over the total weight accrued.

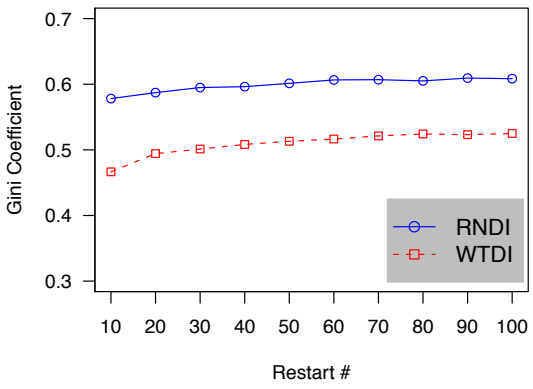


Figure 3.7: Gini coefficients for weight distribution every 10 restarts on sample Rand-200 soluble instance.

We used the weight profiles generated on the 5 instances from the Rand-200 soluble set with 100 restarts and a cutoff of 500 nodes mentioned above. The Gini

coefficient was calculated after every 10 restarts. The results for a sample instance are shown in Figure 3.7. The Gini coefficient for WTDI was consistently less than that for RNDI. This confirms that RNDI provides greater discrimination amongst variables.

One possible explanation for the difference is that RNDI accrued considerably more weight than WTDI during each probe, and thus the proportion of weight on the top variables may be similar but the weights in RNDI are much greater. However, when we compare the Gini coefficients between weight profiles on the same instance with roughly the same sum of weighted degrees in Figure 3.7 (e.g. RNDI after 10 restarts versus WTDI after 30 restarts) we find that the Gini coefficient for RNDI is still significantly higher. Indeed, the lowest Gini for RNDI here (0.578 after 10 restarts) was still greater than the highest Gini coefficient with WTDI (0.525 after 100 restarts), even though the weight accrued by WTDI was nearly 5 times greater.

We evaluated the stability of the weight rankings across restarts using the top-down rank correlation coefficient [112]. We chose this statistic as we are primarily interested in the correlation amongst the variables identified as "globally" contentious by the probing methods. Furthermore, there are often a number of variables which are rarely involved in a domain wipeout. Inclusion of these low-ranked variables can produce spuriously high rank correlations when using an unweighted method such as Spearman's rank correlation coefficient.

We first assessed the consistency of the weight profiles produced by RNDI and WTDI across restarts. We calculated the weight *increase* per variable for each block of 20 restarts on an instance (i.e. a variable's weight increase in restarts 1-20, 21-40, etc.). This was done to ensure independent rankings for the statistic.

The average top-down correlation amongst these augmented weight profiles is given in Figure 3.8 for each sample instance. RNDI was clearly more consistent than WTDI in its identification of the globally contentious variables. This is further underlined by the overall range of correlations found for both methods. The augmented weight profiles produced by WTDI had correlations ranging from 0.69 – 0.91, compared to a range of 0.80 – 0.95 for RNDI.

We next assessed the consistency of RNDI across different initial random seeds. We calculated top-down correlations amongst weight profiles generated

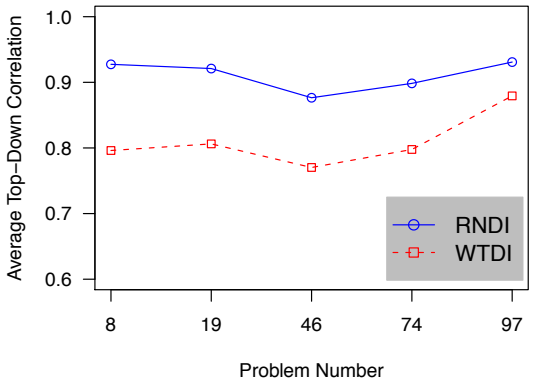


Figure 3.8: Average top-down correlation between weight *increments* every 20 restarts on 5 sample Rand-200 soluble instances, with 100R500C restarting strategy for both RNDI and WTDI.

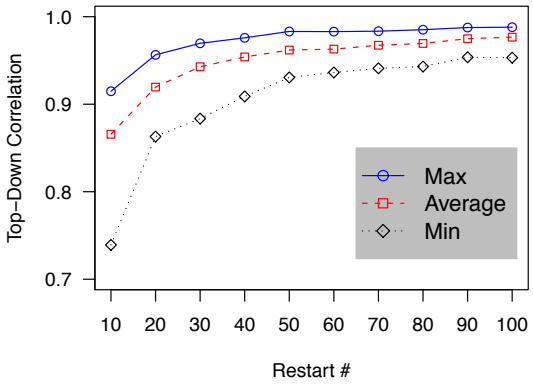


Figure 3.9: RNDI (100R500C) restarting strategy, 5 sample Rand-200 soluble instances. Overall maximum, average, and minimum top-down correlations every 10 restarts across five runs on each instance.

by RNDI with five different seeds, on the five sample instances, after every 10 restarts. In Figure 3.9, we plot the minimum, average, and maximum correlations over the five instances, per 10 restarts. The results show that after 40 restarts, correlations amongst weight profiles generated with different seeds were all above 0.9. This confirms that RNDI is consistent at identifying globally contentious variables in these problems, while the results also suggests that 10 restarts is not sufficient to adequately sample the search space.

We hypothesized earlier that a large number of short runs is better than a few long runs for identifying globally contentious variables. To test this, we generated weight profiles for the restarting strategies 100R50C and 10R500C with five different seeds on each of the five sample instances, and calculated top-down correlations amongst rankings on the same instance.

Our analysis of the correlations (as illustrated in Figure 3.10) confirms that taking a more varied sample produces more consistent information, notwithstanding the shorter cutoff. Indeed, the minimum correlation between a pair of weight profiles generated using 100R50C was 0.90, compared to a minimum of 0.71 for the 10R500C setting.

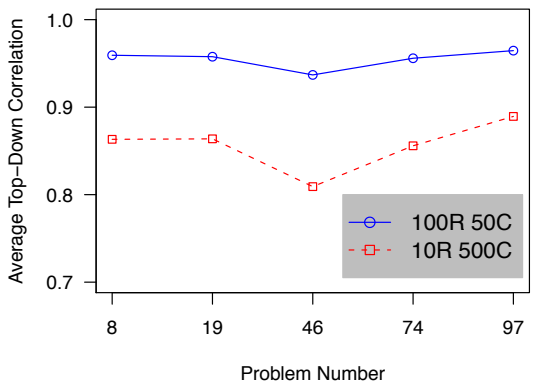


Figure 3.10: Average top-down correlation across five runs per instance for RNDI, comparing weight profiles of 100R50C with 10R500C restarting stragies.

We also generated weight profiles for 10 open shop scheduling instances in

the set ost-5-95 (these were chosen as none were solved during preprocessing by either method). For each instance, we generated the weight profile for WTDI and two weight profiles for RNDI with different seeds, a sample plot is given in Figure 3.11 for instance ost-5-95-3. Here, WTDI clearly identified a small subset of variables as contentious and focused search on these. RNDI, on the other hand, had a wider spread of weight albeit still providing greatest discrimination between its top ranked variables.

Upon further investigation, we find that this instance contains five insoluble cores, three on the machines and two on the jobs. These were insoluble cores as the sum of durations of the five tasks in the job/machine exceeded the latest allowed finishing time of the tasks (95%BKM). Thus, no matter the ordering, it is impossible for the five tasks to be processed sequentially in the allotted time.

There are six variables which partake in two insoluble cores. The five variables weighted by WTDI form an insoluble core, they are five tasks of the one job. Three of these variables are also in an insoluble core on their associated machines. Analysis of the top ranked variables based on weights from RNDI show that the same variable was ranked first after preprocessing with the two different seeds (albeit tied first in one case). This variable participated in two insoluble cores, one on its machine and one on its job (not the same job as identified by WTDI).

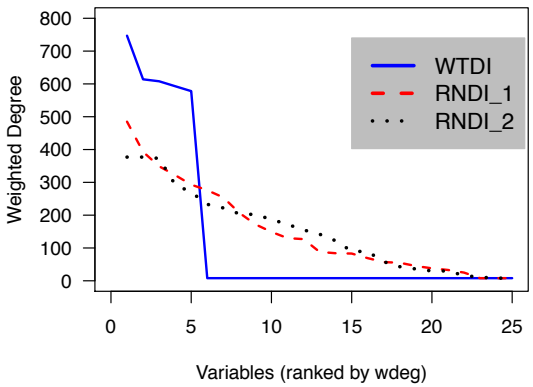


Figure 3.11: ost-5-95-3 weight profiles after preprocessing with 50R40C for both WTDI and RNDI.

Other variables ranked highly by both runs of RNDI include the variables that WTDI identified as an insoluble core. Interestingly, RNDI weighted the variables with the largest duration in the insoluble core highly, while WTDI weighted all five variables in its insoluble core highly. Indeed, the top ranked variable by WTDI after preprocessing had a relatively small duration (37), while the two lowest ranked variables in the insoluble core had largest durations (83 and 93).

This may explain why RNDI proved insolubility on this instance in 8400 nodes on average (with seven runs proving insolubility in 7347 nodes, two runs proving insolubility in 6457 nodes and one run taking 19050 nodes), compared to WTDI which needed 20557 nodes to prove insolubility. Assigning the variable in the insoluble core with the largest duration first is best for two reasons, firstly it has the smallest domain (since the initial domain size of a variable x is (95%(*BKM*) - dur_x)), and secondly it will result in the largest reduction of its neighbors domains after propagation.

The Gini coefficients for the weights distributions on these scheduling instances give further evidence of the ability of WTDI to focus on a subset of variables. The average Gini coefficient over the 10 weight profiles was 0.73 for WTDI, compared to an average of 0.46 for the RNDI weight profiles. On further inspection, we find that the Gini coefficients for WTDI were generally between 0.75 and 0.85 with two exceptions which had Gini coefficients of 0.42 and 0.54 respectively. These two exceptions occurred for the instances which WTDI failed to solve within the million node cutoff.

Weighted degree rankings of the variables were compared using the top-down correlation coefficient. The correlations for the two RNDI rankings (with different seeds) ranged from 0.45 to 0.90 across the ten instances, with an average of 0.68. Thus the weight profiles generated by RNDI on these scheduling instances were much less stable than for the random binary instances discussed earlier.

However, as shown in Table 3.13, RNDI still solved most of these instances (9.4 on average), compared to WTDI which solved eight. As we have discussed for the sample instance, once WTDI identified an insoluble core it concentrated its weight therein. RNDI spread its weight across different insoluble cores but, on occasion, found an insoluble core with a smaller proof. This may also explain why RNDI performed worse when weights were frozen, RNDI may jump between

insoluble cores with frozen weights whereas allowing learning to continue on the run to completion may concentrate search in one insoluble core.

Overall, our analysis has supported a number of hypothesis regarding the results of our earlier experimentation. Firstly we have shown that both methods are adept at identifying insoluble cores, be it in academic problems such as the queens-knights or more real-world examples such as the scheduling instances of Taillard. As expected, WTDI concentrates its weight on an insoluble core once one is discovered (hence the high Gini coefficient for the insoluble scheduling instances), while RNDI spreads its weight across insoluble cores if more than one exists.

Secondly, when problems had less structure, WTDI, unlike RNDI, struggled to separate locally contentious variables from those globally contentious as shown by plots of ranked weighted degrees, and by their Gini coefficients. Furthermore, RNDI resulted in more stable patterns of weight distribution, i.e. variables had similar rankings after runs of RNDI with different seeds. This was clearest for the top ranked variables, which shows that there were measurable differences in the levels of contention associated with different variables in the instance.

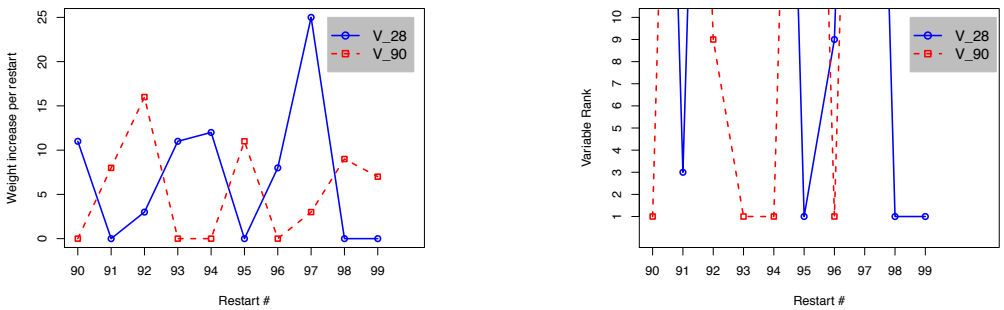
3.6 Discussion

3.6.1 Variable Convection

WTDI suffers from something that we refer to as *variable convection*, which describes the rise and fall of variables within the heuristic's ordering during search. This effect occurs because variables which have a large weight (from the previous runs) are selected at the top of the search tree. The first few variables chosen rarely receive any weight in a given run, since failures generally don't occur until at least 4 or 5 variables have been assigned. Eventually these variables fall back down the ordering and are replaced by variables which have received weight in the previous run(s), and the cycle repeats.

Clearly this is beneficial in the context of restarting, however it does affect the method's ability to clearly identify globally difficult elements. As a result, although WTDI can identify a subset of variables which are sources of contention,

it does not offer good discrimination between them. A similar effect has been observed in "Squeaky Wheel Optimization" [115] where difficult elements get handled earlier upon restart, and because they are handled earlier they cause less "trouble", and eventually they fall back down the ordering to the point where they are sources of difficulty again.



(a) Weight Increase Per Restart

(b) Depth at which Variable Selected

Figure 3.12: Variable Convection for WTDI on sample instance, convection effect on two top ranked variables (V_28,V_90).

Figure 3.12 illustrates the notion of variable convection on two highly ranked variables of a sample instance from the Rand-200 soluble problem set. The two variables were selected because one or the other was chosen first by the heuristic on 8 out of the last 11 runs: Var-90 was chosen first on restarts 90, 93, 94, 96; Var-28 was selected first on restarts 95, 98, 99 and on the final run (which isn't included since there was no cutoff). The weight increase per restart for the two variables over the final 11 restarts out of 100 is shown in Fig 3.12(a), while the depth at which the variable was selected during search is shown in Fig 3.12(b) (due to varying fail-depths we only show the depth if the variable was selected in the first 10 variable selections).

On the runs where the variable was chosen first, its weight increase was 0 (Var-28 also had no weight increase on restart 91 where it was selected third). Then, when other variables achieved a weight larger than theirs, they fell back down the ordering and their weights began to increase again. Indeed in most cases where they were not selected first, they were not selected in the first ten variable

choices in a run (Fig 3.12(b)). Also, there were 5 different variables selected first over the last 11 runs for this instance which further emphasizes the effects we are discussing, i.e. that the variable ordering at the top of the search tree is constantly changing with WTDI, even after a large number of restarts.

A similar phenomenon was observed in the insoluble scheduling instance discussed in the previous section. Three different variables were selected first over the last ten runs of WTDI. The variable representing the task with the largest duration was selected first on four runs, while the variable representing the task with the second smallest duration was selected first on four of the other runs. When the variable was selected first it received no weight increase on its constraints.

Note that this effect is not restricted to the restarting case, it can occur for any form of depth-first search with a weighted degree based heuristic. Variables chosen at the top of the search tree are likely to receive little weight, and will thus be replaced by other variables if search backtracks to this point again.

This also explains why freezing the weights after random probing generally improved search: when search backtracks to the top of the search tree, the variables selected based on the probing weights would no longer have the largest weight if the weights weren't frozen and so would not be selected.

However it should be noted that the impact of variable convection is less likely for problems with large arity constraints, as a variable chosen near the top of the search tree may still receive weight if its neighbors in the large arity constraints are uninstantiated.

3.6.2 Blame Assignment

As in most learning from search approaches, there are important blame assignment issues [187]. First, one doesn't know the actual root of any insoluble subtree without knowing the solution, because search can backtrack above any given instantiation (obviously except for the first variable selected in search).

Secondly, when propagating assignments using arc consistency, one doesn't know that a given constraint is truly to blame for a variable's domain going empty. It could be that the constraint of a neighboring variable had removed so many values that support was lost for neighboring domains and thus it should be the

former which is weighted.

In this context, a significant advantage of the weighted degree heuristic is that the weight it gives for each individual wipeout is small. It is the cumulative effect of a variable's constraints directly causing domain wipeouts on a number of occasions that determines whether that variable will be selected over another. Similarly, RNDI uses weights accumulated over a large number of runs and should be less susceptible to noise for the top choices. We will look at this issue in more detail in the following chapter.

3.7 Chapter Summary

Domain/weighted degree is a powerful heuristic that already demonstrates the benefits of learning from failure. Nevertheless, we have shown that further information can be extracted and utilised to improve search further. Both of our approaches improved on the original heuristic when tested on problems with clearly defined sources of global difficulty, in some cases by orders of magnitude.

When the sources of global difficulty were not as clear cut, e.g. on the random binary and random 3-coloring problems, RNDI consistently outperformed both $dom/wdeg\text{-}nores$ and WTDI, although the magnitude of the improvement was not appreciable on most problem sets when one considers the preprocessing. However, the results on the 6-coloring problem set show that weights from RNDI can have a negative interaction with the domain factor of the heuristic, even if the weights themselves are meaningful.

A significant finding in connection with the random variable selection procedure was that the number of restarts is a more important parameter than the cutoff-level. This conclusion was supported by the post-hoc analysis which showed that, for conditions in which the maximum allotment of nodes was the same, a greater number of restarts was more beneficial than a higher cutoff.

Through analysis of the weight distributions produced by the two strategies, we obtained evidence that supports our explanation of the deficiencies of WTDI and also helps explain the improvement observed when restarting is combined with random variable selection on the random problems. Thus, we have shown that when variables are ranked in terms of their weighted degree, the level of

discrimination between the top ranked variables is much greater with RNDI. We have also shown that the rankings of these variables are much less stable across restarts with WTDI than they are with RNDI. This led us to introduce the notion of variable convection for search using weighted degree heuristics.

Although variable convection does decrease the likelihood of identifying globally difficult elements, it also implies that the weighted degree heuristic might be better suited to a universal restarting strategy which would still guarantee completeness than to a fixed cutoff method such as WTDI. We will investigate this further in Chapter 6.

The most surprising result of this chapter was that learning on the final run can sometimes greatly hinder search when using random probing. Although this can also be explained by the effects of variable convection, it further emphasizes the need for careful consideration when developing approaches combining sampling and learning during search.

One final point to note is that information from WTDI/RNDI can also be used to provide feedback to a client regarding bottlenecks in the problem. For example if one is solving a scheduling problem such as scheduling the machines for a factory, the weights produced by these probes of the problem may indicate that machine X is a bottleneck. Purchasing a second machine of the same type as X may result in greater production and robustness for the client.

Chapter 4

Exploration of alternative constraint weighting techniques

4.1 Introduction

Constraint weighting has been shown to be an extremely efficient tool for guiding search decisions in solving CSPs. However, due to the complex nature of search when these adaptive heuristics are incorporated, there remains limited understanding as to why these methods work. In this chapter, we develop further insight into the nature of search both for the standard method and for search with weights initialized by random probing.

The weighted degree heuristic combines two related principles, the Fail First principle of Haralick and Elliott [97] and the *Contention Principle*, as introduced by Wallace [92], which states that:

If a constraint is identified as a source of contention, then a variable associated with the constraint is more likely to cause failure after instantiation than variables not associated with such a constraint.

We first investigate whether any form of contention can be used to improve the base heuristic. There are a number of ways of assessing contention, in the previous chapter we simply used constraint violations. However it is possible to glean more discriminatory information from search for little extra cost.

We consider a number of alternative measurements of contention such as

weighting based on the effective propagation of a constraint (i.e. when propagation of the constraint removes at least one value from a domain), and discriminating between constraint failures / effective propagations based on the size of the reduction caused by the constraint (i.e. the number of values it removes during propagation). We also assess whether these new forms of identifying contentious variables yield an improvement in the search performance of the two restarting methods introduced in the previous chapter.

Incrementing weights based on the size of reduction caused by propagation of a constraint deals with one aspect of the blame assignment issue for constraint weighting, whereby all constraints that cause failure are weighted equally even though some may be more indicative of global contention than others. By discriminating based on the number of values removed we assign 'blame' in proportion to the impact it had on the variable's domain.

We next evaluate the quality of information learnt by different sampling strategies, in particular we are interested in the extent to which maintaining arc consistency during random probing improves discrimination between global and local sources of contention. Although we have shown that random probing with MAC is effective at weighting global bottlenecks in problems, it is possible that there are less expensive methods to gather the same quality of information. We examine two such methods, the first uses a weaker level of consistency during random probing while the second uses weights generated during local search.

The latter is of particular interest as there have been a number of approaches previously proposed which combined local search with systematic search, where one facet of the local search method was to identify bottlenecks for guiding systematic search (e.g. [60],[147]). For example, Eisenberg and Faltings used local search as a preprocessing tool, which either returned a solution to the problem or returned a set of weights which were used to order variables [60].

Finally, we investigate a number of hypotheses regarding the basis for improvements in search performance when using a weighted degree heuristic, and when using weights initialized by random probing. In particular, we assess the importance of initial choices for improvements with random probing. We then investigate whether the depth at which failures occur reflects the level of associated contention, and the importance of sampling across the search space when using

random probing. Following this, we compare the heuristics in terms of two policy measurements of heuristic performance (promise and fail-firstness [24]), and in terms of their association with factors associated with heuristic action (buildup of contention versus simplification of the future problem) using the factor analysis approach of Wallace [213].

4.2 Sampling Different Forms of Contention

Weighting constraints based on failures is one method of identifying which constraints are the most contentious. Here, we look at alternative weighting methods for assessing contention. In some cases these may provide better discrimination between contentious constraints. In other cases, they simply allow us to assess the viability of different methods of evaluating contention. The purpose of these methods is to test the hypothesis that the performance of the heuristic can be improved by providing more discriminatory information, i.e. does the blame assignment issue discussed in the previous chapter have a negative impact on the heuristic?

In this section, we look at a number of methods for assessing contention. Specifically, we consider the following (where the naming convention was *type-of-contention* **By** *type-of-increment*):

- "WipeBydel": When a domain wipeout (DWO) occurs, the relevant constraint weight is increased by the number of values deleted by the constraint in the DWO variable, i.e. the size of the domain prior to propagation of this constraint.

- "AlldelBy1": Whenever a constraint deletes at least one value of a variable during propagation, the weight of that constraint is incremented by 1.

- "AlldelBydel": Whenever a constraint deletes at least one value of a variable during propagation, the weight of that constraint is incremented by the number of values deleted from the variable's domain.

- "Del/noWipeBy1": Whenever a constraint deletes at least one value of a variable during propagation, the weight of that constraint is incremented by 1 *except* when propagation of the constraint resulted in a DWO.

These different methods had the following expectations:

- $WipeBydel$: Incrementing by the size of the reduction when a DWO occurs should provide better discrimination between constraint failures. Consider the $queens\text{-}knights$ problem of the previous chapter, a DWO on a queen is generally the result of the removal of a relatively small number of values by propagation of a constraint, while a DWO on a knight is the result of a relatively large number of values being removed. However in the normal weighting approach these two events are considered equivalent.

  In the case of $dom/wdeg\text{-}nores$ (and WTDI) this should result in earlier use of the information as larger increments are more likely to result in the heuristic selecting the variable, i.e. the weighted degree factor dominating the domain factor of the heuristic.

- $AlldelBy1$: Incrementing based on deletions allows earlier use of contention information for the heuristic and should increase diversification. The increase in diversification for $dom/wdeg\text{-}nores$ may be quite beneficial. For random probing, however, one would expect performance to deteriorate as the information may obscure sources of global contention. Similarly, this may have an adverse impact on WTDI as the restarts are sufficient for diversification.

- $AlldelBydel$: This is a combination of the two previous strategies. Incrementing by the size of the reduction for each constraint should give a better estimate at how contentious the constraint is compared to $AlldelBy1$. It also allows for earlier use of weight information in the case of $dom/wdeg\text{-}nores$.

- $Del/noWipeBy1$: This final strategy was included to evaluate a form of sampling that could be distinguished from DWOs. This may provide evidence that sampling is related to contention rather than to failure in particular.

It should be noted that, due to the bi-directionality of the wipeouts in binary CSPs (i.e. if no value in variable X has a support in variable Y, this implies that no value in the domain of Y has a support in X), a number of methods could

have been used for the first case where weights are incremented by the size of the domain reduction upon wipeout. One could look at incrementing by the sum of the domain sizes of variables in the scope of the constraint prior to propagation resulting in a wipeout, the average of the domain sizes, or simply to increment by the size of the domain of the variable that was wiped. We chose the latter. The bi-directionality of wipeouts does not hold for deletions.

One must also bear in mind how these methods will impact the performance of $dom/wdeg$ compared to $wdeg$. As the constraint weights increase, the weighted degree will become the dominant factor in the heuristic to the point where the domain size may be reduced to a tie-breaking role. We showed in the previous chapter that this can have an adverse effect on search for some problem types.

4.2.1 Alternative Forms of Contention Experiments

The purpose of the following experiments is to assess how the different algorithms behave using these alternative measurements of contention, and to determine whether these measurements of contention provide better/worse indicators of the globally difficult elements.

Based on the findings of the previous chapter, our restarting regimen is 100 restarts with a cutoff of 30 *failures* per run for both RNDI and WTDI. The use of failures for the cutoff guarantees learning will occur on every run, while also reducing the need for problem-specific parameterization. For the experiments involving random probing, results are again averages of ten experiments. Where statistical significance is discussed between samples below, we refer to paired comparison t-tests at the 95% confidence interval unless otherwise stated.

As we are solely interested in how an algorithm behaves when the different forms of contention information are used for guidance, the RNDI algorithm used here only terminates when the problem is solved on the run to completion. In other words, when a problem is solved during random probing it does not terminate as done in the previous chapter. Therefore, the value for nodes explored with RNDI does not include nodes explored during the probing phase. The WTDI algorithm, on the other hand, does terminate during the probing phase as the information is being used throughout. Hence, the nodes explored for WTDI does include those

explored during the probing phase.

We first experimented on a set of 100 random binary problems with parameters ⟨50,10,0.183,0.369⟩ which, although relatively easy to solve, are in the critical complexity region. These were generated by Richard Wallace. Results are presented in Table 4.1, for both $dom/wdeg$ and $wdeg$. The latter was used to remove the impact of the domain factor, in order to more directly assess the performance of the different forms of measuring contention. For comparison, we note that $dom/fdeg$ and $fdeg$ averaged 1621 and 2625 nodes respectively on these instances.

Table 4.1: Search Efficiency with Different Sampling Strategies: Random Binary Problems

	dom/wdeg			wdeg		
	$nores$	RNDI	WTDI	$nores$	RNDI	WTDI
Normal	1538	1211	4489	2070	1713	3931
WipeBydel	1592	1204	4313	2140	1754	4097
AlldelBydel	1496	1392	4979	2259	2474	5540
AlldelBy1	1523	1302	5149	2284	2106	4678
Del/noWipeBy1	1530	1314	5017	2297	2173	4605

Notes. ⟨50,10,0.183,0.369⟩ problems. Mean search nodes per instance. For comparison, $dom/fdeg$ and $fdeg$ averaged 1621 and 2625 nodes *resp*. RNDI and WTDI had restarting regimen of 100 restarts with a cutoff of 30 failures per run. RNDI data doesn't include nodes from probing, and didn't terminate until instance was solved on run to completion.

For $dom/wdeg$-$nores$ sampling either deletions or failures gave comparable results for search, while sampling directly related to failures was slightly better for $wdeg$-$nores$ (albeit only statistically significant when comparing "Normal" with "AlldelBy1" and "DelnoWipeBy1"). Indeed all measurements of contention for both approaches showed improvement over the associated non-weighted version of the heuristics (i.e. $dom/fdeg$ and $fdeg$). These results suggest that any of these events can serve as an indicator of contention.

On the other hand, direct sampling of failure is better than sampling deletions for both RNDI and WTDI. This is clearer when WTDI and RNDI were combined with $wdeg$. Here there was no statistically significant difference between the two

methods that only use information regarding failures, but there was a statistically significant difference between each of these and the other three measures of contention.

Open Shop Scheduling

We further tested these approaches on the open shop scheduling problems (OSPs) described in the previous chapter (Section 3.4.4). The constraints of these problems have varying tightness and so the results may reflect more inherent differences between the approaches. In particular, one would expect weighting based on the size of the reductions to have a greater impact on the choices made.

Table 4.2: Alternative Measurements of Contention: $dom/wdeg$-$nores$. Open Shop Scheduling Problems, # Solved.

	$dom/fdeg$	$Normal$	$Wipe$ $Bydel$	$Alldel$ $Bydel$	$Alldel$ $By1$	$Del/noWipe$ $By1$
ost-4-95	10	10	10	10	10	10
ost-4-100	9	10	10	10	10	10
ost-5-95	5	**6**	**6**	5	5	5
ost-5-100	2	2	**4**	3	2	2
ost-5-105	6	**9**	**9**	**9**	7	8
$Total$	32	37	**39**	37	34	35

Notes: Number of instances solved (out of ten) for each problem set.
Figures in bold represent the best result over the different methods

Results are presented for the 3 sets (ost-n-95, ost-n-100, ost-n-105) for n=4 and n=5, except for the ost-4-105 set which were trivially easy for all methods. Each set contains 10 instances. An overall limit of one million nodes per instance was used. The results for average nodes include cases where the million node cutoff was reached and so are a lower bound.

Tables 4.2 and 4.3 show the results for $dom/wdeg$-$nores$ per problem set in terms of number of instances solved and average search nodes respectively. For comparison we also include the results for $dom/fdeg$. The first point to note is that, as for the random binary problems, all forms of assessing contention outperformed $dom/fdeg$ on these instances, both in terms of number of instances solved

and in search nodes on where all instances of a problem set were solved by all methods (ost-4-95).

Table 4.3: Alternative Measurements of Contention: $dom/wdeg$-$nores$. Open Shop Scheduling Problem, Average Nodes.

	$dom/fdeg$	$Normal$	$Wipe$ $Bydel$	$Alldel$ $Bydel$	$Alldel$ $By1$	$Del/noWipe$ $By1$
ost-4-95	56.3	3.7	5.3	**2.4**	4.2	2.8
ost-4-100	246.1	17.2	16.5	17.4	14.4	**12.7**
ost-5-95	596	**410**	418	527	536	538
ost-5-100	895	830	820	**717**	853	962
ost-5-105	406	134	**115**	119	331	277
Sum	2,199	1,395	**1,375**	1,382	1,740	1,794

Notes: Average search nodes per instance, includes failed attempts where search hit the million node cutoff.
Figures in bold represent the best result over the different methods

The best approach based on number of instances solved is "WipeBydel", with the approaches that focus neither on failures nor on the size of reduction performing poorest. However, if we compare the sum of the average nodes (Table 4.3) for the three best sampling strategies, we see that there is little difference between the three even though "WipeBydel" solved two more instances than the other two.

In particular, if we compare the average nodes for solving ost-5-100 using "Normal" and "WipeBydel", we see that there is only 10,000 nodes in the difference even though "Normal" solved two instances less. This implied that the two extra instances solved by "WipeBydel" were solved just under the node limit, which was confirmed by analysis of individual results. Indeed there is little difference in performance between the sampling methods on these instances, which was also the case for the random binary problems in the previous section. Therefore there is no evidence to suggest that the blame assignment issue negatively affects $dom/wdeg$-$nores$.

For RNDI, the results once again show that weighting based solely on failures is best for this form of sampling (Tables 4.4 and 4.5). "Normal" weighting with RNDI solved instances in the fewest nodes on average for all five sets. Furthermore, in most sets we see a clear difference in terms of average nodes between

Table 4.4: Alternative Measurements of Contention: RNDI. Open Shop Scheduling Problems, # Solved.

	$Normal$	$Wipe$ $Bydel$	$Alldel$ $Bydel$	$Alldel$ $By1$	$Del/noWipe$ $By1$
ost-4-95	10.0	10.0	10.0	10.0	10.0
ost-4-100	10.0	10.0	10.0	10.0	10.0
ost-5-95	9.6	**9.7**	9.2	8.7	8.2
ost-5-100	**3.7**	3.5	2.1	1.6	2.0
ost-5-105	**10.0**	9.9	9.6	9.8	9.8
$Total$	**43.3**	43.1	40.9	40.1	40.0

Notes: Number of instances solved (out of ten) for each problem set. RNDI had probing regimen of 100 restarts with a 30 failure cutoff per run. Weights were updated on run to completion. **N.B.** Algorithm doesn't stop if solved during probing phase.

the two RNDI approaches that use failure information only and the other RNDI approaches.

Table 4.5: Alternative Measurements of Contention: RNDI. Open Shop Scheduling Problems, Average Nodes.

	$Normal$	$Wipe$ $Bydel$	$Alldel$ $Bydel$	$Alldel$ $By1$	$Del/noWipe$ $By1$
ost-4-95	**1.3**	2.3	1.6	1.7	1.6
ost-4-100	**5.7**	6.7	12.1	16.3	16.1
ost-5-95	**88**	94	175	175	212
ost-5-100	**738**	760	858	870	853
ost-5-105	**18**	20	48	28	30
Sum	**850**	882	1,096	1,091	1,113

Notes: Average search nodes per instance, includes failed attempts where search hit the million node limit, doesn't include nodes explored during probing. RNDI had probing regimen of 100 restarts with a 30 failure cutoff per run. Weights were updated on run to completion. **N.B.** Algorithm doesn't stop if solved during probing phase.

A surprising result is that the worst approach for RNDI was still better than the best approach for $dom/wdeg$-$nores$ both in terms of number of instances solved and sum of average nodes. This confirms that any of these methods can be used as

indicators of contention, and also emphasizes the importance of making informed decisions at the top of the search tree for these problems.

Finally, we present the results for WTDI combined with the different measures of contention in Tables 4.6 and 4.7. All measures are extremely effective on these problem sets, with only one set where not all instances were solved (ost-5-100). On this set, the two methods which increment weights based solely on failures were two of the three best.

In terms of average nodes explored, the ordering becomes somewhat clearer. In the two cases where two approaches solved the same number of instances, there was approximately ninety thousand nodes in the difference (i.e. "WipeBydel" was more efficient than "AlldelBy1", and "AlldelBydel" was more efficient than "Del/noWipeBy1").

Table 4.6: Alternative Measurements of Contention: WTDI. Open Shop Scheduling Problems, # Solved.

	$Normal$	$Wipe$ $Bydel$	$Alldel$ $Bydel$	$Alldel$ $By1$	$Del/noWipe$ $By1$
ost-4-95	10 (1)	10 (1)	10 (1)	10 (1)	10 (1)
ost-4-100	10 (10)	10 (10)	10 (8)	10 (9)	10 (8)
ost-5-95	10 (0)	10 (0)	10 (0)	10 (0)	10 (0)
ost-5-100	**6** (3)	5 (4)	2 (0)	5 (3)	2 (0)
ost-5-105	10 (10)	10 (10)	10 (10)	10 (10)	10 (10)
$Total$	46 (24)	45 (25)	42 (19)	45 (23)	42 (19)

Notes: Number of instances solved (out of ten) for each problem set. Numbers in brackets are number of instances solved during probing. WTDI had probing regimen of 100 restarts with a 30 failure cutoff per run.Weights were updated on run to completion.

The experiments on random binary and open shop scheduling problems show firstly that any of these measures can be used successfully as indicators of contention. Inclusion of the various forms of contention measurement never led to a deterioration in performance with respect to the non-weighting strategy. Furthermore, sampling these different forms of contention during RNDI, and WTDI for the scheduling problems, led to an improvement over the $nores$ approaches in most cases. Together, these results show that contention can be successfully

sampled even when ignoring failures.

Table 4.7: Alternative Measurements of Contention: WTDI. Open Shop Scheduling Problems, Average Nodes.

	$Normal$	$Wipe$ $Bydel$	$Alldel$ $Bydel$	$Alldel$ $By1$	$Del/noWipe$ $By1$
ost-4-95	4.4	4.2	4.4	**3.9**	4.5
ost-4-100	**0.4**	0.5	1.8	32.7	1.9
ost-5-95	**31**	131	32	115	84
ost-5-100	**485**	503	815	577	854
ost-5-105	0.2	**0.1**	0.2	0.2	**0.1**
Sum	**522**	640	853	729	944

Notes: Average search nodes per instance per problem set, includes failed attempts where search hit the million node limit, includes nodes explored during probing. WTDI had probing regimen of 100 restarts with a 30 failure cutoff per run. Weights were updated on the run to completion.

Overall, we found that sampling events directly related to failure performed best. This was clearest for RNDI, where the benefits of increased diversification plays a smaller role than for the other two strategies. In this case, the key to improved performance is identifying those variables which are likely to cause failure, and the results show that this is best done by directly sampling wipeout events. Nonetheless, it is significant that search is very efficient in all cases.

4.2.2 Analysis of Weight Distributions Produced by Unbiased Sampling

We generated ten weight profiles (with different seeds) with RNDI for each weighting method on five sample instances from the 50-variable random binary problem set, and on a sample of the open shop scheduling problems, both soluble and insoluble (which are referred to as *Sched_Sol* and Sched_Insol in the following tables and figures). The instances selected from the random binary problem set were the five hardest when solved by $dom/wdeg$-$nores$, while for the scheduling instances we generated weight profiles for the first two instances of the ost-5-95 set and the first two instances of the ost-5-100 set. We also generated weight profiles for

ost-5-95-3, which was the instance with clearly defined insoluble cores discussed in the weight analysis of the previous chapter.

For the random instances, incrementing weights based on the size of the reduction had relatively little impact (the average weighted degree of "WipeBydel" was 170 compared to 133 for normal weighting, similarly AlldelBydel had an average weighted degree of 10705 compared to 6701 for AlldelBy1). For scheduling instances, on the other hand, there was a much larger impact as predicted (the average weighted degree for "WipeBydel" was 6313 compared to 256 for normal weighting, similarly AlldelBydel had an average weighted degree of 177081 compared to 4464 for AlldelBy1).

Table 4.8: Top-down correlation coefficients for different weighting methods across ten runs

	Random		Sched_Sol		Sched_Insol	
	Avg	Min	Avg	Min	Avg	Min
Normal	0.94	0.85	0.84	0.63	0.89	0.69
WipeBydel	0.93	0.86	0.84	0.53	0.89	0.75
AlldelBydel	0.96	0.84	0.85	0.52	0.90	0.69
AlldelBy1	0.96	0.84	0.89	0.70	0.90	0.70
Del/noWipeBy1	0.96	0.84	0.88	0.63	0.89	0.67

Notes. Sample of instances from each set.
Results in terms of average ("Avg)") and minimum ("Min").

Correlations across different seeds were calculated for each method using the top-down correlation coefficient [112]. The results are shown in Table 4.8. Correlations on random instances were high for all methods, greater than 0.92 on average and always greater than 0.85. There was greater variation in the weight profiles generated on the scheduling instances, with minimum correlations ranging from 0.52 to 0.75 for the five weighting methods.

For the insoluble scheduling instance discussed in the previous chapter (*ost*-5-95-3), we find that the top three variables weighted by all methods are variables from the same insoluble core. However, the top ranked variable of the failure only methods had the smallest domain, while the "Alldel" methods weighted a variable with a slightly larger domain size. Obviously this will impact the size of the proof of insolubility.

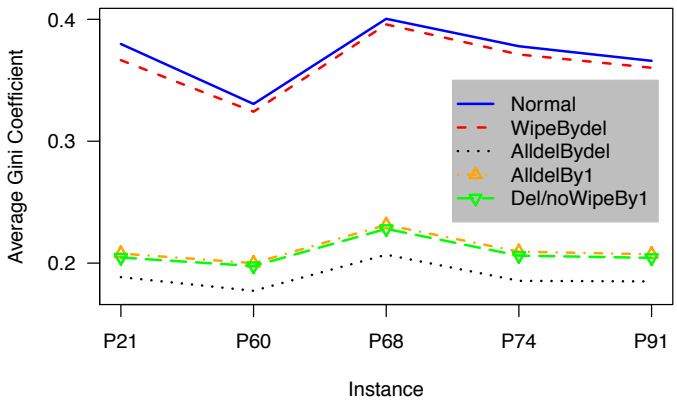


Figure 4.1: Average Gini coefficients of weights distributions for five sample random instances

We next assessed the distribution of weights after random probing. For the random instances, the Gini coefficient for the methods which only considered failures was roughly 0.36 on average for both (Figure 4.1). On the other hand, the Gini coefficients for the three methods that considered any effective propagation of a constraint were all approximately 0.21.

This difference was even more prominent for the scheduling instances, where each of the three "Alldel" methods again had an average Gini coefficient of 0.22 / 0.23, while the "WipeBydel" method had an average Gini coefficient of 0.54 compared to 0.46 for weights produced from the normal method of weighting. However, one would expect a smaller Gini coefficient when all effective propagations are weighted, since weight is increased for a greater number of contention events.

An alternative way to view the distribution of weight is to assess the level of discrimination between the top and bottom ranked variables. In particular, we take the difference in weighted degree for each pair of successively ranked variables. We then sum these differences for the top and bottom 50% of ranked variables, and compute the ratio of top discrimination to bottom discrimination. A value below 1 would mean there is greater discrimination amongst the bottom ranked variables, while a value above 1 would mean the opposite.

We plot the results for the average ratio per instance sample for each problem type in Figure 4.2. The main point to note is that the methods which only sam-

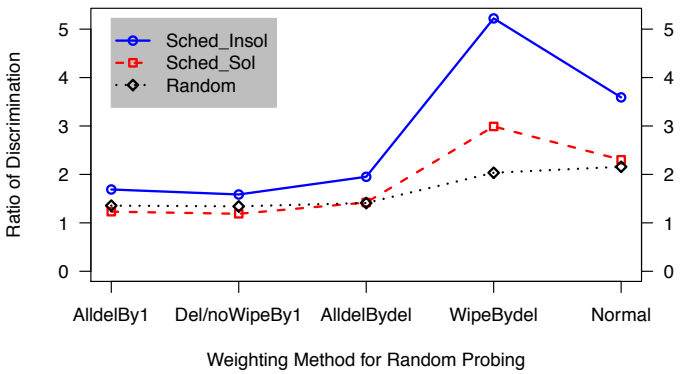


Figure 4.2: Ratio of Discrimination for Ranked Variables, Top 50% Versus Bottom 50%. Averages per instance sample.

pled DWOs always had a larger ratio of discrimination. The results also show that for these two methods, the ratio of discrimination was greater when incrementing based on the size of the reduction for the scheduling instances. As discussed previously the insoluble scheduling instances have a subset of insoluble cores, which explains why these have the largest ratios on average. Furthermore, for some runs on both random and scheduling instances the "Alldel" methods gave a ratio lower than 1, i.e. there was greater discrimination amongst the bottom ranked variables than the top ranked.

4.3 Sampling Based on Different Search Procedures

In this section we assess the importance of the MAC algorithm to the quality of information learnt by RNDI. We compared the normal MAC method of random probing with alternative methods for information gathering, which weight constraints that either cause domain wipeouts in systematic search or cause conflict in local search. These weights are then used by either $dom/wdeg$ or $wdeg$ to solve the problem.

The first method merely replaces the MAC algorithm with a weaker level of consistency, *forward checking* (FC), for the information gathering phase. For both FC-RNDI and MAC-RNDI there were 100 restarts with a *failure* cutoff of 30. All methods were followed by complete search with MAC.

The second method is a version of the "Breakout" algorithm (Morris [153]). We tried three variations of the breakout method for generating a weight profile. The first approach was to simply run the breakout method with min-conflicts heuristic from an initial random assignment until a total weight of 3000 had been generated. Thus, the total weight increase was the same as that of the two random probing procedures. Since we are solely interested in the quality of information learnt by each method, if the problem was solved before the cutoff during this local search phase it was restarted with a new random assignment (and reinitialised weights) until a run produced a weight increase of 3000. We refer to this method as BO-nores.

The second breakout approach we tried (BO-res), has more in common with random probing. Here the breakout method was repeatedly run to a fixed weight cutoff of 30. After each run, weights were added to a $total\text{-}weight$ counter and then reset. Thus weights from previous runs were not used by the min-conflicts procedure and so did not bias the new weights learnt. Each run had a different initial random assignment to the variables. After 100 such runs, the accumulated weights in the $total\text{-}weight$ counter were used by either $dom/wdeg$ or $wdeg$ as before.

The final breakout approach (BO-bias) is identical to the previous method, with the exception that weights were not reset after each run (and so a separate $total\text{-}weight$ counter was not used). Thus weights learnt from previous runs affected the choices made by the min-conflicts heuristic in subsequent runs and so biased the weights learnt. This approach is more consistent with that of Eisenberg and Faltings, in that they restarted the local search algorithm after reaching a predefined number of iterations.

It should be noted that all approaches, except forward-checking, solved some instances during preprocessing. However the algorithms only terminate on each instance after the run to completion with a weighted-degree heuristic, since we are solely interested in the quality of information collected by these preprocessing approaches.

We first tested these sampling strategies on the set of random binary problems with parameters $\langle 50,10,0.183,0.631 \rangle$. Table 4.9 presents the results, in terms of average nodes explored on the run to completion, for each sampling method fol-

lowed by systematic search with MAC using one of four heuristics: $dom/wdeg$, $wdeg$, $dom/frowdeg$ and $frowdeg$. The two latter heuristics use weights frozen after the sampling phase, i.e. weights were not updated on the run to completion, thus allowing a more direct evaluation of the quality of information learnt in preprocessing.

Table 4.9: Search Efficiency with Information Gathered under Different Search Procedures

	dom/wdeg	wdeg	dom/frowdeg	frowdeg
MAC-RNDI 100R 30WC	**1211**	**1731**	**1223**	**1632**
FC -RNDI 100R 30WC	1444	2122	1579	3434
BO-nores 3000WC	1619	2227	1848	3893
BO-res 100R 30WC	1271	1843	1271	2119
BO-bias 100R 30WC	1394	1954	1389	2204

Notes: $\langle$50,10,0.183,0.631$\rangle$ problems. Basic sample is 100 instances. All results are for the run to completion only and are the mean search nodes across ten experiments. "$frowdeg$" means weights were not updated on the run to completion. For reference, the average search nodes for $dom/wdeg$ and $wdeg$ without restarting were 1538 and 2070 respectively.

Weights learnt by random probing with MAC produced the best results in every case. The differences in means between MAC-RNDI and the other sampling methods were statistically significant at the 95% confidence interval for all except BO-res. In this case, the difference was only statistically significant when the weights were frozen and the domain factor was removed.

In order to further investigate the effectiveness of this breakout procedure we tested on harder problem sets (both insoluble and soluble) with parameters $\langle 200, 10, 0.0153, 0.55\rangle$. The heuristic used was $dom/wdeg$ and weights were frozen. For these instances the weights learnt by BO-res resulted in worse search performance than if no preprocessing was done. As previously shown MAC-RNDI with frozen weights improved search effort by roughly 30% on these instances.

The poor performance of FC-RNDI is not surprising as one would not expect it to be able to identify globally contentious variables as well as MAC-RNDI, since propagation in forward checking only occurs on constraints between the assigned variable and its neighbors. We tested whether improvement could be found

by increasing the diversity of the sample, with the expectation that a variable's repeated involvement in localized sources of conflict across different parts of the search space should be more indicative of global contention.

In particular, we ran FC with two alternative probing regimens, 30R100WC and 300R10WC, and compared with the results above. The strategy combining the largest number of restarts with the smallest cutoff did indeed produce the best results, however these were still significantly worse than those for MAC-RNDI.

It has also been suggested that weights learnt by FC-RNDI may work better for search with forward checking. This was tested by comparing FC-RNDI and MAC-RNDI where both were followed by systematic search with *forward checking* on the run to completion. Weights produced by MAC still resulted in superior performance (by an order of magnitude when the domain factor was removed) which further supports our hypothesis that MAC-RNDI is better at identifying contentious variables.

With regard to the breakout methods, it is clear that for these problems gathering a diverse sample is better than simply running breakout to a large cutoff. The results further show that unbiased sampling is better, matching our findings for random binary problems in the previous chapter.

Table 4.10: Average Nodes. Open Shop Scheduling Problems

		*-*nores*	MAC RNDI	FC RNDI	BO nores	BO res	BO bias
dom/wdeg	# Sol	37	43.3	41.4	**44.2**	41.1	42.2
	Nodes	1,395K	850K	1,038K	797K	996K	921K
wdeg	# Sol	34	**42.9**	42	42.5	41.1	39.5
	Nodes	2,059K	922K	1,106K	1,001K	1,168K	1,274K
dom/frowdeg	# Sol	-	**39.3**	36.8	**39.3**	38.2	38.5
	Nodes	-	1,152K	1,469K	1,204K	1,230K	1,255K
frowdeg	# Sol	-	**37.2**	32.3	34.4	29.5	29.6
	Nodes	-	1,541K	2,025K	1,794K	2,211K	2,240K

Notes: "#Sol" refers to the average solved over the 50 instances. "Nodes" refers to the sum of average nodes per problem set, including cases where the million node cutoff was reached. Restarting parameters were 100R30WC for all sampling strategies except BO-nores which had a cutoff of 3000WC.

We further tested the sampling methods on the open shop scheduling problems

(Table 4.10). Thornton found in his dissertation [198] that constraint weighting in local search is more suited to structured problems with clearly distinguishable subsets of easy and hard constraints, so one would expect the weights learnt by the breakout methods to result in better performance here than for the random binary problems. (Once again we remind the reader that all methods only terminate after the run to completion with a weighted degree heuristic, irrespective of whether a problem was solved during the sampling phase. In the case of BO-nores, if a problem was solved during breakout, the algorithm was restarted with reinitialized weights until a run generated a weight increase of 3000.)

The results firstly show that the weights learnt by any of the sampling strategies resulted in superior performance over the case where no weight information was available at the start of search (i.e. "$*$-$nores$"). This implies that there is a clear structure to these problems which can be identified even when simply running forward checking.

The improved performance of all the breakout methods reflects the findings of Thornton. Interestingly, BO-nores performed best when followed by search with $dom/wdeg$, whereas for the random binary problems it was consistently the worst sampling strategy. However, when we take a more direct assessment of the quality of information learnt (by freezing the weights and/or removing the domain factor from the heuristic), we see clearer differences between the methods.

In particular, the results for search followed by "$frowdeg$" show that the quality of information learnt by MAC-RNDI was better than that of the other methods. It is somewhat surprising that the best methods overall were MAC-RNDI and BO-nores as, in terms of sampling, these are polar opposites (extremely biased versus extremely unbiased). The other two breakout methods fall in between in terms of level of bias during sampling. However, we note that WTDI also performed extremely efficiently on these problems which indicates that biased sampling is suited to these types of structured problem.

4.3.1 Analysis of Weight Profiles Produced by Different Sampling Strategies

In this section we look at the weights generated by these different sampling methods, both for the problem sets tested and for the queens-knights problem introduced in the previous chapter (Section 3.4.1). We first examine the consistency of the rankings produced across ten experiments for each method on the same sample of problems used previously, i.e. the same 5 problems from the 50-variable random binary problem set, and the same sample of insoluble and soluble scheduling problems.

Table 4.11: Top-down correlation coefficients for different sampling methods across ten runs

	Random		Sched_Sol		Sched_Insol	
	Avg	Min	Avg	Min	Avg	Min
MAC-RNDI	**0.94**	0.85	0.84	0.63	0.89	0.69
FC-RNDI	0.88	0.77	0.80	0.52	0.82	0.54
BO-nores	0.55	0.16	0.68	0.22	0.89	0.74
BO-res	**0.92**	0.75	**0.96**	0.88	**0.95**	0.83
BO-bias	**0.95**	0.85	**0.95**	0.83	**0.95**	0.83

Notes. Sample of problems from each set.
Correlations greater than 0.9 are marked in bold.

Table 4.11 presents the top-down correlation coefficients on each problem, averaged over the problem type. For the random binary problems, MAC-RNDI and the two restarting breakout methods all had correlations greater than 0.9 on average. FC-RNDI and BO-nores were the least consistent across runs, with the latter in particular quite poor as the minimum correlation of 0.16 illustrates. A similar pattern occurred for the soluble scheduling problems with FC-RNDI and BO-nores the least consistent, although not to the same extent as for the random problems in the latter case.

The next three figures (4.3 4.4, 4.5) show the average weighted degree over ten weight profiles for a sample problem from each problem set. For the random binary problems (Figure 4.3), it is clear that RNDI assigns less weight to its bottom ranked choices than the other sampling methods.

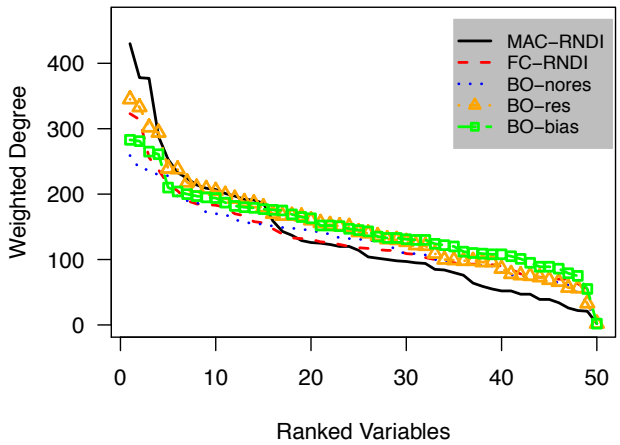


Figure 4.3: Averaged weight profiles over ten runs for sample random binary problem

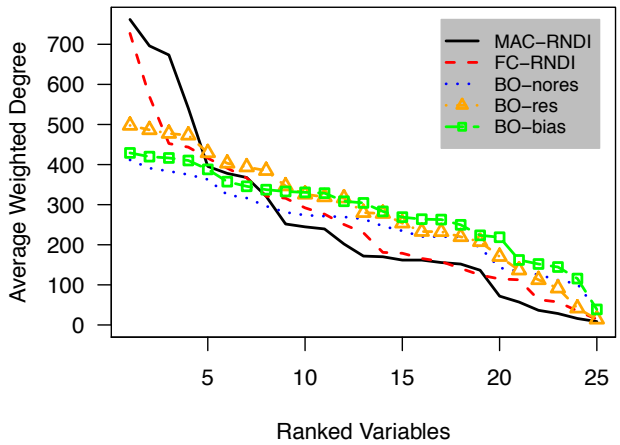


Figure 4.4: Averaged weight profiles over ten runs for a sample insoluble scheduling problem (ost-5-95-3)

In Figure 4.4, the breakout methods have clearly less discrimination than the systematic search methods for the insoluble scheduling problem. This may be the result of the problem containing a number of insoluble cores. Each time the constraints in conflict are incremented during breakout, at least one constraint in each insoluble core must be in conflict and will have its weight incremented. Thus weight will be spread across all insoluble cores mre consistently than with MAC-RNDI.

The final figure (4.5) shows the averaged weight profiles for a sample problem from the soluble scheduling set. All approaches appear to have a similar level of discrimination, albeit MAC-RNDI has slightly greater discrimination for the top ranked variables.

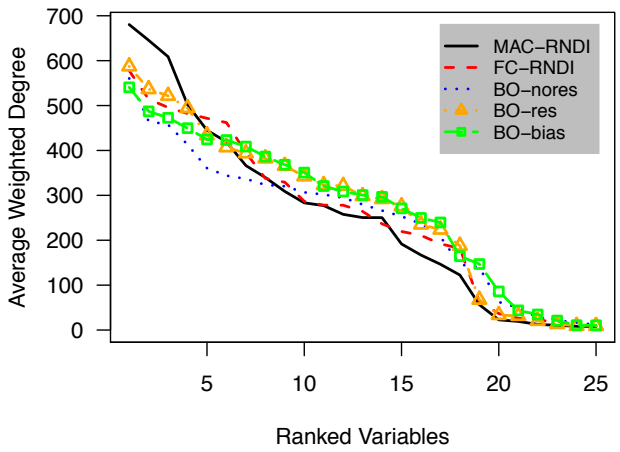


Figure 4.5: Averaged weight profiles over ten runs for a sample soluble scheduling problem (ost-5-100-0)

To further investigate this we compared the weight difference between successively ranked variables for each sampling method. Table 4.12 presents results in terms of the ratio of discrimination for the top 50% versus the bottom 50% of ranked variables, and in terms of the average weight difference between successively ranked variables.

The results show that the systematic search methods provided greater discrimination for the top ranked variables. Indeed the breakout methods gave near equal discrimination between top ranked and bottom ranked variables. Furthermore, the

average weight difference between successively ranked variables was consistently less for the different breakout methods of sampling.

This may partially explain the good performance of BO-nores on the scheduling problems when followed by search with $dom/wdeg$. Smaller differences between successively ranked variables means (a) the domain factor of the heuristic will play a larger role in the selections made; and (b) weights learnt during the run to completion are more likely to affect the variable ordering. In other words, smaller differences between successively ranked variables after sampling results in the heuristic being more sensitive to information from the current search state on the run to completion.

However, this is obviously not the sole reason for the good performance of BO-nores since the other breakout methods didn't perform as well despite having a similar level of discrimination after sampling. BO-nores is clearly identifying globally contentious variables at the top of the search tree.

Table 4.12: Analysis of level of discrimination between successively ranked variables on sample of soluble and insoluble scheduling problems.

	Ratio of Discrimination		Average Weight Difference	
	Sched_Sol	Sched_Insol	Sched_Sol	Sched_Insol
MAC-RNDI	2.07	3.52	26.8	31.8
FC-RNDI	2.66	2.13	28.5	29.0
BO-nores	0.83	1.05	20.5	20.6
BO-res	1.18	0.84	23.5	20.6
BO-bias	0.70	0.68	20.9	19.0

We also calculated the Gini coefficients of the weight distributions for each run. For the random binary problems, MAC-RNDI consistently had the largest Gini coefficient (averaging 0.37 over 10 runs on the 5 sample problems), BO-res had the next largest Gini coefficient on average (0.28), while the other three methods all had an average Gini coefficient of 0.22/0.23. Comparing these findings with the search performance of the different methods on the random problems, we see that the best methods were more consistent in their ranking of the variables after sampling (high correlations across ten experiments), and were more discriminatory (larger Gini coefficients).

The Gini coefficients for the scheduling problems support the findings in Table 4.12. MAC-RNDI had the most skewed weight distributions with an average Gini coefficient of 0.49 (over 10 runs on the 5 sample scheduling problems). FC-RNDI also had a high average Gini coefficient of 0.45, compared to average Gini coefficients of 0.37, 0.33 and 0.30 for BO-res, BO-nores and BO-bias respectively.

Finally we generated weight profiles for a sample queens knights problem qk-15-5-add, containing 15 queens and 5 knights. All methods weighted the knight variables highest, however our interest is in the weight accumulated on the queen variables, given in Figure 4.6. For clarity, we show the weight increase (i.e. the weighted degree minus the degree) per queen variable after sampling.

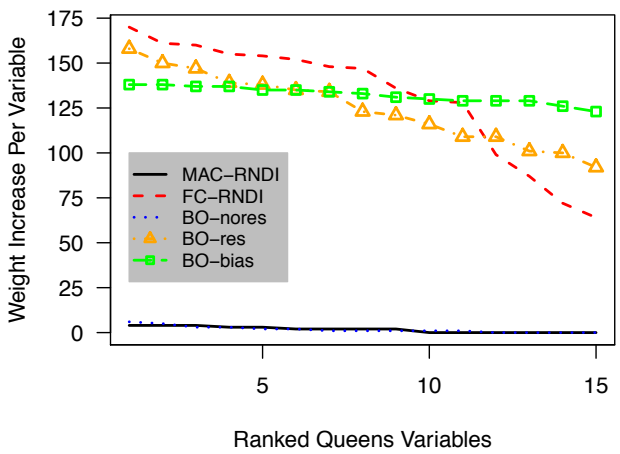


Figure 4.6: Sample weight profile for qk-15-5-add. Weight increase on queens only.

The figure shows that MAC-RNDI and BO-nores were least susceptible to local sources of contention (i.e. conflicts between queen variables), with an average weight increase of 1.7 per queen variable. The other three methods, on the other hand, had an average weight increase of over one hundred.

For FC-RNDI, this shows that the weak propagation of forward checking results in constraints that are local sources of contention receiving a high weight. For the breakout methods, it is clear that there are a number of queens in conflict at the first local minimum. The breakout method quickly finds a solution to the queens problem and then proceeds to consistently weight the knight variables. However,

each time the local search algorithm is restarted, some queens are weighted.

Overall, we have shown that MAC-RNDI leads to greatest discrimination amongst variables, while remaining relatively consistent in its ranking of variables. The restarting breakout algorithms were extremely consistent in their rankings but did not provide as clear a distinction between the levels of contention of the variables.

4.4 Importance of Initial Choices

One hypothesis for the improved search performance of $dom/wdeg$ when using weights from random probing is that it is mainly due to better variable selections at the top of the search tree. We tested this hypothesis by restricting the depths at which the information from probing is used.

More specifically, we perform random probing and then use one of the following strategies on the run to completion:

- select variables from depth 1 to depth x using $dom/fdeg$, and use $dom/frowdeg$ for all choices beneath this depth (referred to as "domfdeg1st" in figures).
- select variables from depth 1 to depth x using $dom/frowdeg$, and use $dom/fdeg$ for all choices beneath this depth (referred to as "domfrowdeg1st" in figures).

We first tested the importance of initial choices on the random binary problem set with parameters $\langle$50,10,0.183,0.631$\rangle$, where all problems are soluble. The results are shown in Figure 4.7. We include three benchmark comparisons, $dom/fdeg$, $dom/wdeg$-$nores$, and random probing followed by $dom/wdeg$ with frozen weights ("RNDIfro"). Furthermore, we note that the peak failure depth on average for the run to completion of RNDIfro was 8, while the maximum failure depth on average was 17.

The figure shows that using weights from random probing to select the first three variables alone already results in improved performance over $dom/wdeg$-$nores$. Similarly, we note that choosing just the first three variables with $dom/fdeg$ and then choosing all others beneath using $dom/frowdeg$ results in a deterioration in performance over RNDIfro.

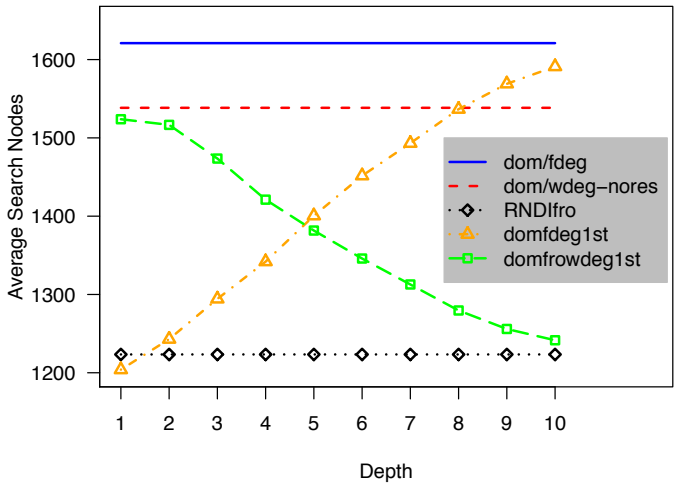


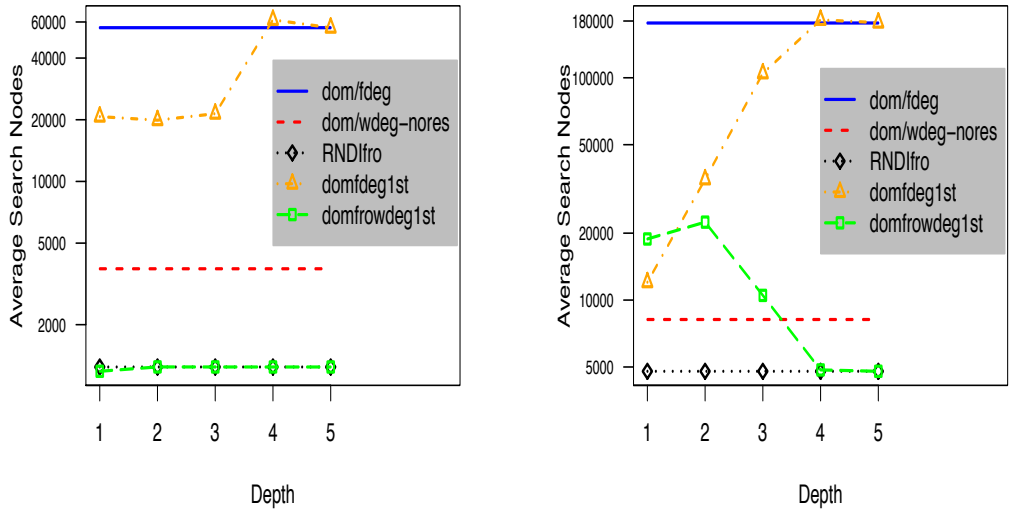
Figure 4.7: Depth to which probe weights are beneficial on random binary problem set ⟨50,10,0.183,0.631⟩

However, we note that the deterioration (*resp.* improvement) in performance is quite consistent for $domfdeg1st$ ($domfrowdeg1st$ *resp.*). Furthermore, the impact of the weights from probing is still evident at a depth of 8, the average peak failure depth, which contradicts the hypothesis that the benefits of weights learnt during random probing is restricted to choices at the top of the search tree for these problems.

We also tested on two sets of the Taillard open shop scheduling problems, the set ost-4-95 consisting of 10 insoluble problems (Figure 4.8(a)), and the set ost-4-100 consisting of 10 soluble problems (Figure 4.8(b)). For the latter set, we only include results for problems which were solved by all methods on all runs, of which there were eight.

Contrary to our findings on the random binary problems, here there is clear evidence that the benefits of probing are restricted to the choices made at the top of the search tree. For the insoluble problems, there was no fall off in performance over RNDIfro when using $dom/frowdeg$ at depth one only. Similarly, choosing the first variable alone with $dom/fdeg$ and then choosing all others with $dom/frowdeg$ was significantly worse than $dom/wdeg$-$nores$ and RNDIfro.

This may be somewhat surprising as $dom/wdeg$-$nores$ will make the same first variable selection. At the same time, it is clear from the results of RNDIfro



(a) ost-4-95 (b) ost-4-100

Figure 4.8: Importance of Initial Choices for Scheduling Problems

that the weights learnt are not of poor quality as it averaged the smallest proofs of insolubility for these problems.

A possible explanation is that these problems contain insoluble cores, where a subset of jobs and/or machines are overconstrained. If the first variable selected by $dom/fdeg$ is part of an insoluble core then $dom/wdeg$ will identify this and restrict search to the variables in the insoluble core. On the other hand, $dom/frowdeg$ may still select variables in the insoluble core which had the highest weight after probing, even if it is not connected to the variable chosen by $dom/fdeg$. In this case, search with the $domfdeg1st$ strategy may be proving insolubility of the highest weighted core from the probing phase for each value of the variable selected by $dom/fdeg$.

An uninformed selection for the first variable has a less negative impact for the soluble problem set. However, at a depth of four in both cases search performance is equivalent to that of search with the heuristic which was used first. These results show the importance of good choices at the top of the search tree, especially for the scheduling problems which are highly sturctured.

4.5 Further Analysis of the Nature of Unbiased Sampling

The majority of the work performed in this section was done by Richard Wallace and was published in [218].

4.5.1 Local Versus Global Contention

The depth at which failures occur may also indicate the globality of the contention, i.e. the deeper in search that a failure occurs, the more specific the context of that failure and thus the more localized the information. One would therefore expect that learning from failures that occur deeper in search when sampling would result in poorer search performance on the run to completion.

To test this hypothesis, sampling in the probing phase was restricted to different ranges of depths, i.e. only failures that occurred when search was in a given depth range were weighted. The ranges tested were 1-5, 6-10, 11-15, 6-15 and 14-18. Experiments were performed on the set of random binary problems with parameters $\langle 50,10,0.183,0.631 \rangle$. The probing regimen was 40 restarts with a cutoff of 30 failures per run. Weights were frozen on the run to completion.

The expectation of this set of experiments was that weights learnt from conflicts in the depth range 1-5 would result in better performance than those learnt from deeper in search. However, contrary to this expectation, there was no discernible difference in search performance on the run to completion when using weights learnt from the different ranges.

An alternative test was to consider the number of variables whose instantiation led to a domain reduction. When a wipeout occurred, weights were only incremented if the number of such "preclusion variables" was within a given range (greater than 3, 4 or 5, and less than 3 or 4). The expectation was that the fewer the number of preclusion variables the more globally contentious the constraint which caused the failure, and thus search performance with this information should be better than when larger numbers of preclusion variables were allowed. Once again, however, this did not turn out to be the case. These two results contradicted the hypothesis that the more specific the context of the failures, the poorer the estimate

of contention.

A final test was performed where random probing was restricted to a specific part of the search space. This was done be selecting variables until a depth i using lexical variable ordering, and then randomly selecting variables below this depth during the random probing phase. The different values of i that were tested were 1, 3, 5, 10, and n (i.e. all variables were selected lexically). The probing regimen was 40 restarts with a cutoff of 50 nodes per run. Here there was a clear deterioration in search performance as the sampling was restricted to a smaller part of the search space.

This experiment was repeated, but with variables chosen by minimum-degree to depth i, instead of lexical ordering. This was done so as to increase the probability that variables which are likely to be sources of global contention, i.e. those with high degree, would receive most weight. Search performance again deteriorated when i was greater than 1.

This last set of experiments supports the idea that the effectiveness of probing for these problems depends on sampling across the entire search space, and that these problems do contain sources of global contention. In light of this, the first two sets of experiments can be interpreted as showing that, as long as one samples across the search space, the same variables and constraints will cause most failures, even when the contexts of those failures differ.

4.5.2 Search after Random Probing: Policy Measures and Heuristic Actions

A further hypothesis with regard to the improved performance when using weights learnt by random probing is that this is due to an improvement in the fail-firstness of the weighted-degree heuristic. This was tested by comparing the heuristic's adherence to the policies introduced by Beck et al. for evaluating heuristic performance [24]. The first policy (*promise policy*) concerns enhancing the likelihood of remaining on a solution path by correctly extending the current partial solution. The second policy (*fail-first policy*) concerns minimizing the size of the refutation when search is not on a solution path, ie. a mistake has been made.

The promise measurement is a sum of probabilities across all complete search

paths, while the fail-first measurement is the mean refutation size (where a refutation is an insoluble subtree, rooted at the initial deviation from the solution path, that was explored). It has been shown that, in general, better heuristics improve adherence to both policies [216]. Similarly Hulubei [110] found that $dom/wdeg$ performed best out of the heuristics tested in their experiments on random problems because it struck the best balance between promise (measured in terms of number mistakes encountered) and fail-firstness, even though it was not the best heuristic in terms of the individual metrics.

Table 4.13: Adherence to Policy Assessments

	Policy Measures	
	Promise	Refutation Size
$dom/fdeg$	0.00041	437
$dom/wdeg$-$nores$	0.00042	366
RNDI	0.00046	282

Notes: $\langle$50,10,0.183,0.631$\rangle$ problems. Averaged over 100 problems. Results for RNDI are for the final run only.

Since variable selection using $dom/wdeg$ is not well defined or easily replicated (either in the non-restarting case or after random probing), each problem was solved and the ordering of the variables on the solution path was stored and reused for calculating promise and fail-firstness measures. Although this means that search is less efficient than the dynamic case, it allows better assessment of the information gained at the end of search for both methods. Furthermore, promise calculations involved an all-solutions search, while fail-firstness measurements were based on search for one solution.

Experiments were carried out on the set of random binary problems with parameters $\langle$50,10,0.183,0.631$\rangle$. The results showed that the main basis for improvement with random probing was indeed down to a greater fail-firstness, as evidenced by smaller refutations on average (Table 4.13), compared to the measurements of promise which are similar for all. We further note that, as expected, the improved performance of $dom/wdeg$-$nores$ over $dom/fdeg$ can also be attributed to enhanced fail-firstness.

However this information does not describe the nature of the refutations themselves. In particular, it could be the case that search after random probing encountered many more mistakes but explored smaller subtrees when refuting them. Thus, it is also of interest to note the depth in search at which mistakes were encountered, and the average refutation size at these depths.

Table 4.14: Refutations per Depth in Search

	$dom/wdeg$-$nores$		RNDI	
Depth	Frequency	Mean Size	Frequency	Mean Size
1	1.3	1883	1.6	1295
2	1.2	597	1.6	349
3	1.1	219	1.2	127
4	1.1	68	0.9	51
5	0.8	24	0.8	26

Notes: $\langle$50,10,0.183,0.631$\rangle$ problems. Averaged over 100 problems. Depth refers to the depth in search at which the initial mistake was made. Results for RNDI are for the final run only.

The results in Table 4.14 show that the biggest differences were for mistakes at the top of the search tree. Here, although $dom/wdeg$-$nores$ encountered slightly fewer mistakes, it needed much larger refutations on average to return to the solution path. This also provides further corroboration of the importance of the choices made at the top of the search tree.

Heuristic Actions

Recent work has shown that, for problems with unspecified structure, there are two distinct forms of heuristic action that result in different patterns of variation in search efficiency across problems [213], [215]. Based on factor analysis [98] of search efficiency of different heuristics, these two forms appear to be related to (i) the buildup of contention, and (ii) the simplification of the underlying subproblem.

Examples of heuristics which are associated with the buildup of contention factor are domain based heuristics such as $dom/fdeg$ and *Brélaz*, while degree only heuristics such as $fdeg$ are associated with the simplification factor. In Ta-

ble 4.15, the results of the factor analysis are presented for a set of random binary problems and a set of coloring problems.

In the table, deg refers to the static degree of the variable, $ff2$ and $ff3$ are two of the fail-first heuristics introduced by Smith and Grant [189]. For both problem sets shown, Factor 1 is the buildup of contention factor and Factor 2 is the simplification factor. (Note that the coloring problems are those studied in the previous chapter (Section 3.4.3), for which RNDI, followed by $dom/wdeg$, performed poorly due to a negative interaction between the domain factor and the weights from the probing phase).

The results show that, while $dom/wdeg$-$nores$ and $wdeg$-$nores$ load on the same factor as their foundation heuristics ($dom/fdeg$ and $fdeg$ respectively), search using frozen weights after random probing had a distinct pattern of correlations. For the set of random binary problems, there was a moderate loading on both factors while there was also a large amount of variance that was unique to the probing methods. For the coloring problems, the results were similar in that there was not a large difference between the loadings on the two factors. However, here the loadings on the two factors were much less, most of the variance was unique to the probing methods.

It has been shown that the best search performance occurs for a weighted-sum combination of two heuristics that load on separate factors, even when the individual heuristics perform relatively poorly [214]. Furthermore, the best combination with MAC was generally when both base heuristics were weighted equally in the weighted-sum. For the random binary problems in Table 4.15, we note that the probing strategies result in more balanced loadings on both factors, compared to the loadings of the associated weighted degree heuristics which load heavily on one or the other factor. This follows previous findings for weighted combinations of heuristics.

4.6 Related Work

Balafoutis and Stergiou have also proposed variations on the constraint weighting method with the goal of improving the quality of information learnt [11]. They firstly tackled one aspect of the blame assignment issue; when a DWO occurs

Table 4.15: Factor Analysis of Heuristic Search Performance

heuristic	Random Problems			Coloring Problems		
	Factor 1	Factor 2	Uniqueness	Factor 1	Factor 2	Uniqueness
deg	0.445	**0.840**	0.097	0.267	**0.787**	0.310
$fdeg$	0.387	**0.910**	0.021	0.109	**0.935**	0.114
Brélaz	**0.749**	0.396	0.282	**0.682**	0.247	0.474
dom/deg	**0.890**	0.437	0.017	**0.830**	0.125	0.296
$dom/fdeg$	**0.887**	0.456	0.005	**0.969**	0.163	0.035
$ff2$	**0.775**	0.412	0.230	0.141	**0.935**	0.105
$ff3$	0.595	0.507	0.389	0.273	**0.747**	0.367
$wdeg$	0.420	**0.854**	0.075	0.134	**0.895**	0.180
$dom/wdeg$	**0.873**	0.466	0.020	**0.967**	0.123	0.050
RNDI-$wdeg$	0.385	0.566	0.531	0.285	0.361	**0.788**
RNDI-$dom/wdeg$	0.487	0.436	0.573	0.380	0.137	**0.837**
SS loadings	4.786	3.975		3.996	3.448	
cumulative var.	0.435	0.796		0.363	0.677	

Notes: Random problems had parameters $\langle 50,10,0.183,0.631 \rangle$ problems. Coloring problems had parameters $\langle 50, 6, 0.27 \rangle$. Correlations greater than 0.6 are shown in bold. Results for RNDI are for the final run only and are for one experimental run (similar results were found with different seeds for RNDI). The probing parameters were 40R50C for the random binary problems and 100R50C for the coloring problems.

the constraint which is weighted is the constraint that *last* removed values from the variable, even though some other, more contentious, constraint(s) may have removed most of its values and thus be more to blame.

They dealt with this blame assignment issue by recording every constraint which removes a value from a variable's domain during propagation. Then, when a DWO occurs for variable x, say, the weights on all constraints that removed at least one value from the domain of x are incremented in one of the following three ways: (i) unit increments; (ii) increment by the number of values the constraint removed from the domain of x; and (iii) increment by the normalized number of values removed from the domain of x by the constraint, i.e. increment by the ratio of number of values removed to the domain size of x.

Based on the observation that different revision ordering heuristics can lead to different DWOs occurring during a pass of AC, they proposed a separate weighting method to identify *potential* DWOs. During a pass of arc consistency, ev-

ery constraint that has a fruitful propagation (i.e. removed at least one value) is recorded. If a DWO occurs, all constraints that had a fruitful propagation during the current pass of AC have their weights increased. They refer to this method as "fully assigned".

Finally, they implemented an "aging" strategy for constraint weighting, where all constraint weights are divided by a constant every x backtracks. A similar approach has been widely used in the SAT community, e.g. [155]. The basis for this is the assumption that more importance should be attached to recent conflicts than older conflicts.

They compared these new methods with $dom/wdeg$ and with the $AlldelBydel$ heuristic introduced earlier. All methods were combined with a geometric restarting strategy [221]. For the problems studied, there was no clear winner although it was interesting to note that for the seven problem classes studied, $dom/wdeg$ was never the fastest, whereas five of the six other methods had a "win" on at least one of the problem classes. Furthermore, aging worked extremely well for certain problem classes, but quite poorly for others.

Balafoutis and Stergiou extended their experimental comparison in [12], testing on a wider range of problems. Here they found that $dom/wdeg$ was fastest overall, closely followed by $AlldelBydel$ and *fully assigned*. These results, combined with our findings in Section 4.2, show that the confluence of failures on the constraints of a variable is more important than the apportioning of blame for individual failures.

In the latter paper Balafoutis and Stergiou also investigated the benefits of using constraint weights to guide revision ordering during arc consistency. They found that, for the instances tested, using $dom/wdeg$ in a variable oriented revision ordering consistently performed best on structured problems, and was best overall on random problems.

Another aspect that affects the information learnt, and thus the performance of $dom/wdeg$, is the value ordering. It has been widely assumed in the CSP community that value ordering does not affect search if one is looking for all solutions or proving a problem insoluble, when using d-way branching. This is because all values in each selected variable's domains will need to be explored. However Mehta and van Dongen [150] showed that the value ordering *does* affect search when one

is using information from previous search states to guide subsequent search such as in the weighted-degree heuristic, even if the problem is insoluble.

However it is still unclear what is the best value ordering heuristic to use in combination with a weighted degree heuristic. Consider a heuristic such as Geelen's $promise$ heuristic [71]. This may remove some noise from the data by avoiding failures which were due to poor value selection (and thus are localized) as opposed to those weights which are indicators of contention.

At the same time, there are disadvantages to such an approach. Firstly these heuristics are often costly to calculate. Secondly, the use of such a heuristic will generally result in failures occurring deeper in search. This means that it will take longer for weights to accrue, and thus affect the variable ordering. There is evidence to suggest that, when combining a weighted degree heuristic with binary branching search, choosing values based on the *fail first policy* may yield better results than a value ordering based on the promise policy ([136]).

There have been a number of hybrid approaches proposed, combining local and systematic search, that use local search to identify difficult elements of a problem This information is then used to guide the branching decisions of a systematic search algorithm. The approach of Eisenberg and Faltings for CSPs [60], discussed earlier, is one example.

Mazure et al. proposed a similar method for the satisfiability domain, which interleaved a GSAT type local search with a DP systematic search algorithm [147]. The local search algorithm was used to identify difficult literals for guiding the systematic search algorithm. In their method, each clause is scored based on the number of times it was falsified during local search. Similarly, each literal is scored based on the number of times it appeared in a falsified clause. At each branching step, the local search algorithm is called with respect to the remainder of the SAT instance. The literal with the highest score is selected

The difference between the approach of Mazure et al. and that of Eisenberg and Faltings is that, for the former, the information supplied by the local search algorithm is specific to the current search state. For the latter, and for the breakout methods we tested, the information supplied by local search was a global overview of the contentious constraints. One would expect the former to result in a smaller search tree, although the cost of running local search at each branching step may

be prohibitive.

Finally, we note the activity-based search strategy (ABS) recently proposed by Michel and Van Hentenryck [151] is similar to the *AlldelBy1* weighting method. The main difference is that, in ABS, the weight on a variable is incremented if it has had its domain reduced after applying a filtering algorithm. For *AlldelBy1*, on the other hand, the weight on a variable (via its constraints) is incremented each time a constraint on the variable caused a domain reduction, either to the variable or to one of its neighbors. Furthermore, an aging strategy is employed in ABS, which focuses search on variables which have recently been "active".

4.7 Chapter Summary

In this chapter we have investigated the quality of information learnt by constraint weighting methods under different conditions. We introduced alternative measurements of contention and showed experimentally that any of these measurements can be used by a weighted degree heuristic to improve search over its associated non-weighted heuristic ($dom/fdeg$ or $fdeg$). We compared these methods across three different paradigms: weighted degree without restarting; weighted degree with restarting; and weighted degree with weights initialized through random probing. In most cases, we found that using measurements of contention directly related to failure resulted in the best search performance. This implies that correctly assigning blame for each individual failure of a variable is not as important as the confluence of weights on the constraints of a variable.

We assessed the importance of maintaining arc consistency during random probing for identifying the critical variables. The results showed that using a weaker level of consistency resulted in poorer search performance on the run to completion. Analysis of the weights produced showed that this form of consistency is more susceptible to local sources of contention. However, in some cases search performance was still better than when no weight information was available at the start of search, which shows that the weaker level of consistency was still available to identify contentious variables.

We also compared weights generated by the breakout algorithm in local search with those generated by random probing with MAC. We tested three variations

with different levels of bias in the information learnt. For random problems the version of breakout, which was restarted a number of times with weights reset each time, performed best of the breakout methods and was only statistically significantly worse than normal random probing when followed by $wdeg$ with frozen weights. The worst breakout method was the non-restarting version, which resulted in worse search performance than if there was no weight information available at the start of search.

However, the opposite was the case for the structured problems, with the non-restarting version of breakout performing best. These results reflect similar findings in the previous chapter, where biased sampling was extremely poor for unstructured problems but performed well on structured instances. Analysis of the weight profiles produced by the different sampling strategies showed that random probing with MAC provided the most discrimination between ranked variables on all problem types.

Our study of the nature of sampling after random probing proved a number of hypotheses. Firstly, the good performance of search after random probing can be mainly attributed to an improvement in choices at the top of the search tree, and an improvement in the fail-firstness of the heuristic. Analysis of the depths at which failures occur showed that, as long as sampling was performed across the search space, the globally contentious variables received the most weight no matter what the contexts of the individual failures.

Finally, search performance with the weighted degree heuristics, in both their basic form and with weights frozen after random probing, were compared using factor analysis. The results showed that the weighted degree heuristics load on the same factors as their base heuristics. On the other hand, most of the variance associated with search when weights were initialized by random probing was unique to this method, and did not load strongly on either of the factors which the other heuristics loaded on.

Chapter 5

Solving Dynamic CSPs Through Failure Reuse

5.1 Introduction

In many real world situations, combinatorial problems are defined in a dynamic environment where changes may occur over time to the problem definition, and each time a problem is altered a new solution may need to be computed. Examples include online scheduling, where problem changes may occur through user interaction (e.g. [73]); and university timetabling where the problem changes slightly each year due to the arrival/departure of lecturers, the addition/removal of courses, etc. (e.g. [61]).

Such cases may be modeled as dynamic constraint satisfaction problems (*DCSPs*), which are sequences of static CSPs. Each CSP in the sequence is an altered version of the preceding CSP. The alterations may be in the form of addition/removal of constraints or variables or domain values for variables.

In order to solve these problems, one could simply apply the same static CSP solving technique to solve each problem in the sequence. However, this may result in a lot of redundant search effort and cause unnecessary disruption to the current solution. Therefore, when solving a DCSP, one would like to take advantage of the fact that one is solving a sequence of similar problems in order to reduce search effort and increase solution stability (i.e. minimize the changes made from the

previous solution in the sequence).

Many methods have been proposed for handling DCSPs, as described in the survey of Verfaillie and Jussien [208]. These methods can be split into two categories: proactive methods (e.g. find robust solutions [217]), where one knows the changes that are likely to occur; and reactive methods (e.g. solution reuse [209]), where information learnt while solving the previous problem is used to solve the subsequent problem.

Our work focuses on reactive techniques, where we assume no knowledge exists regarding the changes that are to occur (however this is not to say that the methods cannot be used in situations where such information is available). Reactive techniques have been further categorized, based on the type of information which is stored, into reasoning reuse approaches (where information regarding inconsistency is stored) and solution reuse approaches (where information regarding consistency is stored). Most research in this area has concentrated on developing/improving proactive or reactive techniques.

However, little attention has been paid to the impact of changes on a problem. In particular, it is acknowledged that solution/reasoning reuse techniques, although generally effective even in the face of large changes, perform poorly when changes occur to problems at the phase transition. It is suggested that this is because even small changes may have a large impact on the nature of the instance to solve at the phase transition. In this case it is proposed that the best approach may be a two-tiered strategy, where solution/reasoning reuse techniques are used for a short time and, if the problem remains unsolved, the problem is solved from scratch [208].

In this work we test the hypothesis with regard to the impact of small changes on CSPs at the phase transition. We show that many important attributes associated with a problem are sensitive to alterations when at the phase transition, e.g. the search effort to find a solution / prove a problem insoluble; the number of solutions to a problem; promise and fail-firstness measurements; and the individual elements of the solution set.

We identify one problem feature which is not greatly affected by alteration, namely the major points of contention in a problem. This led to the development of a new approach for solving DCSPs, based on contention reuse. We show experi-

mentally the benefits of such an approach for a range of alteration types, including cases where alteration affected the solubility of the problem. The advantages, and disadvantages, of this approach are investigated.

We also address the issue of solution stability by adding a simple enhancement to our approach, namely a value ordering heuristic based on the solution to the preceding problem in the sequence. This compares favorably with a traditional solution reuse method, both in terms of solution stability and search effort.

The breakdown of the chapter is as follows. The next section gives a formal definition of the DCSP and outlines the main objectives for solving the problem. We then provide a brief overview of the techniques which have been proposed in the literature for handling DCSPs. Section 5.4 describes the experimental methods which are used in the chapter. In the following section we present our investigation of the impact of small changes on problems at the phase transition.

We assess the stability of the points of contention and introduce a new approach for DCSPs in Section 5.6. This approach is tested on a wide variety of DCSPs with different forms of alteration. The section thereafter is focused on solution stability, including an examination of a traditional solution reuse method, analysis of the nearest solution possible for the DCSPs tested, and an enhancement of our basic method to incorporate a solution reuse component. The final section provides conclusions of the work presented in the chapter.

5.2 Background

Using the definition of Dechter and Dechter [56], a dynamic constraint satisfaction problem (DCSP) is a sequence of static CSPs where each CSP is an altered version of the preceding CSP in the sequence. These alterations are generally the addition and/or deletion of some of the basic elements of the problem (i.e. constraints/variables/domain values). The addition of constraints/variables and the deletion of values are forms of problem restriction, while deletion of constraints/variables and addition of values are forms of problem relaxation.

More formally, a DCSP of length l is a sequence of CSPs $\{P_0, P_1, \ldots P_l\}$ s.t $P_i = (P_{i-1} + E_{a_i} - E_{r_i})$ for $i = \{1 \ldots k\}$. E_{a_i} is the set of elements (constraints, variables, domain values) added to problem P_i while E_{r_i} is the set of elements

that were in problem P_{i-1} but are not in problem P_i. We refer to P_0 as the *base* problem, and the set of changes $(E_{a_i} - E_{r_i})$ as a *perturbation*.

The above definition is quite broad, indeed any pair of independent CSPs (P_j, P_z) form a DCSP, where $E_{a_z} = P_z$ and $E_{r_z} = P_j$. Thus, it is implicitly assumed when referring to DCSPs that the changes between two problems in the sequence are limited, i.e. only a fraction of the elements are altered per perturbation.

The main requirements when solving a DCSP, as outlined by Verfaillie and Jussien in [208], are:

1. Limit the time taken to compute a new solution after the problem has been altered.

2. Limit the disruption caused by the new solution, i.e. one would like a new solution to be as similar as possible to the previous solution.

3. Limit as much as possible the need for successive online problem solvings (we do not want to be continually changing the solution, e.g. truck drivers do not want their route constantly changing due to small changes in traveling conditions).

4. Keep producing consistent and optimal solutions.

We will use the following notation in discussing instances of DCSPs throughout the remainder of the chapter. Per$i$-j refers to the jth problem generated by altering base problem i (Per$i$-0). For DCSPs of length greater than 1, Per$i$-sj refers to the jth problem in a sequence of successive alterations beginning at base problem i. For means, since we are sampling over DCSPs generated from a set of base problems, this notation can be simplified to Per$j$/Pers$j$. For example, Per2-0 refers to the second base problem generated, Per2-3 refers to the third perturbed problem generated by altering base problem Per2-0, while Per2-s3 refers to the third perturbed problem in the sequence of successive alterations beginning at base problem Per2-0.

5.3 Previous Techniques

There are two different types of approach for solving DCSPs. The first type involves *proactive* methods for satisfying the goals of minimizing search effort and minimizing changes to the solutions. The second type involves *reactive* methods. Proactive methods aim to find solutions that will be most resistant to changes in subsequent problems, or that can be easily modified to find alternative solutions if changes negate the validity of the current solution.

Proactive methods can be further split into two categories, those that attempt to find *robust* solutions and those that attempt to find *flexible* solutions. Robust solutions are those which, given a model of the possible changes, are most likely to remain a solution after problem alteration. An example of such an approach is one where probabilities are available for the values in the domains of the variables which may change [217], [222].

A flexible solution (which can be a partial/complete solution, or a set of solutions) is one which can be easily modified to produce a new solution should the current solution become invalid after problem alteration. One method is to store the set of solutions in a binary decision diagram (BDD) [36] or an automaton [207]. This is known as a conditional solution. An alternative method would be to store the interchangeable values (where they exist) [66] for each value in the current solution, providing a quick, compact method for storing a large number of solutions.

Another example of a flexible solution is a *super-solution* [103] which is a solution with an associated tuple (a,b) such that if at most a variables have invalid values in a new problem in the sequence, then at most b other variables will also need to have their values changed to form a new solution, (e.g. a (1,1)-super-solution means that if only one variable loses its value in a solution then only one other variable at most need also have its value changed to produce a new solution).

In this work, we focus on *reactive* methods. Like proactive methods, these can be further split into two categories, *solution reuse* and *reasoning reuse* methods. As their name suggests, both involve reusing information learnt while solving the previous problem in the sequence, to solve the subsequent problem.

Reasoning reuse methods store information regarding *inconsistency* in a prob-

lem, typically in the form of nogoods which can then be used to prune the search space on the subsequent problem in the sequence (e.g. [180],[61]). Note that if the alterations are solely in the form of problem restrictions then any nogood of the preceding problem remains valid.

Solution reuse methods store information regarding *consistency* in a problem, in particular the solution to the previous problem in a sequence is typically reused, either in guiding a depth-first search approach [204], or as an initial assignment to the variables in local search methods and local repair methods (e.g. Local Changes [209]). Local Changes is a complete algorithm designed to find solutions to an altered problem while conserving as much of the original assignment as possible. It works by determining a minimal set of variables that must be reassigned, and undoing old assignments only when they are inconsistent with the new ones.

Obviously, these two reuse methods are not mutually exclusive. Angles-Domínguez and Terashima-Marín proposed two different combinations of these reuse techniques [3]. They tested these methods on randomly generated DCSPs based on the resource allocation problem, which contained binary intensional constraints of the form $<$, $>$, and $\neq$.

However the problems they studied were in the easy region and thus are not applicable to our work. For example, all base problems had 40 variables, domain size 5, and 14 constraints. Thus, at least 12 variables in each base problem were not in the scope of any constraint (and this is for the case where each of the 28 other variables were only in the scope of one constraint). The maximum increase in constraints per perturbation was 7, which still results in an extremely underconstrained problem.

Finally we note that approaches can also be segregated based on their primary objective. Most of the approaches described above are concerned with reducing search effort, with minimizing solution disruption a secondary objective. However, there are a number of methods where this objective order is reversed, i.e. minimizing solution disruption is the primary objective (e.g. [25],[19]).

5.4 Experimental Methods

The majority of our experiments were performed on random binary problems, generated in accordance with Model B [72] as they allow for greater control. Additional experiments were performed on open shop scheduling problems [196]. All perturbed problem sets were generated by Richard Wallace.

For the random binary problems, DCSPs were generated using one of the following types of alteration:

- Addition and deletion of k constraints of the previous CSP in the sequence (for simplicity we will refer to this condition as $k$ad)
- Replacement of k relations of the previous CSP in the sequence (we will refer to this condition as kr)

In both cases the number of constraints in the problem remains the same. Changes were carried out in the first case such that additions and deletions did not overlap. The second method of alteration can be viewed as a special case of the first, where the scope of an added constraint is the same as that of a deleted constraint.

For each set of DCSPs we generated 25 independent base CSPs. In most cases, three CSPs were then generated from each base problem, giving three DCSPs of length 1. This was sufficient to illustrate the effects we are interested in. We also generated DCSPs of length 20, again starting with 25 independently generated base instances, where successive perturbed instances were produced by altering the previous instance in the sequence.

All problems had 50 variables, each with initial domain size of 10, and tightness of 0.369. Most sets had density of 0.184 (giving 225 constraints), where all instances were soluble. Density of 0.19 (giving 233 constraints) was also used for some sets where we wished to assess differences in behavior between soluble and insoluble DCSPs. All problems are in the critical complexity region (with the 0.19 density sets being closer to the peak), while still small enough to allow extensive experimentation be performed in a reasonable amount of time.

Search was performed until either a solution was found or the instance was proven insoluble. The variable ordering heuristics used were $dom/fdeg$, $ff2$, and

$dom/wdeg$, consistency was maintained using the MAC-3 algorithm. In order to avoid vagaries in results due to the value ordering, in most cases each instance was solved 100 times with values chosen randomly, and the results are presented in terms of means over the 100 runs.

We also generated DCSP's from open shop scheduling problems (as previously described in Section 3.4.4), the base problems were the ost-4-100 and ost-4-95 sets. For each base instance, 5 DCSPs of length 1 were generated. Instances in the ost-4-100 set were altered by decreasing the upper bound on the domains of 4 randomly selected variables by 10 units, while instances in the ost-4-95 set were altered by increasing the upper bound on the domains of 6 randomly selected variables by 10 units. The size of the increment/decrement (10 units) was chosen as this was the difference between the upper bounds of the same variables in the ost-4-100 and ost-4-95 sets. All of the ost-4-100 DCSPs are soluble, while all of the ost-4-95 DCSPs are insoluble.

It should be noted that, unlike the random binary perturbed problems, the scheduling perturbed problems should be more difficult than their respective base problems:

- For the soluble set, domains of 4 variables have been decreased which will either reduce the number of solutions to the instance or leave the same number of solutions.

- For the insoluble set, domains of 6 variables have been increased which will either result in an instance with a minimal refutation of the same size as the base, or a larger minimal refutation than that required for the base instance.

5.5 Impact of small changes at the phase transition

We tested the hypothesis, put forward by Verfaillie and Jussien in [208], that small changes to problems at the frontier between consistency and inconsistency have a profound effect on the nature of the instances to solve. To the best of our knowledge, research on this topic has not been published. The majority of the work in this section was performed by Richard Wallace.

We first experimented on the set of soluble random binary DCSPs with parameters $\langle 50, 10, 0.184, 0.631 \rangle$,f where problems were altered by adding and deleting five constraints (condition $5ad$). Table 5.1 presents results for the first five DCSPs (similar results were found for the other twenty DCSPs). We see that small changes can have a large impact on the search effort required to solve an instance. When solving the fifth DCSP with $fdeg$, for example, Per5-1 required nearly double the search effort of the base instance on average, while Per5-3 required less than half the search effort of the base instance.

Table 5.1: Impact of Small Changes on Search Performance: Random Binary Problem

DCSP i	Per$i$-0	Per$i$-1	Per$i$-2	Per$i$-3
			$fdeg$	
1	600	1303	705	1266
2	2136	4160	2407	1569
3	1682	1794	1697	2027
4	318	755	586	1507
5	2804	4996	1425	1270
			$ff2$	
1	670	1280	1412	1004
2	3222	3990	2521	1582
3	924	1385	2385	968
4	713	1129	1027	941
5	3359	4549	2952	1780

Notes. $\langle 50, 10, 0.184, 0.631 \rangle$ problems. Mean search nodes over 100 runs with random value ordering. Instances altered by adding and deleting 5 constraints. Per$i$-0 is base instance for each of three altered instances found on the same row. These are therefore separate DCSPs of length 1.

Furthermore, there were cases where one heuristic performed better on a base instance but the other heuristic performed better on an associated perturbed instance (e.g. $fdeg$ solved the fourth base instance in less than half the nodes it took $ff2$ on average, yet explored nearly twice as many search nodes as $ff2$ on Per4-3). Since these heuristics are associated with different heuristic actions on random binary problems (as discussed in Section 4.5.2 of the previous chapter), this shows that small alterations can affect the relative amenability of a problem to one heuristic action over the other.

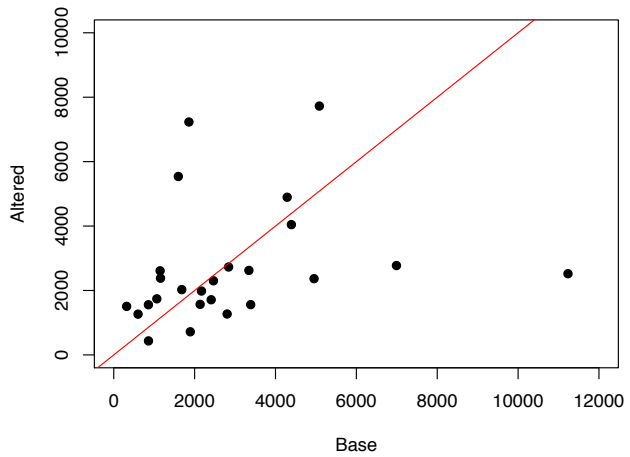


Figure 5.1: Scatter plot of search effort (mean nodes over 100 runs) with $fdeg$ on base versus Per$i$-3 instances with five constraints added and deleted. (Overall correlation in performance between base and altered instances is 0.24.)

The variation in search performance before and after alteration can be seen most clearly in Figure 5.1, which is a scatter plot of search effort on the 25 base problems versus search effort on the perturbed problem set Per$i$-3. For clarity, we include the line $x = y$ where points near or on this line occur for cases where search performance was similar for both a base instance and its associated perturbed instance. The figure shows the extremes that can occur, with search effort on perturbed instances ranging from a factor of 5 less to a factor of 4 more than that for the associated base instance.

We further assessed the impact of small changes on search performance by comparing results for ten sets of 25 independently generated instances with results for ten sets of 25 instances where each set was generated by perturbing instances in a common base problem set. Alterations again took the form of addition and deletion of five constraints. Problems were solved using the heuristic $fdeg$.

Results were compared using the coefficient of variation, which is equal to the ratio of the standard deviation to the mean [191]. The coefficient ranged from 0.61–1.08 over the ten sets of independently generated instances, while the range over the ten sets of perturbed instances was 0.43–0.63. This shows that the variability in search performance found after small problem perturbations is a sizable proportion of that found with independently generated problems.

The scheduling DCSPs generated resulted in even greater variation in search performance, as shown in Table 5.2. We remind the reader that, unlike for the random binary DCSPs, the perturbed scheduling instances were expected to be harder due to the form of alteration used. However, we find orders of magnitude difference in search performance between base and perturbed instances for both soluble and insoluble sets, in *both* directions. In other words, there were cases where perturbed instances were much harder than their base instance as expected, but also cases where the perturbed instances were much easier!

Table 5.2: Impact of Small Changes on Search Performance: Scheduling Problems

DCSP i	Per$i$-0	Per$i$-1	Per$i$-2	Per$i$-3
		$os - taillard - 4 - 95$		
0	51	578	76	557
1	390,845	10,172	1,255,215	1,865,802
2	397	22,140	58,181	66,174
3	1755	22,745	38,060	41,111
4	167,719	51,148	320,280	569,975
		$os - taillard - 4 - 100$		
0	17	18	31	41
1	4,745,944	2,635,513	463,465	6,934,799
2	52,377	81,762	53,060	4,812
3	507,254	233,397	103,630	169,570
4	558,176	4,569,500	9,429	3,481

Notes. Table shows first 5 base instances. Data for 4-100 series is mean search nodes for 50 runs with random value ordering. Variable ordering heuristic was $dom/fdeg$.

In Table 5.3, we present Pearson product-moment correlation coefficients comparing search performance on base instances and on their associated perturbed instances for a number of different conditions for the random binary problem sets. The perturbed instances are arbitrarily grouped based on the number of perturbed instances independently generated from the base, i.e. "Per1" refers to the first perturbed instance generated by altering the base instance, "Per2" the second perturbed instance, etc.

It should be emphasized that this is a somewhat crude measure of problem change, since correlations significantly less than $+1.0$ will only occur if the rel-

ative values of a base and its associated perturbed instance, in comparison with other instances in the corresponding set of base or perturbed instances, are considerably different for a sufficient number of DCSPs. Lower values for a given type of change are of most interest in the table, as these show how large an impact the form of alteration can have.

Table 5.3: Search Performance Correlations for Different Forms of Alterations

	$fdeg$			$ff2$		
condition	Per1	Per2	Per3	Per1	Per2	Per3
1ad	.81	.83	.70	.81	.84	.67
5ad	.49	.83	.24	.34	.54	.31
25ad	.64	.34	.55	.46	.45	.23
1r	.92	.86	.92	.77	.71	.83
5r	.76	.84	.67	.51	.71	.56
25r	.71	.80	.85	.82	.13	.55

Notes. $\langle 50, 10, 0.184, 0.631 \rangle$ problems. Single solution search with repeated runs on each instance. Condition "$k$ad" is k additions and deletions; "kr" is k altered relations. All instances contain 225 constraints.

As expected, correlations tend to decrease as the number of problem changes increases. However, we see that the addition and deletion of just one constraint (or just one relational change) produces correlations as low as 0.7. Furthermore, changes to just one forty-fifth of the constraints (conditions 5ad and 5r) resulted in correlations of 0.5 and lower.

Changes restricted to relations had less impact on search performance for $fdeg$ than for $ff2$. Alterations of this form do not affect the constraint graph, therefore degree only heuristics, such as $fdeg$, will have the same variable ordering before and after perturbation. Here, we still see correlations as low as 0.67 for the $5r$ condition. Furthermore, when lexical value ordering was used (therefore variables *and* values were selected in the same order), correlations of less than 0.5 were found for this condition.

One possible explanation for the variation in search performance could be the impact the alterations have on the number of solutions. In other words, it could be that the perturbed instances required more (resp. less) search effort because they

had less (resp. more) solutions than their associated base instance. However, the results given in Table 5.4 refute this hypothesis.

For example, there is an order of magnitude increase in the number of solutions for Per1-2 over Per1-0, yet search effort increased for both heuristics on Per1-2, by a factor of 2 in the case of $ff2$ (cf. Table 5.1). The overall correlation between average search nodes explored and number of solutions was -0.2 for the 100 instances (25 base and 75 perturbed instances). Although this is in the expected direction, i.e. negatively correlated, the size of the correlation shows that very little of the variation in search performance is attributable to this factor.

Table 5.4: Solution Counts for Sample Instances

DCSP i	Per$i$-0	Per$i$-1	Per$i$-2	Per$i$-3
1	7,846	43,267	109,480	12,065
2	18,573	29,722	47,392	79,147
3	65,735	3,550	8,843	28,427
4	26,505	37,253	17,751	7,282
5	20,156	16,434	12,033	79,384

Notes. Instances taken from Table 5.1

In Table 5.5 we appraised the impact of small problem alterations on two measurements of heuristic performance, *promise* and *fail-firstness* (as introduced in Section 2.3). Although correlations are quite low for all measurements, we see that those for promise were much lower than fail-firstness on average. This implies that changes have a greater impact on the ability of the heuristic to remain on the solution path than on its ability to recover from mistakes.

Table 5.5: Correlations of Heuristic Measurements

	$fdeg$				$ff2$			
measure	Per1	Per2	Per3	x(1-10)	Per1	Per2	Per3	x(1-10)
prom	0.22	0.74	0.34	0.46	0.5	0.43	0.16	0.41
ff-1	0.49	0.67	0.74	0.69	0.52	0.47	0.45	0.58
ff-2	0.75	0.72	0.72	0.72	0.77	0.84	0.46	0.70

Notes. Random binary DCSPs, 5 constraints added and deleted per alteration. "prom" refers to the promise measurement, ff-1 and ff-2 refer to the mean mistake tree size of mistakes rooted at levels 1 and 2 respectively.

Overall, we have shown that even changes as small as one constraint added and deleted can have a profound impact on search performance. Although alterations did have a large effect on the number of solutions, the low correlation with search performance means that this is not the sole cause of search performance variation. Finally, we found that changes had a larger bearing on a heuristic's ability to remain on a solution path than its ability to recover from a mistake.

5.6 New approach - Contention Reuse

In the previous section we showed the marked effects that small changes can have for problems at the phase transition. However, it is possible that there are alternative problem features which are less affected by these forms of alteration. In particular, we investigate the impact of these changes on the major sources of contention. If these remain relatively stable, then contention information from a problem can be reused to guide search in solving altered versions of the problem more efficiently.

5.6.1 Stability of points of contention

We tested the hypothesis that major points of contention remain stable in the face of small problem alteration, by measuring the degree of correlation between weight profiles produced by random probing before and after alteration. A sample of DCSPs were randomly selected. Weight profiles were generated using random probing with 100 restarts and a cutoff of 30 failures per restart. Variables were ranked by their weighted degree after probing, and we compared these rankings for instances before and after change using the top-down correlation coefficient [112].

Figure 5.2 shows results for the correlations between five base instances and each of their three perturbed instances, for the $5r$, $5ad$, and $25ad$ problem sets. We see that changes to relations had little impact on the weight profiles after random probing, with correlations ranging from 0.87 to 0.98. The top ranked variables were slightly less consistent after addition and deletion of five constraints, with correlations in the range 0.73 to 0.98, averaging 0.91 overall. Although there was

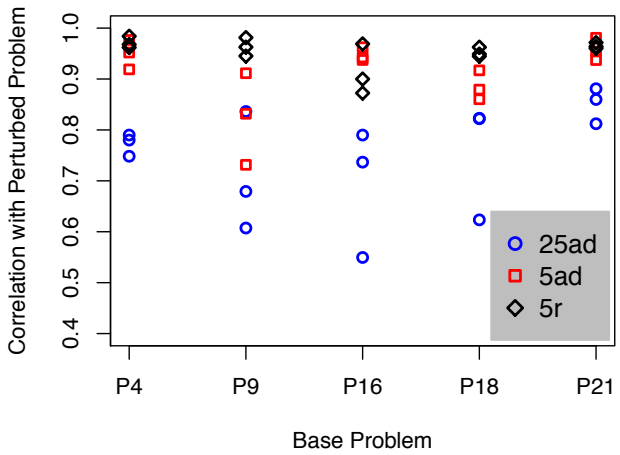


Figure 5.2: Top-down correlation coefficients between 5 sample base instances and each of their 3 perturbed instances for conditions $5r$, $5ad$, and $25ad$.

a more noticeable decrease in correlations for the $25ad$ instances, these still remained relatively high (averaging 0.75).

For comparison, top-down correlations amongst weight profiles for the same instance, but generated with different random seeds for RNDI, ranged between 0.87 and 0.99, while correlations amongst weight profiles for a sample of independently generated instances ranged from -0.11 to 0.27. In light of this, the correlations found on the above DCSPs show that sources of contention remain relatively stable in spite of alterations.

A decrease in correlation is to be expected for the $25ad$ problem set. For the $5ad$ condition only one forty-fifth of the constraints are added/deleted, while for the $25ad$ condition one ninth of the constraints are added/deleted. Most reuse techniques would struggle in the face of such large changes.

Finally, we consider the case where a constraint, which was a source of global contention, is removed. When a constraint is removed, its weight is also removed from the weighted degree of its associated variables. Thus, even in a case where bottlenecks were impacted by change, the weights learnt should not result in performance degradation compared to the case where no weight information is available at the start of search.

5.6.2 DCSP search with weighted degree heuristics

The stability of the major points of contention implies that a contention based heuristic may perform well on the perturbed problems, even when only using information regarding contentious variables of the base problem. In the following experiments we tested this by solving DCSPs using four different forms of weighted degree:

(i) $dom/wdeg$: Normal search with no restarting.

(ii) RNDI: Random probing is used to initialize weights for each problem.

(iii) RNDI-reuse: This approach is identical to RNDI, except no probing is performed on the perturbed problems. Instead, weights on perturbed problems are initialized to the weights after random probing on the base problem.

(iv) $dom/wdeg$-reuse: This approach is similar to (i). Problems are solved with $dom/wdeg$. However here the weight profile from the final search state on the base problem is stored, and constraint weights in the perturbed problems are initialized to these weights.

For the weight-reuse approaches, (iii) and (iv), there are a number of ways one could initialize the weight on the new constraints (when alterations involve the addition and deletion of constraints). One could take the average weight of constraints on the variables in the scope of the new constraint, the average weight over all constraints, or simply assign them an initial weight of 1. The latter approach was used in the following experiments.

There were one hundred runs with random value ordering for the random problem sets (conditions $5ad$ and $5r$). Random probing was performed once on an instance, and these weights were used as initial weights for each of the hundred runs. In the case of RNDI-reuse, the weights from a single phase of probing on a base instance were used as initial weights for all runs on the base and its associated perturbed instances.

The purpose of $dom/wdeg$-reuse is to investigate whether weights can be simply carried over from one instance to the next in a sequence, Thus, the weights were stored after the *first* run of $dom/wdeg$ on the base instance and used as initial weights on all runs on the associated perturbed instances. If the accumulated or average weight over the hundred runs on the base instance was used instead, then

weights would be learnt from a larger sample of the search space, which doesn't evaluate the core issue we are investigating.

We assessed the performance of each approach in terms of average nodes over the perturbed instances (75 for random, 50 for scheduling). Nodes explored during the probing phase are not included, since we are interested in examining quality of performance using the information gained by probing. These amounted to about $3300-4400$ nodes per instance for the entire phase, depending on the problem type. For the same reason, the algorithm only terminates on the run to completion for the probing strategies. The results are given in Table 5.6.

Table 5.6: Search Performance of Weighted Degree Approaches

problems	$dom/wdeg$	$dom/wdeg$ -reuse	RNDI	RNDI -reuse
random-5ad	1617	1875	1170	1216
random-5r	1729	2055	1344	1317
ost-4-95-per	16,745	11,496	4139	5198
ost-4-100-per	11,340	20,283	7972	5999

Notes. Mean search nodes across all altered instances. Random instance parameters are $\langle 50, 10, 0.184, 0.631 \rangle$, 5$ad$ refers to 5 constraints added and deleted, 5r refers to 5 relational changes Scheduling instances altered as described in text.

Reusing weights obtained with random probing on the base instance consistently resulted in an improvement in search performance over the case where no weight information is available at the start of search. Furthermore, there was little fall-off in performance between RNDI-reuse and probing on each individual instance.

This was confirmed through paired comparison t-tests [101] at the 95% confidence interval. There was no statistical significance between the results of RNDI and RNDI-reuse on any of the sets, but there was statistical significance between the performance of each of the probing methods and that of $dom/wdeg$ on all problem sets except the soluble scheduling instances. In the latter case there was no statistical significance between the results of any pair of methods due to variability across instances.

Reusing weights from the final search state of $dom/wdeg$, on the other hand, led to a degradation in performance on the perturbed instances in most cases. Differences between $dom/wdeg$ and $dom/wdeg$-reuse were statistically significant at the 95% confidence interval for the random binary instances. This illustrates the importance of unbiased sampling when gathering information regarding contention.

Interestingly, $dom/wdeg$-reuse performed better than $dom/wdeg$ on the insoluble scheduling perturbed set. A possible explanation is that the insoluble cores may be unaffected by change for some perturbed instances, in which case the weights learnt by $dom/wdeg$ on the base would guide search directly to an insoluble core in the perturbed instances. Thus the search effort to prove insolubility would be greatly reduced.

We have shown that reusing weights from the probing phase on the base instances can result in similarly good performance on the perturbed instances as performing probing on each individual instance. However, this does not necessarily mean that reusing contention information will result in more consistency in search performance between base and perturbed instances.

Table 5.7: Search Effort Correlations for Weighted Degree Approaches

	Random-5ad			Random-5r		
	Per1	Per2	Per3	Per1	Per2	Per3
$dom/wdeg$	0.49	0.82	0.26	0.76	0.77	0.68
$dom/wdeg$-reuse	0.37	0.89	0.48	0.81	0.78	0.76
RNDI	0.58	0.76	0.38	0.70	0.86	0.59
RNDI-reuse	0.59	0.80	0.43	0.77	0.85	0.68

Notes. Search effort correlations between base and altered instances. Random problem parameters are $\langle 50, 10, 0.184, 0.631 \rangle$, 5ad refers to 5 constraints added and deleted, 5r refers to 5 relational changes.

We calculated the Pearson product-moment correlation coefficient between search performance on base instances and on perturbed instances. The results, presented in Table 5.7, for the random problem sets with conditions $5ad$ and $5r$ show that reusing contention information did *not* reduce variation in search performance. Indeed correlations are similar to those for the non-adaptive heuris-

tics shown in Table 5.3. A possible explanation for this is the low correlations for promise shown earlier, which showed that there is considerable variation in a heuristics ability to remain on the solution path.

5.6.3 Insoluble Problems

The stability of the points of contention, and the higher correlations for fail-firstness measurements compared to promise, suggest that there should be less variation in search performance on insoluble DCSPs. To test this we generated a set of insoluble random DCSPs with identical parameters to those already studied, with the exception that the density was increased slightly (to 0.19) so as to reduce the likelihood of generating soluble instances.

The results in Table 5.8 support our hypothesis, where we show search effort correlations between base and associated perturbed instances on the sets of soluble and insoluble random binary DCSPS with five constraints added and deleted. We see consistently high correlations on the insoluble DCSPs, whereas correlations as low as 0.3 are observed for the soluble DCSPs.

Table 5.8: Search Performance Correlations: Soluble versus Insoluble Random Binary DCSPs

	$fdeg$			$ff2$		
condition	Per1	Per2	Per3	Per1	Per2	Per3
Soluble	.49	.83	.24	.34	.54	.31
Insoluble	.80	.85	.84	.90	.91	.93

Notes. Soluble instances: $\langle 50, 10, 0.184, 0.631 \rangle$
Insoluble instances: $\langle 50,10,0.19,0.369 \rangle$
Alterations involved the addition and deletion of 5 constraints.

However, we showed in Section 5.5 that alterations had a large impact on both soluble *and* insoluble scheduling instances, which seems to contradict the hypothesis. This apparent contradiction can be explained by the following observations in regard to the insoluble scheduling DCSPs. Firstly, as previously mentioned, the perturbed scheduling instances are either as difficult or harder than their respective base instances, whereas the random instances have the same basic level of diffi-

culty before and after alteration. In particular, the alterations may remove some insoluble cores for the insoluble scheduling DCSPs. Thus some instances will be considerably more difficult than their base, while others may remain completely unaffected.

Secondly, the initial choices of the heuristic used earlier to solve the insoluble scheduling instances ($dom/fdeg$) were directly impacted by the kind of problem alterations made. It is possible that a heuristic whose initial choices are not impacted by the changes would suffer from less variation. We tested this hypothesis with the heuristic $wdeg$ (due to the symmetry in the constraint graph, $fdeg$ provides little in the way of discrimination on these instances).

The search correlations given in Table 5.9 confirm our hypothesis. We see that, while correlations were high for $dom/fdeg$ in most cases on the insoluble instances, a correlation of 0.03 was found. For $wdeg$, on the other hand, the minimum correlation was 0.98. On the soluble scheduling instances, correlations of 0.51 or lower were found for both heuristics (correlations on these instances for average search effort over 50 runs with the value ordering randomized were the same or slightly lower).

Table 5.9: Search Performance Correlations: Soluble versus Insoluble Scheduling DCSPs

	$dom/fdeg$					$wdeg$				
	Per1	Per2	Per3	Per4	Per5	Per1	Per2	Per3	Per4	Per5
Soluble	0.52	0.98	0.99	1.00	0.27	0.99	0.85	0.93	0.51	0.99
Insoluble	0.03	0.98	0.99	1.00	0.98	1.00	0.98	0.99	1.00	1.00

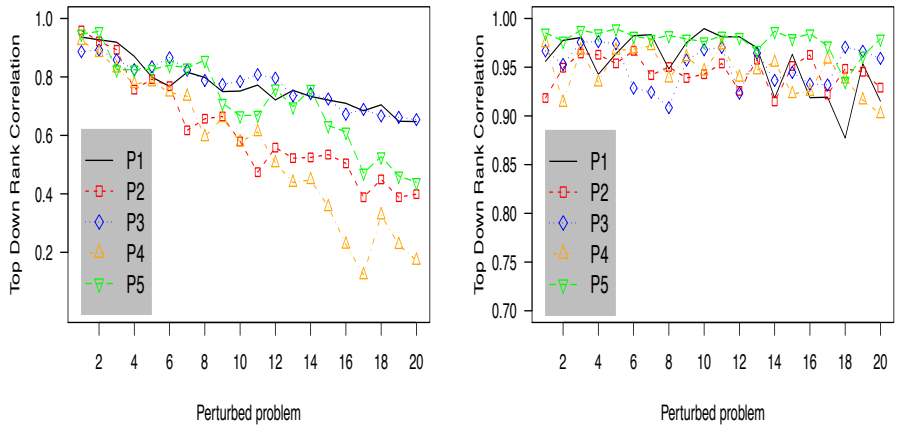
Notes. Instances altered by increasing/decreasing a subset of domains.
Value ordering was alternate between lb and ub, with no randomization.

Finally, upon closer inspection of the results for $dom/fdeg$, we see that only three out of the fifty perturbed instances were solved more efficiently than their base for the insoluble set, compared to over half of the soluble perturbed instances. A similar pattern was observed for $wdeg$. Since the perturbed instances *should* be either as difficult or harder than their associated base instances, this shows that the impact of small changes was greater on the soluble instance set than on the

insoluble instance set.

5.6.4 Probing with successive changes

The main disadvantage of the approach RNDI-reuse, in contrast to other reuse methods (be they reasoning or solution based), is that information is not carried over from the *preceding* instance in a sequence of perturbed instances (i.e. DCSP of length $\gg 1$), but from the *first* instance in the sequence. Indeed, once a perturbed instance has become sufficiently different from that which random probing was performed on, the weights may no longer be beneficial to search and thus random probing would need to be rerun on the current CSP in the sequence.



(a) $5ad$ (b) $5r$

Figure 5.3: Top Down correlations with base after successive changes for (a) $5ad$ condition and (b) $5r$ condition

An important question, therefore, is at what point do the weights from random probing cause a deterioration in performance. We investigated this by generating sets of random binary DCSPs of length 20, where each perturbed instance is the result of alterations to the preceding instance in the sequence. We tested alterations for both the $5ad$ and $5r$ problem sets.

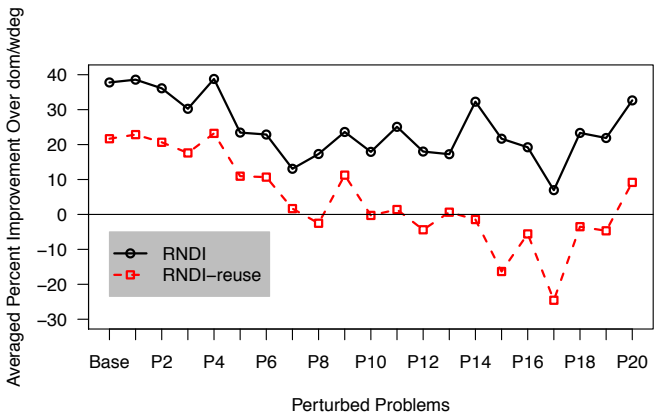


Figure 5.4: Percent improvement of RNDI and RNDI-reuse in comparison with dom/wdeg on sets of successive perturbed instances. Problem set k+1 in sequence derived by adding and deleting 5 constraints in each instance in set k (Pk in figure).

We first examined the stability of the major points of contention across the first 5 DCSPs for both conditions, again using the top-down rank correlation coefficient. We calculated correlations between the weight profile generated by random probing on the base instance and each of the perturbed instances. The weight profiles were all generated with the same initial seed. The results are shown in Figure 5.3.

For the $5ad$ condition, correlations for the first five perturbed instances ranged from 0.75 to 0.96 and continued to decrease thereafter. This is unsurprising given that, after five successive alterations, the instances should be have a similar relation to the base instance as for the $25ad$ condition discussed earlier. On the other hand, when changes were restricted to altering the set of supports in a constraint, correlations were consistently high across all twenty perturbed instances, ranging from 0.94 to 0.99. This shows that major points of contention remain relatively stable for this form of change, providing further support to our earlier findings.

In Figures 5.4 and 5.5, we compare the search performance of RNDI and RNDI-reuse with $dom/wdeg$ for the $5ad$ and $5r$ conditions respectively. Each data point is the average percent improvement, in terms of nodes explored, of RNDI or RNDI-reuse over $dom/wdeg$. The results mirror the weight profile correlations, with search performance of RNDI-reuse deteriorating as the distance

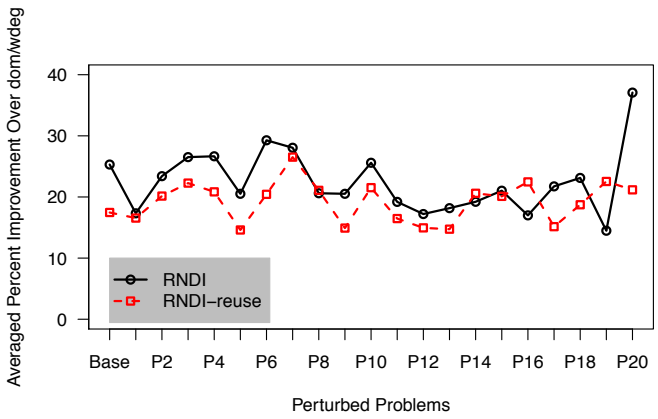


Figure 5.5: Percent improvement of RNDI and RNDI-reuse in comparison with dom/wdeg on sets of successive perturbed instances. Problem set k+1 in sequence derived by altering 5 relations in each instance in set k (Pk in figure).

from the base increases for the 5ad condition, while performance is consistently better across all sets for the 5r condition.

Since alterations of the 5r problems do not add nor remove constraints, the same weight profile is used on all perturbed instances which contributes to the consistency of the performance, i.e. it is a 1-1 mapping of constraint weights (which isn't the case for the 5ad problems as a subset of constraints are new and thus have a weight of 1 and a subset are removed as are their weights). For the 5ad instances we note that weights were no longer beneficial after the seventh perturbed instance in the sequence.

5.6.5 Alterations Affecting the Solubility of a Problem

We next assessed whether the contention reuse method would be able to maintain performance levels in the face of alterations which change a soluble instance to an insoluble instance or vice versa. All random instances had the same parameters as the insoluble instances discussed earlier, $\langle$50,10,0.19,0.369$\rangle$, and changes involved the addition and deletion of five constraints.

The results (Table 5.10) are quite similar to those shown previously, with both random probing methods averaging fewer search nodes than $dom/wdeg$ in finding a solution to the the perturbed instances, while there was no statistical significance

Table 5.10: Search Performance for Random Problems where Alterations Changed the Solubility

	Soluble −> Insoluble		Insoluble −> Soluble	
	Base	Perturbed	Base	Perturbed
dom/wdeg	2,858	6,321	4,985	2,390
RNDI	2,262	5,162	3,954	1,886
RNDI-reuse	2,224	5,227	3,943	1,906

Notes. Instances altered by adding and deleting 5 constraints. In the first set the base instances were soluble and altered instances were not. The opposite was the case for the second set.
Mean search nodes (25 base instances, 75 perturbed instances).

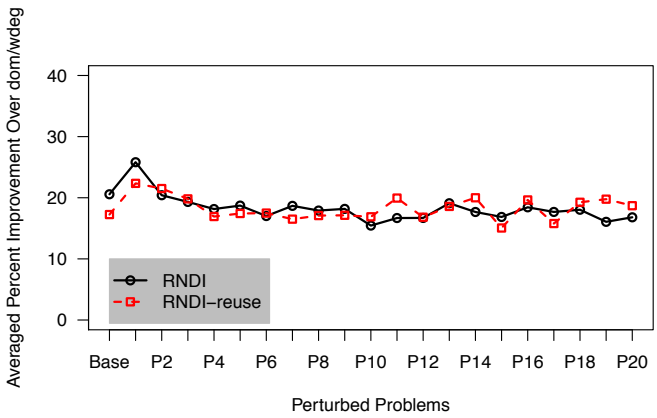


Figure 5.6: Percent improvement of RNDI and RNDI-reuse in comparison with dom/wdeg on sets of successive perturbed instances with 40-60% soluble. Problem set k+1 in sequence derived by altering 5 relations in each instance in set k (Pk in figure).

to the difference in means of RNDI and RNDI-reuse. The issue of solubility had no impact on the gains obtained with RNDI-reuse.

Figure 5.6 shows the percent improvement of the random probing methods over *dom/wdeg* on a set of DCSPs with successive changes. For these DCSPs, all base instances had solutions while perturbed sequences were selected so that 40-60% of the instances had solutions. There were 5 relational changes per perturbation. As we can see, the gains were extremely consistent, with little difference in search performance between RNDI and RNDI-reuse on most sets of perturbed

instances.

DCSPs based on the taillard scheduling instances, with perturbations changing a soluble instance to insoluble and vice versa, were also studied. For these, the domains of four variables in the ost-4-100 instances were decreased by ten, and insoluble instances were selected. Five sets of DCSPs of length one were generated for each base instance. Once again we see that both probing methods yield a large improvement in search performance over $dom/wdeg$ (Table 5.11).

Table 5.11: Search Performance for Scheduling Problems where Alterations Changed the Solubility

	Base	Perturbed
$dom/wdeg$	17,159	49,085
RNDI	6,524	19,439
RNDI-reuse	6,524	19,208

Notes. ost-4-100 instances altered by decreasing 4 domains. Mean search nodes (10 base instances, 50 perturbed instances). Results for random probing methods are averaged over 10 runs per instance.

An alternative method for comparing learning on soluble (resp. insoluble) base instances and testing on insoluble (resp. soluble) perturbed instances, involves the CSP solver competition version of the Taillard open shop scheduling instances [†]. The problem sets are referred to as ost-n-$*$, where n is the number of jobs/machines and * is either 95, 100, or 105. As described previously, instances in the ost-n-105 set were obtained by increasing the domain size of all variables in the corresponding ost-n-100 instance by a small fixed amount, instances in the ost-n-95 set were obtained by decreasing the domain size of all variables in the corresponding ost-n-100 instance by the same fixed amount. The ost-n-95 instances are insoluble, the others are all soluble.

Our experimental setup was to learn weights with random probing on instances in one set, and use the weights learnt as initial weights on the respective instances in the other two sets. We experimented on the sets with n = 4 and 5. For the latter, an overall search limit of one million nodes was used. We present the results in

[†]http://www.cril.univ-artois.fr/ lecoutre/benchmarks/ benchmarks.html

Table 5.12 in terms of average search nodes explored for the ost-4-$*$ problem sets, and in terms of number of instances solved on average for the ost-5-$*$ problem sets.

The results reveal some interesting points. Firstly, in all but one case, RNDI-reuse gave a significant improvement over $dom/wdeg$ be it in terms of average nodes for the ost-4-$*$ problem sets, or number of instances solved for the ost-5-$*$ problem sets. This is quite impressive given that the size of the changes are much greater in some cases than for the scheduling DCSPs studied so far.

Table 5.12: Search Performance on Scheduling Instances

problems	$dom/wdeg$	RNDI	RNDI -L95	RNDI -L100	RNDI -L105
ost-4-95	3748	1263	1263	1496	2916
ost-4-100	17159	5706	8358	5706	19203
ost-4-105	16	23	97	57	23
Mean	6972	2331	3239	2420	7381
ost-5-95	6	9.6	9.6	9.6	9.3
ost-5-100	2	3.7	2.8	3.7	4.2
ost-5-105	9	10	9.8	9.8	10
Total	17	23.3	22.2	23.1	23.5

Notes. RNDI-$L*$ refers to weights learnt by random probing on ost-n-$*$ and tested on all three sets. Results for RNDI methods are averages of ten experiments. For ost-4 instances, the metric is mean search nodes explored. For ost-5 instances, the metric was number of instances solved within the million node limit on search.

Given the performance of RNDI-$L*$ for all other training sets, it was somewhat surprising that it resulted in a deterioration in performance over $dom/wdeg$ when learning from ost-4-105. Further analysis of the results reveals that this is probably because the instances are extremely easy, thus there were no major bottlenecks to be identified.

For this set, instances were solved on over 60 of the random runs (out of 100) on average during the probing phase, and in most cases they were solved mistake-free. In comparison, an instance was solved on 15% (resp. 11%) of the probing

runs for the ost-4-100 (resp. ost-5-105) set, while an instance was solved on less than 1% of the probing runs on average for the ost-5-100 set.

Analysis of the weights generated on the ten instances in the os-4-105 set shows that generally there were less than half the number of failures during probing than when an instance is never solved during the probing phase. Top-down rank correlations as low as -0.01 were found for weight profiles generated with different seeds on the same instance, with an average of 0.28 for the minimum correlation per instance. This raises an important issue for our method and indeed for random probing in general. If the random probing method is unable to identify global bottlenecks in a problem, then weights learnt may result in a deterioration in performance. This may occur for "easy" problems or for problems which do not contain global bottlenecks due to symmetries, etc. (an example of the latter would be the pigeon hole problem).

In order to reduce the likelihood of this occurring, the probing method could be monitored and if an instance is solved on over 40% of the first 30 runs, say, then weights are not carried over. Alternatively, a relatively low correlation between weights learnt on the first 20 runs and the next 20 runs could be used as an indication that the probing method is not identifying global bottlenecks. In either case, one could abandon the probing phase and either use $dom/wdeg$ with uninitialized weights on the next instance in the sequence, or continue to perform monitored probing on each successive instance until either neither condition is met, or all instances have been solved.

5.7 Solution Stability

In the work so far we have concentrated on our primary objective for solving DCSPs, namely minimizing the search effort required to solve the problem. We now concern ourselves with the secondary objective: maximizing the solution stability. The solution stability can be measured in a number of different ways, with the most common being the Hamming distance between solutions to successive CSPs in a DCSP [25], [3]. Let V be the set of variables in common between two successive perturbed problems, P_{i-1}, P_i, and let s_a^{i-1}, s_b^i be the respective solutions found for these problems. Then the Hamming distance between the two solutions

is given by

$$hdist(s_a^{i-1}, s_b^i) = |\{x \in V : s_a^{i-1}(x) \neq s_b^i(x)\}| \tag{5.1}$$

where $s_b^i(x)$ is the value variable x took in s_b^i. Let $Sol(P_i)$ be the set of solutions to P_i. We say that s_b^i is the *nearest solution* to s_a^{i-1} if

$$hdist(s_a^{i-1}, s_b^i) \leq hdist(s_a^{i-1}, s_c^i) \quad \forall s_c^i \in Sol(P_i), c \neq b \tag{5.2}$$

i.e. if it has *minimal* Hamming distance to the solution to the previous problem in the sequence (note there may be more than one solution with minimal hamming distance to the solution found for the preceding problem, so there may be more than one "nearest solution").

We first show how changes impact a traditional solution reuse method, *Local Changes* [209], for problems at the phase transition. We then analyze the maximal solution stability attainable on the problems studied. Finally, we introduce a simple augmentation of our contention reuse method to increase the solution stability of our approach.

5.7.1 Performance of an Algorithm Based on Solution Reuse: Local Changes

In this section we assess the performance of a traditional solution reuse approach on DCSPs in the critical complexity region. The approach we consider is Local Changes [209], which is a complete method that starts with the previous solution assigned to the variables. For each constraint in conflict, a variable in its scope is selected and unassigned. Then each unassigned variable is selected in turn and assigned a new value. The process of unassigning and reassigning variables is repeated until all constraints are satisfied or all values have been tried (i.e. the problem has been proven insoluble). The majority of the work in this section was performed by Richard Wallace.

Our version of *Local Changes* updates the classical description by using MAC; it also makes use of the data structures and style of control used in our basic MAC implementation. All base instances were solved with lexical value ordering. The

solution to the base instance was then passed to the Local Changes algorithm for solving the perturbed instances.

We tried three different value ordering heuristics for Local Changes on the random problems: lexical, min-conflicts [152], and "min-violations". The latter heuristic is that proposed by Verfaillie and Schiex in the original Local Changes paper [209] and involves selecting the value with fewest conflicts with *assigned* neighbors of the current variable. Min-conflicts, on the other hand, chooses the value with fewest conflicts with values in the domains of *unassigned* neighbors. For scheduling problems, we used the value ordering described earlier (alternate between lower and upper bounds of the domain) as min-conflicts is too costly to compute on such large domains.

We found in preliminary experiments with Local Changes that min-conflicts resulted in the best search performance for the random problems studied, and it is this heuristic that was used for Local Changes in the experiments reported in Table 5.13. In the table, we compare the performance of Local Changes with that of depth first search for three different variable ordering heuristics. Results for depth first search are grand means over 100 runs with random value ordering per perturbed instance, although we note that with lexical value ordering, the largest difference with the averages reported in the table was only 400 nodes.

Table 5.13: Solution Reuse Search Performance: Random DCSPs

		Heuristic		
Algorithm	Condition	$fdeg$	$ff2$	$dom/wdeg$
DFS	$5ad$	2,601	3,561	1,617
	$5r$	3,423	3,080	1,729
Local Changes	$5ad$	7,463,591	191,503	377,596
	$5r$	8,931,099	285,796	564,097

Notes. Comparison of depth first search and Local Changes with different variable heuristics. Mean search nodes across all (75) altered instances. Random problem parameters are $\langle 50, 10, 0.184, 0.631 \rangle$. DFS results are averaged over 100 runs with random value ordering.

We see that this form of solution reuse is consistently orders of magnitude worse than solving the perturbed instances from scratch. This is because Local Changes attempts to maintain the previous solution, repeatedly undoing assign-

ments, resulting in an enormous amount of thrashing. The differences are even more impressive given that over 15% of the altered instances in the $5ad$ condition were solved in zero nodes, i.e. the solution to the base instance was still valid after problem alteration. For $fdeg$, there were 13 such cases, while there were 12 for $ff2$.

When changes were restricted to relations, where we have shown that the major points of contention remain relatively stable across successive perturbed instances, there was a deterioration in the performance of Local Changes compared to the $5ad$ condition. Here, there were only 5 (resp. 4) cases where the solution to the base instance remained valid for $fdeg$ (resp. $ff2$). The results imply that solutions at a small distance from the base solution are rarely found.

Table 5.14: Solution Reuse Search Performance: Scheduling DCSPs

	Heuristic	
Algorithm	$dom/fdeg$	$dom/wdeg$
DFS	367,291	11,340
Local Changes	$>$ 2,104,397	$>$ 2,148,399

Notes. ost-4-100 instances, altered by reducing 4 domains. Mean search nodes across all (50) altered instances. 10 million node search limit.

The performance of Local Changes on the soluble scheduling DCSPs was even worse (Table 5.14), even though 70% of the instances were solved by Local Changes in zero nodes for both heuristics. Yet of the fifteen instances where the previous solution was invalid, Local Changes failed to solve ten instances with $dom/fdeg$, and eleven instances with $dom/wdeg$, within the limit of ten million nodes. It is also interesting to note that, unlike depth first search, $dom/wdeg$ was not the best heuristic when combined with Local Changes on either the random DCSPs or the scheduling DCSPs.

5.7.2 Nearest Solution Analysis

The poor search performance of Local Changes implies that small changes on problems at the phase transition have a profound effect on the elements in the

solution set of a problem. In order to develop our understanding of the effects of these minor alterations, we look at the Hamming distance between solutions to the base instances and their nearest solution in the perturbed instances. Experiments were carried out on random DCSPs with 50 variables, so the Hamming distances have a possible range of 0 (meaning the solution to the base was still a solution for the perturbed instance) to 50 (meaning no value in the solution to the base instance was part of any solution to the perturbed instance).

Similar experiments have been performed in Ran et al. [168], where the Hamming distance was used to define the optimal solution stability. However in their experimental setup, they generated a set of random binary CSPs. For each CSP, they then randomly generated a complete assignment which they referred to as the "infringed solution", and used branch and bound search to find the nearest solution of the infringed solution in the *same* problem. We would argue that our method is more realistic in that the only constraints which may not be satisfied by the old solution are those added (a small fraction of the total number of constraints). In the experimental setup of [168], on the other hand, a large number of constraints may be in conflict with the random value assignments.

We first report results of an experiment performed by Emmanuel Hebrard, using the solver Mistral [†]. The experiment was designed to test the impact of small changes on the minimal Hamming distance as instances go from the easy to hard region. Problem parameters were the same as for the soluble random problems studied earlier (i.e. 50 variables, domain size 10, and density 0.184), with the tightness ranging from 0.1 to 0.36 in steps of 0.01. In the easy region 500 base instances were generated for each value of tightness, while in the hard region 100 base instances were generated. For each base instance, one perturbed instance was generated by adding and deleting three constraints. The nearest solution in the perturbed instance was found using limited discrepancy search.

The results are shown in Figure 5.7. We see that solutions remain relatively unaffected by change until instances enter the critical complexity region, at which point there is a sharp rise in the minimum minimal Hamming distance. Indeed, for some instances at the phase transition, there were *no* values in the base solution which were part of any solution to the perturbed instance.

[†] http://4c.ucc.ie/~ehebrard/mistral/doxygen/html/main.html

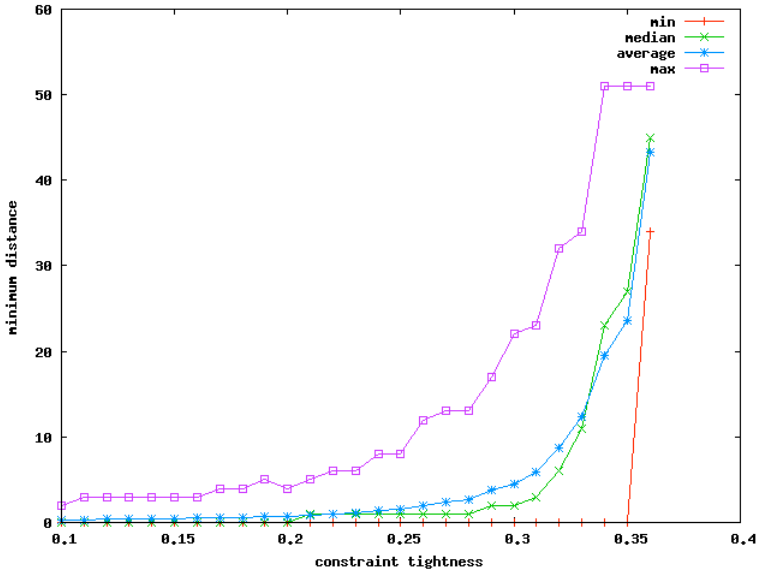


Figure 5.7: Hamming distances for nearest solution of perturbed instances to their base instances for varying tightness.

For our experiments, we focused primarily on the perturbed set of instances involving the addition and deletion of five constraints, with additional experiments reported on instances involving the alteration of five relations. Each of the 25 base instances was solved 100 times with random value ordering. For each solution to a base instance, branch and bound search was then used to find the nearest solution in the three associated perturbed instances. The variable ordering heuristic used for the base and the perturbed instances was $dom/wdeg$.

We present the results in terms of average, median, minimum, and maximum minimal Hamming distance over the 100 runs for each perturbed instance (Figure 5.8). For clarity, the instances have been ordered in terms of increasing average minimal Hamming distance. A similar set of results was obtained on instances with the $5r$ condition.

The average minimal Hamming distance per altered instance, over 100 runs, ranged from 0.4 to 42.7. The overall average minimal Hamming distance was 21.2. Clearly, a local repair method such as Local Changes would suffer greatly in the face of such changes to the individual solution set. Interestingly, for 57 of the 75 perturbed instances there was at least one solution to the base that was still

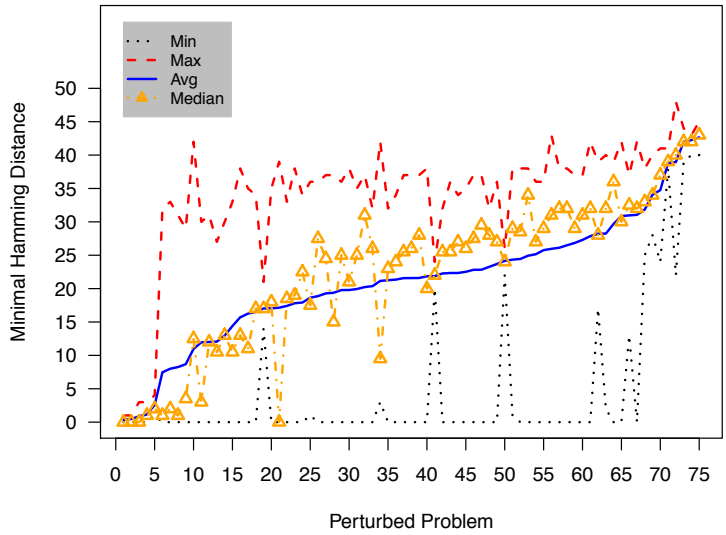


Figure 5.8: Minimal Hamming distance statistics for perturbed instances in the $5ad$ condition.

valid after problem alteration. Overall, however, only 12% of the 7500 runs had a minimal Hamming distance of 0.

These figures were lower for the $5r$ condition, where only 45 out of the 75 perturbed instances had at least one minimal Hamming distance of 0 and a solution to the base remained valid on only 6% of the 7500 runs (although here the overall average minimal Hamming distance was just slightly higher at 22.2 than for the $5ad$ DCSPs). These results explicate the difference in search performance of Local Changes for these two conditions (cf. Table 5.13).

The extremes between different perturbed problems, in terms of minimal Hamming distances, can be seen more clearly in Figure 5.9, where we plot the minimal Hamming distance for each run of 3 sample instances. For clarity, the runs are ordered for each instance in terms of increasing Hamming distance. For Per11-2, the minimal Hamming distance was always low which shows that the problem alterations had little impact on individual solutions. The opposite was the case for instance Per6-2, which had a consistently high minimal Hamming distance ($>$

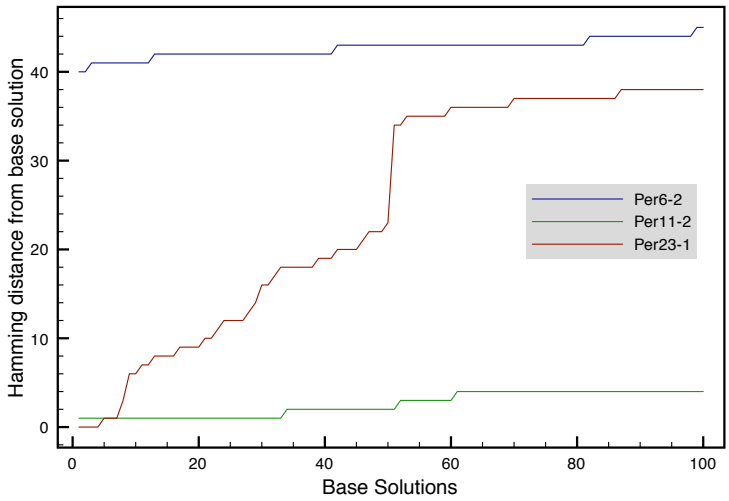


Figure 5.9: Hamming distances for nearest solution of 3 perturbed instances from 100 different solutions to their base instance.

40 for all base solutions). In between such extremes were cases such as Per23-1, where the minimal Hamming distance had a large range, with base solutions still valid in the best case while in the worst case only 25% of the solution could be maintained.

These results illustrate why Local Changes performed poorly on these problems. Starting from a complete assignment of the previous solution and working backwards is extremely inefficient if there is a large distance between the previous solution and the nearest solution in the current problem.

5.7.3 Solution guidance for failure reuse

The contention reuse method we proposed can be enhanced by adding a solution reuse component in the form of a simple value ordering heuristic. The heuristic chooses the value that the variable took in the solution to the previous problem in the DCSP sequence (similar heuristics have been previously proposed, e.g. [204]).

However, as noted in [208], depth-first tree search methods do not explore the space of possible assignments in increasing distance to the heuristic assignment. To improve the solution stability, we added a tie-breaker for the case where the

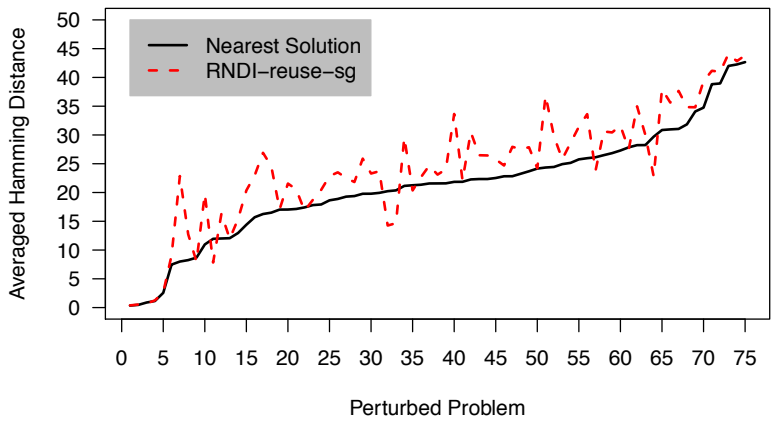


Figure 5.10: Hamming distance comparison of adaptive methods with solution guidance compared to minimum obtainable.

value from the previous solution is not in the current domain of the variable. For each value, v, in the domain of the variable, we count the number of unassigned neighbors of the variable for which v supports the *solution* value of the neighbor, i.e. the value the neighbor took in the previous solution. The heuristic chooses the value that supports most solution values over the unassigned neighbors. (Since the algorithm incorporates MAC, each value in the current domain supports the same number of solution values in the *assigned* neighbors of the variable.)

In Figure 5.10, we compare the average Hamming distance of solutions found by our solution guided contention reuse method (RNDI-reuse-sg) with the average Hamming distances of the nearest solution. Instances were ordered in terms of increasing average Hamming distance of the nearest solution (it should be noted that the solutions to the base instances were different for the two methods, hence there are a couple of instances for which RNDI-reuse-sg had a lower average than that of the "nearest solution"). We see that the addition of the value ordering heuristic for our method results in solutions which are close to optimal in most cases. The overall average Hamming distance was 24.3 for RNDI-reuse-sg, compared to an average of 21.2 for the minimal Hamming distance.

The search performance of probing with solution guidance was closely related to the minimal Hamming distance. We plot in Figure 5.11 the search nodes explored in increasing order for 100 runs on the same three instances studied in

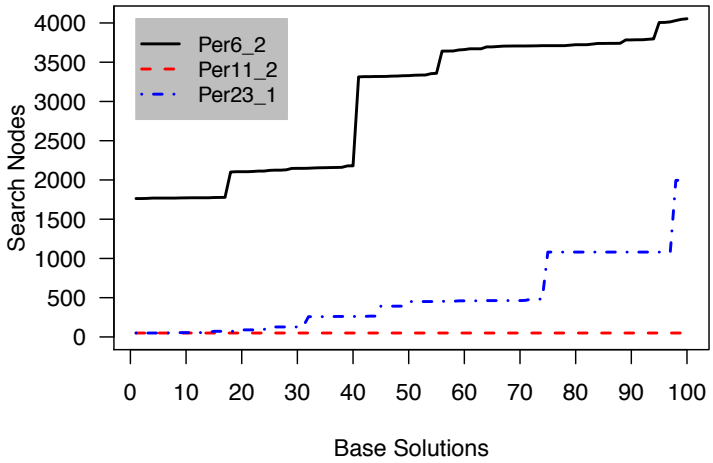


Figure 5.11: Search Performance of RNDI-reuse-sg on instances studied in Figure 5.9.

Figure 5.9. Correlations between weight profiles generated on these perturbed instances and on their associated base instances were 0.92 or higher, which shows that the major points of contention were unaffected by extreme changes to the solution set. It is also interesting to note that for Per11-2, the instance was always solved without encountering a single mistake, even though the base solution wasn't always valid. This can be attributed to the effects of propagating the solution values at the top of the search tree.

To allow for a more direct comparison of solution stability with Local Changes, we ran RNDI-reuse with solution guidance on the perturbed instances, but with solutions to the base instances found using lexical value ordering and either $fdeg$ or $ff2$. Thus the same base solutions were used on the perturbed instances by both RNDI-reuse-sg and Local Changes. We also generated the *minimal* Hamming distances for each base solution of the two heuristics. The results are given in Table 5.15.

We see that, in terms of solution stability, min-violations is the best value ordering heuristic for Local Changes. This is unsurprising as min-conflicts is used

solely for search effort reduction. Min-violations attempts to maintain the solution with respect to the original instantiated variables, if most of these assignments are undone during search then the solution stability will be greatly reduced. On average, less than 50% of the original solution was maintained by Local Changes with min-violations. Somewhat surprisingly, our simple value ordering heuristic resulted in better solution stability than any of the Local Changes approaches.

Table 5.15: Solution Stability Comparison

	Algorithm				
Heuristic	LC -$lexval$	LC -$minconflicts$	LC -$minviolations$	RNDI-reuse -sg	$Minimal$ $Hdist$
$fdeg$	30.6	30.4	25.8	22.7	18.8
$ff2$	31.3	31.1	28.6	24.5	20.7

Notes. $5ad$ problem set. Base instances solved by either $fdeg$ or $ff2$ with lexical value ordering. Average Hamming distance between solution found on base and perturbed. LC is Local Changes.

Finally, we note that the benefit of solution guidance to our method wasn't restricted to an increase in solution stability. Search performance also improved over RNDI-reuse. A reduction in search effort of nearly 50% was found on average for the $5ad$ perturbed instances, for both RNDI-reuse and $dom/wdeg$ when combined with solution guidance (average nodes of 701 and 954 respectively). This is because little search effort is required to solve perturbed problems where the solution to the base is still valid or a solution quite close to the base solution exists.

5.8 Chapter Summary

The Dynamic Constraint Satisfaction Problem is an important subfield in the area of constraint satisfaction and optimization. However, there has been little work done on understanding the impact of changes, especially on problems in the critical complexity region. In this chapter we have shown that even small alterations can have a profound impact on a number of problem features, such as search per-

formance, the size of the solution set, the individual elements in the solution set, and promise and fail-firstness measurements for heuristics.

In the course of this work we identified one feature which does not appear to be as sensitive to change, namely the major points of contention. Based on these findings we have proposed a new method for solving DCSPs, in particular DCSPs at or near the phase transition. We have identified both advantages (equally adept at solving insoluble DCSPs as soluble DCSPs) and disadvantages (contention information cannot be carried over from one problem to the next in the sequence) of our method. However, we have still shown that it is orders of magnitude more efficient than the local repair method of Verfaillie and Schiex (Local Changes).

Analysis of the optimal solution stability clearly showed why solution repair methods will struggle on the type of DCSPs studied, where on average nearly 50% of the solution could not be maintained. We further supplemented our method with a simple value ordering heuristic to improve the solution stability. The results showed that not only was it more efficient in solving problems than Local Changes, it also resulted in better solution stability.

Chapter 6

Alternative restarting strategies

6.1 Introduction

Fixed cutoff restarting strategies have proven very adept at solving many CSPs, where the runtime distribution is used to identify the optimal cutoff (Gomes et al. [82]). However, often the runtime distribution is not known in advance of solving the problem, and a trial-and-error approach is required to identify the optimal fixed cutoff. A poor choice of cutoff can have disastrous consequences: too low and the problem will never be solved; too high and the algorithm will spend a large amount of time exploring unpromising parts of the search space.

Universal restarting strategies were proposed in order to approximate the optimal cutoff. They retain the benefits of restarting while guaranteeing completeness. These strategies have cutoffs that increase periodically, eventually tending to infinity. The most popular universal restarting strategies are the Luby strategy (Luby et al. [140]) and the Walsh (geometric) strategy [221], and variations thereof (Wu [231]).

The traditional method of ensuring diversification of search across restarts is to introduce an element of randomness into the search algorithm, typically through randomization of the variable and/or value ordering heuristics (Harvey [99], Gomes et al. [82]). More recently, adaptive heuristics, such as the weighted-degree heuristic or the impact-based search strategy of Refalo [169], have been successfully combined with restarting strategies where updates to the constraint

weights / value impacts are generally sufficient for diversification across restarts.

Impact-based search (IBS) uses alternative information from previous search states to guide subsequent search. It stores information about the *impact* of a value, i.e. the proportional reduction in the size of the search space after assigning, and propagating, a value. These impacts are then used to guide both variable and value selection.

The aim of this chapter is to identify the best general purpose strategy for solving CSPs. To this end we firstly provide a thorough evaluation of the different restarting strategies and adaptive heuristics on a large testbed of instances of varying types. We then investigate the behavior of the approaches on a subset of problems, to ascertain why the different approaches result in good/bad performance. In particular, we identify advantages and disadvantages of constraint weighting for different models of the open shop scheduling problem.

The next section provides a description of the solver used and the different search strategies tested. The section thereafter describes the results for the weighted degree heuristic with the different restarting strategies and a comparison of weighted degree with impact-based search.

Sections 6.4 and 6.5 provide more detailed analysis of the behavior of the different approaches on two problem types, open shop scheduling and radio link frequency assignment. The final section presents the conclusions of the work.

6.2 Solver Description and Techniques Tested

In order to test the different methods on a wide testbed of problems, we implemented the probing strategies in the CSP solver, *Mistral*. This solver is extremely robust at solving CSPs (Lecoutre et al. [137]), it was the top ranked solver in three of the five categories of the 2009 CSP Solver Competition and was one of the top three ranked solvers for the other two categories[†].

Mistral is a C++ constraint programming library developed by Emmanuel Hebrard. It contains a number of automated modeling steps such as: the choice of representation for a variable (bitset, list, Boolean, range); the constraint propagator

[†] `http://www.cril.univ-artois.fr/CPAI09/results/ranking.php?idev=30`

for each constraint (general algorithms for enforcing arc consistency and bounds consistency, and dedicated constraint propagators for handling global constraints). We tested the methods using the black-box solving method of Mistral which has been submitted to the CSP solver competitions.

The depth first search method used by Mistral involves binary branching, which has been shown to be much more powerful than d-way branching for backtracking search (Hwang and Mitchell [111]). An important aspect of Mistral, given the impact two different but logically equivalent models will have on search with the weighted degree heuristic, is the method used to handle constraints which are specified as predicate trees. Figure 6.1 shows a sample predicate tree for the following constraint:

$$3X + 4Y + 8Z \leq 18 \tag{6.1}$$

The default method for handling constraints of this type in Mistral is to represent each internal node of the predicate tree as a variable. For the sample constraint given, the model used by Mistral would involve the following variables and constraints:

$$3X + 4Y = V_1 \tag{6.2}$$

$$8Z + V_1 = V_2 \tag{6.3}$$

$$V_2 \leq 18 \tag{6.4}$$

where V_1 and V_2 are auxiliary variables, and the domain of V_1 (similarly for V_2) is thus

$$(3 * min(D(X)) + 4 * min(D(Y)), \ldots, 3 * max(D(X)) + 4 * max(D(Y)))$$

Furthermore, the unary constraint (6.4) is simply enforced at the root node, by reducing the maximum domain value of V_2 to 18. These auxiliary variables are not included in the decision variables of the problem, however they will impact the spread of the weight in the problem.

The strategies we tested in Mistral involved combinations of the following components:

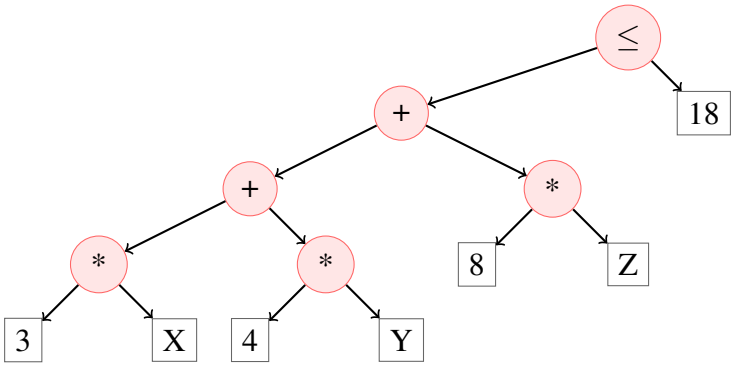


Figure 6.1: Sample predicate tree.

- Variable ordering heuristics: $dom/fdeg$, $dom/wdeg$, and impact.
- Initialization for adaptive heuristics: Random probing for $dom/wdeg$, singleton arc consistency for impact.
- Restarting: All heuristics were tested with and without restarting. Geometric restarting was used with randomized $dom/fdeg$ and with impact, while three restarting strategies were tried in combination with $dom/wdeg$: WTDI, Luby [140] and Geometric [221].

6.2.1 Restarting Parameters for weighted degree approaches

Probing Strategies

The probing parameters used for RNDI were again 100 restarts with a cutoff of 30 failures per restart, as we have shown that these are generally sufficient to identify problem bottlenecks. A ten minute cutoff for the probing phase was included as a safety measure, i.e. if problems are extremely expensive to search on then spending the majority of the time performing random probing could prove detrimental.

The purpose of WTDI is to combine a rapid restart phase with an intensive search phase. We tried various cutoff values for WTDI while keeping the number of restarts fixed at 100. The cutoff values tested were $C \in \{100, 500, 1000, 2000, 5000\}$.

Universal Restarting Strategies

Following on from [232], we include a scale parameter s in both universal strategies tested. The Luby sequence can be described as a strategy where "all run lengths are powers of 2, and each time a pair of runs of a given length are completed, a run of twice the length is immediately executed" [140].

More formally, the Luby sequence is a sequence L=$(t_1, t_2, t_3, \dots)$ where:

$$t_i = \begin{cases} s * 2^{k-1} & if\ i = 2^k - 1 \\ t_{i-2^{k-1}+1} & if\ 2^{k-1} \leq i < 2^k - 1 \end{cases}$$

The first terms of this sequence for $s = 100$ are:

$$100, 100, 200, 100, 100, 200, 400, 100, 100, 200, 100, 100, 200, 400, 800, \dots$$

We tested a large range of values for the scale parameter $s \in$\{10, 100, 500, 1000, 2000, 5000, 10000\}. Larger cutoffs than those for WTDI were tested as the Luby strategy may take longer to scale up for problems suited to intensive search.

The geometric strategy is a sequence of the form s, sr, sr^2, sr^3, where s is a scale parameter and r is the multiplicative factor. This is one of the most popular restarting strategies and indeed has been combined with the weighted degree heuristic previously (Lecoutre et al. [135], Balafoutis and Stergiou [12]). The values we tested for the parameters were r=\{1.1, 1.3, 1.5, 2\} and s=\{10,100,1000\}, in a "fully crossed" design.

The geometric strategy is less influenced by the scale factor as Luby (within reason); here, the parameter r is the more important factor. Too low a value for r and a large enough cutoff may not occur in the sequence within a reasonable amount of time; too high a value and the optimal cutoff may be bypassed. We will refer to the strategy combining the weighted degree heuristic with Luby restarting as *Lubywtd* and that with geometric restarting as *Geowtd*.

6.2.2 Randomized non-adaptive heuristic

There are a number of options for randomizing the variable heuristic, the two most common methods are "forced ties" (e.g. Wu and van Beek [232]) and "heuristic

equivalence" (Gomes et al. [82]). We chose the former, where ties were forced by considering the top ten choices of $dom/fdeg$ as equivalent, and randomly selecting from amongst this subset. The randomized heuristic was combined with geometric restarting, with a scale factor of 10 and a multiplicative factor of 1.5. We will refer to this method as *Georand*.

6.2.3 Impact-Based Search

Impact-based search (IBS) was tested with and without initialization by singleton arc consistency (SAC), and with and without restarting. For the former, when impacts were not initialized by SAC some initial value must be chosen for the impacts. Provided all values have the same initial impact value, > 0, the heuristic will behave identically to $min\text{-}domain$ up until at least one backtrack has occurred (this is similar to initializing all constraint weights to one when using the weighted degree heuristic, resulting in the heuristic behaving identically to $max\text{-}degree$).

Different initial values for the impacts, however, will have a profound effect on the subsequent search. Consider the case where search is at a choice point after several backtracks and there are a number of unassigned variables with equal minimum domain size, including at least one previously tried variable, y, which was assigned the value a prior to backtracking.

If impacts are initialized to a very low value, e.g. 0.0001, then the impact of $y = a$ is likely to have increased. Therefore the variable y is likely to be selected ahead of other variables with the same domain size which haven't been tried. For this variable, since the impact of $y = a$ has increased, a previously untried value will be selected.

On the other hand, if impacts are initialized to a very high value, e.g. 0.9999, then the impact of $y = a$ is likely to have decreased. Thus a previously untried variable of equal domain size will be selected ahead of y. If the variable y is eventually selected again, it will be assigned a previously tried value as these are likely to have an impact lower than 0.9999.

Overall, this means that a low initial value for the impacts will lead to diversification of the value selection, while a high initial value will diversify the variable

selection. Given these considerations, we tested both small and large initial values for the impacts. When IBS was combined with geometric restarting, the same parameters as for randomized $dom/fdeg$ were used, i.e. a scale factor of 10 and a multiplicative factor of 1.5.

6.3 Full Empirical Study

6.3.1 Problem Sets

All problems are taken from the benchmarks website of the CSP Solver Competition([202]). There were 1800 instances in our test set, separated into 30 problem types. In some cases we grouped instances based on an overall type, such as random extensional or scheduling problems, and randomly selected a sample of 100 instances. For any problem type, the largest number of instances was 100 (so if there were more than 100 instances of the type a sample was randomly selected). Due to the number of problem sets tested, we do not describe these here although reference will be made during the discussion to certain problem types. A full description of each problem set is given in Appendix A; alternatively problem descriptions can be found at the benchmarks website[†].

6.3.2 Experimental Setup

All algorithms had a time limit of 1200 seconds per instance. The results for the randomized algorithms (RNDI and randomized $dom/fdeg$) are the averages of ten runs. No ordering heuristics were used for the consistency methods, and values were chosen lexically for all approaches except IBS which used the impact value ordering heuristic. Experiments were again performed on an Intel Xeon 2.66GHz machine with 12GB of RAM on Fedora 9.

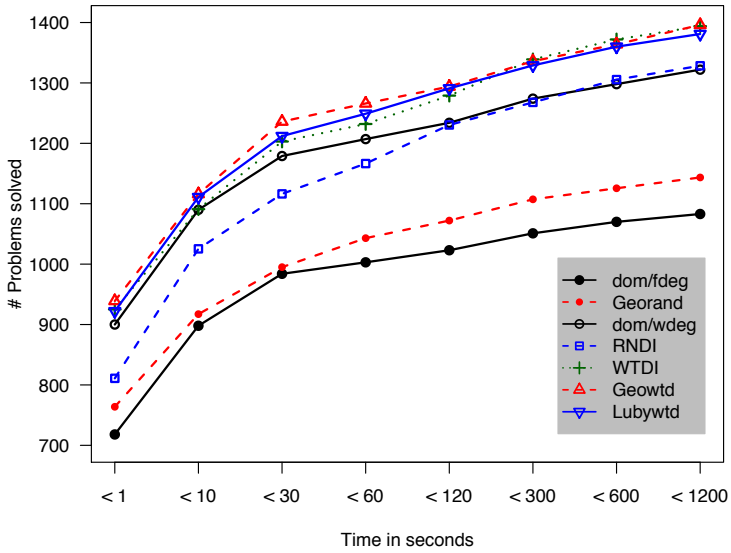


Figure 6.2: Results summary.

6.3.3 Experimental Results for Weighted Degree Approaches

We first present our results in terms of number of instances solved within increasing time limits (Figure 6.2 and Table 6.1) by each method with its best parameter setting (except results for IBS approaches which are given in Section 6.3.4). The results for the methods that combined $dom/wdeg$ directly with restarting, are given for the best parameter settings for each (2000C for Wtdi, 5000C for Lubywtd, and 1000C with a multiplicative factor of 1.5 for Geowtd).

The results for Georand illustrate two points. Firstly, as expected, restarting combined with an element of randomization resulted in improved performance. More importantly, however, these results show that the benefit of the weighted degree heuristic isn't restricted to merely increasing the diversification of the base heuristic ($dom/fdeg$). The weights learnt are clearly meaningful. Indeed, comparing Georand with $dom/wdeg\text{-}nores$, we observe that constraint weighting is more effective at removing thrashing than randomizing $dom/fdeg$ with restarting.

[†]http://www.cril.univ-artois.fr/ lecoutre/research/benchmarks/benchmarks.html

Overall, we found that the weighted degree heuristic combined with a geometric restarting strategy was best. However it is interesting to note that WTDI would have solved the most instances if a cutoff of five or ten minutes was used. Indeed, directly combining the weighted degree heuristic with any of the restarting strategies was consistently better than the two cases where the weighted degree heuristic was only used on a run to completion ($dom/wdeg$-$nores$ and RNDI). Surprisingly, on average RNDI only solved 6.4 more instances than $dom/wdeg$-$nores$. Furthermore, the cost of probing meant that RNDI was generally slower on the "easy" instances. Indeed, if a cutoff of five minutes, or less, had been used then $dom/wdeg$-$nores$ would have solved more instances.

Table 6.1: Full Results By Runtime

Time	$dom/fdeg$	Georand	$dom/wdeg$ -$nores$	RNDI 100R 30WC	WTDI 100R 2000WC	Geowtd s=1000 r=1.5	Lubywtd s=5000
$<$ 1s	718	763.8	900	810.9	925	**939**	922
$<$ 10s	898	917.2	1090	1025.3	1091	**1116**	1111
$<$ 30s	984	995	1179	1116.3	1203	**1236**	1212
$<$ 60s	1003	1043	1207	1166.5	1232	**1266**	1249
$<$ 120s	1023	1072.1	1234	1230.4	1279	**1294**	1291
$<$ 300s	1051	1107.3	1274	1267.8	**1339**	1336	1329
$<$ 600s	1070	1125.6	1298	1305.5	**1372**	1364	1360
$<$ 1200s	1083	1143.5	1322	1328.4	1394	**1396**	1381

In Figure 6.3 we show the number of problem sets for which each method performed best (in terms of number of instances solved), and uniquely best (i.e. solved more instances of the type than any of the other methods). We find that WTDI matched the best performance on eighteen of the thirty problem sets, two more than Geowtd, and was uniquely best on two more problem sets than Geowtd. Interestingly, RNDI was also uniquely best on four problem sets.

The methods Geowtd and WTDI both attempt a tradeoff between diversification and intensification, albeit in very different ways, in order to have more robust performance across problems. WTDI starts with total diversification (fixed short cutoff, many runs), and follows with total intensification (unbounded search). Ge-

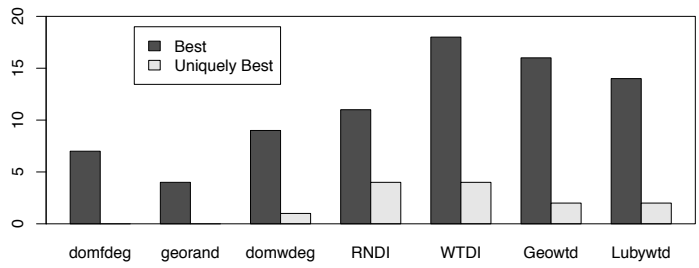


Figure 6.3: Number of problem sets (out of 30) where method was best / uniquely best.

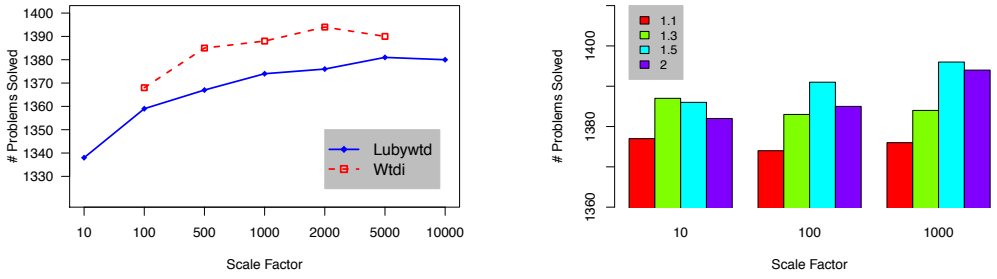
owtd, on the other hand, gradually moves from diversification oriented to intensification oriented search.

We found that both achieved the aim of robust performance to a certain degree. We compared these two methods, in terms of number of instances solved per problem set, with the more intensification-oriented strategy ($dom/wdeg$-$nores$) and the more diversification-oriented strategy (Lubywtd). Over the thirty problem sets, we found that $dom/wdeg$-$nores$ solved at most one more instance in a problem set than WTDI, and a maximum of two more instances than Geowtd, whereas $dom/wdeg$-$nores$ solved a maximum of ten more instances than Lubywtd.

The results were slightly worse for robust performance on problem sets suited to diversification. Lubywtd solved, per problem set, a maximum of nine more instances than WTDI and five more instances than Geowtd. However, we note that both WTDI and Geowtd also solved a maximum of nine more instances than Lubywtd over the thirty problem sets.

In Figure 6.4, we show the number of instances solved by the three weighted restarting methods for their different parameter settings. Observe that, outside of a cutoff of 100 failures, WTDI always solved more instances than the best cutoff (5000) for Lubywtd. For Geowtd, the smallest multiplicative factor (1.1) was consistently worst across the different scale factors. There is greater variation in the results for the other multiplicative factors, although the multiplicative factor of 1.5 was generally best. We also observe that as the scale factor increased, so

did the number of instances solved with multiplicative factors of 1.5 and 2.



(a) Scale Factor for Wtdi, Lubywtd

(b) Scale/Multiplicative Factor for Geowtd

Figure 6.4: Number of instances solved with different parameter settings.

We compare the runtime performance of Geowtd with Wtdi in Figure 6.5, where we provide the boxplots of runtimes on the commonly solved instances, along with average and standard deviation of the runtimes. For clarity, we removed "easy" instances that were solved in less than one second by both methods, leaving 460 instances. The boxplot is a graphical depiction of the five point summary of data involving: the minimum, 1st (lower) quartile, median, 3rd (upper) quartile, and maximum. The "box" goes from the lower quartile to the upper quartile, with the median illustrated by a dark line across the middle of the box.

The inner quartile range (IQR) is used to calculate the "whiskers" and outliers as follows. The IQR is multiplied by 1.5 to give the *step*. Any data points above (3rd quartile + step) are considered to be outliers. A whisker is drawn at the largest data point that is not an outlier. Similarly, any data points below (1st quartile - step) are considered to be outliers and the lowest data point that is not an outlier is the lower whisker in the plot. The results show that although both methods had comparable performance on average, Geowtd was generally quicker than WTDI in solving these instances.

The reason for RNDI's relatively poor overall performance, as can be seen in Table 6.2, is mainly due to one problem type, the balanced incomplete block design problem (*BIBD*). RNDI solved 20 fewer instances than $dom/wdeg\text{-}nores$ out of 83 instances in this problem set, and approximately 40 instances fewer than

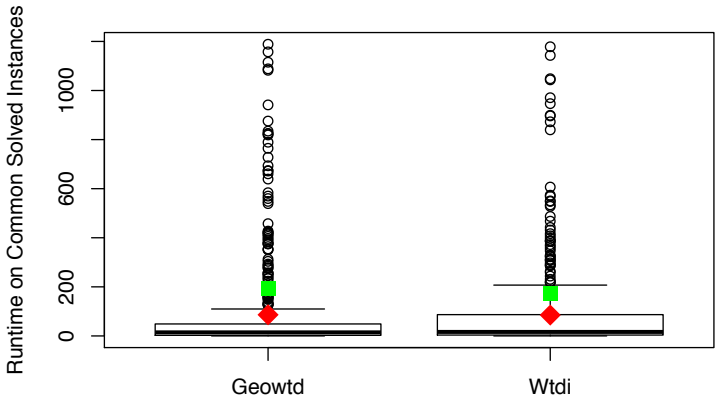


Figure 6.5: Boxplots of runtime performance for WTDI and Geowtd on 460 commonly solved, non-trivial, instances. Average (red diamond) and standard deviation (green square) also depicted.

the best algorithm (the Luby restarting strategy).

Given a tuple of natural numbers (v, b, r, k, λ), a BIBD involves arranging v distinct objects into b blocks. Each block contains exactly k distinct objects, each object occurs in exactly r different blocks, and every two distinct objects occur together in exactly λ blocks. Since both the objects and blocks are interchangeable, the matrix X has total row and total column symmetry (Frisch et al. [67]). These symmetries mean that there are no global bottlenecks for RNDI to find.

We generated weight profiles for RNDI, with two different seeds, and WTDI on a sample BIBD that was unsolved during the probing phase by either method. The same probing parameters were used for WTDI as for RNDI (cutoff of 30 failures with 100 restarts). The sample instance involves 3,264 (Boolean) variables, with 3,040 constraints. 2,880 constraints have arity 3 and take the form ($x_1 * x_2 = x_3$), while the other 160 constraints involve the global constraint *weighted sum*, of which 24 have arity 16 and 136 have arity 24.

The breakdown of the variables in this instance, in terms of degree, is 2,880 variables of degree 2 and 384 variables of degree 17. We observe that the heuristic

Table 6.2: Full Results By Problem Type

Problem Type	#Inst	$dom/fdeg$	Geo-rand	$dom/wdeg$	RNDI	WTDI	Geowtd	Luby-wtd
All Intervals	25	9	13.2	19	**23**	22	22	21
All Squares	74	33	33.7	40	40.1	**41**	**41**	39
Bdd	70	70	70	70	70	70	70	70
BIBD	83	22	62.4	55	34.2	69	71	**74**
Bmc	24	**24**	23.7	**24**	**24**	**24**	**24**	**24**
Boolean (S)	100	81	77	**92**	91	91	90	87
Chess Color	20	11	**15.2**	13	15	15	15	15
Coloring (S)	100	87	86.2	89	88.9	**91**	90	90
Costas Array	11	9	9	9	9.9	**10**	9	**10**
Crosswords (S)	100	69	67.1	92	93.7	93	**95**	94
Driver Log	7	7	7	7	7	7	7	7
Fapp (S)	100	16	9.4	48	48.9	**57**	**57**	52
Fischer (S)	100	38	37	63	67.1	**68**	65	61
Golomb	28	21	22.9	23	**23.4**	22	22	22
Langford	75	53	**54**	53	**54**	**54**	**54**	53
Multi-Knapsack	6	**6**	**6**	**6**	**6**	**6**	**6**	**6**
Patat	46	4	4	5	7	17	**29**	26
Primes (S)	50	39	38.8	40	40.3	**44**	41	41
PseudoBool (S)	100	51	56.3	64	**69.3**	67	66	68
Pseudo-GLB (S)	100	25	24.2	37	35.4	**40**	35	**40**
Qcp/Qwh (S)	100	93	93.7	95	95.3	**96**	95	94
Radar	100	89	96.3	**100**	**100**	**100**	**100**	**100**
Ramsey	16	5	8.3	5	8.2	9	**10**	**10**
Random (S)	100	81	71.4	85	**86.1**	84	84	75
RLFAP	71	43	47.4	68	69.9	**70**	**70**	69
Scheduling (S)	100	56	63.1	73	73.5	**74**	**74**	**74**
Schurr's Lemma	10	**9**	8	**9**	8.5	**9**	**9**	**9**
Social Golfer	12	1	4.4	3	3.8	4	**5**	**5**
Tdsp	42	1	4.6	5	4.9	10	10	**15**
Traveling Salesman	30	**30**	29.2	**30**	**30**	**30**	**30**	**30**
Total	1800	1083	1143.5	1322	1328.4	1394	**1396**	1381

Notes: "S" refers to a sample of instances. Problem descriptions given in Appendix A.

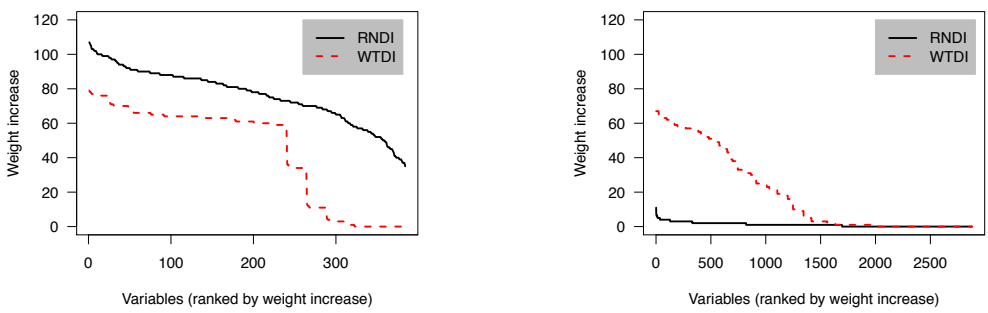
$dom/wdeg$ will behave identically to $wdeg$ on these instances, as it only contains Boolean variables. Thus, it will branch on a number of the 384 variables with degree 17 first (until sufficient weight has accrued on a variable of degree 2).

RNDI, on the other hand, is likely to branch primarily on the variables of

degree 2 here, given the probability of selecting a variable of degree 17 first is 0.12 compared to a probability of 0.88 for selecting a variable of degree 2. We note that the nodes explored during the probing phase by WTDI were 15,389, while there were over 79,000 nodes explored by RNDI during the probing phase in both cases. This shows that, as expected, it took much longer for failures to occur when randomly selecting variables.

Surprisingly we found that the Gini coefficient was greater than 0.84 for both runs of RNDI, compared to a Gini coefficient of 0.61 for WTDI. This suggests that RNDI was weighting a subset of the variables much more highly than WTDI. We investigated this further by calculating the Pearson product-moment correlation coefficient comparing the variable degree with the weighted degree. For both weight profiles of RNDI, this correlation was 0.98, while the correlation was 0.48 for WTDI. Thus we find that the weight profiles generated by RNDI loaded heavily on the degree factor, compared to WTDI.

This loading by RNDI on the degree can be clearly seen in Figure 6.6, where we plot the weight increase for WTDI and a run of RNDI on the variables separated by degree. Figure 6.6(a) shows the weight increase on the 384 variables of degree 17, while Figure 6.6(b) shows the weight increase on the variables of degree 2.



(a) Variables with degree 17

(b) Variables with degree 2

Figure 6.6: Weight increase per variable with RNDI/WTDI on sample Bibd.

We see that most failures occurred on the variables of degree 17 during random probing, while WTDI weighted a large number of the variables of degree 2 highly,

and a number of variables of degree 17 received no weight increase during the probing phase of WTDI. We separated the variables into two subsets based on degree, and found that both weight profiles of RNDI had a Gini coefficient of less than 0.12 on the variables with degree 17, and a Gini coefficient of 0.58 on the variables of degree 2. The weight profile of WTDI had a Gini coefficient of 0.35 on the variables of degree 17 and 0.64 on the variables of degree 2. This confirms the lack of discrimination amongst the variables of these problems by RNDI.

6.3.4 Comparison with Impact-Based Search

We compared three heuristics ($dom/fdeg$, $dom/wdeg$ and IBS) under two conditions, with and without restarting. The same geometric restarting approach was used for all, with an initial cutoff of 10 failures and a multiplicative factor of 1.5. These methods were tested on 26 of the 30 problem sets, using the same general experimental setup as before.

We removed problem sets containing variables with range domains as impact is not straightforward to implement for such variables. A range domain is where, due to the size of the domain, only the lower and upper bounds on the value that the variable can take are stored. The method used by Mistral for the impact of values in a range domain involves storing the impacts of the lower and upper bounds. However, the value these bounds take can change drastically during search as the bounds are updated.

Initializing impacts using singleton arc consistency resulted in worse performance overall, so the impact results presented are without SAC initialization. For geometric restarting the best strategy was to initialize the impacts to a large value (0.9999). Interestingly, the opposite was the case for IBS without restarting where the best method was to initialize the impacts to a low value (0.0001). The impact results presented in Table 6.3 are only for the best methods for the two cases.

The results are presented in Table 6.3. Comparing Geowtd with Geoimpact, we see that there were 12 problem sets where Geowtd solved more instances than Geoimpact, while there were only 3 problem sets where the opposite was the case. The relative performance of $dom/wdeg$ was even more impressive when restarting was not involved, solving 91 instances more than IBS. Furthermore, there was

Table 6.3: Number of Instances Solved: Impact-Based Search Comparison

		W/o Restarting			Geometric Restarting		
		dom/ fdeg	dom/ wdeg	IBS	Georand	Geowtd	Geo- impact
All Intervals	25	9	**19**	14	13.2	**25**	16
All Squares	74	33	**40**	32	33.7	**40**	33
Bdd	70	**70**	**70**	**70**	**70**	**70**	**70**
BIBD	83	22	**55**	36	62.4	**69**	66
Bmc	24	**24**	**24**	21	23.7	**24**	22
Boolean (S)	100	81	**92**	91	77	**90**	**90**
Chess Color	20	11	**13**	**13**	15.2	15	**16**
Coloring (S)	100	87	**89**	**89**	86.2	**90**	**90**
Costas Array	11	**9**	**9**	**9**	**9**	**9**	**9**
Crosswords (S)	100	69	**92**	71	67.1	**94**	84
Driver Log	7	**7**	**7**	**7**	**7**	**7**	**7**
Fapp (S)	100	16	**48**	31	9.4	**54**	37
Golomb	28	21	**23**	22	22.9	22	**23**
Langford	75	**53**	**53**	52	**54**	**54**	52
Multi-Knapsack	6	**6**	**6**	**6**	**6**	**6**	**6**
Patat	46	4	**5**	1	4	26	**28**
PseudoBool (S)	100	51	**64**	**64**	56.3	**66**	**66**
Pseudo-GLB (S)	100	25	**37**	**37**	24.2	**33**	**33**
Qcp/Qwh (S)	100	93	**95**	93	93.7	**94**	**94**
Radar	100	89	**100**	**100**	96.3	**100**	**100**
Ramsey	16	**5**	**5**	3	8.3	**10**	7
Random (S)	100	81	**85**	79	71.4	**82**	80
RLFAP	71	43	**68**	66	47.4	**69**	65
Schurrs Lemma	10	**9**	**9**	**9**	8	**9**	8
Social Golfer	12	1	3	**4**	4.4	**6**	3
Traveling Salesman	30	**30**	**30**	**30**	29.2	**30**	**30**
Totals	1508	949	**1141**	1050	1000	**1194**	1135

Notes: "S" refers to a sample of instances. Problem descriptions given in Appendix A.

only one problem set where $dom/wdeg$-$nores$ solved fewer instances, while there were 15 problem sets where it solved more instances. (We note that Balafoutis and Stergiou had similar findings in their comparison of $dom/wdeg$ and IBS [12]).

Overall, there were 1098 instances that were solved by both Geowtd and Geoimpact. In terms of runtime over these instances, we found that Geowtd was faster by 6.8 seconds on average (17.6 versus 24.4 seconds). If we remove the instances that were solved in under a second by both, we find that Geowtd was

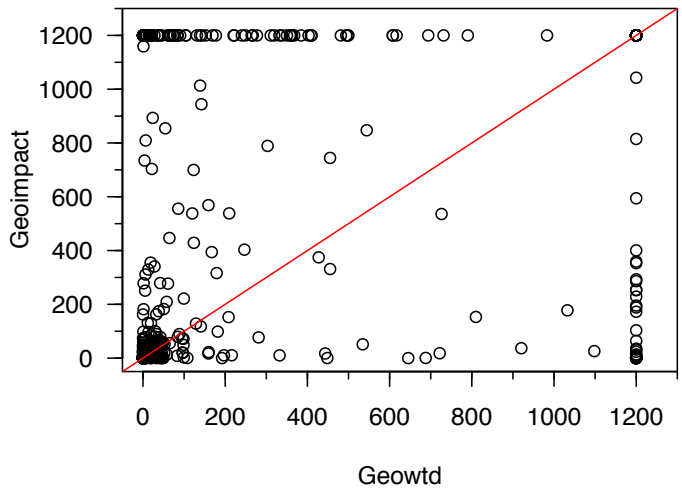


Figure 6.7: Scatter plot of runtimes for Geoimpact versus Geowtd.

nearly 30% faster on average (49.0s versus 68.3s) over the 393 "hard" instances, and had a median runtime of 6.0s compared to Geoimpact which had a median runtime of 10.6s.

A scatter plot of the runtimes over all instances is shown in Figure 6.7. The most striking aspect of the figure is that there were a large number of instances that Geowtd solved quickly while Geoimpact found extremely difficult and vice versa. This implies that many instances were suited to either constraint weighting or impact-based search, but not both. Further analysis of the results reveals that of the 1508 instances, Geowtd was at least two orders of magnitude faster than Geoimpact on 80 instances, while the opposite was the case on 41 instances.

The complementary nature of these results suggest that the best generic strategy for solving CSPs would be to run both in parallel. For example, if we had run both in parallel then 1231 instances would have been solved. Furthermore the average runtime of running both in parallel would have been 8.3 seconds on the 1098 commonly solved instances discussed above, which is less than 50% of the average runtime of Geowtd.

6.4 Case Study I: Open Shop Scheduling Problems

We have previously shown that the probing based methods are extremely efficient at solving open shop scheduling problems, due to their ability to identify the key variables and thus avoid the early mistake problem. We investigate whether the universal restarting strategies are more adept at solving such problems.

The default method used in Mistral for handling problems with disjunctive constraints is to introduce auxiliary variables which explicitly state the possible precedences. For a disjunctive constraint between tasks t_i and t_j, two Boolean variables b_{ij}, b_{ji}, and two range variables r_i, r_j, are added. These are linked via the following constraints:

$$(t_i + p_i) = r_i \tag{6.5}$$

$$(r_i \leq t_j) = b_{ij} \tag{6.6}$$

$$(t_j + p_j) = r_j \tag{6.7}$$

$$(r_j \leq t_i) = b_{ji} \tag{6.8}$$

$$b_{ij} \vee b_{ji} \tag{6.9}$$

where p_i is the duration of task t_i. Assigning a Boolean variable b_{ij} the value 0 enforces the precedence between tasks t_i and t_j, stating that task t_j must finish before task t_i can start. The decision variables used are the task and Boolean variables.

This model has two advantages, it speeds up propagation and can also reduce search effort by branching on the Boolean variables. This is because a consistent assignment to all Boolean variables creates a partial order on the tasks, where tasks are fully ordered on each job and machine. In other words, the Boolean variables form a strong backdoor to the problem. If a problem is unsatisfiable then there is no ordering of the tasks which will satisfy the constraints, if the problem is satisfiable then an assignment to all Booleans which satisfies all constraints proves a solution exists (and each task variable can simply be assigned the minimum value in its domain to find the solution).

In this section, we compare this model with the non-adapted model which contains only the tasks as variables. We will refer to the former as the *Aux-Task*

model and the latter as the *Task-Only* model. Due to the difference in propagation speeds, we define the overall limit per instance in terms of total nodes explored. The limit was set to one million nodes. Value ordering was again lexical, which involves branching on the lower bound of the domain of the selected variable for range variables.

The restarting parameters used are the same as for the previous section, with the exception that the base value for the geometric restarting method was 10 failures for both Geowtd and Georand. This is because these problems are suited to restarting, and so the lower initial cutoff resulted in slightly better performance due to the extra short runs. We performed some initial experiments to assess whether WTDI or Lubywtd would also benefit from a reduction in (initial) cutoff, and found that this was not the case generally.

We first tested the various methods (excluding impact-based search as these problems involve variables with very large domains) on the five largest sets of Taillard open shop instances, ost-n (where n refers to the number of jobs/machines, so there are n^2 tasks per instance). We remind the reader that each set contains thirty instances, at least twenty of which are satisfiable (ten ost-n-100 and ten ost-n-105 instances). The other ten instances (ost-n-95) are unsatisfiable in most cases. The instances range in size from 25 to 400 tasks to be scheduled, with domain sizes ranging from ~200 values (for ost-5-95 instances) to ~1200 values (for ost-20-105 instances).

The results in Table 6.4 show that the methods using the $dom/fdeg$ heuristic are quite poor on these problem sets, due to the auxiliary Boolean variables. Since these variables all have the same initial domain size and degree, the heuristic cannot discriminate amongst the $2n^2(n-1)$ such variables. The weighted degree approaches, on the other hand, are extremely adept at solving these instances with $dom/wdeg$-$nores$ alone solving nearly two thirds of the instances. Indeed, the poor discrimination of $dom/fdeg$ further underlines the power of $dom/wdeg$ on these instances, as the initial decisions made by the heuristic are completely uninformed.

RNDI solved a further 22 instances more than $dom/wdeg$-$nores$ on average even though, outside of the ost-5 set, an instance was rarely solved during the probing phase. This shows the ability of the method to identify the key variables

Table 6.4: Taillard Open Shop Scheduling Problems (Aux-Task model)

		$dom - fdeg$	Geo rand	$dom - wdeg$	RNDI	WTDI	Geo wtd	Luby wtd
ost-5	# Solved	19	**30**	**30**	**30**	**30**	**30**	**30**
	Sol Time (s)	1.13	0.07	0.04	0.09	0.03	**0.02**	0.04
	Sol Nodes (K)	46	2.5	1.2	3.7	0.8	0.5	1.1
	# Pre Solved	-	-	-	11.7	30	-	-
ost-7	# Solved	0	8.4	29	29.3	**30**	**30**	28
	Sol Time (s)	-	1.63	5.91	1.82	4.08	**3.62**	2.78
	Sol Nodes (K)	-	37	87	31	49	52	38
	# Pre Solved	-	-	-	2.4	27	-	-
ost-10	# Solved	0	1.1	17	26	**29**	28	28
	Sol Time (s)	-	12.08	18.42	8.11	7.99	6.63	11.23
	Sol Nodes (K)	-	132	231	101	98	74	136
	# Pre Solved	-	-	-	0.1	20	-	-
ost-15	# Solved	0	0	13	19.8	**21**	**21**	**21**
	Sol Time (s)	-	-	40.48	33.97	4.11	**3.9**	4.84
	Sol Nodes (K)	-	-	152	87	15	15	20
	# Pre Solved	-	-	-	0	21	-	-
ost-20	# Solved	0	0	10	16.7	**22**	**22**	**22**
	Sol Time (s)	-	-	76.67	347.57	65.94	54.31	**48.1**
	Sol Nodes (K)	-	-	85	184	54	46	43
	# Pre Solved	-	-	-	0	22	-	-
Total Solved		19	39.5	99	121.8	**132**	131	129

Notes: Results for Georand and RNDI are averaged over ten runs. "#Pre Solved" refers to the number of instances solved during the probing phase by RNDI/WTDI.
"Sol" Time/Nodes refers to the average time taken / nodes explored over the instances solved by the method.

through their constraint weights. The best approaches were those directly combining the weighted heuristic with a restarting strategy, with all three methods solving over 85% of the 150 instances.

For the ost-15 and ost-20 problem sets, unsatisfiability was never proven by any method due to the size of the search space. We note that one of the ost-15-95 instances is satisfiable, as are two of the ost-20-95 instances. This is because the best known upper bound for the optimization instance was not optimal when the instances were generated. Thus, there were 17 instances between these two sets which were not solved by any method.

The performance of the weighted restarting approaches is even more impressive if we restrict our attention to the (known) satisfiable instances. Of the 103 such instances, Lubywtd only failed to solve two, while WTDI and Geowtd solved all satisfiable instances.

Table 6.5: Gueret-Prins Open Shop Scheduling Problems (Aux-Task model)

		$dom - fdeg$	Geo rand	$dom - wdeg$	RNDI	WTDI	Geo wtd	Luby wtd
osgp-10 Sat	# Solved	0	4.6	**10**	**10**	**10**	**10**	**10**
	Sol Time (s)	-	3.40	12.88	1.28	5.71	**0.88**	13.66
	Sol Nodes (K)	-	30	71	15	28	5	77
	# Pre Solved	-	-	-	0.5	9	-	-
osgp-10 Unsat	# Solved	0	0.1	9	**10**	9	**10**	9
	Sol Time (s)	-	3.47	12.28	**4.98**	10.71	13.24	2.97
	Sol Nodes (K)	-	24	56	39	52	75	13
	# Pre Solved	-	-	-	0	8	-	-
Total Solved		0	4.7	19	**20**	19	**20**	19

Notes: Results for Georand and RNDI are averaged over ten runs. "#Pre Solved" refers to the number of instances solved during the probing phase by RNDI/WTDI.
"Sol" Time/Nodes refers to the average time taken / nodes explored over the instances solved by the method.

We further experimented on a set of OSPs (of size 10) originally proposed by Guéret and Prins [95], which were converted to CSP format by Naoyuki Tamura [†] for the CSP solver competition ([202]). These were generated with the expectation that they would be significantly harder than the Taillard instances. This was done by ensuring that there was a sufficient gap between the optimal makespan value and the trivial lower bound (which is the maximum of the sum of task durations for each job/machine). There are ten instances in the original set. These were converted into a set of satisfiable instances by setting the upper bound on the domains of the tasks to the optimal makespan for the instance, and into a set of unsatisfiable instances by setting the upper bound on the domains of the tasks to (optimal makespan - 1).

[†] `http://bach.istc.kobe-u.ac.jp/tamura.html`

We see in Table 6.5 a similar pattern in the results to those of the Taillard instances, with the adaptive methods significantly outperforming the two algorithms which use $dom/fdeg$. Interestingly, restarting does not appear to be as necessary for good performance here, with $dom/wdeg$-$nores$ solving the same number of instances as WTDI and Lubywtd, albeit with a slightly greater average runtime.

RNDI and Geowtd were the only methods to solve all instances, with RNDI outperforming the latter in terms of average runtime. Furthermore, of the 95 runs where RNDI solved a satisfiable instance on the run to completion, seven of those were solved backtrack-free. (We note that when a smaller cutoff of 30 failures was used for WTDI, it solved all instances and had the lowest average runtimes (0.5s for the satisfiable instances and 3.6s for the unsatisfiable instances).) The impressive performance of the weighted approaches on these instances is surprising given the aforementioned expectation that the instances in optimization format were much harder than the Taillard instances (Guéret and Prins [95]).

In Table 6.6 we present results on the same Taillard problem sets using the Task-Only model, so each ost-n instance has n^2 (range) variables. The most noticeable trend is in the difference between the non-adaptive versus adaptive methods. The two algorithms which use $dom/fdeg$ solved roughly 20 instances more than before, while the adaptive methods all solved 20 instances *fewer* than with the Aux-Task model.

The improvement in performance of the non-adaptive methods can be attributed to greater discrimination amongst the heuristic's choices. This is particularly evident for the two largest sets. Interestingly, although the performance of $dom/wdeg$-$nores$ on these larger sets improved somewhat, overall it solved 25 fewer instances than with the previous model. Similar results were found for the other adaptive methods, with all solving at least 20 more instances with the Aux-Task model.

Furthermore, we found that search was markedly slower with this model. For example, there was approximately two orders of magnitude difference in the runtime of the random probing phase between the previous model and this model on the ost-15 instances (three seconds versus three hundred seconds). However, one interesting result with RNDI was that, of the 39.4 instances solved on average between the ost-15 and ost-20 sets, over 75% were solved backtrack-free on the run

Table 6.6: Results For Open Shop Scheduling Problems (Task-Only model)

		$dom - fdeg$	Geo rand	$dom - wdeg$	RNDI	WTDI	Geo wtd	Luby wtd
ost-5	# Solved	12	11.6	27	27	28	**29**	**29**
	Sol Time (s)	124.42	12.86	113.19	148.86	153.37	**44.82**	94.34
	Sol Nodes (K)	74	20	61	86	81	24	55
	# Pre Solved	-	-	-	10.8	26	-	-
ost-7	# Solved	4	8.7	10	16.3	**17**	**17**	14
	Sol Time (s)	1025.89	42.71	614.84	677.89	481.29	**356.88**	191
	Sol Nodes (K)	251	40	156	181	117	83	49
	# Pre Solved	-	-	-	4.8	15	-	-
ost-10	# Solved	4	9.8	9	15	19	**20**	**20**
	Sol Time (s)	5.26	32.74	709.33	163.07	106.93	**150.8**	456.71
	Sol Nodes (K)	0.6	22	150	31	18	26	78
	# Pre Solved	-	-	-	1.2	19	-	-
ost-15	# Solved	11	16.9	15	19	**21**	20	**21**
	Sol Time (s)	24.99	59.92	126.62	327.8	**88.85**	30.44	125.92
	Sol Nodes (K)	3	18	42	29	8	2	14
	# Pre Solved	-	-	-	1.2	21	-	-
ost-20	# Solved	9	18.8	13	20.4	**22**	**22**	**22**
	Sol Time (s)	37.67	108.77	290.4	646.03	**138.62**	525.37	293.5
	Sol Nodes (K)	0.5	16	16	30	7	29	21
	# Pre Solved	-	-	-	0.6	22	-	-
Total Solved		40	65.8	74	97.7	107	**108**	106

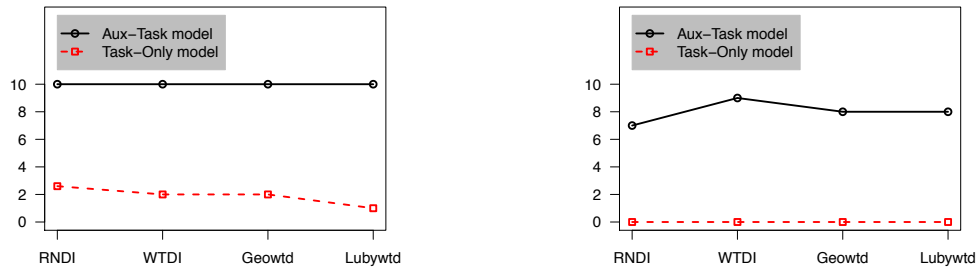
Notes: Results for Georand and RNDI are averaged over ten runs. "#Pre Solved" refers to the number of instances solved during the probing phase by RNDI/WTDI. "Sol" Time/Nodes refers to the average time taken / nodes explored over the instances solved by the method.

to completion (a similar result was found [89] using the solver JaCoP [†]).

Upon closer inspection, we find that the main difference between the performance of the adaptive heuristics on the two models is due to the problem sets ost-7 and ost-10, and in particular the unsatisfiable instances of those problem sets. As shown in Figure 6.8, no method was able to prove unsatisfiability on any of the ost-10-95 instances with the Task-Only model, whereas a minimum of seven were solved with the Aux-Task model. (Note that the difference in propagation speeds

[†] www.jacop.eu

cannot be an explanation for this as search had an overall *node* limit.)



(a) ost-7-95 (b) ost-10-95

Figure 6.8: Number of unsatisfiable instances solved using two different models.

6.4.1 Analysis of weight profiles

We first tested whether the same tasks would be highly weighted by the two different models using random probing. We selected a satisfiable instance (ost-7-100-0), as the weights would be spread across insoluble cores with an unsatisfiable instance. We ranked the task variables of both models by their weighted degree and found that the top-down rank correlation coefficient was 0.93. Furthermore, as can be seen in Figure 6.9, the four most highly weighted task variables were the same for both models. With regard to the Boolean variables, at least one of the two top ranked task variables shared a constraint with the five top ranked Boolean variables in the Aux-Task model.

Refutation size comparision for over-constrained job/machine

To better understand the results on the unsatisfiable instances, let us consider the case where an instance has an over-constrained job/machine. (We note that global constraints for the unary resources such as Edge-Finder (Carlier and Pinson [40]), etc., would be able to discover this inconsistency directly.) In order to prove unsatisfiability in the Task-Only model, one will need to search an insoluble core

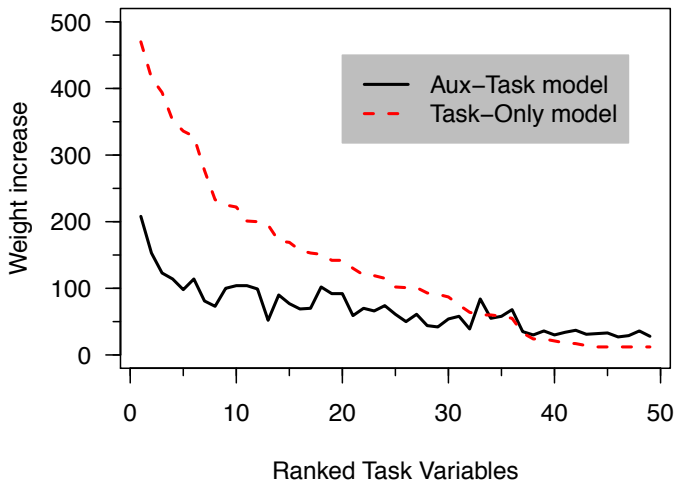


Figure 6.9: Weight increase on task variables after random probing for instance ost-7-100-0 using different models. Variables ranked by weight on Task-Only model.

involving n variables, all with large domains. On the ost-7-95 instances, this involves 7 variables each with domain size greater than 300. For the ost-10-95 instances, this involves 10 variables, each with domain size greater than 500.

For the Aux-Task model, on the other hand, branching solely on the Boolean variables amongst tasks in an over-constrained job/machine is sufficient to prove the instance unsatisfiable. There are $n(n-1)$ Boolean variables per job/machine. So for the ost-7-95 instances there are 42 Boolean variables per job/machine, while there are 90 Boolean variables per job/machine in the ost-10-95 instances. However, only half of these are necessary to instantiate, as there are two Booleans per disjunct and assigning one to 0 will result in the other taking the value 1.

Analysis of the search trees explored, and the weight profiles generated, confirms that branching on the Boolean variables is key to proving an instance unsatisfiable. We compared the two models on the instance ost-7-95-1 using the Geowtd search strategy. We firstly note that this instance was solved with Geowtd in 104 nodes using the Aux-Task model, compared to over 800,000 nodes with the Task-Only model.

We found that, on the restart where the instance was solved with the Aux-Task model, search branched on just nine Boolean variables. These Boolean variables

were all of the same job. Moreover, only five of the seven tasks of this job appeared in the constraints on the nine Booleans. The sum of the durations of these five tasks is greater than the makespan allowed, and thus they form an insoluble core.

For the Task-Only model, we generated a weight profile with Geowtd after 1000 nodes of search, since this was sufficient to illustrate the behaviour of the algorithm on this model. Only ten of the forty nine tasks received any weight increase, six of which had a weight increase of approximately 300, with the other variables receiving a weight increase of at most ten. As expected, we find that these six tasks are all tasks of the same over-constrained job (albeit a different over-constrained job to that found by Geowtd with the Aux-Task model).

Since these six tasks form an insoluble core, once identified, no other variables will be weighted as these variables will repeatedly be selected and thus will accrue all subsequent weight. Yet, there were over 800,000 nodes explored by Geowtd before the instance was proven to be over-constrained. This can be explained by the interaction of variable convection with binary branching. Although binary branching guided by a weighted degree heuristic clearly has advantages over d-way branching, in that it allows earlier use of weight information and results in greater diversification, it also has disadvantages when solving problems with an insoluble core (as alluded to in Chapter 3.4.1).

Let x be the first variable of the insoluble core selected at the top of the search tree. For d-way branching, each value of x is tried and found to be inconsistent, so at level 1 there will be $|\text{dom}(x)|$ values tried. With binary branching, the first value of x will be tried and found to be inconsistent. However, due to the weight accrued while proving this value inconsistent, a different variable, y say, may be selected next at level 1. If the heuristic repeatedly flips between just these two variables, then there will be ($|\text{dom}(x)|$ + $|\text{dom}(y)|$ - 1) values tried at level 1. This effect will obviously not occur if the variables are Boolean.

Identification of over-constrained jobs/machines

We generated weight profiles on an unsatisfiable instance of the largest set, ost-20-95-0, with the three weighted restarting approaches. We found that even for instances of this size, insoluble cores in the shape of over-constrained jobs/machines

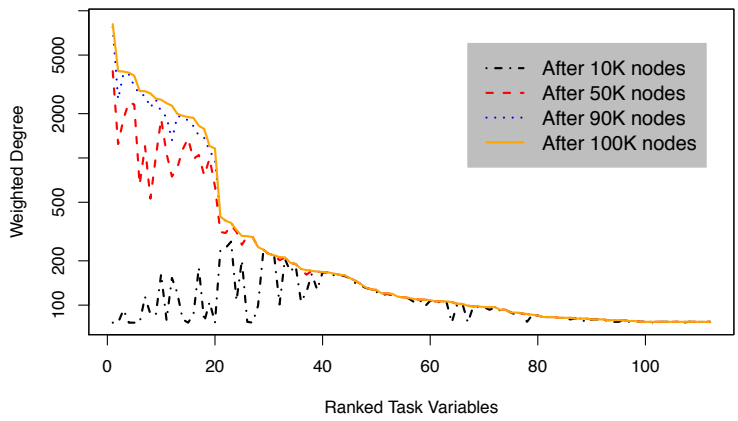


Figure 6.10: Evolution of constraint weights on task variables (which received a weight increase of at least one), during Geowtd search using Aux-Task model, on instance ost-20-95-0.

are quickly identified with the Task-Only model. For example, after two thousand nodes of search with Geowtd, the weights converge on a single over-constrained job.

However, weight profiles generated for the Aux-Task model reveal that it takes much longer for weights to converge on an over-constrained job. This can be attributed to the combination of weak propagation and lack of initial discrimination amongst the 15,200 Boolean variable. For example, we analyzed weight profiles at different points during search with Geowtd using the Aux-Task model on the same unsatisfiable instance. Since this instance was not proven unsatisfiable within the one million node limit, we outputted the weighted degrees every ten thousand nodes to assess the evolution of the weights.

In Figure 6.10, we plot the weighted degree of each task variable, which received a weight increase of at least one, at four arbitrary points during search with Geowtd: after 10,000, 50,000, 90,000, and 100,000 nodes. Variables are ranked by their weighted degree after 100,000 nodes. The first point to note is that an over-constrained job was eventually identified (the first twenty variables in the figure). This was the same job as was identified using the Task-Only model after 2,000 nodes of search. However, here it took roughly 90,000 nodes for the weights to converge on this job (note that the weighted degrees after 90,000 and 100,000

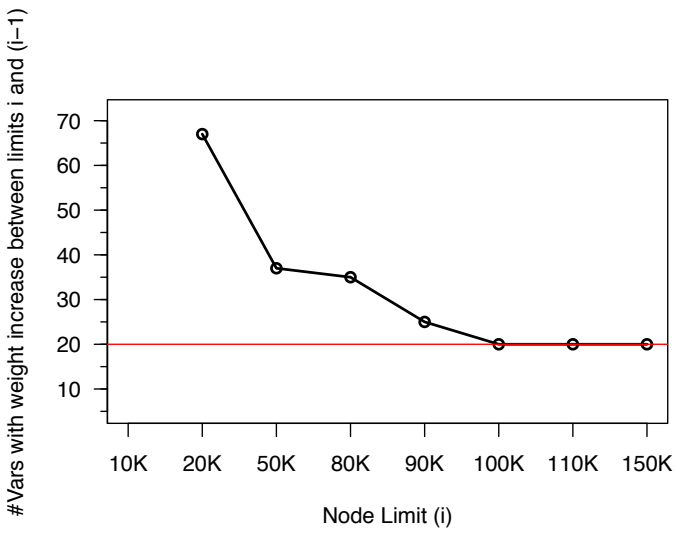


Figure 6.11: Geowtd search using Aux-Task model on instance ost-20-95-0. Number of task variables whose weighted degree increased between node limits i and (i - 1) in sequence (10K, 20K, 50K, 80K, 90K, 100K, 110K, 150K).

nodes only differ on the top twenty ranked variables). Even after 50,000 nodes there were tasks of other jobs/machines receiving weight. A similar result was found with WTDI.

This can be seen more clearly in Figure 6.11. Here, we look at the number of variables whose weight increased during successive periods of search. We generated weight profiles after each of the following node limits (10K, 20K, 50K, 80K, 90K, 100K, 110K, 150K). For each limit (from 20K on), we subtracted the weighted degree of each task variable after the previous limit, from the weighted degree after the current limit. The results show that after 90,000 nodes search converged on twenty variables (further analysis revealing that they are all tasks on one over-constrained job), and these twenty variables were the only variables weighted during search thereafter. Overall, this analysis shows that the Aux-Task model is better at proving problems infeasible, even though it can take much longer to identify an unsatisfiable core.

6.5 Case Study II: Radio Link Frequency Assignment Problems

The radio link frequency assignment problem (RLFAP) involves assigning frequencies to a number of radio links in a communications network, such that interference is minimized between radio sites wishing to communicate with one another. CELAR (the french "Centre d'Electronique de l'Armement") designed a set of simplified versions of this problem (RLFAPs), based on data from a real network. These instances were then made public in the framework of the European EUCLID project CALMA ("Combinatorial Algorithms for Military Applications")[†].

In CSP terms the variables are the radio links, the possible frequencies are the domains of the variables, and the constraints define distances between frequencies. More formally, Cabon et al. [37] describe the problem as follows. We are given a set X of unidirectional radio links. For each link $i \in X$, a frequency f_i has to be chosen from a finite set D_i of available frequencies for that link. Binary constraints are defined on pairs of links (i,j). These constraints can take two forms. The first type specifies that the distance between the frequencies of i and j must be greater than a given constant $d_{i,j}$:

$$|f_i - f_j| < d_{i,j}$$

The other type of constraint specifies that the distance between the frequencies of i and j must be equal to a given constant $\delta_{i,j}$ (238 in all CELAR instances):

$$|f_i - f_j| = \delta_{i,j}$$

The objective of this optimization problem is to find a solution which uses the minimum number of frequencies.

The RLFAPs tested are again taken from the CSP Solver Competition website, and are converted to satisfaction problems by fixing the frequencies available to each variable. The problem is then to find a solution, if one exists, using the limited set of frequencies. There are 12 instances in the set we test, which are all

[†] `http://www.win.tue.nl/~wscor/calma.html`

modified versions of the same base instance "scen11". The instance was modified by removing the highest f frequencies, so scen11-f3 is the scen11 instance with the three highest frequencies of the original instance removed from the domains of the variables. Each instance contains 680 variables, 4103 constraints, and with maximum domain size ranging from 32 (for -$f12$) to 43 (for -$f1$). All instances are unsatisfiable.

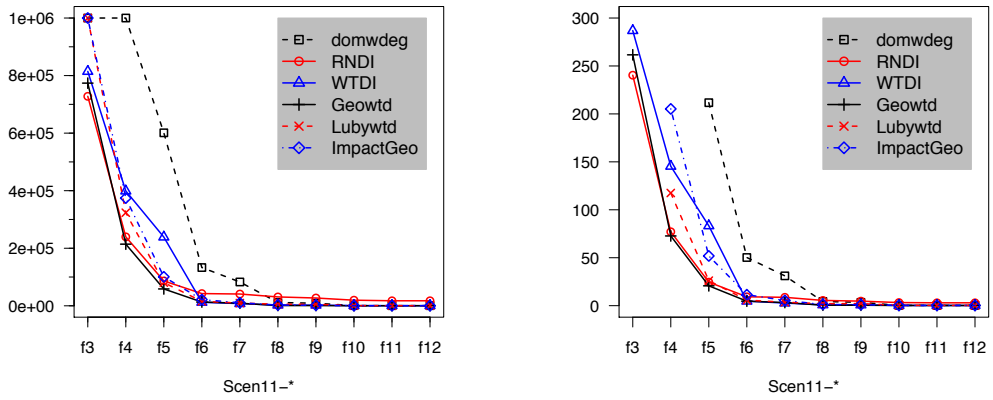
Table 6.7: Results For RLFAP Modified Scen11 Problems

	$\frac{dom}{wdeg}$	RNDI	WTDI	Geowtd	Lubywtd	Geoimpact
# Solved	8	**10**	**10**	**10**	9	9
# Pre Solved	-	0	7	-	-	-
Mean Sol Time (s)	37.71	37.93	51.55	**36.45**	37.62	30.77
Mean Sol Nodes (K)	105	125	148	**107**	120	57

The results in Table 6.7 and Figure 6.12 once again show the ability of the weighted degree heuristic in solving unsatisfiable instances, once it is able to make informed decisions at the top of the search tree. Neither $dom/fdeg$ nor Georand were able to solve any of these instances. The impact heuristic was also effective at solving these instances. However, with the same parameters, geometric restarting was consistently more efficient when combined with the weighted degree heuristic than with impact-based search.

Random probing performed marginally better on the hardest instance as the minimal refutation was of a considerable size. This also explains the poor performance of the Luby restarting strategy and, to a lesser extent, WTDI on these large instances as the cutoff(s) were too low to prove unsatisfiability on the harder instances. However, the opposite was the case for the instances with most frequencies removed ($f8$-$f12$), where the cost of random probing outweighed the benefits. We also note that when a cutoff of 30 failures was used for WTDI, it solved the same 10 instances in 31.93 seconds on average, 4 seconds faster than Geowtd.

Weight profiles were generated for all constraint weighting methods on the instance scen11-f6. All methods except RNDI had a node limit of 10,000. For RNDI,



(a) Nodes (b) Time (s)

Figure 6.12: Individual results for RLFAP modified Scen11 Instances.

we outputted weights after the probing phase. Lubywtd, WTDI and $dom/wdeg$-$nores$ all resulted in a weight increase on only 19 variables out of 680 after 10,000 nodes of search, and the same 19 variables were weighted by all. These 19 variables also had their weight increased when using Geowtd, along with just one other variable. Analysis of the instance reveals that these variables form an insoluble core of the instance.

On the other hand, 150 variables had a weight increase of at least one during the random probing phase of RNDI. We believe that this is because the more over-constrained a problem, the more likely it is that a variable will be part of some insoluble core, and thus RNDI is likely to spread its weight across a number of variables, We investigated this hypothesis by generating weight profiles after random probing for a number of the scen11-fn instances. Note that the larger the value of n, the more values that have been removed from the domains of the variables, and thus the more over-constrained the instance. If our theory holds, then RNDI should weight fewer variables for scen11-f1 than for scen11-f6.

Weight profiles were generated with RNDI for instances scen11-f1, scen11-f3, scen11-f6, and scen11-f10. The results shown in Figure 6.13 support the hypothesis. We see that the more domain values that were removed, the greater the number of variables which received a weight increase. Indeed, for scen11-f1, only

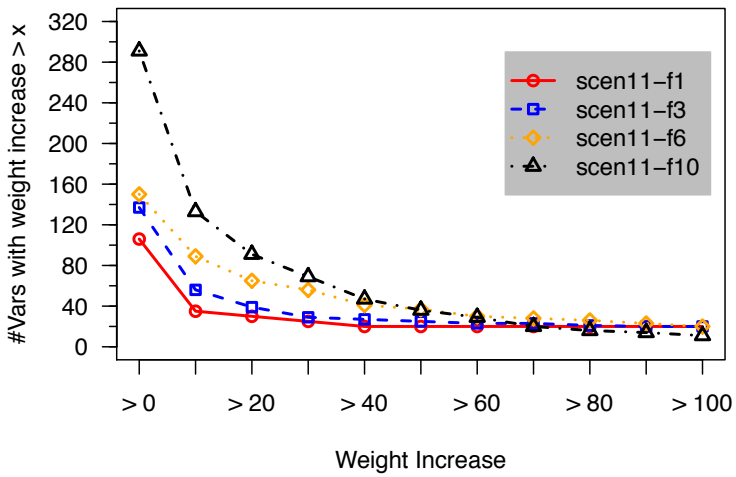


Figure 6.13: Spread of weight after random probing on RLFAP modified scen11 instances.

35 variables received a weight greater than 10, and only 20 variables received a weight greater than 40. These 20 variables all received a weight increase of at least 100 during probing, and we find that these are the same 20 variables as were weighted by Geowtd on the instances scen11-f1 and scen11-f6. Similarly we note that the top twenty variables after random probing on scen11-f3 were the same as on scen11-f1.

6.6 Chapter Summary

In this chapter, we have provided an extensive empirical analysis of different restarting strategies for the weighted degree heuristic. The best strategies were the pure restarting approaches, with little overall difference between WTDI and the geometric restarting strategy of Walsh in particular. It should be noted that the benefits of restarting combined with the weighted degree heuristic for generic problem solving is widely accepted in the community, as evidenced by the number of solvers in the last CSP solver competition which used this combination (Lecoutre et al. [137]).

Furthermore, we found that the weighted degree heuristic outperformed impact-based search on a wide range of problems. Of equal interest was the behavior of

the two heuristics, where we observed a number of instances, for both heuristics, where one was an order of magnitude faster than the other.

Experimental analysis of open shop scheduling problems revealed that the weighted degree heuristic, combined with a restarting strategy, is extremely efficient at solving this type of problem. These results mirrored the general pattern seen earlier, with the best methods being the restarting methods that move from diversification (short runs) to intensification (long run(s)), i.e. WTDI and Geowtd.

We compared two models for this type of problem and found that, despite the lack of initial discrimination by the heuristic, the model that works best was the one incorporating auxiliary Boolean variables to represent the disjunctive constraints. The ability of constraint weighting to identify the critical variables in these problems was further illustrated by the fact that RNDI solved a large number of instances backtrack-free on the run to completion using one of the models. Analysis of the results and the weight profiles produced on the different models revealed that the Aux-Task model was better for proving unsatisfiability, even though it can take longer to identify insoluble cores due to the lack of initial discrimination and the number of auxiliary variables in the larger instances.

Finally we tested the different approaches on a set of difficult Radio Link Frequency Assignment Problems. The results illustrated the ability of the methods which combine the weighted degree heuristic directly with a restarting strategy to quickly identify a small subset of variables which formed an insoluble core, and prove the instances infeasible by eventually shifting these variables to the top of the search tree upon restarting. On the other hand, RNDI spread its weight out more on the instances which were more over-constrained. This has a negative impact as search on the run to completion may jump between unconnected insoluble cores.

Chapter 7

A Generic Approach for Disjunctive Scheduling Problems

.

7.1 Introduction

Scheduling problems have proven to be fertile research ground for constraint programming and other combinatorial optimization techniques. There are numerous such problems occurring in industry, and whilst relatively simple in their formulation, they typically involve only *Sequencing* and *Resource* constraints, they remain extremely challenging to solve.

The most efficient methods for solving disjunctive scheduling problems like *Open shop* and *Job shop* scheduling problems are usually dedicated local search algorithms, such as tabu search (Nowicki and Smutnicki [157, 158]) for job shop scheduling and particle swarm optimization (Sha and Hsu [185]) for open shop scheduling. However, constraint programming often remains the solution of choice. It is relatively competitive (Beck [21], Watson and Beck [224], Malapert et al. [142]) with the added benefit that optimality can be proven.

The best CP models to date are those based on strong inference methods, such as *Edge-Finding* (Carlier and Pinson [40], Nuijten [159]), and specific search strategies, such as *Texture* (Fox et al. [64]). Indeed, the conventional wisdom is

that many of these problems are too difficult to solve without such dedicated techniques. After such a long period as an active research topic (over half a century since Johnsons seminal work [114]) it is natural to expect that methods specically engineered for each class of problems would dominate approaches with a broader spectrum.

In the previous chapter we showed that combining the weighted degree heuristic with a universal restarting strategy is extremely efficient for Open shop scheduling problems. We explore this result further in this chapter, comparing state-of-the-art approaches for a number of disjunctive scheduling problems with our "light model" (weighted degree heuristic combined with bounds consistency) implemented in Mistral. We will empirically show that the complex inference methods and search strategies currently used in state of the art constraint models can, surprisingly, be advantageously replaced by simple propagation algorithms and the *weighted degree* heuristic [29].

Our approach relies on a standard constraint model and generic variable/value ordering heuristics and restart policy. Each unordered pair of tasks that cannot be run in parallel, whether because they share a machine (OSP/JSP) or belong to the same job (OSP), are associated through a *disjunctive* constraint, with a Boolean variable standing for their relative ordering. Following the standard search procedure for this class of problems, the search space can thus be restricted to the partial orders on tasks. Once the tasks are totally ordered on each resource the residual problem is a *Simple Temporal Problem* (STP) [57] and can be solved in polynomial time.

The key components of our algorithm are as follows. The choice of the next (Boolean) variable to branch on is made by combining the current domain sizes of the associated two tasks with the weighted degree of the corresponding ternary disjunctive constraint. The current "best" solution is used as a value ordering heuristic. Finally, we use a standard restarting policy together with a certain amount of randomization, and nogoods are recorded from restarts using the method of Lecoutre et al. [135].

In the next section we describe the general type of scheduling problem considered in this chapter and outline the main CP methods that have been developed for tackling these problems. In section 7.3, we introduce our basic model and the

different components of our search algorithm.

The following six sections deal with different variants of the disjunctive scheduling problem: open shop scheduling; job shop scheduling; job shop scheduling with setup times; job shop scheduling with maximal time lags; no-wait job shop scheduling; and finally job shop scheduling with earliness and tardiness costs. We show how our basic model can easily be adapted to handle each variant, and provide compelling empirical proof of the benefits of our method compared with state-of-the-art exact and approximate methods.

Section 7.10 provides a detailed evaluation of the different components in our algorithm, identifying those that are key to the performance. Analysis of the weight profiles produced on the problem variants is given in Section 7.11. Finally, conclusions of the chapter are outlined in Section 7.12, along with avenues for future research.

7.2 Background

The scheduling problems we study can be generally defined as the problem of scheduling n jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$, on a set of m resources $R = \{R_1, \ldots, R_m\}$. A job J_i consists of a set of tasks $\mathcal{T} = \{t_1, \ldots, t_k\}$, where each task has an associated processing time and an associated resource on which it must be processed. A resource has an associated capacity which cannot be exceeded at any time point.

Scheduling problems can be distinguished based on the type of resource: *disjunctive scheduling* where a resource can only handle one task a time (in this case the resource is referred to as a machine); or *cumulative scheduling* where a resource can execute several tasks at a time provided its capacity isn't exceeded.

Problems can also be distinguished based on the type of task: *non-preemptive scheduling* where a task must run to completion once it has started; or *preemptive scheduling* where tasks can be interrupted during processing. All the problems we study are *disjunctive, non-preemptive, scheduling problems*, i.e. resources can only process one task at a time and tasks cannot be interrupted once started.

Table 7.1 presents the notation that will be used throughout this chapter. Furthermore, we use the usual classification of Graham et al. [87] to define our problems under the notation $\alpha|\beta|\gamma$. These parameters describe respectively the ma-

chine environment, the job characteristics, and the optimality criterion. Here, α is either J or O, where J (resp. O) means the problems are job shop (resp. open shop). β refers to the characteristics of the jobs, for example if jobs have a release date before which they cannot start and a due date for when they should finish then β = "$rd_i dd_i$".

Table 7.1: Table of Notation.

J_j	Job j
M_m	Machine m
t_i	Task i
st_i	Start time of t_i
p_i	Processing time of t_i
C_j	Completion time of J_j
rd_j	Release date of J_j
dd_j	Due date of J_j
P_j	Processing time of $J_j = \sum_{t_i \in J_j} p_i$
P_m	Processing time of $M_m = \sum_{t_i \in M_m} p_i$
L_j	Lateness of J_j = $(C_j - dd_j)$
E_j	Earliness of J_j = $\max(-L_j, 0)$
T_j	Tardiness of J_j = $\max(L_j, 0)$
w_j^e	Cost of J_j being early per unit
w_j^t	Cost of J_j being tardy per unit
$s_{i,k}$	Setup time between t_i finishing and t_k; $t_i, t_k \in M_m$
tl_j	Maximum time lag allowed between t_i finishing and t_{i+1} starting in J_j
est_i	Earliest start time of t_i, i.e. $\underline{st}_i$
eft_i	Earliest finish time of t_i, i.e. $\underline{st}_i + p_i$
lst_i	Latest start time of t_i, i.e. $\overline{st}_i$
lft_i	Latest finish time of t_i, i.e. $\overline{st}_i + p_i$
Ω	A subset of tasks on a resource
st_Ω	$min_{t_i \in \Omega}(st_i)$
ft_Ω	$max_{t_i \in \Omega}(st_i + p_i)$
p_Ω	$\sum_{t_i \in \Omega} p_i$

The optimization criterion γ, refers to the objective function which one intends to minimize/maximize. There are a number of such criterion, however we only consider the following two objectives:

- Makespan: C_{max} = $\max C_j$
  The objective is to find the schedule with minimum overall duration. If we consider time 0 to be the earliest starting time, then C_{max} is the latest finishing time over all jobs.

- Earliness Tardiness Costs: $ETcost$ = $\sum_{\mathcal{J}}(E_j * w_j^e + T_j * w_j^t)$
  In certain situations, where there is a due date associated with each job, there may be a penalty for a delay in the completion of a job. Similarly there may be a cost associated with earlier completion of a job (e.g. the storage of the product until it is ready to be shipped on its due date).

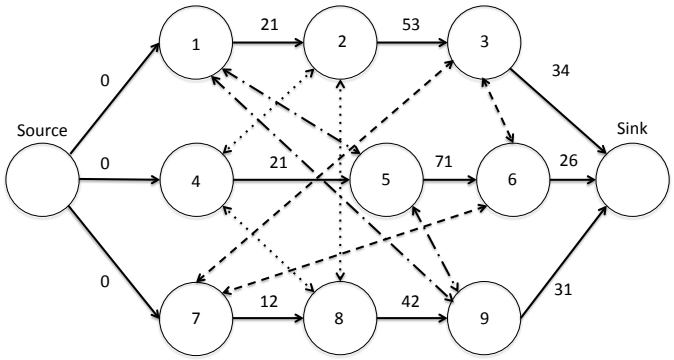
Many approaches use the disjunctive graph representation (Roy and Sussman [172]) for these problems, G = (N, D, A). There is a set of nodes N, where each node in the graph represents a task (outside of the source and sink dummy nodes). The set D of directed arcs represents the precedence constraints and the set A of bidirectional dashed arcs represents the disjunctive constraints. A job is a maximal path in (N,D), while a resource is a maximal clique in (N,A). For each arc in A, a direction must be chosen for the arc which defines the order of the two tasks on the machine. The length of the arc is the duration of the task on the tail of the arc.

A solution to the scheduling problem is an acyclic graph where all bidirectional arcs are replaced with directed arcs. The makespan of the scheduling problem is the *critical path* from the source node to the sink node, i.e. the longest path from the source to the sink. To find the optimal solution, one therefore needs to direct the bidirectional arcs such that the critical path is minimized.

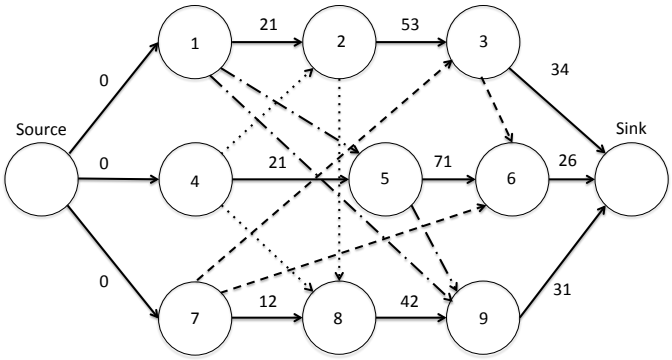
We introduce a sample job shop scheduling problem with three jobs and three machines to illustrate the disjunctive graph. The problem is outlined in Table 7.2, while Figure 7.1 presents the disjunctive graph representation of this problem, along with a sample graph of an optimal solution. The critical path of the optimal solution is: "Source" $\rightarrow$ 4 $\rightarrow$ 2 $\rightarrow$ 8 $\rightarrow$ 9 $\rightarrow$ "Sink", which gives optimal C_{max} of 147 (=0+21+53+42+31).

Table 7.2: Sample 3×3 Job Shop Instance.

	t_1		t_2		t_3	
	M_i	p_i	M_i	p_i	M_i	p_i
J_1	2	21	1	53	3	34
J_2	1	21	2	71	3	26
J_3	3	12	1	42	2	31



(a) Disjunctive Graph



(b) Optimal makespan

Figure 7.1: Disjunctive Graph for sample 3×3 JSP.

7.2.1 Traditional CP approach ("Heavy Model")

We first provide a brief overview of some of the main inference techniques and search heuristics developed for the non-preemptive machine scheduling problems. Modeling this type of problem as a COP is quite straightforward: the variables are the tasks, their domains are the possible starting times of the task, and the

constraints specify relations between the tasks, e.g. no two tasks sharing a resource can overlap. The objective function is a criteria for deciding what the best schedule should be, e.g. minimize C_{max}.

Unary resource constraint propagation algorithms

The *temporal* constraint (or precedence constraint) enforces bounds consistency on ordered pairs of sequential tasks. For an ordered pair of tasks t_i, t_{i+1} on a job, propagation of the temporal constraint enforces that

$$st_i + p_i \leq st_{i+1}. \tag{7.1}$$

The *disjunctive* constraint handles pairs of tasks (t_i t_j) sharing a resource (or sharing a job for OSPs). Since two tasks cannot overlap, either t_i finishes before t_j starts or vice versa:

$$(st_i + p_i \leq st_j) \vee (st_j + p_j \leq st_i). \tag{7.2}$$

The *time-table* propagation rule (Pape [161]) identifies time periods for which a resource *must* be used by a task. It computes the required resource usage for each time point t, maintaining a set of 0-1 variables $X(t_i, t)$ which takes the value 1 *iff* t_i must use the resource at time point t. Since the resources are unary, the following constraint is added:

$$\sum_{i=1}^{n} X(t_i, t) = 1 \forall t \in T. \tag{7.3}$$

where T is the set of time periods, T=(0,lft_{M_y}).

For example, if we have a task t_i with $p_i = 7$ and domain [0...3], then its earliest possible processing time would require the resource in the time interval [0 6] and its latest possible processing time would require the resource in the time interval [3 9]. Thus this task will always require the resource in the time interval [3 6], the intersection of the earliest and latest processing time intervals, no matter what its starting time. The earliest possible start time of all other tasks on this resource, with $p_i > 4$, must then be 6 (since they cannot be processed before this

task), and their domains are updated accordingly.

The previous two propagation methods involve simple reasoning. We now describe some of the more complicated filtering techniques that have been proposed for the unary resource constraint. One of the most popular filtering techniques for the unary resource constraint is known as *Edge-Finding* (Carlier and Pinson [40], Nuijten [159]).

Let T denote a set of tasks sharing a unary resource, Ω denote a subset of T, and let $t_i \ll \Omega$ (respectively $t_i \gg \Omega$) denote that t_i must start before (*resp.* after) the set of tasks Ω, for $t_i \notin \Omega$. Edge-Finding involves detecting that a task must be scheduled first (Eqn 7.4) or last (Eqn 7.5) in the set of tasks $(\Omega \cup t_i)$. Equations 7.6 and 7.7 illustrate how the variable st_i is updated:

$$\forall \Omega, \forall t_i \notin \Omega, [(lft_{\Omega \cup t_i} - est_{\Omega}) < (p_{\Omega \cup t_i})] \;\Rightarrow\; t_i \ll \Omega \tag{7.4}$$

$$\forall \Omega, \forall t_i \notin \Omega, [(lft_{\Omega} - est_{\Omega \cup t_i}) < (p_{\Omega \cup t_i})] \;\Rightarrow\; t_i \gg \Omega \tag{7.5}$$

$$t_i \ll \Omega \;\;\Rightarrow\;\; st_i + p_i \;\leq\; \min_{\emptyset \neq \Omega' \subseteq \Omega} (lft_{\Omega'} - p_{\Omega'}) \tag{7.6}$$

$$t_i \gg \Omega \;\;\Rightarrow\;\; st_i \;\geq\; \max_{\emptyset \neq \Omega' \subseteq \Omega} (est_{\Omega'} + p_{\Omega'}) \tag{7.7}$$

The complement to the above is the "Not First, Not Last" filtering technique which detects that a task $t_i \notin \Omega$ cannot be scheduled first or last in the set of tasks $\Omega \cup t_i$, in which case the domain of t_i is updated accordingly (Baptiste and Pape [16], Torres and Lopez [200]).

The *Shaving* filtering technique (Carlier and Pinson [41], Martin and Shmoys [145]) updates the task time windows by assessing the earliest and latest start times. For each unassigned task t_i, at each node, it temporarily assigns t_i a starting time (either est_i or lst_i) and propagates the assignment using the filtering algorithms (Edge-Finding, etc). If this results in a failure, it updates the relevant domain bound. The process iterates until a fixed point is reached. This can be viewed as a form of Singleton Bounds Consistency.

There are many variations on the above constraints, as well as further filtering techniques such as the *balance* constraint (Laborie [126]), which are beyond the scope of this dissertation. The reader is pointed to [17, 126] for further details on

inference techniques for constraint based scheduling, and the dissertation of Vilím [210] for further details and improvements to unary resource filtering techniques in particular.

Variable and Value Ordering Heuristics

Instead of searching by assigning a task a starting time on a left branch, and forbidding this value on the right branch, it is common to branch on *precedences*. An unresolved pair of tasks t_i, t_j is selected and the constraint $st_i + p_i \leq st_j$ is posted on the left branch whilst $st_j + p_j \leq st_i$ is posted on the right branch.

Most heuristics are based on the identification of critical variables. The branching scheme *Profile* introduced in Beck et al. [23] (which is an extension of the *ORR/FSS* heuristic (Sadeh and Fox [176])) involves selecting a pair of *critical* tasks sharing the same unary resource and ordering them by posting a precedence constraint on the left branch. The criticality is based on two texture measurements, *contention* and *reliance* (Sadeh [175]). Contention is the extent to which tasks compete for the same resource over the same time interval, reliance is the extent to which a task must use the resource for given time points (e.g. in the above example for the time-table constraint, t_i relies on its resource in the time interval [3 6]).

The heuristic determines the most constrained resources and tasks. For each task on each resource, the probability of the task requiring the resource is calculated for each time period. This probabilistic profile is referred to as the individual demand curve. The contention is based on the aggregated demand over all tasks on the resource. At each node, the resource and the time point with the maximum contention are identified by the heuristic, then a pair of tasks that rely most on this resource at this time point are selected (provided the two tasks are not already connected by a path of temporal constraints).

Once the pair of tasks has been chosen, the order of the precedence has to be decided. For that purpose, a number of randomized value ordering heuristics have also been proposed [23], such as the centroid heuristic. The centroid of a task on a resource is based on the individual demand curve for the task on the resource. and is computed for the two critical tasks. The centroid of a task is

the point that divides its probabilistic profile equally. If the centroids are at the same position, a random ordering is chosen. A similar contention based approach has been proposed by Laborie, based on the detection and resolution of *minimal critical sets (MCS)* [127].

An alternative measurement of criticality was proposed by Smith and Cheng [190]. They used the "slack" between two tasks sharing a resource to identify the pair of critical tasks. For all pairs of tasks sharing a resource the $Bslack$ was calculated as follows:

$$slack(t_i \rightarrow t_j) = lft_j - est_i - (p_i + p_j) \tag{7.8}$$

$$S = \frac{\min(slack(t_i \rightarrow t_j), slack(t_j \rightarrow t_i))}{\max(slack(t_i \rightarrow t_j), slack(t_j \rightarrow t_i))} \tag{7.9}$$

$$Bslack(t_i \rightarrow t_j) = \frac{slack(t_i \rightarrow t_j)}{\sqrt{S}} \tag{7.10}$$

The pair of tasks with the smallest $Bslack$ is chosen and ordered so as to leave the most slack.

7.3 Light Weighted Model (LW)

The CP model introduced in the previous section can be viewed as a reasoning-intensive model. Our model is more search intensive, with little in the way of domain-specific propagators/heuristics. We mainly concern ourselves with the objective of minimizing C_{max}. The start time of each task t_i is represented by a variable $st_i \in [0, \ldots, (C_{max} - p_i)]$.

We use an improved version of the Mistral default model described in the previous chapter (Section 6.4). For each pair of tasks sharing a resource, we introduce a Boolean variable b_{ij} which represents the relative ordering between t_i and t_j. A value of 0 for b_{ij} means that task t_i precedes task t_j, whilst a value of 1 stands for the opposite ordering. The variables st_i, st_j and b_{ij} are linked by the following constraint:

$$b_{ij} = \begin{cases} 0 \Leftrightarrow st_i + p_i \leq st_j \\ 1 \Leftrightarrow st_j + p_j \leq st_i \end{cases}$$

Bounds consistency (BC) is maintained on these constraints (note that bounds consistency is equivalent to arc consistency for these constraints). Here, the scope of the constraint involves three variables, therefore BC can be achieved in constant time for a single constraint, by applying simple rules. For n jobs and m machines, this model involves $nm(n-1)/2$ Boolean variables (and as many ternary disjunctive constraints) for the job shop scheduling problem. Using an AC3 type constraint queue, the worst case time complexity for achieving bounds consistency on the whole network is therefore $O(n^2 m C_{max})$. However, it rarely reaches this bound in practice.

Note that this model follows the disjunctive graph representation: we include a precedence constraint for each edge in D and a disjunctive constraint for each edge in A. We do not use information about the maximal paths in D and maximal cliques in A (which define a resource), unlike the "heavy" model with global constraints such as Edge-Finding.

It has been observed (for instance in Meiri [149]) that the existence of a partial ordering of the tasks (compatible with start times and durations, and such that its projection on any job or machine is a total order) is equivalent to the existence of a solution. In other words, if we successfully assign all Boolean variables in our model, the existence of a solution is guaranteed. This is equivalent to replacing all the bidirectional arcs with directed arcs in the disjunctive graph representation.

7.3.1 Variable Ordering

We use the domain/weighted-degree heuristic [29]. However, at the start of the search, this heuristic is completely uninformed since every Boolean variable has the same domain size and the same degree (i.e. 1). We therefore used the domain size of the two tasks t_i, t_j associated to every disjunct b_{ij} to alleviate this issue. The domain size of task t_i (denoted $dom(t_i)$) is the number of possible starting times of t_i, i.e. $dom(t_i) = (lst_i - est_i + 1)$

With regard to the weighted component of the heuristic, there is a number of ways to incorporate failure information. We focused on the following two methods. In the first, we simply use the weight on the Boolean variable (denoted $w(i,j)$), i.e. the number of times the search failed while propagating the constraint

between st_i, st_j and b_{ij}.

The heuristic then chooses the variable minimizing the sum of the tasks' domain size divided by the weighted degree:

$$\frac{dom(t_i) + dom(t_j)}{w(i,j)} \tag{7.11}$$

Our second method uses the weighted degree associated with the task variables instead of the Boolean variable. Let $\Gamma(t_i)$ denote the set of tasks sharing a resource with t_i. We call $w(t_i) = \sum_{t_j \in \Gamma(t_i)} w(i,j)$ the sum of the weights of every ternary disjunctive constraint involving t_i. Now we can define an alternative variable ordering as follows:

$$\frac{dom(t_i) + dom(t_j)}{w(t_i) + w(t_j)} \tag{7.12}$$

We refer to these heuristics as $Tdom/Bwt$ and $Tdom/Twt$, where $tdom$ is the sum of the domain sizes of the tasks associated with the Boolean variable, and bwt (twt) is the weighted degree of the Boolean (associated tasks *resp.*). Ties were broken randomly.

It is important to stress that the behaviour of the weighted degree heuristic is dependent on the modelling choices. Indeed two different, yet logically equivalent, sets of constraints may distribute the weights differently. The relative light weight of our model allows the search engine to explore many more nodes, thus quickly accruing information in the form of constraint weights.

7.3.2 Value Ordering

Our value ordering is based on the solution guided method (SGMPCS) proposed by Beck for JSPs [21]. This approach uses previous solutions as guidance for the current search, intensifying search around a previous solution in a similar manner to the tabu search algorithm of Nowicki and Smutnicki [158]. In SGMPCS, a set of elite solutions is initially generated. Then, at the start of each search attempt, a solution is randomly chosen from the set and is used as a value ordering heuristic. When an improving solution is found, it replaces the solution in the elite set that

was used for guidance. (A similar solution guided approach, known as "progress saving", has recently been proposed in SAT [162].)

The logic behind this approach is its combination of intensification (through solution guidance) and diversification (through maintaining a set of diverse solutions). Note that this is a generic technique that can be applied to both COPs and CSPs (Heckman [104]). In the latter case, partial solutions are stored and used to guide subsequent search.

Interestingly Beck found that the intensification aspect was more important than diversification for solving JSPs. Indeed, for the instances studied, there was little difference in performance between an elite set of size 1 and larger elite sets (although too large a set did result in a deterioration in performance). We use an elite set of 1 for our approach, i.e. once an initial solution has been found this solution is used, and updated, throughout our search.

Furthermore, up until the first solution is found, we use a value ordering working on the principle of best *promise* [71]. The value zero for b_{ij} is visited first *iff* the domain reduction directly induced by the corresponding precedence ($st_i + p_i \leq st_j$) is less than that of the opposite precedence ($st_j + p_j \leq st_i$). We use a static value ordering heuristic for breaking ties, based on the tasks' relative position in their jobs. For example, if t_j is the fourth task in its job and t_i is the sixth task in its job, then the value zero for b_{ij} sets the precedence $st_j + p_j \leq st_i$.

7.3.3 Additional Features

In the previous chapter we showed the efficiency gained by combining the weighted degree heuristic with a restarting strategy for these types of problem. Here, we use a geometric restarting strategy (Walsh [221]) with random tie-breaking for the variable heuristic. The geometric strategy is of the form $s, sr, sr^2, sr^3, \ldots$ where s is the base and r is the multiplicative factor. In our experiments the base was 256 failures and the multiplicative factor was 1.3.

We also incorporate the nogood recording from restarts strategy of Lecoutre et al. [135], where nogoods are generated from the final search state when the cutoff has been reached. In other words, a nogood is stored for each right branch on the path from the root to the search node at which the cutoff was reached.

7.3.4 Search Strategy

There are two main phases to our approach (Algorithm 4), the first involves a dichotomic search phase (lines 8-17), which is followed by branch-and-bound search if optimality hasn't been proven (line 22). Initially the lower bound on C_{max} is set to the duration of the longest job/machine, while the upper bound ub is initialized by a greedy algorithm in most cases (in one case it is initialized by simply summing the durations of every task).

Furthermore, we observed that there was a large improvement in search performance when guided by a solution, even one of relatively poor quality. Thus we added an initial step prior to the dichotomic phase, where the C_{max} was set to the initial upperbound and search was run for approximately 3 seconds in order to quickly initialize the solution (line 3 of Algorithm 4). Since these initial bounds are often quite weak, the dichotomic search allows us to quickly reduce the gap between lb and ub.

The function `Solve` handles the scheduling instance as a decision problem with a latest starting time of $(C_{max} - p_i)$ for all t_i, where C_{max} is the current upperbound on the objective. This function returns two values. The first is the final state: "solution", "Limit reached", "no solution". The second value returned is the solution if the problem was solved. Note that when the decision problem is solved, the solution may have a better objective value than the C_{max} for the decision problem. Hence the objective value for the solution is calculated (lines 5 and 12).

In the dichotomic search phase, the decision problem is repeatedly solved with C_{max} fixed to $\frac{ub+lb}{2}$, updating lb and ub accordingly, until they have collapsed. Each dichotomic step has a fixed cutoff, if the problem is unsolved the lb is updated, although not stored as the best proven lb.

In order to make the results reproducible, we implemented the cutoff for each dichotomic step in terms of propagations since the vast majority of constraints are the same (ternary disjuncts). The observed propagation speed for a subset of problems was approximately 20M/s, the cutoff was then calculated by multiplying this by 30 (as this would generally result in a time cutoff of 30 seconds).

If the problem has not been solved to optimality during the dichotomic search,

Algorithm 4: *LW_Solve*

Input : Scheduling problem P, variable heuristic varH, value heuristic valH, initial cutoff, dichotomic search step cutoff, and overall cutoff

Output: Integer `bestMkp` **and** Boolean `optimal`

```
1  runtime ← 0.0, bestLb ← 0
2  Initialize lb and ub
   // Try solving with ub and very short cutoff
   // to initialize solution store
3  ⟨result,solution⟩ ← Solve (P,ub,initCutoff,varH, valH)
4  if result = SolutionFound then
5      Set ub to objective value of solution
6      Store solution for value ordering
7  bestMkp ← ub
   // Dichotomic Search
8  while lb < ub do
9      mkp ← ⌊(lb+ub)/2⌋
10     ⟨result,solution⟩ ← Solve (P,mkp,dsCutoff,varH, valH)
11     if result = SolutionFound then
12         Set ub to objective value of solution
13         Store solution for value ordering
14     else
15         lb←mkp+1
16         if result ≠ LimitReached then
17             bestLb=mkp+1
18 if bestLb = ub then // Optimal solution found
19     return (ub, true)
20 else
       // Branch and Bound Search
21     bnbCutoff ← (totalCutoff- runtime)
22     result ← BnbSolve (P,ub, bestLb, bnbCutoff,varH, valH)
23 return result
```

we perform a branch and bound search (`BnbSolve`) with the best C_{max} from the dichotmic search as our upper bound, and the best proven lb as our lower bound. Branch and bound search is performed until either optimality of a solution is proven or an overall cutoff is reached. The function returns the best objective value found and a Boolean value for whether optimality was proven or not.

7.4 Open Shop Scheduling

7.4.1 Problem Description

An $n \times m$ Open Shop Scheduling Problem (OSP) involves scheduling a set of n jobs $\mathcal{J}$, each consisting of m tasks, to be processed on a set of m machines $\mathcal{M}$, and is written as ($O \mid\mid \boldsymbol{C_{max}}$) in the Graham three-field notation. Each task of a job has an associated duration (or processing time), p_i, and is associated with a different machine, i.e. each machine processes exactly one task in each job. This was shown to be NP-hard for $m \geq 3$ in Gonzales and Sahni [84].

More formally, our problem can be defined as:

$$minimize\ C_{max}$$

subject to

$$C_{max} \geq st_i + p_i \qquad \forall t_i \in \mathcal{T} \tag{7.13}$$

$$(st_i + p_i \leq st_j) \ \vee \ (st_j + p_j \leq st_i) \qquad \forall J_x \in \mathcal{J},\ t_i \neq t_j \in J_x \tag{7.14}$$

$$(st_i + p_i \leq st_j) \ \vee \ (st_j + p_j \leq st_i) \qquad \forall M_y \in \mathcal{M},\ t_i \neq t_j \in M_y \tag{7.15}$$

Figure 7.2 presents a Gantt chart of the optimal solution to the sample JSP introduced in Table 7.2 converted to open shop by replacing precedence constraints with disjunctive constraints in the jobs. The optimal value for C_{max} is 123. Note that a symmetrical solution can be found by setting the start time of each task t_i to (C_{max} - t_i - p_i).

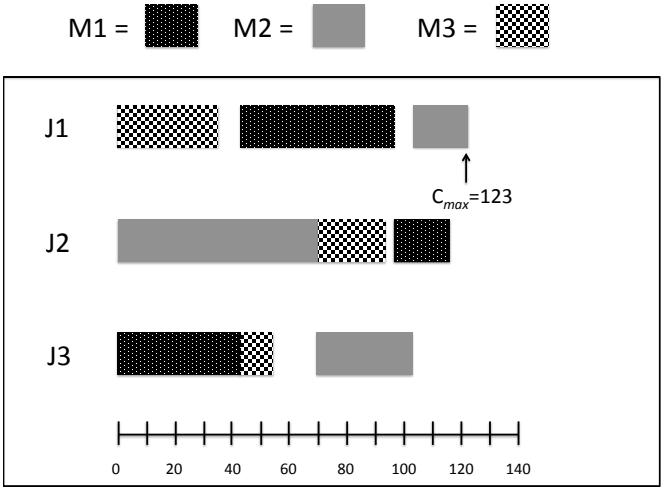


Figure 7.2: Optimal solution to sample 3×3 OSP.

7.4.2 State of the art

We compared our model with the CP model of Malapert et al. [142] (denoted RRCP-OSP), implemented in Choco [46], which has recently been shown to be state-of-the-art on these problems [142]. It outperformed both metaheuristic approaches, such as particle swarm optimization [185] and ant-colony optimization [28]; and exact methods, such as the branch and bound methods of Dorndorf et al. [59] and Gueret et al. [96], and the method of Laborie [127].

The approach combines a number of traditional filtering techniques for the unary resource global constraint (Edge-Finder; Not-First, Not-Last) with a specialized filtering method for OSPs, *Forbidden Intervals* [94] which are time intervals in which no task can start or finish in an optimal solution. Furthermore, a precedence constraint network is used to improve efficiency on propagation of temporal constraints, using previous techniques developed for the Simple Temporal Problem (Dechter et al. [57]).

Symmetry breaking was added to remove the solution symmetries of the OSP (simply reversing the order of the tasks on each job and machine in a solution is also a solution). To achieve this, the task with the longest processing time was selected and forced to start in the first half of the schedule, i.e. the constraint

$st_i \leq \frac{C_{max}-p_i}{2}$ was added, where task t_i has the longest processing time. The branching scheme was the *Profile* heuristic described earlier (with the precedence decided by the *centroid* heuristic).

However, Malapert et al. found that both the propagation methods and the heuristics perform poorly when the upper bound on C_{max} is very far from the optimal. To counteract this, a randomized constructive heuristic was used, without propagation, to generate a good initial upper bound. They used a common priority dispatching rule (the longest processing time) to construct a nondelay schedule by repeatedly appending tasks to a schedule.

Finally, the best restarting scheme for the approach of Malapert et al. was found to be Luby [140] of order 3 (i.e. all run lengths are powers of 2 as normal, but each time a *triplet* (instead of a pair) of runs of a given length are completed, a run of twice that length is performed). Nogood recording form restarts [135], as described earlier, was also incorporated.

7.4.3 Implementation of our model

There is little alteration required for our basic model for these problems, other than to consider each job as a resource. For n jobs and m machines, we have $m*(m-1)/2$ Boolean variables per job and $n*(n-1)/2$ Boolean variables per machine, so overall our OSP model involves $nm(m+n-2)/2$ Boolean variables and as many ternary constraints.

7.4.4 Experimental Evaluation

All experiments reported in this chapter were again run on an Intel Xeon 2.66GHz machine with 12GB of RAM on Fedora 9, unless otherwise stated. Each algorithm run on an instance had an overall time limit of 3600s. There were 10 runs per instance. The heuristic used in our algorithm was $Tdom/Bwt$.

Benchmarks

There are three sets of instances which are widely studied in the literature, Taillard [196] (60 instances, denoted *tai-**), Brucker [35] (52 instances, denoted *j-**), and

Guéret-Prins [95] (80 instances, denoted *gp-**). All instances involve "square" problems, i.e. where the number of jobs and machines are equal. The instances range in size from 3×3 to 20x20, with 192 instances overall.

The Taillard instances were randomly generated, and the ten most difficult, based on the performance of a tabu search approach of Taillard, for each pair of n, m were selected [196]. Although this problem set contains the largest instances (400 tasks for the 20x20 set compared to 100 (64) tasks for the largest Gueret-Prins (Brucker *resp.*) instances), the Taillard instances are considered to be relatively easy [95]. This is because there is generally no proof of optimality required, the basic lower bound (LB_{basic}, the maximum job/machine duration over all jobs and machines) is the optimal value for all instances with $n, m > 5$.

In response to observations regarding the easiness of the Taillard instances, Brucker et al. proposed instances which were generated so as to be much more difficult to solve. They defined the *workload* of an instance to be the average workload on its machines:

$$Workload = \frac{total\ processing\ time}{m * LB_{basic}}.$$

They stated that generating instances with a workload close to 1 would be difficult to solve, as this would imply that the processing times of all jobs and machines are in a small range. Thus, there would be little chance of finding a solution with C_{max} near LB_{basic}.

Gueret and Prins proposed an improved method of finding a lower bound for OSPs [95]. Based on this, they developed a problem generator which produced instances with a large distance between LB_{basic} and their new lower bound, and by extension the optimal value of C_{max}.

Forty of these OSPs remained open up until recently, when Laborie closed the remaining 34 open Gueret-Prins instances and 3 of the 6 open Brucker instances [127] in 2005. The final 3 open Brucker instances were closed by Tamura et al [197] in 2006 (although 2 of these were solved by dividing the problem into 120 subproblems, which were solved running on 10 machines in parallel, taking 6 and 13 hours respectively).

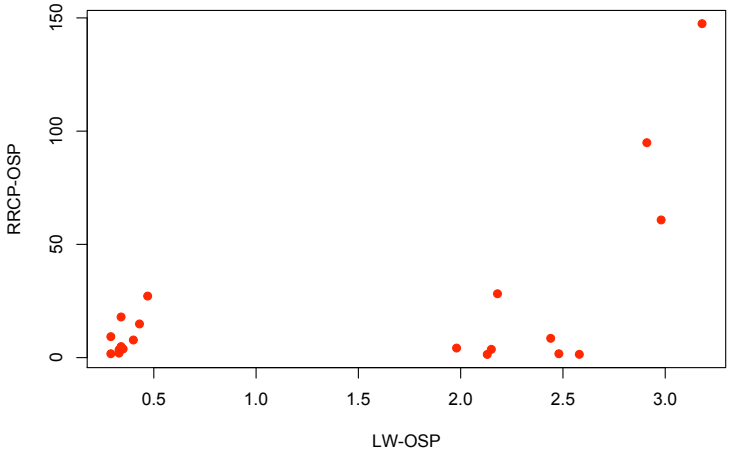


Figure 7.3: Average runtimes on Taillard instances.

Results

We first compare the Mistral implementation of LW-OSP with RRCP-OSP. Both proved optimality within the 1 hour cutoff on every instance, so we present our results in terms of average time taken per instance over the 10 runs. Results on the Taillard instances are given in Figure 7.3. Although both methods were extremely efficient on these instances, LW-OSP solved all in under 4 seconds (note the difference in scale for the two axes), while RRCP-OSP took over a minute to solve some of the 20x20 instances.

These results clearly show that, contrary to popular belief, scheduling specific filtering techniques and heuristics are not necessary for solving these problems. However it may be the case that, when a proof of optimality is required, the stronger inference techniques of RRCP-OSP are advantageous. Thus the Gueret-Prins and Brucker instances are of more interest.

Results for the Gueret-Prints instances are shown in Figure 7.4. These instances were generated to be hard to solve, and have proven difficult for metaheuristic approaches [185]. However, exact methods have been much more successful, with the MCS method of Laborie closing the 34 open problems in under 5 seconds each [127]. Our LW-OSP method solved all instances in less than a

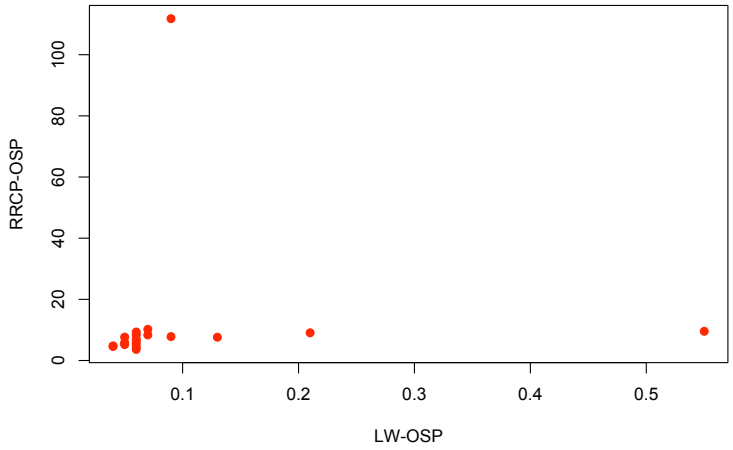


Figure 7.4: Average runtimes on Gueret-Prins instances.

second, while RRCP-OSP was slower on the larger instances.

The Brucker instances are generally the most difficult, as shown in Figure 7.5 (for clarity we have also plotted the line $x = y$). Indeed, RRCP-OSP was the first exact method to solve the 2 most difficult instances without dividing the problem into subproblems and running in parallel (as done in Tamura et al. [197]). However, LW-OSP is still consistently faster than RRCP-OSP on the hard instances. These results show that our method is efficient both at solving problems quickly and proving optimality.

In order for a more direct comparison, Arnaud Malapert implemented LW-OSP in Choco with the following alterations: the value ordering was lexical throughout; and the dichotomic search phase was replaced by the same randomized constructive heuristic used to initialize the upper bound in RRCP-OSP. Therefore both methods start the branch-and-bound phase with the same bounds on C_{max}.

Figure 7.6(a) is a log-log plot of the average time taken by the two Choco models to solve each of the 192 open shop instances, while Figure 7.6(b) is a log-log plot of the average nodes explored by the two models. The light model is generally faster on the harder instances, even though (as expected) it explores more nodes.

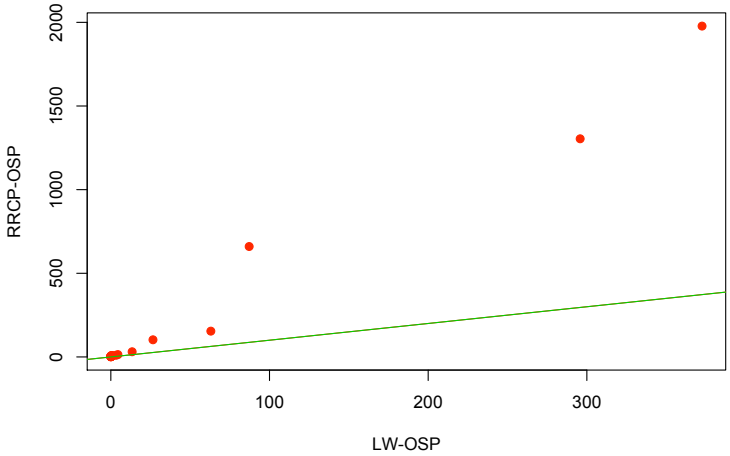
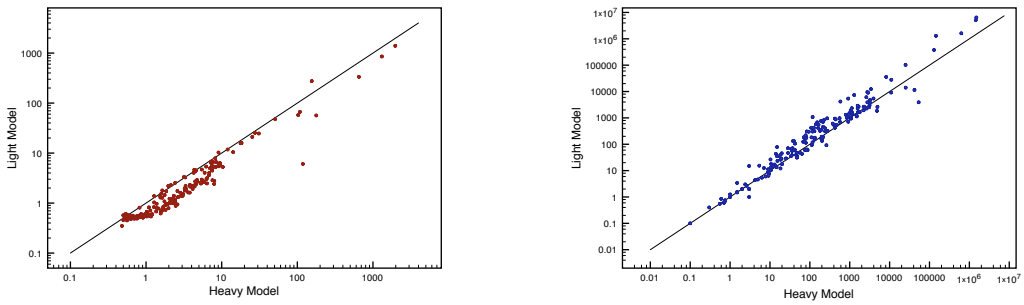


Figure 7.5: Average runtimes on Brucker instances.



(a) Time Comparison (s)

(b) Node Comparison

Figure 7.6: Log-log plots of Light vs Heavy Choco models on open shop instances.

We selected a subset of 11 of the hardest instances based on the overall results of Figure 7.6(a), choosing instances for which at least one approach took over 20 seconds on average to solve. This subset consisted of one Gueret-Prins, four Taillard and six Brucker instances, of order 10, 20 and 7-8 respectively.

In Table 7.3 we compare LW-OSP implemented in Choco ("Light-dw") with the RRCP-OSP model ("Heavy-prof") on this hard subset. In order to better understand the importance of the heuristics to the two models, we ran the same experiments but with the heuristics swapped (algorithms "Light-prof" and "Heavy-dw" in Table 7.3).

The results are presented in terms of average time and average nodes (where a model failed to prove optimality its time is taken as 3600s, in these cases the average time is a lower bound). For reference, we note that the Mistral implementation of LW-OSP averaged 79s on these instances, while the same model, but with lexical value ordering, averaged 93s. We see that, similarly to the results in Figure 7.6, Light-dw is generally much faster than Heavy-prof (only slower on 1 of the 11 instances), even though it explores many more nodes.

Table 7.3: Results (Time) For Hard Open Shop Scheduling Problems

	Light-dw		Heavy-prof		Light-prof		Heavy-dw	
	$\bar{t}$	$\bar{n}$	$\bar{t}$	$\bar{n}$	$\bar{t}$	$\bar{n}$	$\bar{t}$	$\bar{n}$
gp-10-1	6.1	4K	118.4	53K	2523.2	6131K	9.6	3K
j7-0-0	854.5	5.1M	1310.9	1.4M	979.1	4.3M	3326.7	2.6M
j7-10-2	57.6	0.4M	102.7	0.1M	89.5	0.4M	109.6	0.1M
j8-0-1	1397.1	6.4M	1973.8	1.5M	1729.3	6.0M	3600	2.4M
j8-10-0	24.6	0.10M	30.9	0.02M	19.7	0.05M	68	0.06M
j8-10-1	275.2	1.3M	154.8	0.1M	92.8	0.3M	796.7	0.7M
j8-10-2	335.3	1.6M	651.4	0.6M	754	2.7M	697.1	0.6M
tai-20-1	25.4	2K	27.5	3K	2524.7	1458K	34.4	3K
tai-20-2	56.5	11K	178	42K	3600	2261K	81.9	10K
tai-20-7	47.8	9K	50.9	11K	3600	2083K	63.7	8K
tai-20-8	66.8	14K	108.3	25K	3600	2127K	84.8	11K
Average	286.1	1.3M	428.9	0.4M	1774.3	2.5M	806.6	0.6M

Average runtime in seconds ($\bar{t}$), and average nodes explored ($\bar{n}$)

Interestingly, on the hardest instances (6 Brucker instances), we observed that

Heavy-prof was slightly quicker on average than Light-dw at finding the optimal solution (310s vs 360s), but was 3 times slower at proving optimality once the solution had been found (385s vs 125s). This may be somewhat surprising since, as mentioned earlier, one would expect that the stronger inference techniques would be more beneficial when a proof of optimality is required.

With regard to the swapping of heuristics, $Tdom/Bwt$ is clearly more effective for the light model than $Profile$. Indeed combining Profile with the light model failed to prove optimality on 30% of the instances and was 8 times slower on average.

For the heavy model, $Tdom/Bwt$ is slower on average than $Profile$. This is mainly due to poor performance on the Brucker instances, although it still proved optimality on 92% of the runs. This is surprising given that bwt is much less discriminatory with the heavy model as each constraint has n variables associated with it. When a failure occurs all variables in that constraint will have their weight increased.

Overall, we have shown that domain-specific filtering techniques and heuristics are not necessary for solving open shop scheduling problems. Our approach of combining a number of generic CSP solving techniques with simple bounds consistency is extremely efficient for these problems.

7.5 Job Shop Scheduling

7.5.1 Problem Description

A Job Shop Scheduling Problem (JSP) is identical to an OSP with the exception that there is a predefined ordering on the tasks of each job, i.e. a task in a job cannot start until the preceding task in the order has finished. In the Graham three-field notation, it is written as $(\mathcal{J}| \mid \boldsymbol{C_{max}})$. The problem is known to be NP-hard for $m \geq 3$ (Garey et al. [70]).

The formal definition can be stated as follows:

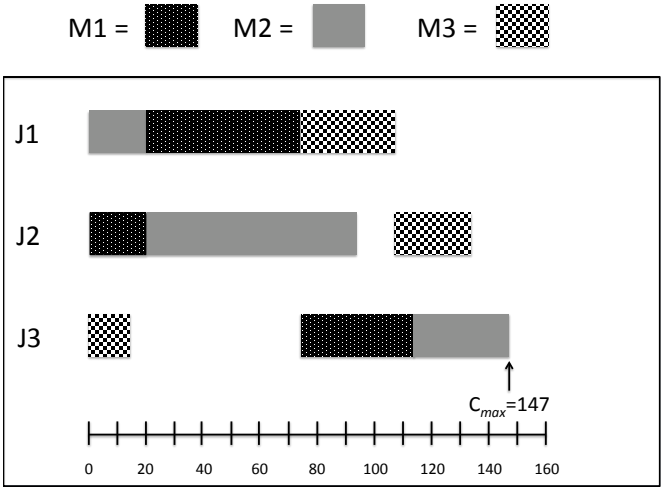


Figure 7.7: Optimal solution to sample 3×3 JSP.

$$(JSP)\ \ minimize\, C_{max}$$

subject to

$$C_{max} \geq st_i + p_i \qquad \forall t_i \in \mathcal{T} \tag{7.16}$$

$$st_i + p_i \leq st_{i+1} \qquad \forall J_x \in \mathcal{J},\ \forall t_i, t_{i+1} \in J_x \tag{7.17}$$

$$(st_i + p_i \leq st_j)\ \vee\ (st_j + p_j \leq st_i) \qquad \forall M_y \in \mathcal{M},\ t_i \neq t_j \in M_y \tag{7.18}$$

In Figure 7.7, we show the optimal solution for the sample JSP introduced in Table 7.2 (this is the same solution as shown in disjunctive graph form in Figure 7.1). The precedence constraints on the tasks in a job result in a larger optimal C_{max} than for the associated OSP. Indeed, the optimal C_{max} to a JSP can at best match the optimal C_{max} for the associated OSP since any solution to a JSP is also a solution to the OSP while the opposite does not hold.

7.5.2 State of the art

Metaheuristic approaches have generally been the most efficient at solving JSPs, in particular the tabu search method of Nowicki and Smutnicki's i-TSAB [158], and the tabu search / simulated annealing hybrid of Zhang et al. [238]. Indeed, the former produced such impressive results that research was carried out independently on why this form of tabu search was so suited to JSPs (Watson et al. [226]). The most competitive exact methods are the i-TSAB/SGMPCS approach of Watson and Beck [224] (hybridizing tabu search and CP) and, from a pure CP point of view, the solution guided multi point constructive search method (SGMPCS) of Beck [21].

The i-TSAB method has three main components as outlined in Watson and Beck [224]: the $N5$ critical-path based move operator (Nowicki and Smutnicki [157]); search intensification through storing and reusing a set of *elite* or high-quality solutions (Nowicki and Smutnicki [157]); and search diversification through the selection of a solution e', a solution equidistant from two randomly selected elite solutions e_i and e_j (by a process known as *path relinking*, Nowicki and Smutnicki [158]). When an elite solution e_i is improved, the new solution replaces the old solution.

The Zhang method differs from i-TSAB in the following ways. Firstly, simulated annealing is used to find the elite solutions, and this provides the diversification. Secondly, the move operator used is the $N6$ neighborhood structure based move operator of Balas and Vazacopoulos [13].

The SGMPCS approach of Beck also uses an elite solution set for guidance, and indeed was inspired by i-TSAB. The set is initialized with a number of randomly generated solutions to provide diversification. Restarted search is then performed, where at each restart an elite solution is selected with probability p. This is used to guide the value ordering until either an improved solution is found or the cutoff is reached. For the former, the guiding solution is replaced with the improved solution. If an elite solution wasn't used for guidance, and a solution of better quality than at least one of the elite solutions is found, then this solution replaces the worst solution in the elite solution set.

The variable ordering heuristic used in SGMPCS is the $Profile$ heuristic de-

scribed earlier (referred to as $Texture$ in [21]). Randomization is added by randomly selecting with uniform probability from the top 10% most critical pairs of $(machine, timepoint)$. Finally, the standard constraint propagation techniques for scheduling are used, such as time-table [161], Edge-Finding [159], and balance constraints [126].

7.5.3 Implementation of our model

For n jobs and m machines, our model involves $nm(n-1)/2$ Boolean variables and as many ternary constraints. The model is as described earlier in Section 7.3.

7.5.4 Experimental Evaluation

We compare our approach with the SGMPCS approach of Beck, for which the code was kindly supplied by Chris Beck. Each approach was run 10 times with a time limit of one hour per instance. SGMPCS was run using Ilog Scheduler 6.2. Due to the number of benchmarks our primary method of comparison is the average percentage relative deviation (*APRD*) per problem set. The *PRD* is given by

$$PRD = \frac{C_{Alg} - C_{Ref}}{C_{Ref}} * 100$$

where C_{Alg} is the makespan found by *Alg* and C_{Ref} is the best known makespan for the instance. In most cases this was either taken from Zhang et al. [238], or where optimality was proven. However, for the set of five instances swv11-15, the best known makespan is taken from the best found by the two methods tested. Finally, the heuristic used in our algorithm was $Tdom/Twt$.

Benchmarks

There are a large number of benchmarks in the literature, stretching back to the 3 ft-$*$ instances proposed by Fisher and Thompson in 1963 [63]. The other benchmarks we consider here are: the 40 Lawrence instances [130] (denoted la-$*$), 5 instances proposed by Adams et al. [2] (denoted abz-$*$), 10 instances proposed by Applegate and Cook [4] (denoted orb-$*$), 4 instances proposed by Yamada

Table 7.4: APRDs on JSPs: LW-JSP versus SGMPCS

Problem Set	LW-JSP		SGMPCS	
	Min	Avg	Min	Avg
abz	**0.360**	0.839	0.840	1.347
ft	**0**	0.157	**0**	**0**
la01-20	**0**	**0**	**0**	**0**
la11-20	**0**	**0**	**0**	**0**
la21-30	0.153	0.292	**0**	0.078
la31-40	**0**	**0**	**0**	**0**
orb	**0**	**0**	**0**	**0**
swv1-10	2.706	4.272	**1.718**	2.99
swv11-15	5.950	8.459	**0**	2.977
swv15-20	**0**	**0**	**0**	**0**
yn	**0.738**	1.670	0.793	1.589
tai11-20	0.637	1.661	**0.228**	1.111
tai21-30	**0.419**	1.056	0.523	1.159
tai31-40	2.468	3.716	**0.687**	1.438
tai41-50	3.588	4.915	**1.882**	3.015
Total Average	1.135	1.803	**0.445**	1.047

Notes: C_{Ref} taken from Zhang et al. [238] (or known optimal values), except for swv11-15 where C_{Ref}=min(LW-JSP, SGMPCS).

and Nakano [234] (denoted yn-$*$), 20 instances proposed by Storer et al. [195] (denoted swv-$*$), and finally 40 of the Taillard instances [196] (denoted tai-$*$). Instances range in size from 6x6 to 50x10.

Results

The results are given in Table 7.4. We see that LW-JSP outperformed SGMPCS on three of the sets (abz, yn, tai21-30). Although our approach was generally competitive on the other problem sets, it is clear from the total averages that SGMPCS was best. Indeed the average of the SGMPCS runs was better than the best found with our method. This was mainly due to the swv instances 1-15 and Taillard in-

stances 31-50, which were the only cases where the APRD of the best solutions found by Mistral was greater than 1.

We further compare the approaches in terms of number of instances (out of the 182 instances tested) where optimality was proven. LW-JSP solved 56 instances to optimality on at least one run, while SGMPCS solved four more instances to optimality on at least one run. Only two of these instances, for both methods, were not solved on every run.

Similar behavior was observed for the number of instances for which a method matched the best solution. LW-JSP matched the best solution on at least one run on 58 instances, whereas SGMPCS matched the best known solution on 61 of the instances on at least one run. Again, the methods had fairly consistent performance across runs, with the best known solution not matched on every run on only three of these instances for both methods.

7.6 Job Shop Scheduling with Sequence Dependent Setup Times

7.6.1 Problem Description

An $n \times m$ job shop problem with sequence-dependent setup times (SDST-JSP) involves the same variables and constraints as a JSP of the same order, with the additional constraint that for each machine and each pair of tasks running on this machine, the machine needs to be setup to accommodate the new task. During this setup the machine must stand idle. In the Graham three-field notation, it is written as $(\mathcal{J}|s_{i,j}|\ \boldsymbol{C_{max}})$.

The duration of the setup depends on the sequence of tasks, that is, for every pair of tasks (t_i, t_j) running on the same machine we are given the setup time $s_{i,j}$ for t_j following t_i and the setup time $s_{j,i}$ for t_i following t_j.

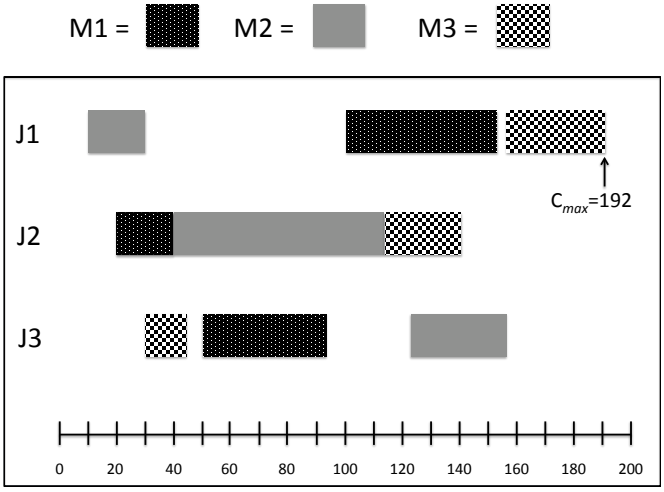


Figure 7.8: Optimal solution to sample 3×3 JSP with sequence dependent setup times.

The objective, once again, is to minimize C_{max}. More formally:

$$(SDST - JSP) \;\; minimize\, C_{max}$$

subject to

$$C_{max} \geq st_i + p_i \qquad \forall t_i \in \mathcal{T} \tag{7.19}$$

$$st_i + p_i \leq st_{i+1} \qquad \forall J_x \in \mathcal{J},\ \forall t_i, t_{i+1} \in J_x \tag{7.20}$$

$$(st_i + p_i + s_{i,j,y} \leq st_j) \ \vee \ (st_j + p_j + s_{j,i,y} \leq st_i) \qquad \forall M_y \in \mathcal{M},\ \forall t_i, t_j \in M_y, t_i \neq t_j \tag{7.21}$$

Figure 7.8 presents a Gantt chart of the optimal solution to the sample 3×3 JSP with additional setup times. For each machine the setup times were permutations on {0, 10, 20}, while the initial setup time was taken from {10, 20, 30}. The optimal C_{max} for this problem is 192, compared to 147 for the basic JSP, due to the setup times (note the gap between tasks of the same machine).

7.6.2 State of the art

This problem presents a challenge for CP and systematic approaches in general, since traditional inference methods for disjunctive constraints (e.g. the Edge-Finding algorithm) are seriously weakened as they cannot easily take into account the setup times. The best complete techniques are two recently introduced CP approaches. The first by Artigues and Feillet [6] (denoted AF08 in Table 7.5) tries to adapt the reasoning for simple unary resources to unary resources with setup times. The approach relies on solving a traveling salesman problem (TSP) with time windows to find the shortest permutation of tasks using a dynamic programming technique, and is therefore computationally expensive.

The second CP approach (which we will refer to as W09) is that of Wolf [230]. Here they introduced a model of the SDST-JSP with time windows and work breaks. A dedicated pruning algorithm was proposed and incorporated in a branch-and-bound approach with a dichotomic search strategy on C_{max}. The branching scheme is the *demand* heuristic proposed for classical JSPs in [229], which branches based on the demand on resources.

A number of metaheuristic approaches have also been proposed. Balas *et al.* introduced a shifting bottleneck algorithm combined with guided local search "SB-GLS" [14] (denoted BSV08 in Table 7.6), where the problem is decomposed into a TSP with time windows (which is solved with dynamic programming). Hybrid genetic algorithms have been proposed by González *et al.* for this problem, firstly a hybrid GA with local search [85] and more recently GA combined with tabu search [86] (denoted GVV08 and GV09 *resp.* in Table 7.6). For both GA hybrids, the problem is modeled using the disjunctive graph representation.

7.6.3 Implementation of our model

Our model is basically identical to the job shop scheduling model. However, the setup time between two tasks is added to the duration within the disjunctive constraints. That is, given two tasks t_i and t_j sharing a machine, let $s_{i,j}$ (resp. $s_{j,i}$) be the setup time for the transition between t_i and t_j (resp. between t_j and t_i), we

replace the usual disjunctive constraint with:

$$b_{ij} = \begin{cases} 0 \Leftrightarrow st_i + p_i + s_{i,j} \leq st_j \\ 1 \Leftrightarrow st_j + p_j + s_{j,i} \leq st_i \end{cases}$$

Since $p_i + s_{i,j}$ is a constant, this constraint has the same form as the disjunctive constraint for the classical JSP. Finally, we note that our approach (like most methods) works only if the setup times respect the triangular inequality:

$$s_{i,j} + s_{j,k} \geq s_{i,k} \quad \forall \forall M_y \in \mathcal{M},\ \forall t_i, t_j, t_k \in M_y \tag{7.22}$$

7.6.4 Experimental Evaluation

The search heuristic used by LW-SDST was $Tdom/Bwt$. The results for the comparison methods are taken from their respective papers with the exception of the results for BSV08, which are taken from `http://www.andrew.cmu.edu/user/neils/tsp/outt2.txt`. In this section and in the following work we will refer to the best known solution of an instance as the *BKS*.

There were two approaches presented by Artigues and Feillet [6] which only differed in the search strategy used, the first was a dichotomic search strategy, the second used a linear search strategy starting from the the lower bound. We provide the best results over the two approaches (although we note that the dichotomic search strategy was generally best).

Balas also reported results for two different approaches, a single run of the SB-GLS algorithm or multiple runs of a randomized version of SB-GLS combined with a two phase strategy. The latter approach, although computationally more expensive, achieved the best results and it is these that are reported.

Benchmarks

We tested our method on a set of benchmarks proposed by Brucker and Thiele [34]. These instances were generated by adding setup times to a subset of the Lawrence job shop instances [130]. The instances in the set t2-ps* correspond to the job shop instances la01-15, instances in the set t2-pss$*$ are variations of t2-ps* where an alternative setup-time distribution was used.

Results

We first compare our method with the best systematic approaches from the literature, AF08 of Artigues and Feillet [6] and W09 of Wolf [230]. The target machines used by the comparison methods are as follows. AF08 was run on a PC with AMD64 architecture under Linux, while W09 was tested on a laptop PC Intel Core 2 Duo (T7700) with 2 GB of RAM and a 2.4 GHz processor.

Table 7.5: SDST-JSP: Comparison with state-of-the-art exact methods.

Instance	Size	AF08		W09		LW-SDST		
	$n \times m$	Best	Time(s)	Best	Time(s)	Best	Avg	Time (s)
t2-ps01		**798**	56.7	**798**	2.1	**798**	798.0	0.03
t2-ps02		**784**	242.3	**784**	7.2	**784**	784.0	0.15
t2-ps03	10x5	**749**	699.3	**749**	15.1	**749**	749.0	0.21
t2-ps04		**730**	251.6	**730**	3.8	**730**	730.0	0.06
t2-ps05		**691**	58.2	**691**	2.7	**691**	691.0	0.08
t2-ps06		**1009**	1797.6	**1009**	1481.8	**1009**	1009.0	18.52
t2-ps07		**970**	781.8	**970**	7538.2	**970**	970.0	41.63
t2-ps08	15x5	**963**	349923	-	TO	**963**	963.0	104.48
t2-ps09		1061	169582	**1060**	31812.1	**1060**	1060.0	967.27
t2-ps10		**1018**	35.1	**1018**	1788.9	**1018**	1018.0	15.05
t2-ps11		1494	916833	-	-	**1437***	1502	3600
t2-ps12		1381	914086	-	-	**1269**	1341	3600
t2-ps13	20x5	1457	895059	-	-	1415	1439	3600
t2-ps14		1483	306899	-	-	**1452**	1501	3600
t2-ps15		1661	792196	-	-	**1479***	1524	3600

Boldface values denote the best known solution to an instance. Underlined values denote that optimality was proven. Values marked with a * denote that our method improved on the BKS

Table 7.5 summarizes the results for the set t2-ps, consisting of 15 instances. No results were given for W09 on the five largest instances. Furthermore, a cutoff of 12 hours was used in W09 and no result was reported for the case where this cutoff was reached (t2-ps09). Of the ten instances tested by all three methods we see that LW-SDST proved optimality on one instance more than both AF08 and W09, and was orders of magnitude faster in most cases, albeit on different machines. On the five largest instances, LW-SDST consistently found better solutions than AF08, despite the latter having an average runtime of 8.8 *days*.

With regard to the state-of-the-art approximate methods for this type of problem, we first note the target machines used by the three comparison techniques. BSV08 was run on a Sun Ultra 60 with UltraSPARC-II processor at 360MHz, while GVV08 and GVV09 were run on Pentium IV (1.7GHz) and Intel Core 2

Table 7.6: SDST-JSP: Comparison with state-of-the-art incomplete techniques.

Instance	GVV08			GVV09			BSV08		LW-SDST		
	Best	Avg	t (s)	Best	Avg	t (s)	Best	t (s)	Best	Avg	t (s)
t2-ps11	1438	1439	31.62	1438	1441	34.92	1470	3047	**1437***	1502	3600
t2-ps12	**1269**	1291	34.28	**1269**	1277	34.57	1305	2173	**1269**	1341	3600
t2-ps13	**1406**	1415	31.14	1415	1416	34.57	1439	2468	1415	1439	3600
t2-ps14	**1452**	1489	27.83	**1452**	1489	37.51	1485	2131	**1452**	1501	3600
t2-ps15	1485	1502	36.58	1485	1496	36.02	1527	3111	**1479***	1524	3600
t2-pss06							1126	980	**1114***	1126	1161.86
t2-pss07							1075	929	**1070***	1070.0	232.47
t2-pss08							1087	794	**1072***	1077	3600
t2-pss09							1181	295	**1161***	1161	3600
t2-pss10							1121	535	**1118***	1118.0	138.25
t2-pss11							1442	3193	**1421***	1440	3600
t2-pss12				**1258**	1266	35.23	1290	2062	1269	1296	3600
t2-pss13				**1361**	1379	36.81	1398	1875	1383	1405	3600
t2-pss14							1453	2060	**1452***	1462	3600
t2-pss15							1435	4111	**1413***	1430	3600

C_{max} and average runtime comparison. Boldface values denote the best known solution to an instance. Underlined values denote that optimality was proven. Values marked with a * denote that our method improved on the BKS.

Duo (2.6GHz) machines respectively.

Table 7.6 provides comparison results on the fifteen remaining open problems. LW-SDST found the first proofs of optimality on three of these open problems (t2-pss- 06,07,10), and improved on the best known solution on seven of the remaining twelve instances. Indeed, there were only three instances where one of the other methods found a better solution than LW-SDST. However there are a couple of caveats to the comparison with the GVV methods. Firstly, they didn't report results on most of the t2-pss* instances. Secondly, as can be seen in average C_{max} columns, these methods were more consistent than LW-SDST across runs despite having a much shorter runtime.

Overall we have shown that LW-SDST is the state-of-the-art systematic method for problems of this nature and is competitive with the best metaheuristic approaches. Our method was orders of magnitude faster than the best known exact methods, closed three open problems and improved on the best known solution on another seven instances.

7.7 Job Shop Scheduling with Maximal Time Lags

7.7.1 Problem Description

An $n \times m$ job shop problem with maximal time lags (TL-JSP) is a JSP with an additional constraint on the time allowed between a task finishing and the next task of a job starting. This is written as $\mathcal{J}|tl_{\mathcal{J}}|\ \boldsymbol{C_{max}}$ in the three-field Graham notation.

Let tl_i denote the maximum duration allowed between the completion of task t_i and the start of task t_{i+1}. Then the problem can be formally defined as follows:

$$(JTL)\ \ minimize\, C_{max}$$

subject to

$$C_{max} \geq st_i + p_i \qquad \forall t_i \in \mathcal{T} \tag{7.23}$$

$$st_i + p_i \leq st_{i+1} \qquad \forall J_x \in \mathcal{J},\ \forall t_i, t_{i+1} \in J_x \tag{7.24}$$

$$st_i + p_i + tl_i \geq st_{i+1} \qquad \forall J_x \in \mathcal{J},\ \forall t_i, t_{i+1} \in J_x \tag{7.25}$$

$$(st_i + p_i \leq st_j)\ \vee\ (st_j + p_j \leq st_i) \qquad \forall M_y \in \mathcal{M},\ \forall t_i \neq t_j \in M_y \tag{7.26}$$

This type of constraint arises in many situations. For instance in the steel industry, the time lag between the heating of a piece of steel and its moulding should be small [228]. Similarly when scheduling chemical reactions, the reactives often cannot be stored for a long period of time between two stages of a process to avoid interactions with external elements [167]. This type of constraint also occurs in the food [106] and pharmaceutical industries [166].

In Figure 7.9 we illustrate the optimal solution to the sample 3×3 JSP with added time lag constraints in a Gantt chart. The time lags between successive tasks of the one job (given in Table 7.7) are the average task duration of the job. Although the optimal C_{max} is the same as for the JSP, 147, we note that the first task of J_3 does not start at 0 in this solution, but at 34 due to the time lag constraint.

Table 7.7: Sample 3×3 Job Shop Instance with maximum time lag constraints.

	t_1		t_2		t_3		
	M_i	p_i	M_i	p_i	M_i	p_i	tl_j
J_1	2	21	1	53	3	34	37
J_2	1	21	2	71	3	26	39
J_3	3	12	1	42	2	31	29

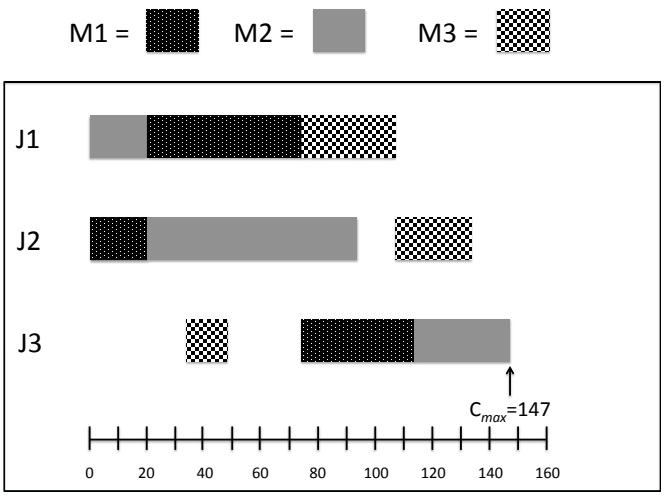


Figure 7.9: Optimal solution to sample 3×3 TL-JSP.

7.7.2 State of the art

The general TL-JSP problem has only recently received attention in the literature, (most methods have been proposed for a special case which we will consider in the following section). Caumond et al. [44] introduced a memetic algorithm, which is a combination of population based search and local search with an emphasis on using problem specific knowledge, for the TL-JSP.

An initial solution is generated by priority dispatching rules. The memetic algorithm then runs an evolutionary process, applying local search to a set of chromosones. The local search method incorporates the longest path algorithm for the disjunctive constraint with positive cycle detection, followed by a Bierwirth

sequence for representing the solution. The local search then adapts this solution.

More recently, there have been two systematic search approaches proposed. The first method, of Karoui et al. [122], proposes a climbing discrepancy search approach to the problem. Unfortunately, as it is a short paper, there is little detail provided either in the description of the method or the results. Artigues et al. [8] proposed a CP branch and bound method using dichotomic search and the branching scheme of Torres and Lopez [200]. The method uses a greedy insertion heuristic to initialize the upper bound, and combines a number of generalizations of constraint propagation rules for JSPs.

In particular they proposed generalizations of three disjunctive constraint propagation rules: Forbidden Precedences (FP) from Brucker [33]; latest starting time of last (LSL); and earliest finishing time of first (EFF). The two latter rules were originally proposed by Caseau and Laburthe [42]. They combined these generalizations with Edge-Finding and Forbidden Precedences with energetic reasoning.

7.7.3 Implementation of our model

Once again our model required little alteration, the time lag constraint being the only addition. The constraint to represent time lags between two tasks of a job are simple precedences in our model. For instance, a time lag tl_i between t_i and t_{i+1}, is represented by the following constraint: $st_{i+1} - (p_i + tl_i) \leq st_i$.

7.7.4 Experimental Evaluation

We compare with the memetic algorithm of Caumond et al. [44], which we will refer to as CLT08; and the CP method of Artigues et al. [8], which we will refer to as AHL11. All results for the memetic algorithm are taken from Caumond et al. [44] (note that the results for the memetic algorithm given in Artigues et al. [8] were taken from the dissertation of Caumond [43], the results of Caumond et al. [44] are slightly better so it is these that we use). Artigues et al. also provided a set of results on larger instances than those studied by Caumond et al, in the technical report Artigues et al. [7].

The upper bound for our method was set to the trivial upper bound of the sum of the durations of all tasks, i.e. start each job only after the preceding job has

finished. The lower bound was set to the maximum job/machine duration. The branching heuristic used was again $Tdom/Bwt$.

Benchmarks

The instances that have been studied in the literature, as presented in Caumond et al. [44], were generated by adding time lag constraints to well known JSP benchmarks. In particular we tested on forty Lawrence instances [130], and four flow shop instances of Carlier [39]. (The flow shop problem is a special case of the job shop problem where the ith task of each job is performed on the same machine for all $i \in \{1, \ldots, m\}$.)

The time lags are defined by a factor, β. For each pair of consecutive tasks in a job the maximum time lag is $\beta * p_{avg}$, where p_{avg} is the average task duration of tasks in the job. This means that the maximum time lag is the same for all pairs of consecutive tasks in a job, but may vary across jobs. (Note that the larger the value of β, the closer the instance is to the original JSP.) The values of β tested were $\beta \in \{0, 0.25, 0.5, 1, 2, 3, 10\}$. The la instances range in size from 10x5 to 30x10, while the four flow shop instances, denoted $car*$, range in size from 10x6 to 8x9.

Table 7.8: TL-JSP. APRD comparison with AHL11 and CLT08 on easy flow/job shop with time lag instances.

Instance Sets	AHL11		$CLT08^*$		LW-JTL	
	Best	Time	Best	STime/Time	Best	Time
car[5-8]_0	-	-	**0.00**	14.5/14.5	**0.00**	8.7
car[5-8]_0,5	-	-	**0.00**	28.9/322.2	**0.00**	1.9
car[5-8]_1	-	-	**0.00**	103.5/273.7	**0.00**	3.4
car[5-8]_2	-	-	**0.00**	199.5/297.1	**0.00**	4.4
la[1-5]_0	12.17	-	1.33	174.4/425.4	**0.00**	1.7
la[1-5]_0,5	**0.00**	633.6	10.23	135.8/277.8	**0.00**	0.5
la[1-5]_1	**0.00**	300.0	3.52	152.6/265.4	**0.00**	0.4
la[1-5]_2	**0.00**	249.6	1.26	104.4/234.6	**0.00**	0.3

* Results for memetic algorithm for *la*[1-5] are taken from AHL11 as these were not given in CLT08. Time is average runtime in seconds. For CLT08 "STime" is time taken to find the best solution. Boldface values denote that the optimum was found.

Results

Experiments with the memetic algorithm were carried out on a 1.8 GHz computer under Windows XP with 512 MO of memory, while the AHL11 experiments were performed on a 2.33 GHz computer with 4 GB of RAM running under Linux Red Hat 4.4.1-2. In Caumond et al. [44], they provided results for four runs of the memetic algorithm per instance. We report the best and average over the four runs.

The results, in terms of the APRD from the optimal solution, for two sets of small instances are given in Table 7.8. Artigues et al. did not test on the flow shop instances, nor did they provide runtimes for the la[1-5] instances where they failed to prove optimality. LW-JTL proved optimality on every instance (on every run), in less than ten seconds on average. For flow shop instances we find that the memetic algorithm performed best on the instances with the tightest time lag, compared to LW-JTL which found these instances to be the hardest.

A similar pattern was observed for both systematic approaches on the la instances, with performance degrading as the time lags decrease. Indeed AHL11 failed to prove optimality on four of the five instances with maximum time lag of 0, whereas it proved optimality on all other instances.

The results on larger Lawrence instances are shown in Table 7.9, again given in terms of PRD from the best solution found over the three methods. Due to the difference in machines and timeouts used, we also include the results of LW-JTL after the dichotomic search phase where runtimes are more comparable with those used by the other methods.

The superior performance of LW-JTL can be seen more clearly in this table, where we closed ten of the fourteen open problems. However, our method was less consistent, failing to prove optimality on every run for four of the ten instances. Comparing the PRDs of AHL11 and CLT08 with those of LW-JTL after the dichotomic search phase, we see that the superior performance of LW-JTL cannot be attributed to a longer runtime or faster machine.

Artigues et al. also provided results on instances la09-25 for maximum time lag $\beta \in \{0, 0.5, 1, 3, 10\}$ in an earlier technical report [7], which we will refer to as AHL10. The method and hardware used for testing were the same as for AHL11.

Table 7.9: TL-JSP. PRD comparison with CLT and AHL on hard job shop with time lag instances.

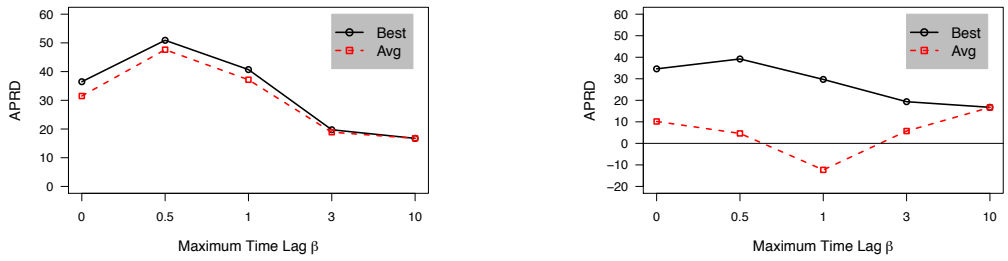
Instance	Best	AHL11		CLT08			LW-JTL Overall			LW-JTL DS		
	C_{max}	Best	Time	Best	Avg	Time	Best	Avg	OptTime	Best	Avg	Time
la06_0	1248	41.83	525	4.57	12.12	3684	**0.00***	1.23	2040 (6)	**0.00**	4.34	134.1
la07_0	1172	31.06	530	9.39	13.67	3967	**0.00**	1.70	-	2.56	5.56	97.6
la08_0	1244	31.83	528	5.47	11.15	3992	**0.00***	1.07	981 (3)	**0.00**	4.34	89.1
la06_0.5	1003	46.66	526	21.44	23.08	2440	**0.00***	0.02	1283 (9)	**0.00**	10.23	100.3
la07_0.5	953	50.05	529	23.71	25.50	2413	**0.00***	**0.00**	1522 (8)	1.15	3.10	108.6
la08_0.5	984	47.76	529	14.23	21.80	2225	**0.00***	**0.00**	393 (10)	**0.00**	0.64	103.7
la06_1	926	50.22	524	17.28	20.14	1876	**0.00***	**0.00**	1612 (10)	0.65	6.48	68.2
la07_1	896	18.86	754	15.18	18.14	1898	**0.00***	0.33	-	**0.00**	3.78	88.3
la08_1	892	17.94	587	17.49	23.04	1838	**0.00***	**0.00**	1060 (10)	0.56	4.99	88.5
la06_2	926	50.22	524	17.28	20.14	1890	**0.00***	**0.00**	1615 (10)	0.65	6.48	68.2
la07_2	896	20.42	659	18.19	19.08	1829	**0.00**	0.44	-	1.34	11.31	72.8
la08_2	892	17.94	587	17.49	23.04	1842	**0.00***	**0.00**	1060 (10)	0.56	4.99	88.4
la06_10	926	0.11	707	**0.00**	**0.00**	758	**0.00***	**0.00**	0.01 (10)	**0.00**	**0.00**	0.01
la07_10	890	26.18	518	**0.00**	**0.00**	58	**0.00**	**0.00**	-	**0.00**	**0.00**	78
la08_10	863	**0.00**	260	**0.00**	**0.00**	5	**0.00**	**0.00**	0.02 (10)	**0.00**	**0.00**	0.02
Average		30.10	552	12.11	15.39	2048	**0.00**	0.32	-	0.50	4.41	79.1

Time is average total time (in seconds). OptTime is the average time taken by LW-JTL over the runs where optimality was proven, the number in parantheses is the number of runs where optimality was proven. Boldface values denote the best known result on an instance was found. Underlined values denote that optimality was proven.
Values marked with a * denote that our method improved on the previously best known result.

Figure 7.10 shows a comparison of the APRD over the 17 instances per time lag, of AHL10 relative to the best and the average found by (a) LW-JTL, and (b) LW-JTL after the dichotomic search phase alone (Caumond only tested on Lawrence instances up to $la08$).

The figure shows that LW-JTL consistently found better solutions than AHL10 while the dichotomic search phase alone had worse average performance than AHL10 for just one maximum time lag ($\beta = 1$). The average runtime for AHL10 over the 85 instances was 740 seconds, LW-JTL had an average total runtime of 1617 seconds and an average runtime for the dichotomic search phase of 75 seconds.

The reason for the poorer performance of the dichotomic search phase is that it did not improve on the initial upper bound on every run for a number of instances, as shown in Table 7.10. AHL10 failed to improve on the initial upper bound on 70% of the instances. This may seem unsurprising given that AHL10 had a much stronger initial upper bound than LW-JTL due to their use of a greedy



(a) APRD of AHL10 vs LW-JTL

(b) APRD of AHL10 vs LW-JTL, dichotomic search only

Figure 7.10: Lawrence instances 09-25 for various maximum time lags, APRD of AHL10 versus (a) LW-JTL and (b) LW-JTL dichotomic search only.

insertion heuristic, i.e. LW-JTL had greater scope for improvement on the initial upper bound.

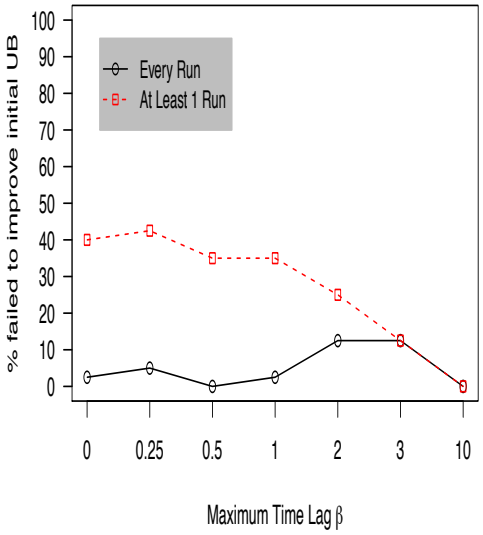
However these problems are somewhat different to typical scheduling problems where a larger upper bound on C_{max} generally makes it easier to find a feasible solution. For the general JSP a solution can be found backtrack-free given a large enough upper bound on C_{max}. This does not hold for the TL-JSP due to the time lag constraint, which greatly reduces the number of solutions to the problem.

Table 7.10: TL-JSP, optimality and failure to improve on initial upper bound.

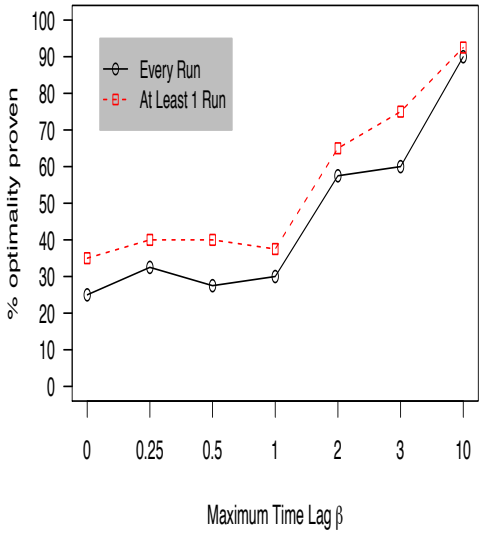
	AHL11	LW-JTL			
		Total		Dichotomic Search	
		At least 1 run	Every run	At least 1 run	Every run
# Proved optimal	4	57	44	40	24
# Failed to improve initub	60	1	0	24	1

85 Lawrence time lag instances (la[09-25], $\beta \in \{0, 0.5, 1, 3, 10\}$).
The average runtime over the 85 instances was 740s for AHL11, 1617s for LW-JTL and 75s for the dichotomic search phase of LW-JTL

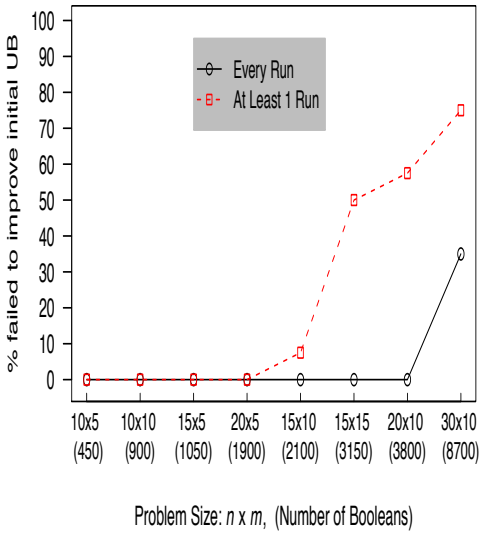
With regard to the ability of the two methods to prove optimality, we see in Table 7.10 that LW-JTL is much more efficient on these instances than AHL10. Of the 85 instances tested, the optimal solution was found for 67% on at least one



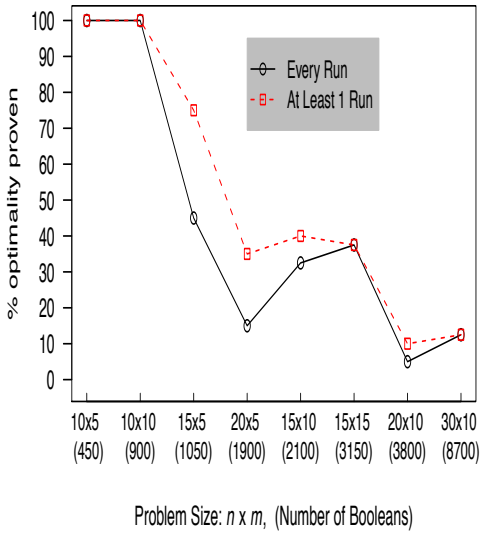
(a) % Failed to improve initial upper bound



(b) % Optimality was proven



(c) % Failed to improve initial upper bound



(d) % Optimality was proven

Figure 7.11: Lawrence instances grouped by time lag (top figures), and by problem size (bottom figures); where LW-JTL failed to improve initial upper bound (left figures), and where optimality was proven (right figures).

run. Indeed, nearly 50% of instances were solved to optimality in the dichotomic search phase alone on at least one run. In comparison, AHL10 proved optimality on less than 5% of the instances.

In Figure 7.11, we report results for LW-JTL on all forty Lawrence instances for seven different maximum time lags: $\beta \in \{0, 0.25, 0.5, 1, 2, 3, 10\}$, giving 280 instances in total. Instances are grouped by time lag (7.11(a), 7.11(b)), and by size (7.11(c), 7.11(d)). Figures 7.11(a) and 7.11(c) show the percentage of instances for which LW-JTL failed to improve on the initial upper bound, while Figures 7.11(b) and 7.11(d) show the percentage of instances for which LW-JTL proved optimality.

AHL11 didn't report results on the instances $la[26\text{-}40]$ as they never improved on the initial upper bound. Similar behavior occurred for LW-JTL where the initial upper bound was not improved on for over 50% of these instances on at least one run (Figure 7.11(c), problem sizes 15x15, 20x10 and 30x10). The problem set with the largest instances ($la[36-40]$, size 30x10) was the only problem set where there were instances that LW-JTL failed to improve the initial upper bound on every run on an instance. This occurred for 14 of the 35 instances. Unsurprisingly, LW-JTL performs best on the instances with largest maximum time lag, and on the smallest instances. However, optimality was still proven on over 30% of the instances with the smallest time lags and on 10% of the largest instances.

Finally, we note that our method has since been improved by Emmanuel Hebrard through the addition of a greedy heuristic, which is used to find a good initial upper bound when the problem isn't solved in the first (short) dichotomic search step (Grimes and Hebrard [90]). This heuristic has the additional benefit of providing a good initial solution for guiding value ordering in the subsequent search. We compare the performance of our method with and without this initialization heuristic in Table 7.11.

The results show that initializing the upper bound in this manner results in a significant improvement in search performance. Obviously the large differences are primarily due to the cases where our method couldn't improve on the basic upper bound. However, the new method *always* improved on the initial upper bound found by the greedy heuristic. This further illustrates the difficulty of these problems when starting from a weak upper bound.

Table 7.11: LW-JTL: Initialization of upper bound with greedy job insertion heuristic.

	APRD					
	Best			Worst		
	w/o greedy	w. greedy	Greedy ub	w/o greedy	w. greedy	Greedy ub
la[6,40]_0_0	12.27	0.18	12.45	151.61	6.54	21.44
la[6,40]_0_0.25	26.91	0.05	11.06	224.70	5.74	21.11
la[6,40]_0_0.5	2.91	0.05	14.65	211.63	5.35	24.58
la[6,40]_0_1	20.78	0.00	21.29	273.06	5.11	29.67
la[6,40]_0_2	121.22	0.00	23.87	239.68	2.97	31.90
la[6,40]_0_3	168.79	0.09	21.38	171.95	2.72	29.36
Average	58.81	0.06	17.45	212.11	4.74	26.35

7.8 No-Wait Job Shop Scheduling

Most algorithms introduced in the literature for the TL-JSP have been designed for a particular case of this problem: the *no-wait* job shop scheduling problem (*NW-JSP*), written as $\mathcal{J}|$ *no-wait* $|\ \boldsymbol{C_{max}}$ in the Graham three-field notation. In this case, the maximum time-lag is zero, i.e. each task must start directly after the preceding task in the job has finished. This has been shown to be NP-hard in the strong sense for the two-machine case alone (Sahni and Cho [177]). Mascis and Pacciarelli [146] found that many of the greedy algorithms (based on list schedules with priority dispatching rules) developed for the classical JSP fail to find a solution to the NW-JSP with high probability.

Figure 7.12 shows the Gantt chart of the optimal solution to the sample 3×3 JSP with added no-wait constraints. Here, the optimal C_{max} is 197, which is the largest makespan over the different variants of this instance that we've tested. As one can see in the figure, each job is one block of tasks which can be moved as long as each successive task starts immediately after the preceding task in the job.

7.8.1 State of the art

A large number of metaheuristic approaches have been proposed in recent years for the NW-JSP. Schuster and Framinan [182] proposed hybridizing a genetic algorithm with simulated annealing; Framinan and Schuster [65] applied a re-

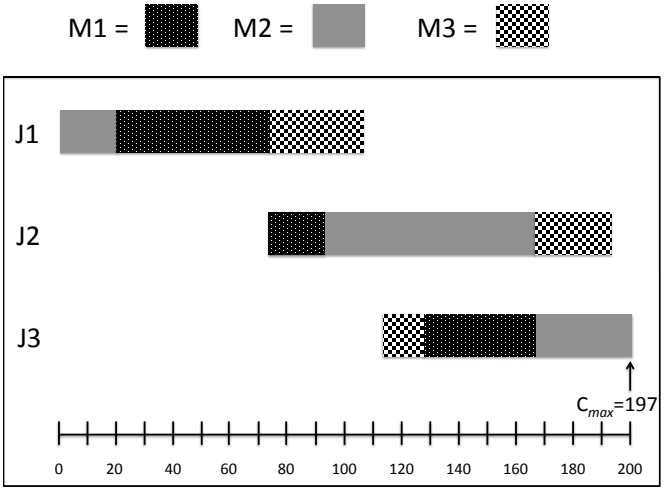


Figure 7.12: Optimal solution to sample 3×3 NW-JSP.

cently proposed technique, complete local search with memory (*CLM*, Ghosh and Sierksma [74]), where all solutions are stored to avoid repeatedly discovering the same solutions; while Schuster [181] also proposed an efficient tabu search algorithm for the NW-JSP.

The best metaheuristic methods, however, are the complete local search with limited memory (*CLLM*) method of Zhu, Li and Wang [239] and a hybrid constructive/tabu search algorithm (*HTS*) introduced by Bożejko and Makuchowski in 2009 [30].

The CLLM method of Zhu et al. decomposes the problem into sequencing and timetabling components. It differs from the CLM method of Framinian and Schuster in the following ways: firstly they augment the timetabling method to perform shift timetabling on top of non-delay and enhanced timetabling techniques; secondly, since the memory required by CLM in storing solutions increases with the size of the problem, they propose limiting the number of solutions that are stored. Furthermore, unlike most methods which use the sequencing and timetabling decomposition, CLLM solves them integrally rather than considering them as two separate subproblems.

The HTS method of Bożejko and Makuchowski uses a constructive job in-

sertion heuristic to generate a set of initial solutions. They also solve a "mirror" instance, which is the instance with the order of all tasks on their jobs reversed. Tabu search is then performed with intensification of search around the best found solution. When the algorithm fails to improve on the solution after a predefined number of iterations, a "backjump" is performed to the best known solution. The size of the tabu list is then incremented and the number of allowed non-improving iterations prior to returning to the best solution is doubled.

The first systematic method for this problem was that of Mascis and Pacciarelli [146], incorporating a generalization of the disjunctive graph representation, which they refer to as the alternative graph. These graphs are identical for the classical JSP, but the alternative graph allows for greater inference on the NW-JSP. Recently, van den Broek [201] has proposed an improvement to this method, where the problem is formulated as a Mixed Integer Programming (MIP) problem incorporating the alternative graph representation.

Van den Broek's algorithm, which we will refer to as vdB09, initializes the upper bound using a problem-specific job insertion heuristic that is guaranteed to find a feasible solution, although in the worst case the trivial feasible solution is found. The branch and bound method applies three immediate selection rules based on the alternative graph, and uses a branching scheme also incorporating information from the alternative graph.

7.8.2 Implementation of our model

Although our generic model was relatively efficient on these problems, we made a simple improvement for the no-wait problem based on the following observation: if no delay is allowed between any two consecutive tasks of a job, then the start time of every task is functionally dependent on the start time of any other task. The tasks of each job can thus be viewed as one block. In other words we really only need one variable in our model to represent all the tasks of a job. We therefore use only n variables: $\{J_x \mid 1 \leq x \leq n\}$.

Let h_i be the total duration of the tasks coming before task t_i in its job. That is, if job $J = \{t_1, \ldots, t_m\}$, we have: $h_i = \sum_{k<i} p_k$. For every pair of tasks $t_i \in J_x, t_j \in J_y$ sharing a machine, we use the same Boolean variables to represent

disjuncts as in the original model, however linked by the following constraints:

$$b_{ij} = \begin{cases} 0 \Leftrightarrow J_x + h_i + p_i - h_j \leq J_y \\ 1 \Leftrightarrow J_y + h_j + p_j - h_i \leq J_x \end{cases}$$

Notice that, although the variables and constants are different, these are still exactly the same ternary disjuncts used in the original model.

The no-wait job shop scheduling problem can therefore be reformulated as follows, where the variables $J_1, \ldots, J_n$ represent the start time of the jobs, $J_{x(i)}$ stands for the job of task t_i, and $f(i,j) = h_i + p_i - h_j$.

$$(NW - JSP) \;\; minimize\, C_{max}$$

subject to

$$C_{max} \geq J_x + \sum_{t_i \in J_x} p_i \qquad \forall J_x \in \mathcal{J} \tag{7.27}$$

$$(J_{x(i)} + f(i,j) \leq J_{x(j)}) \vee (J_{x(j)} + f(j,i) \leq J_{x(i)}) \qquad \forall M_y \in \mathcal{M},\; t_i, t_j \in M_y \tag{7.28}$$

The initial upper bound for the dichotomic search is set to the trivial upper bound, i.e. the sum of the durations of every task.

7.8.3 Experimental Evaluation

The best heuristics for our method were $Tdom/Twt$, and $Tdom$ with the weight on the Booleans used as a tie-breaker ("$Tdom$+Bwt"). This was quite surprising given that neither metric, $tdom$ alone nor $tdom/twt$, offer much discrimination amongst Booleans. Indeed all Booleans between tasks of the same pair of jobs will have the same $tdom$ value and the same $tdom/twt$ value, since all tasks are replaced by their job in the improved model. Therefore the heuristic $Tdom$+Bwt will choose the pair of jobs with the smallest sum of domain sizes, and choose the Boolean variable with the largest weighted degree from the Boolean variables of the selected pair of jobs.

Furthermore, the promise value ordering performed poorly in finding initial solutions so we used a simple static value ordering heuristic, which we refer to as

JobRank. Here each pair of tasks on a machine are ordered based on their relative position in their respective jobs. For example, given a pair of tasks (t_i, t_j) on a machine, if t_i is fourth in the order on its job, and t_j is second in the order on its job, then the Boolean variable b_{ij} is set up so that $b_{ij} = 0 \Leftrightarrow t_j + p_j < t_i$. (We will return to these issues when assessing the different factors of our method in Section 7.10.)

Comparison of NW-JSP model with TL-JSP model

We first compare the no-wait model with the general time lag model on the set of Lawrence NW-JSPs. The results, given in Table 7.12, show that the problem specific model results in a large improvement over LW-JTL on these instances. For both heuristics tested LW-JNW outperformed LW-JTL in all metrics of comparison: much lower APRD; proved optimality on a larger number of instances; never failed to improve on the initial upper bound; and was an order of magnitude faster on the ten instances that were solved to optimality by all three methods tested.

Interestingly, $Tdom$+Bwt was much better than $Tdom$/Twt at proving optimality, yet the opposite was the case when the heuristics are compared in terms of APRD. $Tdom$/Twt was better at finding good solutions, but had much greater variation in performance than $Tdom$+Bwt. This can be seen even more clearly in the following section.

Table 7.12: NW-JSP: Comparison of NW-JSP and TL-JSP models.

Model	Heuristic	APRD	Proved Optimal	Common OptTime	Failed to Improve InitUB
LW-JTL	$Tdom$/Bwt	12.7	10/14	46.04	1/16
LW-JNW	$Tdom$+Bwt	0.8	**26/26**	**0.53**	**0/0**
	$Tdom$/Twt	**0.3**	14/16	0.90	**0/0**

Notes: Results are for 40 Lawrence no-wait JSPs. i/j refers to every run / at least one run. "Common OptTime" is average runtime in proving optimality on the 10 instances which were solved by all methods on every run. InitUB is the initial ub on C_{max}. For PRD C_{ref} is best C_{max} over the methods.

Comparison with state of the art

The branch and bound method of van den Broek [201] (vdB09) was run on an Intel core T8300, with a 2.4 GHz processor and 2GB of internal memory. The metaheuristic methods CLLM [239] and HTS [30] were run on PCs with a Pentium 4 processor and an AMD Duron 800 MHz processor respectively. CLLM had 20 runs per instance, thus we report the results in terms of best and average performance. For HTS, the authors reported two sets of results on the set of "hard" instances, where one run was "without limit of computation time".

We first compare our approach with the best systematic method, vdB09, in terms of average runtime on the set of instances for which both proved optimality (Table 7.13). The results for the (small) instances that were closed by Mascis and Pacciarelli [146] are not included as all methods solved these to optimality in under three seconds on average. On the medium sized instances we find again that $Tdom/Twt$ is quite poor at proving optimality on these problems.

Comparing vdB09 with $Tdom$+Bwt, we see that on average vdB09 took over twice as long, albeit on a different machine. In van den Broek [201], they gave proofs of optimality on 13 further instances which LW-JNW was unable to solve. However, the proofs of optimality for these instances took over 21 hours on average, with only one instance solved in under one hour, while the longest runtime on an instance was over three days. These results show that LW-JNW is competitive with the state of the art systematic method.

In Table 7.14 we compare the makespans found by both metaheuristic and systematic methods on a set of "easy" instances. The set consists of two *ft* instances of sizes 6x6 and 10x10, two *abz* instances and ten *orb* instances all of size 10x10, and fifteen *la* instances split into three groups of five, of size 10x5, 10x10 and 15x5 respectively. The results show that the systematic methods are extremely efficient on these instances, finding the optimal solution in less than ten seconds on average for vdB09 and $Tdom$+Bwt ($Tdom/Twt$ had longer average runtime as it failed to prove optimality on one run on one instance).

We give the APRDs over the remaining 53 (hard) instances in Table 7.15. The runtime used by vdB, where there was a limit on the computation time, was that given for CLLM for each instance in Zhu et al. [239]. Similarly HTS, where

Table 7.13: NW-JSP: Runtime comparison with vdB09 for proofs of optimality.

Instances	Opt	vdB	LW-JNW	
	C_{max}		$Tdom$+Bwt	$Tdom/Twt$
La06	1248	40	57	104
La07	1172	22	37	50
La08	1244	16	16	30
La09	1358	38	61	121 (9)
La10	1287	21	18	41
La12	1414	3506	2278	-
La21	2030	226	290	-
La22	1852	285	303	-
La23	2021	490	380	-
La24	1972	356	372	-
La25	1906	133	114	-
La36	2685	695	426	-
La37	2831	1963	1962	-
La38	2525	1106	786	-
La39	2660	2515	1827	152 (7)
La40	2564	1505	1746	-
Ft20	1532	11830	2565	-
swv05	2333	10593	3308	-
	Average	1963	919	-
	Max	11830	3308	-

Notes: Results for LW-JNW are averages of ten runs, except for two cases with $Tomd/Twt$ where the number in parantheses is the number of runs for which optimality was proven.

Table 7.14: NW-JSP: APRD comparison with state of the art on "easy" instances.

	vdB	CLLM		HTS	$Tdom$+Bwt	$Tdom$/Twt
		Best	Avg		Avg	Avg
APRD	0	0.005	0.615	1.068	**0**	**0**
Avg Runtime	7.5	-	73.1	1.3	7.4	25.5

Notes. 29 Instances: *abz*5-6, *ft*6, *ft*10, *orb*1-10, *la*1-10, *la*16-20.
Results weren't included by vdB on *la*1-5, and by CLLM on *abz*5-6.

there was a limit on the computation time, used the runtime of the tabu search algorithm in Schuster [181] for each instance. For vdB where there was no limit on computation time, every instance was solved to optimality (as discussed with regard to Table 7.13 above). Unfortunately, Bozėjko and Makuchowski did not provide any further runtime information for the "unlimited" case.

The results show that, outside of HTS and vdB with unlimited computation time, LW-JNW had a lower average APRD with both heuristics than the other methods. $Tdom$/Twt produced the best solutions, within 2% of the best known solution on average. Furthermore it improved on the best known solution for one of the 27 open problems (finding an upper bound on C_{max} of 5769 for swv17, whereas the previous best known upper bound was 5780, found by HTS with unlimited computation time).

This is quite surprising given its relatively poor performance in terms of proving optimality as previously shown in Table 7.13 above. LW-JNW with $Tdom$+Bwt proved optimality on 13 of the 53 "hard" instances compared to $Tdom$/Twt which only proved optimality on one instance. However the quality of solutions found by $Tdom$/Twt was much less consistent than $Tdom$+Bwt across the ten runs.

Finally, we note that this method has also been recently improved by Emmanuel Hebrard (Grimes and Hebrard [90]). The basis for the improvement is a tighter model through the identification of *conflict intervals* for pairs of jobs. Each disjunct between a pair of tasks defines a conflict interval for their respective jobs.

Using the notation $J_{x(j)}$ for the job x of task t_j, then for two tasks t_i and t_j,

Table 7.15: NW-JSP: comparison with state of the art on "hard" instances.

	vdB		CLLM		HTS		$Tdom$+Bwt		$Tdom$/Twt	
Instances	Lim	Unlim	Best	Avg	Lim	Unlim	Best	Avg	Best	Avg
la11-15	1.33	**0**	3.44	6.27	6.47	0.89	0.25	0.25	0.85	2.54
la21-25	**0**	**0**	1.22	3.59	6.20	**0**	**0**	**0**	**0**	0.63
la26-30	5.63	**0**	5.73	9.75	10.47	0.85	3.53	3.54	3.57	6.27
la36-40	2.92	**0**	2.17	5.22	7.29	0.33	**0**	**0**	0.75	3.92
swv1-5	4.78	**0**	1.03	4.30	3.67	**0**	2.40	2.60	0.69	2.35
la31-35	11.52	-	5.95	10.32	8.26	**0**	8.34	9.68	2.46	5.52
swv6-10	7.51	-	2.05	4.23	4.19	**0**	5.32	5.79	1.46	4.54
swv11-15	-	-	-	-	1.86	**0**	24.19	25.14	4.97	8.27
swv16-20	-	-	-	-	3.98	**0**	7.12	10.24	1.30	3.99
ft20	-	**0**	-	-	4.96	**0**	**0**	**0**	1.11	2.89
yn1-4	-	-	-	-	8.51	**0**	4.54	5.67	3.27	6.37
abz7-9	-	-	-	-	8.61	**0**	3.71	3.85	1.98	5.04
Avg1	4.81	-	3.08	6.24	6.65	**0.30**	2.83	3.12	1.40	3.68
Avg2	-	-	-	-	6.17	**0.20**	5.38	6.04	1.89	4.41

Notes: Lim/Unlim refers to limited/unlimited computation time.
Avg1 is the average for the subset of instances that CLLM and vdB-Lim solved.
Avg2 is the average over all instances.

sharing a resource we have the following conflict interval:

$$J_{x(j)} \notin \ ]J_{x(i)} - f(j,i), J_{x(i)} + f(i,j)[$$

and vice versa for $J_{x(i)}$. However, these intervals may overlap or subsume each other. It is therefore possible to tighten this encoding by computing larger intervals, that we refer to as *maximal forbidden intervals*, hence resulting in fewer disjuncts. (Further details can be found in [90].)

Table 7.16: NW-JSP: comparison with new model.

Instances		Metric	$Tdom$+Bwt		$Tdom$/Twt	
Set	#		New	Old	New	Old
easy	29	Average Runtime	**1.78**	7.4	2.71	25.49
hard	53	APRD	2.86	5.38	**1.31**	1.89
		# Opt	**21**	13	8	1
		# Bks	2	0	**3**	1

For *hard* set, metrics are APRD of best solution relative to that of previous BKS; number of instances where optimality was proven; number of improved solutions found.

Table 7.16 summarizes the performance gains achieved by the new improved model. For both heuristics, we provide four different comparisons of the new and old models. The APRD is relative to the previous best known solution, which was either the optimal solution found by vdB or the best solution found by HTS with unlimited computation time. The results show that the new model improves both the ability to prove optimality and the quality of the solutions found where optimality was not proven. New upper bounds were found for four instances ($abz9$, $la34$, $yn2$ and $yn4$).

7.9 Job Shop Scheduling with Earliness/Tardiness Objective

7.9.1 Problem Description

All the problems studied so far have involved minimizing the latest completion time over all jobs (given the earliest starting time over all jobs is 0). An alternative objective, which is important in industry, is the minimization of the cost of a job finishing early/late. An example of a cost for early completion of a job would be storage costs incurred, while for late completion of a job these costs may represent the impact on customer satisfaction.

In the job shop problem with earliness tardiness objective ($\mathcal{J} \mid rd_x dd_x \mid w_x^e \sum E_x + w_x^t \sum L_x$ in the Graham notation), each job J_x has a release date, rd_x, and a due date, dd_x. There is a cost associated with completing a job before its due date, w_x^e, and similarly with the tardy completion of a job, w_x^t. The cost of a job is then given by

$$cost_{J_x} = \begin{cases} w_x^e(dd_x - C_x) \text{ if } C_x \leq dd_x \\ w_x^t(C_x - dd_x) \text{ if } C_x > dd_x \end{cases}$$

where C_x refers to the completion time of job J_x. The total cost is the sum of the costs of each job: $ET_{sum} = \sum_{J_x \in \mathcal{J}} cost_{J_x}$. (Note that these problems differ from Just in Time job shop scheduling problems (Baptiste et al. [18]), where there is a cost associated with the completion time of each *task*.)

The problem is formally defined as follows:

$$(ET - JSP)\ \ minimize\ ET_{sum}$$

subject to

$$ET_{sum} = \sum_{J_x \in \mathcal{J}} (w_x^e E_x + w_x^t L_x) \tag{7.29}$$

$$E_x = \max(dd_x - C_x, 0) \qquad \forall J_x \in \mathcal{J} \tag{7.30}$$

$$T_x = \max(C_x - dd_x, 0) \qquad \forall J_x \in \mathcal{J} \tag{7.31}$$

$$st_i + p_i \leq st_{i+1} \qquad \forall J_x \in \mathcal{J},\ \forall t_i, t_{i+1} \in J_x \tag{7.32}$$

$$(st_i + p_i \leq st_j)\ \vee\ (st_j + p_j \leq st_i) \qquad \forall M_y \in \mathcal{M},\ t_i \neq t_j \in M_y \tag{7.33}$$

We again illustrate the problem using a Gantt chart (Figure 7.13) for the optimal solution to the sample 3×3 JSP with additional due dates and weights as given in Table 7.17. In the optimal solution, J_1 and J_3 finished on time while J_2 finished 4 units early, giving a minimum cost of 120 (4*w_2^e=4*30). We also note that C_{max} for the optimal solution was 160.

Table 7.17: Sample 3×3 Job Shop Instance with due dates and earliness/tardiness penalties.

	t_1		t_2		t_3				
	M_i	p_i	M_i	p_i	M_i	p_i	dd_j	w_j^e	w_j^t
J_1	2	21	1	53	3	34	160	100	50
J_2	1	21	2	71	3	26	130	30	40
J_3	3	12	1	42	2	31	150	60	20

7.9.2 State of the art

The best complete methods for this type of problem are the CP/LP hybrid of Beck and Refalo [22], the MIP approaches of Danna et al. [53] and Danna and Perron [51], while more recently Kebel and Hanzalek proposed a pure CP approach [124]. Unfortunately the latter didn't provide results on the benchmarks widely studied in the literature, so we do not provide a comparison with their method. Danna and

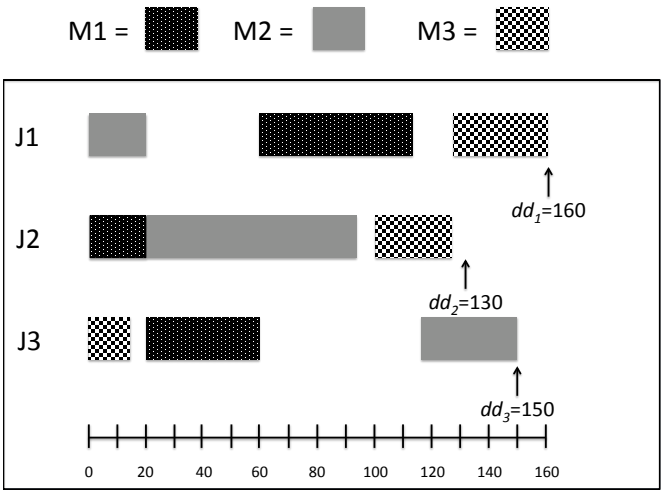


Figure 7.13: Optimal solution to sample 3×3 ET-JSP.

Perron [51] also proposed an incomplete method based on large neighborhood search method, while a number of genetic algorithms have been proposed for problems of this type (see the survey of Vázquez and Whitley [206]).

The complete method of Beck and Refalo (*CRS-ALL*) involves applying CP and LP algorithms at each search node. The CP method is based on probe backtrack search [178] and is used to infer domain reductions and precedence constraints on the resources. The LP algorithm is used to maintain an optimal solution to a relaxation of the problem where resource constraints are not considered. The branching heuristic then selects a pair of conflicting tasks (due to the resource constraints) from this solution and posts a precedence constraint between them. This constraint is added to both the CP and LP models.

The key component of their technique is the creation and solving of the cost relevant subproblem (CRS), which only contains those tasks and resources which directly impact the objective function. In other words, the CRS contains the last task of each job (since the cost is solely based on the completion time of these tasks) and the resources that these tasks require for processing. The CRS is solved using the same CP method (an augmented form of probe backtrack search). The optimal solution to the CRS is then either extended to a full solution or it is proven

that no solution exists for this cost, using a pure CP scheduling approach. In the former, a new upper bound has been found, while in the latter a new lower bound has been found.

The MIP methods proposed by Danna et al. [53] (RINS), and Danna and Perron [51] (uLNS and sLNS) are based on large neighborhood search [186]. *Relaxation induced search* (RINS) is a heuristic method for restricting search to subspaces, referred to as sub-MIPs. A standard MIP algorithm is used to produce a solution (of the continuous relaxation). The RINS method takes an incumbent solution and fixes a subset of the variables to the values they took in the incumbent solution. Branch-and-cut is then performed on the restricted search space for a limited number of nodes / amount of time.

Unstructured large neighborhood search, or uLNS, is an extension of RINS which incorporates a tree traversal strategy known as *guided dives*, which guides search towards the neighborhood of the incumbent. The structured large neighborhood search method (sLNS) they proposed is a traditional LNS scheduling method embedded in a dedicated CP algorithm. The algorithm uLNS is a complete method, whereas sLNS is incomplete.

The comparison methods are as follows:

- *MIP*: Default CPLEX in [53], run using a modified version of ILOG CPLEX 8.1
- *CP*: A pure constraint programming approach introduced by Beck and Refalo in [22], run using ILOG Scheduler 5.3 and ILOG Solver 5.3
- *CRS-ALL*: A CP/LP hybrid approach proposed by Beck and Refalo in [22], run using ILOG CPLEX 8.1, ILOG Hybrid 1.3.1, ILOG Scheduler 5.3 and ILOG Solver 5.3
- *uLNS*: An unstructured large neighborhood search MIP method proposed by Danna and Peron in [52], run using a modified version of ILOG CPLEX 8.1
- *sLNS*: A structured large neighborhood search CP/LP method proposed by Danna and Peron in [52], run using ILOG Scheduler 5.3, ILOG Solver 5.3 and ILOG CPLEX 8.1

7.9.3 Implementation of our model

Although the only change to the problem is the objective function, our model requires a number of additional elements. In particular we introduce $4n$ additional variables. For each job J_x we have a Boolean variable e_x that takes the value 1 iff J_x is finished early and the value 0 otherwise. In other words, e_x is a reification of the precedence constraint $st_{xm} + p_{xm} < dd_x$.

Moreover, we also have a variable E_x representing the duration between the completion time of the last task of J_x and the due date dd_x when J_x finishes before its due date: $E_x = e_x(dd_x - st_{xm} - p_{xm})$. Symmetrically, for each job J_x we have a Boolean variable l_x taking the value 1 iff J_x is finished late, and an integer variable T_x representing the delay.

We model the earliness/tardiness objective as follows:

$$minimize\ ET_{sum} \text{ subject to :}$$

$$ET_{sum} = \sum_{J_x \in \mathcal{J}} (w_x^e E_x + w_x^t L_x) \tag{7.34}$$

$$e_x \Leftrightarrow (st_{xm} + p_{xm} < dd_x) \qquad \forall J_x \in \mathcal{J} \tag{7.35}$$

$$E_x = e_x(dd_x - st_{xm} - p_{xm}) \qquad \forall J_x \in \mathcal{J} \tag{7.36}$$

$$l_x \Leftrightarrow (st_{xm} + p_{xm} > dd_x) \qquad \forall J_x \in \mathcal{J} \tag{7.37}$$

$$T_x = l_x(st_{xm} + p_{xm} - dd_x) \qquad \forall J_x \in \mathcal{J} \tag{7.38}$$

Unlike the case where the objective involves minimizing C_{max}, branching only on the disjuncts is not sufficient for these problems. Thus we also branch on the early and late Boolean variables, and on the variables representing the start times of the last task of each job.

7.9.4 Experimental Evaluation

We tested our method on two sets of benchmarks which have been widely studied in the literature. The comparison experimental results are taken from [53] and [51], where all experiments were performed on a 1.5 GHz Pentium IV system running Linux. For the first benchmark, these algorithms had a time limit of 20

minutes per instance, while for the second benchmark the algorithms had a time limit of 2 hours.

The results for LW-ETJSP are given in terms of the best and worst performance over the ten runs. The variable heuristic used was $dom/wdeg$, where dom was $tdom$ and $wdeg$ was twt for the Booleans representing the disjuncts, and had the normal defintions for the other variables.

This heuristic will naturally branch on the e_x and l_x variables at the top of the search tree (up until sufficient failures have been encountered during the previous restarts) as their domain size is 2 compared with the $tdom$ of the Boolean disjunct variables. Assigning these variables to the value 0 (i.e. lexically) will result in search for a solution of minimum cost (as the last task of each job will be set such that it finishes on the due date of the job where possible).

Observe, however, that forcing this branching strategy (of initially searching for a solution of minimum cost) would result in poor performance if a solution with this cost did not exist. In this case, search must prove the underlying scheduling problem infeasible. If this was not achieved within the cutoff in the first dichotomic step, then increasing the upper bound on the cost for the subsequent dichotomic steps would have no impact on search as the branching strategy would repeatedly search for a solution with the same minimum cost.

Benchmarks

The first benchmark consists of 9 sets of problems, each containing 10 instances. These were generated by Beck and Refalo [22] using the random JSP generator of Watson et al. [225]. For problem size $\mathcal{J}$x$\mathcal{M}$, three sets of JSPs of size 10x10, 15x10 and 20x10 were generated, each set containing ten instances. For each JSP instance, three ET-JSP instances were generated with different due dates and costs. The costs were uniformly drawn from the interval [1, 20]. The due dates were calculated based on the Taillard lower bound (*tlb* [196]) of the base JSP instance, and were uniformly drawn from the interval:

$$[0.75 \text{ x } tlb \text{ x } lf, 1.25 \text{ x } tlb \text{ x } lf]$$

where *lf* is the looseness factor and takes one of the three values $\{1.0, 1.3, 1.5\}$. Thus for each problem size, there are 30 instances (10 for each looseness factor).

Jobs do not have release dates in these instances.

The second benchmark is taken from the genetic algorithms (GA) literature and was proposed by Morton and Pentico [154]. There are 12 instances, with problem size ranging from 10x3 to 50x8. Jobs in these problems do have release dates. Furthermore earliness and tardiness costs of a job are equal in these problems.

Results

We present results on the first benchmark in Table 7.18 in terms of number of instances solved to optimality by each of the complete algorithms. Here, the "best" column for our method refers to the number of instances solved to optimality on at least one of the ten runs on the instance, while "worst" refers to the number of instances solved to optimality on all ten runs.

Table 7.18: ET-JSP - Random Problems, Number Proven Optimal

Looseness Factor	MIP	CP	uLNS	CRS-All	LW-ETJSP	
					Best	Worst
1.0	0	0	0	7	**10**	8
1.3	14	6	**30**	**30**	**30**	**30**
1.5	27	6	**30**	**30**	**30**	**30**

Notes: Comparison results taken from [53], except uLNS results taken from [52]. Figures in bold are the best result over all methods.

While there is little difference in the performance of our method and that of uLNS and CRS-ALL on the looser instances (looseness factors of 1.3 and 1.5), we see that our method is able to close three of the 23 open problems in the set with looseness factor 1.0. One possible explanation for the improvement with our method is the difference in time limits and quality of machines. However, analysis of the results reveals that of the 68 instances solved to optimality on every run of LW-ETJSP, only 8 took longer than one second on average, and only one took longer than one minute (averaging 156s). Furthermore, uLNS only solved two instances to optimality when the time limit was increased to two hours [52]. Clearly our method is extremely efficient at proving optimality on these problems.

The previous results suggest that CRS-ALL is much better than uLNS on these instances. However, as was shown by Danna et al. [53], this was not the case when the algorithms were compared in terms of the sum of the upper bounds found over the 30 "hard" instances (i.e. with looseness factor 1.0). In Table 7.19 we assess whether there was a similar deterioration in the performance of LW-ETJSP as for CRS-ALL on the instances where optimality wasn't proven.

Table 7.19: ET-JSP - Random Problems, Upper Bound Sum

	MIP	CP	uLNS	sLNS	CRS-All	LW-ETJSP	
lf						Best	Worst
1.0	654,290	1,060,634	156,001	52,307	885,546	**30,735**	38,416
1.3	26,930	1,248,618	**8,397**	**8,397**	**8,397**	**8,397**	**8,397**
1.5	7,891	1,672,511	**6,964**	**6,964**	**6,964**	**6,964**	**6,964**

Notes: Comparison results taken from [53], except uLNS and sLNS results [52]. Figures in bold are the best result over all methods.

On the contrary, we find that the performance of LW-ETJSP is even more impressive when algorithms are compared using this metric. The two large neighborhood search methods found the best upper bounds of the comparison algorithms with sLNS the most efficient by a factor of 2 over uLNS. There are a couple of points that should be noted concerning sLNS. Firstly it is an incomplete method so cannot prove optimality, and secondly the sum of the worst upper bounds found by our method was still significantly better than that found by sLNS. Indeed, there was very little variation in performance for LW-ETJSP across runs, with an average difference over the 30 instances of just 256 between the best and worst upper bounds found per instance.

Danna and Perron also provided the sum of the best upper bounds found on the hard instances over all methods they tested, which was 36,849 [51]. This further underlines the quality of the performance of LW-ETJSP on these instances. Finally, we once again considered the hypothesis that the different time limit and machines used for experiments might explain these results. We compared the best and worst upper bounds found by our method after the dichotomic search phase, where the maximum runtime of this phase over all runs per instance was 339s. The upper bound sums on the hard instances were 32,299 and 49,808 for best and

worst respectively, which refutes this hypothesis.

Table 7.20 provides results on the second of the benchmarks (taken from the GA literature). Following the convention of previous work on these instances ([206][22][53]), we report the cost normalized by the weighted sum of the job processing times. We include the best results found by the GA algorithms as presented by Vázquez and Whitley [206]. We also provide an aggregated view of the results of each algorithm using the geometric mean ratio (GMR), which is the geometric mean of the ratio between the normalized upper bound found by the algorithm and the best known normalized upper bound, across a set of instances.

Table 7.20: ET-JSP - GA Problems, Normalized upper bounds

		MIP	CP	uLNS	sLNS	CRS	GA	LW-ETJSP	
Instance	Size					-All	Best	Best	Worst
jb1	10x3	**0.191***	0.474	**0.191***	**0.191**	**0.191***	0.474	**0.191***	**0.191***
jb2	10x3	**0.137***	0.746	**0.137***	**0.137**	0.531	0.499	**0.137***	**0.137***
jb4	10x5	**0.568***	0.570	**0.568***	**0.568**	**0.568***	0.619	**0.568***	**0.568***
jb9	15x3	**0.333***	0.355	**0.333***	**0.333**	1.216	0.369	**0.333***	**0.333***
jb11	15x5	0.233	0.365	**0.213***	**0.213**	**0.213***	0.262	0.221	0.235
jb12	15x5	**0.190***	0.239	**0.190***	**0.190**	**0.190***	0.246	**0.190***	**0.190***
	GMR	1.015	1.774	1	1	1.555	1.610	1.006	1.017
ljb1	30x3	**0.215***	0.847	**0.215***	**0.215**	0.295	0.279	**0.215**	0.221
ljb2	30x3	0.622	1.268	**0.508**	**0.508**	1.364	0.598	0.590	0.728
ljb7	50x5	0.317	0.614	0.123	**0.110**	0.951	0.246	0.166	0.256
ljb9	50x5	1.373	1.737	1.270	1.015	2.571	**0.739**	1.157	1.513
ljb10	50x8	0.820	1.569	0.558	0.525	1.779	0.512	**0.499**	0.637
ljb12	50x8	1.025	1.368	0.488	0.605	1.601	**0.399**	0.537	0.623
	GMR	1.943	3.233	1.213	1.170	4.098	1.220	1.299	1.686
Overall GMR		1.329	2.434	1.084	1.068	2.305	1.408	1.118	1.256

Comparison results taken from [53]. Figures in bold indicate best upper bound found over the different algorithms. "*" indicates optimality was proven by the algorithm.

The performance of our method was less impressive for these instances, solving two fewer instances to optimality than uLNS, and achieving a worse GMR than either of the large neighborhood search methods. However, we remind the reader that all comparison methods had a 2 hour time limit on these instances, except the GA approaches for which the time limit was not reported. We further note that LW-ETJSP improved on the best known solution for one instance (ljb10)

and outperforms all methods other than uLNS and sLNS.

Overall, we have shown that our model can be successfully adapted to handle the objective of minimizing the sum of earliness/tardiness costs. These problems have traditionally proven quite troublesome for CP approaches due to the weak propagation of the sum objective [51].

7.10 Analysis of different factors in model

In this section we assess the relative importance of the different components of our algorithm to its overall performance. The motivation for this is twofold. Firstly, the results provide further insight into the behavior of our algorithm, which could be of use to other researchers considering a similar approach. Secondly, the results provide insight into the problems themselves, which can also clarify why certain components of our approach are more suited to one type of problem than another.

For each problem type we compared the default method with the following variations:

- Variable Ordering: $Tdom$, Bwt, and Twt alone; and the non-default of $Tdom$/Bwt and $Tdom$/Twt.

  The default for OSP, SDST-JSP and TL-JSP was $Tdom$/Bwt; the default for JSP and ET-JSP was $Tdom$/Twt; ad finally the default for NW-JSP was $Tdom$+Bwt.

- Value Ordering: Promise and lexical (for the job shop problem and its variants, this is the static heuristic as described in Section7.8.3). The default used solution guided value ordering.

- Without nogood recording from restarts (*noNgd*). The default used nogood recording from restarts.

- Dichotomic Search: None versus 3 or 300 second cutoffs (*noDs/Ds3s/Ds300s*). The default used a 30 second cutoff for dichotomic search.

- Randomization: None versus randomly select from top 2 or top 3 choices (*noRand/RandTop2/RandTop3*). The default used random tie breaking.

- Restarting: None versus Luby restarting strategy (*noRestart/Luby*). For the latter, the scale factor was 4096 failures. The default used the geometric restarting strategy.

For each variant of an algorithmic component, all other components used the default settings (for example *Luby* combined the Luby restarting strategy with the default variable heuristic for the problem type, solution guided value ordering, nogood recording from restarts, 30 second cutoff for each dichotomic step, and random tie breaking).

7.10.1 Experimental Setup and Benchmarks

There were ten runs per instance with an overall time limit of twenty minutes per run on an instance (except obviously for the non-randomized variant for which there was only one run). A subset of ten benchmarks were randomly selected for each problem type. However the selection process was biased so as to contain a number of difficult instances (based on the results of the previous sections), and to have problems of varying size.

The algorithms were compared in terms of % runs where optimality was proven, average runtime, and best and average APRD where C_{ref} was the best objective found on the instance over the different variants. For ET-JSPs the sample contained five of the instances generated by Beck et al. [22] and five of the instances from the GA literature [154]. The APRD for the latter was calculated relative to the normalized objective as described in the previous section, and the overall results for ET-JSPs are averaged over the two sets.

The results for four of the problem types (OSP, JSP, SDST-JSP, and ET-JSP) are given in Table 7.21. The rankings are relatively consistent in terms of the key algorithmic factors across the different problem types. The weighted component of the variable heuristic was most important, as evidenced by the performance of $Tdom$.

In particular, when compared in terms of % proven optimal, $Tdom$ performed significantly worse in three of the four problem types and only solved 3% more than the worst on the other problem set. For the ET-JSPs, the reason for the large APRD for $Tdom$ is that it failed to improve on the initial upperbound for a number

Table 7.21: Analysis of algorithm components - OSP, JSP, SDST-JSP, ET-JSP

OSP					JSP				
Alg	APRD		% Opt	Time	Alg	APRD		% Opt	Time
	Best	Avg		Avg		Best	Avg		Avg
Ds3s	**0**	**0**	**100**	**74**	Luby	**0.09**	**0.92**	54	**595**
noDs	**0**	**0**	**100**	**74**	RandTop2	0.13	1.01	51	601
Default	**0**	**0**	**100**	94	RandTop3	0.27	1.08	51	602
noRand	**0**	-	**100**	96	Ds3s	0.31	1.14	53	596
LexVal	**0**	**0**	**100**	99	Ds300s	0.42	1.08	50	603
Promise	**0**	**0**	**100**	100	Tdom/Bwt	0.48	1.66	52	602
noRestart	**0**	**0**	**100**	103	Default	0.53	1.29	52	601
Luby	**0**	**0**	**100**	106	noDs	0.57	1.77	51	599
RandTop2	**0**	**0**	**100**	109	noNgd	0.63	1.31	50	610
RandTop3	**0**	**0**	**100**	114	noRand	0.68	-	**60**	596
Ds300s	**0**	**0**	**100**	148	Twt	2.85	3.93	50	606
noNgd	**0**	**0**	**100**	190	Promise	2.89	3.65	50	604
Bwt	**0**	0.06	99	167	LexVal	3.80	4.91	50	604
Tdom/Twt	**0**	**0**	90	180	noRestart	5.01	6.60	50	617
Twt	**0**	0.54	74	458	Bwt	6.09	6.90	50	608
Tdom	0.39	0.85	39	810	Tdom	9.41	10.25	39	804

SDST-JSP					ET-JSP				
Alg	APRD		% Opt	Time	Alg	APRD		% Opt	Time
	Best	Avg		Avg		Best	Avg		avg
RandTop3	**0.08**	0.94	**39**	795	RandTop2	**0.80**	9.76	69	427
RandTop2	0.10	1.07	38	792	Luby	1.64	**6.43**	69	429
Tdom/Twt	0.16	1.10	38	**780**	RandTop3	2.33	9.25	67	443
Default	0.16	**0.84**	35	818	Ds300s	2.35	11.20	**70**	447
Ds3s	0.18	1.06	38	794	Tdom/Bwt	2.47	13.17	60	516
Twt	0.25	1.40	31	844	Ds3s	2.48	14.31	68	404
Bwt	0.29	1.58	30	849	Default	3.73	9.49	69	434
noNgd	0.30	1.11	34	847	noNgd	4.74	9.91	50	600
Ds300s	0.31	0.97	35	824	Bwt	4.99	79.00	68	**398**
Luby	0.32	1.09	37	795	noRand	7.68	-	**70**	405
noDs	0.89	1.82	35	812	noDs	8.01	28.76	58	529
noRand	1.26	-	30	845	LexVal	15.58	23.69	**70**	413
LexVal	1.37	2.04	31	852	Promise	17.65	26.52	69	431
Promise	1.50	2.35	32	844	noRestart	61.81	220.92	63	508
noRestart	2.99	5.55	33	874	Twt	566.98	3447.30	34	803
Tdom	12.93	13.69	10	1082	Tdom	10016.34	55098.77	37	757

of runs on three of the five Beck instances, and in one case it didn't find a solution on any run.

For the JSP and its variants, solution guided value ordering, restarting and, to a lesser extent, randomization were also important to the performance of the algorithms. The other factors, such as dichotomic search and nogood recording, generally resulted in only minor gains.

The key factors for the open shop instances were nogood recording from restarts and, more importantly, the two variable heuristic components, bwt and $tdom$. We see that search guided by the weights on the Boolean variables alone solved every instance but one to optimality on every run. The poorest performance occurred when weights were ignored, with optimality proven on only four of the ten instances.

One further point of note for the OSPs is that dichotomic search with a relatively large cutoff (greater than 3 seconds) performed poorly. This was because, during dichotomic search, the algorithm repeatedly tries to prove infeasibility of makespans which are below the optimal C_{max}. On the other hand, a proof of infeasibility is only required once during the branch and bound phase (for (C_{max} -1)).

For problems involving time lag constraints, the ranking of the components was quite different, as shown in Table 7.22. The most obvious difference is that here the weighted degree component of the variable heuristic was much less important than the (tasks) domain size, indeed if anything it appeared to be detrimental to search. Similarly, solution guided value ordering and restarting appear to have much less influence on these problems.

The reason for the large differences in APRD on the TL-JSP instances is that a number of the algorithms failed to improve on the initial upper bound, either on a subset of runs or on all (see for example the best and average APRD for $noNgd$ and *noDs*).

One hypothesis to explain why $Tdom$ performed better than $Tdom/Bwt$ on the NW-JSP instances is that $Tdom$ concentrates search on neighboring variables. For these problems $Tdom$ would usually select the same pair of jobs until all Booleans between them are assigned, before moving on to the next pair of jobs. This is because assigning a Boolean variable between a pair of jobs will often

Table 7.22: Analysis of algorithm components - Time lag JSPs

TL-JSP					NW-JSP				
Alg	APRD		% Opt	Time	Alg	APRD		% Opt	Time
	Best	Avg		Avg		Best	Avg		avg
noRestart	**1.74**	**5.95**	**50**	719	Tdom/Twt	**0.19**	**2.29**	40	721
noNgd	2.49	126.55	46	726	Tdom/Bwt	1.30	4.50	40	728
noDs	2.53	166.44	49	674	Promise	2.85	16.56	**60**	540
Tdom	4.62	14.74	37	866	noDs	2.93	7.52	**60**	**526**
LexVal	8.73	182.12	49	680	noRestart	3.14	3.87	**60**	543
Ds300s	55.60	112.55	48	683	Tdom	3.38	3.83	**60**	553
Luby	56.07	65.70	48	690	RandTop2	3.38	3.80	**60**	547
Default	56.08	123.89	44	703	RandTop3	3.43	3.81	**60**	552
RandTop2	56.63	162.73	**50**	664	Default	3.62	3.84	**60**	552
Ds3s	57.35	182.19	48	687	Luby	3.63	3.87	**60**	545
RandTop3	57.50	168.37	49	672	LexVal	3.66	4.15	**60**	552
Promise	58.03	125.75	49	671	noRand	3.71	-	**60**	538
Tdom/Twt	188.42	230.42	**50**	654	Ds3s	3.74	4.14	**60**	528
noRand	189.87	-	**50**	**641**	Ds300s	3.79	4.19	**60**	603
Twt	190.62	272.14	46	727	noNgd	4.07	4.28	**60**	605
Bwt	260.47	261.17	43	743	Twt	5.42	8.43	40	792
					Bwt	6.88	10.21	40	798

Notes: Algorithms ranked by best APRD. Figures in bold indicate best result according to metric.

result in the greatest domain reduction on the unassigned Booleans between the same pair of jobs. Since this pair of jobs had the smallest $tdom$ at the previous choice point it is likely to be selected again.

On the other hand, as the weights grow, $Tdom/Bwt$ would be more likely to lead search to unconnected variables. In the case of $Tdom/Twt$, this effect would be less pronounced as its selections are restricted to pairs of jobs (all Booleans between the same pair of jobs will have the same ratio of $tdom$ to twt). However, when a Boolean variable is assigned, its weight is removed which may result in a Boolean from a different pair of jobs being selected subsequently.

We tested this hypothesis on the NW-JSP instances by using the heuristic $Tdom/Twt$ only to select a pair of jobs. Each Boolean variable between the pair of jobs would then be assigned before $Tdom/Twt$ selected the next pair of jobs. However, our results showed that this did not perform significantly better than $Tdom/Twt$ which contradicts the hypothesis. Furthermore, it is clear that while

the weights were detrimental for proving optimality, they did result in much better solutions on the harder instances.

Overall these results show that the variable heuristic is the key factor in our algorithm, although the relative importance of the two components of the heuristic varies depending on the problem type. Furthermore, it is clear that our algorithm could be improved further by fine tuning the various parameters for the different problem types. However, the goal of this work was to illustrate that a relatively simple generic method can be successfully applied to scheduling problems without the need for parameter tuning for each specific problem type.

7.11 Weight analysis

We generated weight profiles on a sample of instances for each problem type. The experimental setup involved running branch and bound search with a 30 second cutoff. Weights were stored after either the cutoff was reached or the instance was solved. The purpose of these experiments is to provide insight into the behaviour of search on the different problem sets and to ascertain whether there was a correlation between search performance and the level of discrimination amongst the variable weights.

7.11.1 Weight profiles for OSPs

We first analyzed the weight profiles generated on OSPs with the heuristic $Tdom/Bwt$, comparing the weight *increment* received by each variable using the Gini coefficient. The instances tested were the 20 largest instances from both Taillard (tai15-$*$, tai20-$*$) and Gueret-Prins sets (gp09-$*$, gp10-$*$), and the 26 largest Brucker instances ($j6per$-$*$, $j7per$-$*$, $j8per$-$*$). Boxplots of the results, in terms of the Gini coefficient of the weight increments, are shown in Figure 7.14.

We see that the Taillard and Gueret-Prins instances always yielded a Gini coefficient of greater than 0.83. The Brucker instances, on the other hand, had Gini coefficients ranging from 0.22 to 0.88, with a median value of 0.58. This means that there was a relatively small set of variables which were identified as the pri-

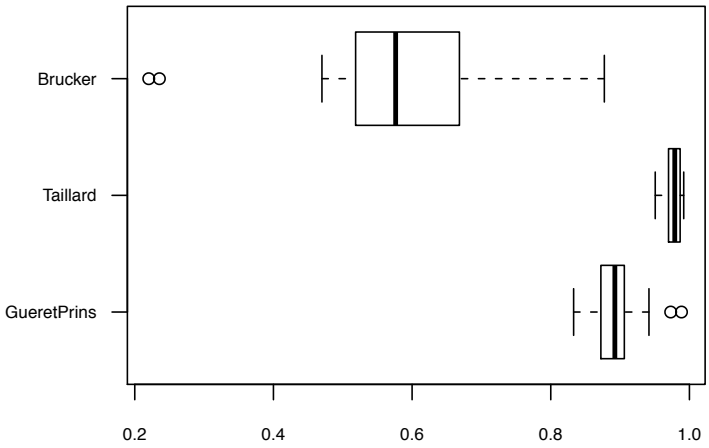


Figure 7.14: Boxplot representation of Gini coefficient of weight increment distribution for largest OSPs of the three sets.

mary sources of conflict for the Taillard and Gueret-Prins instances, whereas there were fewer clearly defined bottleneck variables in most of the Brucker instances.

However, there are a couple of caveats that should be noted. Firstly, optimality wasn't proven for 5 of the 26 Brucker instances, and thus these ran for the full 30 seconds, encountering many more failures. Indeed, the number of failures encountered during search was much less for the Taillard and Gueret-Prins instances than for the Brucker instances (the median number of failures for the latter was over twice that for either of the other two sets). Secondly, the Brucker instances involved at most 448 Boolean variables, compared to 900 for the largest Gueret-Prins and 7600 for the largest Taillard instances.

To illustrate the impact of the size of the problem on the Gini coefficient, let us consider the large Taillard instances. The median number of failures encountered on the 20 instances was 564. Suppose each of 564 failures occurred on a different variable in a 20x20 instance, this would still result in a Gini coefficient of 0.93 (564/7600 variables with a weight increment of 1). The other extreme, where all

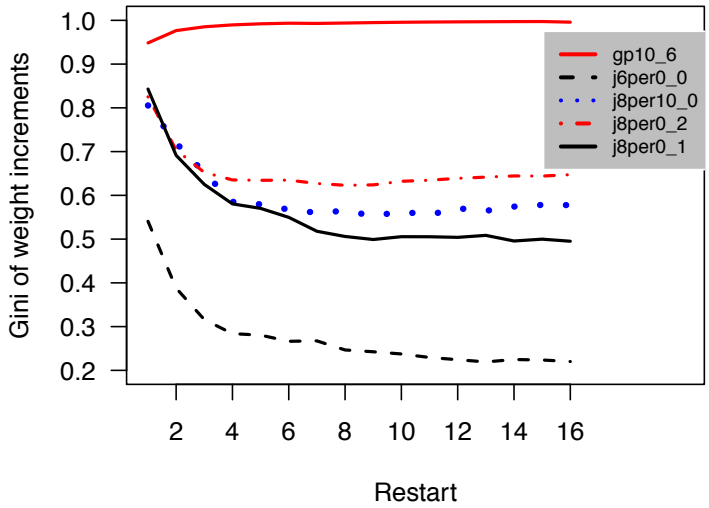


Figure 7.15: Evolution of weight increment distribution across restarts for sample OSPs (gp∗ is Gueret-Prins instance, j∗per∗ is Brucker instance).

564 failures occurred on the same variable, would obviously yield a Gini coefficient of 1. The range of the Gini coefficients on the ten $tai20*$ instances was 0.97 to 0.99, with a median of 0.987, so the weight distribution was clearly closer to amassing all weight on one variable than an even distribution of weight.

Note that if we calculated the Gini coefficient of the weighted degree of the variables (as opposed to the weight increment), then there would be a similar issue. If we consider the 20x20 case where all 564 failures occurred on the same variable, i.e. all other variables have a weighted degree of 1, the Gini coefficient would be 0.07 here instead of 1! This is mainly an issue when we calculate the Gini coefficient on problems after a small number of failures, as the number of failures increases the Gini coefficient of the weighted degrees and the weight increments converge.

Figure 7.15 illustrates the evolution of the distribution of weight increments across restarts for a sample of OSPs. These instances were chosen as there were sufficient failures to be of interest. The results again show that the heuristic found

it more difficult to identify bottleneck variables in the Brucker instances than in the Gueret-Prins instance. The important point is that there was a sharp decline in the Gini coefficient on all Brucker instances over the first three restarts (approximately 1000 failures). This refutes the hypothesis that the results in Figure 7.14 were due to a greater number of failures encountered on the Brucker instances.

Focusing on the instances GP10_6 and j8per0_2, we first note that search effort was similar on both instances (∼50K/60K failures). Analysis of the weight profiles on these two instances further corroborates the findings of Figure 7.15. Here only 33% of variables received any weight increment during search on $gp10_6$, compared to 85% of the variables in $j8per0_1$.

7.11.2 Weight profiles for the JSP and its variants

The results on the OSPs suggest that the weight discrimination is the reason for the good performance of $Tdom/Bwt$. We tested this hypothesis further on variants of the JSP, generating weight profiles with both $Tdom/Bwt$ and $Tdom/Twt$. The benchmarks tested were a sample of ten Lawrence instances la06-15, as these have SDST-JSP, TL-JSP, and NW-JSP variants, i.e. the same base JSP is used for each variant. Of particular interest is the weight discrimination on the NW-JSPs, as $Tdom/Bwt$ performed relatively poorly on these instances.

Figure 7.16 shows the boxplots of the Gini coefficients for the ten instances of the different variants, with $bwts$ generated with $Tdom/Bwt$ and $twts$ generated with $Tdom/Twt$. We include two sets of SDST-JSPs (denoted ps-$*$ and pss-$*$ in our earlier experiments, here *sds1* and *sds2* respectively), and two sets of TL-JSPs (for time lag β = 0, 1, denoted $jtl0$, $jtl1$ respectively in the figure). We do not include weight profiles for the base JSPs as these were solved in under 0.1 seconds in most cases, encountering few failures.

The clearest pattern observable in the figure illustrates the difference between the distribution of $bwts$ and $twts$ on the Boolean variables. The Gini coefficients of the $bwts$ were consistently much higher than those of the $twts$, while there was greater variation in the Gini coefficients of the $twts$ across instances for each variant except on the NW-JSP instances which were consistently low.

However, there was clearly no correlation between search performance and the

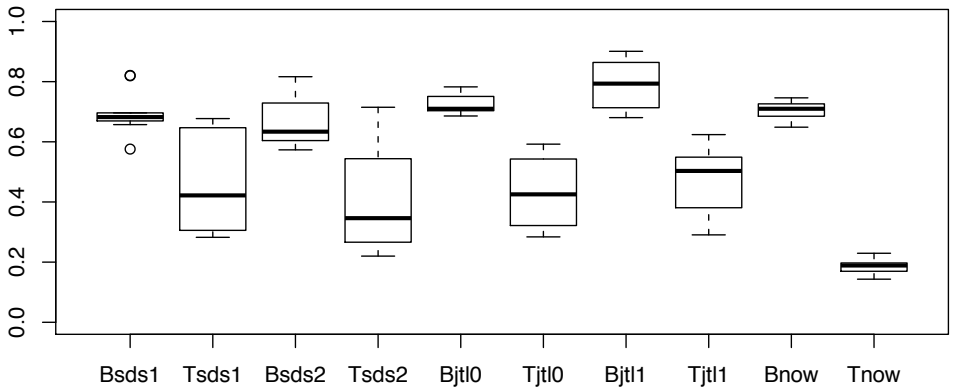


Figure 7.16: Gini coefficients of twt increment (T$*$) and bwt increment (B$*$) for different variants of sample Lawrence instances.

Gini coefficients in these experiments as $Tdom/Twt$ here performed much better (both in terms of proving optimality and objective found) than $Tdom/Bwt$ on both sets of SDST-JSP instances and on the NW-JSP instances (although the opposite was the case on the two TL-JSP sets). Indeed, in the previous section we showed that $Tdom/Bwt$ performed extremely poorly on the NW-JSP instances, yet we see that the level of discrimination of the weights was similar for bwt across all problem sets.

One interesting discovery we made during initial weight analysis on the OSPs, was that incrementing the weight on the global nogood constraint resulted in a deterioration in performance on some of the harder instances. This may prove counter intuitive at first glance, as all variables are involved in the constraint. To illustrate why this would occur, let us consider two variables a and b, where a has $taskdom$ of 30 and b has $taskdom$ of 35. Further let us first consider the case where the bwt of a is 10 and the bwt of b is 100. Here b will be selected ahead of a ($0.35 < 3$).

However, suppose there is a weight of 1000 on the global nogood constraint. This reduces the impact of differences between weights on the disjunctive con-

straints. In our example, with bwt of a now 1010 and bwt of b now 1100, a will be chosen ahead of b (0.0297 $<$ 0.0318). For this reason the nogood constraint was not weighted in our experiments (and indeed, based on this finding, the default black box solver of Mistral does not weight constraints with arity $> n/2$, where n is the number of variables in the problem instance).

7.12 Chapter Summary

In this chapter we introduced a new approach for solving disjunctive scheduling problems, combining relatively simple inference with a number of generic CP techniques such as restarting, weighted degree variable ordering, and solution guided value ordering. We took a minimalistic approach to modeling the problem, simply decomposing the unary resource constraints into primitive disjunctive constraints. In comparison, most CP techniques model the unary resource constraint as a global constraint, devising specialized filtering algorithms for the constraint.

We demonstrated the benefits of our approach on a variety of problems, and in so doing we have refuted the conventional wisdom that problem specific heuristics and, in particular, problem specific inference methods are necessary to achieve good performance on problems of this nature. The advantages of using a weighted degree based heuristic for these problems, and indeed in general, is twofold: it can identify critical variables without (a) costly calculations at each node and (b) costly inference methods.

We have further shown that our basic method can be easily adapted to handle a number of side constraints (setup times and maximum time constraints) and the objective of minimizing the earliness/tardiness costs. This is important as these side constraints and the alternative objective have proven troublesome for traditional CP methods due to their impact on the dedicated inference methods. Since our method is more search intensive than inference intensive, it suffers less from these issues.

However, our method cannot be considered to be completely generic as domain-specific knowledge was incorporated to good effect in certain cases. Firstly, the variable heuristic was improved by adding information regarding the domain sizes of the tasks, and in some cases weight information of the tasks. Secondly, our

method for solving the NW-JSP was improved by using a model specific to the problem based on the simple observation that each task in a job is functionally dependent on the other tasks of the job. This encoding has since been further strengthened by applying the concept of forbidden intervals. The addition of a dedicated metaheuristic for finding good initial upper bound has also been used to improve performance on the TL-JSPs.

These results show that there is still room for improvement through the combination of generic techniques with problem specific information. However, it should be noted that no problem specific propagators were employed by our approach. Indeed we showed that, for the OSPs, global constraints negated the ability of the variable heuristic to discriminate due to the large arity constraints.

One concern with our method is that the number of disjunctive constraints (and Boolean variables) is quadratic in n. In comparison, the most popular filtering algorithms for the unary resource constraint employ the Edge-Finding, Not-First/Not-Last and Detectable Precedence rules with a $O(n \log n)$ time complexity Vilím [211]. Therefore one would expect that our method would be less efficient as n grows. However this did not prove to be the case on the academic benchmarks tested.

Chapter 8

Conclusion and Future Work

The central thesis defended in this dissertation was the following:

Many constraint satisfaction problems contain globally difficult elements. These elements can be identified through preprocessing. This information can then be used to both solve problems efficiently and provide valuable feedback to the user.

We demonstrated this by applying our preprocessing techniques to a wide range of problems ranging from highly random to highly structured. Analysis of the weight profiles revealed that for many problems there were a small subset of variables which were the primary source of contention across the search space in the respective problem.

This dissertation addressed two general issues when solving a CSP. The first issue is that a user is often not an expert in the field, and thus will not know the best search method to use for a problem. We have shown that not only is combining the weighted degree heuristic directly with a restarting strategy one of the best generic methods for solving CSPs, it can even outperform specialized heuristics and reasoning techniques for some widely studied problems (Chapter 7). Here, our method was able to close a number of open problems, and find a large number of improved upper bounds. We further showed that the weight profiles generated can provide the user with valuable insight, both into the bottlenecks of the problem and into the behavior of conflict-directed search on the problem.

The second issue addressed was that of the impact of thrashing on the amount of search effort required to solve a problem, or prove that no solution exists. We

demonstrated that the approaches we proposed can greatly reduce search effort by identifying the critical variables through their consistent involvement in failures in disparate parts of the search space. Assigning these globally contentious variables at the top of the search tree resulted in improved search performance, in some cases it even resulted in backtrack-free search on difficult instances (e.g. Section 6.4).

8.1 Contributions

8.1.1 Boosting the weighted degree heuristic through information gathering

In this dissertation we investigated methods designed to boost conflict directed heuristics [29]. The motivation for this work was a combination of the lack of information available to the heuristic at the start of search, and the lack of information gathering in the randomized restarting approaches proposed by Gomes et al. [82]. These new methods were designed to have the dual benefit of improving both the conflict-directed heuristic and the randomized restarting approach.

We proposed two methods and empirically showed that both improved over standard search with $dom/wdeg$, for example achieving orders of magnitude improvement on a number of problems with insoluble cores, and solving 15-20% more open shop problems. We identified advantages and disadvantages to both. When the test problem had clearly defined structure, the approach whose preprocessing phase consisted of a number of short runs guided by the $dom/wdeg$ heuristic generally identified a subset of critical variables and concentrated search on this subset thereafter, shuffling the critical variables upon restart. However this shuffling of variables, which we refer to as variable convection, can also be disadvantageous when solving problems with less clearly defined sources of contention. In this case, the weights used on the run to completion may not reflect the globality of the sources of contention.

The second method we proposed, which randomly samples the search space for information, produced a more global view of the sources of contention in a problem than the previous method. This is particularly valuable as a feedback

mechanism to the user as it may be difficult to identify the problem bottlenecks through static analysis of a problem. The disadvantage of the random probing method, as evidenced on the balanced incomplete block design problems, is that highly symmetrical problems can result in symmetrically equivalent variables in unconnected components receiving a large weight. Search guided by these weights will then jump between unconnected components, whereas weights learnt by the heuristic in a localized context would have resulted in a more efficient search. One further disadvantage, compared to the previous method, is that a problem is much less likely to be solved during the preprocessing phase.

8.1.2 Blame assignment issue for constraint weighting

We investigated whether alternative measurements of contention would improve the heuristic. These measurements were designed to produce more discriminatory information and address the blame assignment issue. However none of these methods dominated the standard method of simply incrementing the weight on a constraint by one when the constraint caused a failure. (Similar results have been found by Balafoutis and Stergiou [11].) This suggests that it is the confluence of failures on the constraints of a variable that is important, rather than the "correct" apportioning of blame for individual failures. This was further supported by analysis of the failure depths, which showed that globally contentious variables received the most weight no matter the context of failures.

8.1.3 Alternative preprocessing techniques

We assessed the quality of information produced by some less expensive preprocessing methods, namely forward checking and the breakout local search technique. We compared these methods with MAC-based random probing and found that these were generally inferior. In particular, we showed that these methods were more susceptible to weighting local sources of contention.

8.1.4 Dynamic CSPs at the phase transition: impact of small changes

We analyzed the impact of small changes on CSPs at the phase transition and showed that, although a number of problem features are dramatically affected, the main sources of contention are relatively unaffected on the problems studied. The poor performance of traditional solution reuse approaches was primarily down to the impact of small changes on the minimal hamming distances between solutions of successive problems. Based on these findings, we introduced a new approach to solving DCSPs that reuses both solution and contention information and empirically showed the benefits of such an approach. In particular, we showed that in some cases this new method offered orders of magnitude reduction in search effort compared to a traditional solution reuse method, while maintaining a similar level of solution stability.

8.1.5 Analysis of various restarting strategies in combination with conflict-directed heuristics

An extensive experimental study with a number of restarting strategies was performed to assess which restarting strategy was best in general for combining with the weighted degree heuristic. The overall results confirmed that combining the weighted degree heuristic with a restarting strategy outperforms both non-restarting weighted degree search, and non-learning randomized restarting search. Geometric restarting and WTDI were found to be the best, with little overall difference in performance between the two. RNDI, on the other hand, performed relatively poorly compared to the methods which combined the heuristic directly with a restarting strategy. This was primarily due to one set of highly symmetrical instances, which did not contain globally contentious variables.

We provided detailed analysis for two problem types: open shop scheduling problems and radio link frequency assignment problems. The results illustrated that the weighted degree heuristic is extremely effective at solving both problem types when combined with a restarting strategy. We further showed that modeling the OSP with auxiliary variables to represent the disjuncts is much more effective

than the basic model of the problem.

8.1.6 Complementary performance between weighted degree and impact-based search

We compared the two heuristics (with and without restarting) on a large testbed of instances and found that the weighted degree heuristic was best both in terms of number of instance solved and runtime on commonly solved instances. However, we also found that for *both* heuristics, there were a number of instances for which the heuristic was orders of magnitude faster than the other. This shows the complementary nature of these two heuristics.

8.1.7 Ability of constraint weighting to identify the critical variables in unary resource scheduling problems

As mentioned above, one of the key discoveries of Chapter 6 was the efficiency of conflict directed heuristics, combined with restarting, in solving open shop scheduling problems. Indeed, throughout this dissertation we have found this to be the case. This led us to develop a new method for solving unary resource scheduling problems, which combines a number of generic CSP/COP search techniques such as the weighted degree heuristic, restarting, solution guided value ordering, etc., with a simple inference method (bounds consistency on pairwise disjunctive constraints). We demonstrated that our method can be easily and successfully adapted to handle additional side constraints, and the alternative objective of minimizing earliness/tardiness penalties.

The results contradicted the conventional wisdom that these scheduling problems require domain-specific heuristics and dedicated global constraints to be solved efficiently. Our method matched or outperformed the state-of-the-art systematic and local search techniques in most cases on a variety of problems, closing a number of open problems and improving on the best known solution for a large number of problems in these sets.

While the individual components of our algorithm are not particularly novel, we hope that this work will provide encouragement to the community to consider

similar techniques for other problem types. In particular, this work shows that (a) combining existing generic techniques can lead to interesting synthesis, where the performance is greater than the sum of the parts; and (b) creating a complex specialized approach to a problem can be unnecessary, sometimes a simple generic technique can also be the best.

8.2 Future Work

There are a number of research directions identified in this dissertation which warrant further exploration.

Random Probing. Correlation analysis of the weight profiles during the random probing phase could be used to assess whether global sources of contention have been identified. Low (top-down) correlations between weights after independent sets of 10 probes may indicate that the probing method is unsuited to the problem. On the other hand, high correlations would indicate that the probing method has identified the critical variables and thus further probing may be unnecessary.

Dynamic CSPs. Our contention and solution reuse method could be further enhanced with a reasoning reuse strategy. Alternatively, it would be interesting to investigate combining random probing with proactive techniques, both as a means of identifying the bottleneck variables which may prove troublesome after alteration, and for guiding search on the subsequent problem.

Combining weighted degree and impact-based search. The complementary nature of the results with these two heuristics, discussed above, suggest that a method of combining the two heuristics would provide the best general performance for solving CSPs. The simplest method would be to run both in parallel. An alternative which has been tried (Hebrard [102]) is to directly combine the two heuristic metrics by choosing the variable with minimum ratio of impact to weighted degree. However, this method generally did not result in improved performance over the weighted degree heuristic [†]. Of greater interest would be to

[†]Personal correspondence with Emmanuel Hebrard

investigate whether there are problem features that can be used to indicate which of the two heuristics is most suited for a given problem. Weight profile analysis after a fixed amount of search could be used to indicate whether the weighted degree heuristic is suited to the problem.

Scheduling. This work can be extended in two directions. Firstly, applying these generic techniques to other scheduling problems such as those with cumulative resource constraints, and indeed applying these techniques to applications outside of the scheduling domain. Secondly, the approaches for the problems studied may be further improved through the addition of domain-specific information (e.g. the addition of dedicated metaheuristics to initialize both the upper bound and the solution pool for the value ordering heuristic).

Bibliography

[1] A. Aamodt and E. Plaza. Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, 1994.

[2] J. Adams, E. Balas, and D. Zawack. The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science*, 34(3):391–401, 1988.

[3] M. I. Angles-Domínguez and H. Terashima-Marín. Stability Analysis for Dynamic Constraint Satisfaction Problems. In *3rd Mexican International Conference on Artificial Intelligence - MICAI'04. LNCS No. 2972*, pages 169–178. Springer, 2004.

[4] D. Applegate and W. J. Cook. A Computational Study of the Job-Shop Scheduling Problem. *INFORMS Journal on Computing*, 3(2):149–156, 1991.

[5] K. R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[6] C. Artigues and D. Feillet. A branch and bound method for the job-shop problem with sequence-dependent setup times. *Annals of Operations Research*, 159(1):135–159, 2008.

[7] C. Artigues, M-J. Huguet, and P. Lopez. Generalized disjunctive constraint propagation for solving the job shop problem with time lags. Technical Report N°10373, LAAS-CNRS, Toulouse, France, 2010. URL `http://hal.archives-ouvertes.fr/hal-00491986/fr/`.

[8] C. Artigues, M-J. Huguet, and P. Lopez. Generalized Disjunctive Constraint Propagation for Solving the Job Shop Problem with Time Lags. *Engineering Applications of Artificial Intelligence*, 24(2):220 – 231, 2011.

[9] F. Bacchus. Extending Forward Checking. In *Proceedings of Principles and Practice of Constraint Programming - CP'00. LNCS No. 1894*, pages 35–51, 2000.

[10] F. Bacchus and P. van Beek. On the Conversion between Non-Binary and Binary Constraint Satisfaction Problems. In *Proceedings of the 15th National Conference on Artificial Intelligence - AAAI'98*, pages 310–318, 1998.

[11] T. Balafoutis and K. Stergiou. Conflict Directed Variable Selection Strategies for Constraint Satisfaction Problems. In *6th Hellenic National Conference on Artificial Intelligence - SETN'10. LNCS No. 6040*, pages 29–38. Springer, 2010.

[12] T. Balafoutis and K. Stergiou. Evaluating and Improving Modern Variable and Revision Ordering Strategies in CSPs. *Fundamenta Informaticae*, 102: 229–261, 2010.

[13] E. Balas and A. Vazacopoulos. Guided Local Search with Shifting Bottleneck for Job-Shop Scheduling. *Management Science*, 44(2):262–275, 1998.

[14] E. Balas, N. Simonetti, and A. Vazacopoulos. Job shop scheduling with setup times, deadlines and precedence constraints. *Journal of Scheduling*, 11(4):253–262, 2008.

[15] L. Baptista, I. Lynce, and J. Marques-Silva. Complete search restart strategies for satisfiability. In *Proceedings of the IJCAI Workshop on Stochastic Search Algorithms (IJCAI-SSA)*, 2001.

[16] P. Baptiste and C. Le Pape. Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling. In *Proceedings of the 15th Workshop of the U.K. Planning Special Interest Group*, 1996.

[17] P. Baptiste, C. Le Pape, and W Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming Techniques to Scheduling Problems*. Kluwer Academic Publishers, 2001.

[18] P. Baptiste, M. Flamini, and F. Sourd. Lagrangian bounds for just-in-time job-shop scheduling. *Computers & Operations Research*, 35(3):906–915, 2008.

[19] R. Barták, T. Müller, and H. Rudová. A New Approach to Modeling and Solving Minimal Perturbation Problems. In *Recent Advances in Constraints, Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2003, LNAI No.3010*, pages 233–249, 2003.

[20] A. Beacham, X. Chen, J. Sillito, and P. van Beek. Constraint Programming Lessons Learned from Crossword Puzzles. In *Proceedings of the 14th Canadian Conference on Artificial Intelligence*, pages 78–87, 2001.

[21] J. C. Beck. Solution-Guided Multi-Point Constructive Search for Job Shop Scheduling. *Journal of Artificial Intelligence Research*, 29:49–77, 2007.

[22] J. C. Beck and P. Refalo. A Hybrid Approach to Scheduling with Earliness and Tardiness Costs. *Annals of Operations Research*, 118(1-4):49–71, 2003.

[23] J. C. Beck, A. J. Davenport, E. M. Sitarski, and M. S. Fox. Texture-Based Heuristics for Scheduling Revisited. In *Proceedings of the 14th National Conference on Artificial Intelligence - AAAI'97*, pages 241–248, 1997.

[24] J. C. Beck, P. Prosser, and R. J. Wallace. Variable Ordering Heuristics Show Promise. In *Proceedings of Principles and Practice of Constraint Programming - CP'04. LNCS No. 3258*, pages 711–715, 2004.

[25] A. Bellicha. Maintenance of Solution in a Dynamic Constraint Satisfaction Problem. In *Proceedings of the Applications of Artificial Intelligence in Engineering VIII*, pages 261–274, 1993.

[26] C. Bessière. Constraint Propagation. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*. Elsevier, 2006.

[27] C. Bessière and J-C. Régin. MAC and Combined Heuristics: Two Reasons to Forsake FC (and CBJ?) on Hard Problems. In *Proceedings of Principles and Practice of Constraint Programming - CP'96. LNCS No. 1118*, pages 61–75, 1996.

[28] C. Blum. Beam-ACO - hybridizing ant colony optimization with beam search: an application to open shop scheduling. *Computers & Operations Research*, 32:1565–1591, 2005.

[29] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting Systematic Search by Weighting Constraints. In *Proceedings of the 16th European Conference on Artificial Intelligence - ECAI'04*, pages 146–150, 2004.

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

[31] D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[32] J. L. Bresina. Heuristic-Biased Stochastic Sampling. In *Proceedings of the 13th National Conference on Artificial Intelligence - AAAI'96*, pages 271–278, 1996.

[33] P. Brucker. Scheduling and constraint propagation. *Discrete Applied Mathematics*, 123(1-3):227–256, 2002.

[34] P. Brucker and O. Thiele. A branch and bound method for the general-shop problem with sequence-dependent setup times. *Operation Research Spektrum*, 18:145–161, 1996.

[35] P. Brucker, J. Hurink, B. Jurisch, and B. Wöstmann. A branch & bound algorithm for the open-shop problem. *Discrete Applied Mathematics*, 76 (1-3):43–59, 1997.

[36] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[37] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J. P. Warners. Radio Link Frequency Assignment. *Constraints*, 4(1):79–89, 1999.

[38] H. Cambazard and N. Jussien. Identifying and Exploiting Problem Structures Using Explanation-based Constraint Programming. *Constraints*, 11 (4):295–313, 2006.

[39] J. Carlier. Ordonnancements à contraintes disjonctives. *R.A.I.R.O Recherche opérationelle / Operations Research*, 12:333–350, 1978.

[40] J. Carlier and E. Pinson. An Algorithm for Solving the Job-shop Problem. *Management Science*, 35(2):164–176, 1989.

[41] J. Carlier and E. Pinson. Adjustment of Heads and Tails for the Job-Shop Problem. *European Journal of Operations Research*, 78:146–191, 1994.

[42] Y. Caseau and F. Laburthe. Improved CLP Scheduling with Task Intervals. In *ICLP*, pages 369–383, 1994.

[43] A. Caumond. *Le problème de jobshop avec contraintes: modélisation et optimisation*. PhD thesis, Universite Blaise Pascal - Clermont Ferrand II, 2006.

[44] A. Caumond, P. Lacomme, and N. Tchernev. A memetic algorithm for the job-shop with time-lags. *Computers & Operations Research*, 35(7):2331–2356, 2008.

[45] K. C. K. Cheng and R. H. C. Yap. Maintaining Generalized Arc Consistency on Ad-Hoc n-Ary Boolean Constraints. In *Proceedings of the 17th European Conference on Artificial Intelligence - ECAI'06*, pages 78–82, 2006.

[46] The choco team. choco: an Open Source Java Constraint Programming Library. In *The Third International CSP Solver Competition*, pages 31–40, 2008.

[47] C. W. Choi, W. Harvey, J. H. M. Lee, and P. J. Stuckey. Finite Domain Bounds Consistency Revisited. In *Proceedings of the 19th Australian Joint Conference on Artificial Intelligence - AI'06. LNCS No. 4304*, pages 49–58. Springer, 2006.

[48] V. A. Cicirello. *Boosting Stochastic Problem Solvers Through Online Self-Analysis of Performance*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 2003. CMU-RI-TR-03-27.

[49] M. Correia and P. Barahona. On the Integration of Singleton Consistencies and Look-Ahead Heuristics. In *Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming-CSCLP'07*, pages 62–75, 2007.

[50] J. M. Crawford and A. B. Baker. Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems. In *Proceedings of the 12th National Conference on Artificial Intelligence - AAAI'94*, pages 1092–1097, 1994.

[51] E. Danna and L. Perron. Structured vs. Unstructured Large Neighborhood Search: A Case Study on Job-Shop Scheduling Problems with Earliness and Tardiness Costs. In *Proceedings of Principles and Practice of Constraint Programming - CP'03. LNCS No. 2833*, pages 817–821, 2003.

[52] E. Danna and L. Perron. Structured vs. Unstructured Large Neighborhood Search: A Case Study on Job-Shop Scheduling Problems with Earliness and Tardiness Costs. Technical report, ILOG, 2003.

[53] E. Danna, E. Rothberg, and C. Le Pape. Integrating Mixed Integer Programming and Local Search: A Case Study on Job-Shop Scheduling Problems. In *5th International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems-CPAIOR'03*, 2003.

[54] R. Debruyne and C. Bessière. Some Practicable Filtering Techniques for the Constraint Satisfaction Problem. In *Proceedings of the 15th Interna-*

tional Joint Conference on Artificial Intelligence - IJCAI'97, pages 412–417, 1997.

[55] R. Dechter. Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition. *Artificial Intelligence*, 41(3): 273–312, 1990.

[56] R. Dechter and A. Dechter. Belief Maintenance in Dynamic Constraint Networks. In *Proceedings of the 7th National Conference on Artificial Intelligence - AAAI'88*, pages 37–42, 1988.

[57] R. Dechter, I. Meiri, and J. Pearl. Temporal Constraint Networks. *Artificial Intelligence*, 49(1-3):61–95, 1991.

[58] J. Denzinger, M. Fuchs, C. Goller, and S. Schulz. Learning from Previous Proof Experience. Technical Report AR99-4, Institut für Informatik, Technische Universität München, 1999. (also to be published as a SEKI report).

[59] U. Dorndorf, E. Pesch, and T. Phan Huy. Solving the open shop scheduling problem. *Journal of Scheduling*, 4(3):157–174, 2001.

[60] C. Eisenberg and B. Faltings. Using the Breakout Algorithm to Identify Hard and Unsolvable Subproblems. In *Proceedings of Principles and Practice of Constraint Programming - CP'03. LNCS No. 2833*, pages 822–826, 2003.

[61] A. Elkhyari, C. Guéret, and N. Jussien. Solving Dynamic Resource Constraint Project Scheduling Problems Using New Constraint Programming Tools. In E. K. Burke and P. De Causmaecker, editors, *Practice and Theory of Automated Timetabling IV, 4th International Conference, PATAT-02*, pages 39–62. Springer, 2002.

[62] V. Ferreira and J. Thornton. Longer-term memory in clause weighting local search for SAT. *AI 2004: Advances in Artificial Intelligence*, pages 730–741, 2005.

[63] H. Fisher and G. L. Thompson. Probabilistic Learning Combinations of Local Job-shop Scheduling Rules. *Industrial Scheduling*, pages 225–251, 1963.

[64] M. S. Fox, N. M. Sadeh, and C. A. Baykan. Constrained Heuristic Search. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - IJCAI'89*, pages 309–315, 1989.

[65] J. M. Framinan and C. J. Schuster. An enhanced timetabling procedure for the no-wait job shop problem: a complete local search approach. *Computers & Operations Research*, 33:1200–1213, 2006.

[66] E. C. Freuder. Eliminating Interchangeable Values in Constraint Satisfaction Problems. In *Proceedings of the 9th National Conference on Artificial Intelligence - AAAI'91*, pages 227–233, 1991.

[67] A. M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, and T. Walsh. Propagation algorithms for lexicographic ordering constraints. *Artificial Intelligence*, 170(10):803–834, 2006.

[68] D. Frost, I. Rish, and L. Vila. Summarizing CSP Hardness with Continuous Probability Distributions. In *Proceedings of the 14th National Conference on Artificial Intelligence - AAAI'97*, pages 327–333, 1997.

[69] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[70] M. R. Garey, D. S. Johnson, and R.Sehti. The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[71] P. A. Geelen. Dual Viewpoint Heuristics for Binary Constraint Satisfaction Problems. In *Proceedings of the 10th European Conference on Artificial Intelligence - ECAI'92*, pages 31–35, 1992.

[72] I. P. Gent, E. MacIntyre, P. Prosser, B. M. Smith, and T. Walsh. Random constraint satisfaction: Flaws and structure. *Constraints*, 6:345–372, 2001.

[73] L. Getoor, G. Ottosson, M. P. J. Fromherz, and B. Carlson. Effective Redundant Constraints for Online Scheduling. In *Proceedings of the 14th National Conference on Artificial Intelligence - AAAI'97*, pages 302–307, 1997.

[74] D. Ghosh and G. Sierksma. Complete local search with memory. *Journal of Heuristics*, 8:571–584, 2002.

[75] C. Gini. Variabilita e mutabilita contributo allo studio della distribuzioni. *Studie Economico-Guiridici della R. Universita di Cagliari*, 1912.

[76] M. L. Ginsberg. Dynamic Backtracking. *Journal of Artificial Intelligence Research (JAIR)*, 1:25–46, 1993.

[77] M. L. Ginsberg. Dynamic Backtracking. *Journal of Artificial Intelligence Research (JAIR)*, 1:25–46, 1993.

[78] M. L. Ginsberg, M. Frank, M. P. Halpin, and M. C. Torrance. Search Lessons Learned from Crossword Puzzles. In *Proceedings of the 8th National Conference on Artificial Intelligence - AAAI'90*, pages 210–215, 1990.

[79] F. Glover and M. Laguna. *TABU search*. Kluwer, 1999.

[80] C. P. Gomes. Randomized Backtrack Search. In M. Milano, editor, *Constraint and Integer Programming: Toward a Unified Methodology*, chapter 8, pages 233–292. Kluwer Academic Publishers, 2003.

[81] C. P. Gomes and T. Walsh. Randomness and Structure. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 18. Elsevier, 2006.

[82] C. P. Gomes, B. Selman, and H. A. Kautz. Boosting Combinatorial Search Through Randomization. In *Proceedings of the 15th National Conference on Artificial Intelligence - AAAI'98*, pages 431–437, 1998.

[83] C. P. Gomes, B. Selman, N. Crato, and H. A. Kautz. Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems. *Journal of Automated Reasoning*, 24(1/2):67–100, 2000.

[84] T. Gonzales and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the Association for Computing Machinery*, 23(4):665–679, 1976.

[85] M. A. González, C. R. Vela, and R. Varela. A New Hybrid Genetic Algorithm for the Job Shop Scheduling Problem with Setup Times. In J. Rintanen, B. Nebel, J. C. Beck, and E. A. Hansen, editors, *ICAPS*, pages 116–123. AAAI, 2008.

[86] M. A. González, C. R. Vela, and R. Varela. Genetic Algorithm Combined with Tabu Search for the Job Shop Scheduling Problem with Setup Times. In *IWINAC (1). LNCS No. 5601*, pages 265–274, 2009.

[87] R. E. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[88] É. Grégoire, B. Mazure, and C. Piette. Extracting MUSes. In *Proceedings of the 17th European Conference on Artificial Intelligence - ECAI'06*, pages 387–391, 2006.

[89] D. Grimes. A study of adaptive restarting strategies for solving constraint satisfaction problems. In *Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science - AICS'08.*, pages 33–42, 2008.

[90] D. Grimes and E. Hebrard. Models and Strategies for Variants of the Job Shop Scheduling Problem. In *Proceedings of Principles and Practice of Constraint Programming - CP'11. LNCS No. 6876*, pages 356–372. Springer, 2011.

[91] D. Grimes and R. J. Wallace. Learning to identify global bottlenecks in constraint satisfaction search. In *20th Conference of the Florida Artificial Intelligence Research Society-FLAIRS'07. AAAI Press*, pages 592–597, 2007.

[92] D. Grimes and R. J. Wallace. Sampling Strategies and Variable Selection in Constraint Satisfaction Search. In *Proceedings of Principles and Practice of Constraint Programming - CP'07. LNCS No. 4741*, pages 831–838, 2007.

[93] V. P. Guddeti and B. Y. Choueiry. Characterization of a New Restart Strategy for Randomized Backtrack Search. In *Recent Advances in Constraints. Papers from the 2004 ERCIM/CologNet Workshop-CSCLP 2004. LNAI No. 3419*, pages 56–70, 2005.

[94] C. Guéret and C. Prins. Forbidden Intervals for Open-Shop Problems. Technical report, École Des Mines de Nantes, 1998.

[95] C. Guéret and C. Prins. A new lower bound for the Open-Shop problem. *Annals of Operations Research*, 92:165–183, 1999.

[96] C. Guéret, N. Jussien, and C. Prins. Using intelligent backtracking to improve branch-and-bound methods: An application to open-shop problems. *Journal of Operations Research*, 127(2):344–354, 2000.

[97] R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–314, 1980.

[98] H. H. Harman. *Modern Factor Analysis*. University of Chicago, Chicago and London, 2nd edition, 1967.

[99] W. D. Harvey. *Nonsystematic backtracking search*. PhD thesis, Stanford University, Stanford, CA, USA, 1995. UMI Order No. GAX95-25840.

[100] W. D. Harvey and M. L. Ginsberg. Limited Discrepancy Search. In Chris S. Mellish, editor, *Proceedings of the 45th International Joint Conference on Artificial Intelligence - IJCAI'95*, pages 607–615, Montréal, Québec, Canada, August 20-25 1995. Morgan Kaufmann.

[101] W. L. Hays. *Statistics for the Social Sciences*. Holt, Rinehart, Winston, 2nd edition, 1973.

[102] E. Hebrard. Mistral, a Constraint Satisfaction Library. In *The Third International CSP Solver Competition*, pages 31–40, 2008.

[103] E. Hebrard, B. Hnich, and T. Walsh. Super Solutions in Constraint Programming. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems-CPAIOR'04. LNCS No. 3011*, pages 157–172, 2004.

[104] I. Heckman. Empirical analysis of solution guided multi-point constructive search. Master's thesis, University of Toronto, 2007.

[105] F. Hemery, C. Lecoutre, L. Sais, and F. Boussemart. Extracting MUCs from constraint networks. In *Proceedings of the 17th European Conference on Artificial Intelligence - ECAI'06*, pages 113–117, 2006.

[106] A. Hodson, A. P. Muhlemann, and D. H. R. Price. A Microcomputer Based Solution to a Practical Scheduling Problem. *The Journal of the Operational Research Society*, 36(10):903–914, 1985.

[107] E. Hopper. *Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods*. PhD thesis, University of Wales, Cardiff School of Engineering, May 2000.

[108] J. Huang. The Effect of Restarts on the Efficiency of Clause Learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence - IJCAI'07*, pages 2318–2323, 2007.

[109] M-J. Huguet, P. Lopez, and W. Karoui. Weight-based heuristics for constraint satisfaction and combinatorial optimization problems. *Journal of Mathematical Modelling and Algorithms*, 11(2):193–215, 2012.

[110] T. Hulubei. *Refutation Analysis for Constraint Satisfaction Problems*. PhD thesis, National University of Ireland, Cork, 2007.

[111] J. Hwang and D. G. Mitchell. 2-way vs. d-way branching for csp. In *Proceedings of Principles and Practice of Constraint Programming - CP'05. LNCS No. 3709*, pages 343–357, 2005.

[112] R. L. Iman and W. J. Conover. A measure of top-down correlation. *Technometrics*, pages 351–357, 1987.

[113] D. S. Johnson and M. A. Trick, editors. *Second DIMACS implementation challenge : cliques, coloring and satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1996. URL `http://dimacs.rutgers.edu/Challenges/`.

[114] S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.

[115] D. Joslin and D. P. Clements. ”Squeaky Wheel” Optimization. In *Proceedings of the 15th National Conference on Artificial Intelligence - AAAI'98*, pages 340–346, 1998.

[116] D. Joslin and D. P. Clements. Squeaky Wheel Optimization. *Journal of Artificial Intelligence Research (JAIR)*, 10:353–373, 1999.

[117] R. J. Bayardo Jr. and R. Schrag. Using CSP Look-Back Techniques to Solve Real-World SAT Instances. In *Proceedings of the 14th National Conference on Artificial Intelligence - AAAI'97*, pages 203–208, 1997.

[118] U. Junker. QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. In *Proceedings of the 19th National Conference on Artificial Intelligence - AAAI'04*, pages 167–172, 2004.

[119] N. Jussien and V. Barichard. The PaLM system: explanation-based constraint programming. In *Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, pages 118–133, 2000.

[120] N. Jussien and O. Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1):21–45, 2002.

[121] W. Karoui, M-J. Huguet, P. Lopez, and W. Naanaa. YIELDS: A Yet Improved Limited Discrepancy Search for CSPs. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems-CPAIOR'07. LNCS No. 4510*, pages 99–111, 2007.

[122] W. Karoui, M. J. Huguet, P. Lopez, and M. Haouari. Climbing discrepancy search for flowshop and jobshop scheduling with time lags. *Electronic Notes in Discrete Mathematics*, 36:821–828, 2010.

[123] G. Katsirelos and F. Bacchus. Generalized NoGoods in CSPs. In *Proceedings of the 20th National Conference on Artificial Intelligence - AAAI'93*, pages 390–396, 2005.

[124] J. Kelbel and Z. Hanzálek. Solving production scheduling with earliness/tardiness penalties by constraint programming. *Journal of Intelligent Manufacturing*, 22(4):553–562, 2011.

[125] S. Kirkpatrick, D. Gelatt Jr., and M. P. Vecchi. Optimization by Simmulated Annealing. *Science*, 220(4598):671–680, 1983.

[126] P. Laborie. Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and new Results. *Artificial Intelligence*, 143(2):151–188, 2003.

[127] P. Laborie. Complete MCS-based search: Application to Resource Constrained Project Scheduling. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence - IJCAI'05*, pages 181–186, 2005.

[128] P. Langley. Systematic and nonsystematic search strategies. In *Proceedings of the first international conference on Artificial intelligence planning systems*, pages 145–152, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers, Inc.

[129] P. Langley. *Elements of Machine Learning*. Morgan Kaufmann, 1994.

[130] S. Lawrence. Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (Supplement). *Gradu-*

ate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania., 1984.

[131] C. Lecoutre. *Constraint Networks: Techniques and Algorithms*. International Scientific and Technical Encyclopedia (ISTE Ltd), John Wiley Inc., 592 pages, June 2009.

[132] C. Lecoutre and S. Tabary. Abscon 109: a generic CSP solver. In *Proceedings of the 2006 CSP Solver Competition*, pages 55–63, 2007.

[133] C. Lecoutre, F. Boussemart, and F. Hemery. Backjump-Based Techniques versus Conflict-Directed Heuristics. In *Proceedings of the 16th International Conference on Tools with Artificial Intelligence, ICTAI'04*, pages 549–557, 2004.

[134] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Last Conflict Based Reasoning. In *Proceedings of the 17th European Conference on Artificial Intelligence - ECAI'06*, pages 133–137, 2006.

[135] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Nogood recording from restarts. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence - IJCAI'07*, pages 131–136, 2007.

[136] C. Lecoutre, L. Sais, and J. Vion. Using SAT Encodings to derive CSP value ordering heuristics. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 1:169–186, 2007.

[137] C. Lecoutre, O. Roussel, and M. R. C. van Dongen. Promoting robust black-box solvers through competitions. *Journal of Constraints*, 15(3): 317–326, 2010.

[138] M. Liffiton, M. N. Mneimneh, I. Lynce, Z. S. Andraus, J. Marques-Silva, and K. A. Sakallah. A branch and bound algorithm for extracting smallest minimal unsatisfiable subformulas. *Constraints*, 14:415–442, 2009.

[139] M. Lombardi, M. Milano, A. Roli, and A. Zanarini. Deriving information from Sampling and Diving. *Fundam. Inform.*, 107(2-3):267–287, 2011.

[140] M. Luby, A. Sinclair, and D. Zuckerman. Optimal Speedup of Las Vegas Algorithms. In *Israel Symposium on Theory of Computing Systems*, pages 128–133, 1993.

[141] A. K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[142] A. Malapert, H. Cambazard, C. Guéret, N. Jussien, A. Langevin, and L-M. Rousseau. An Optimal Constraint Programming Approach to the Open-Shop problem. *submitted to INFORMS, Journal of Computing*, 2008.

[143] D. L. Mammen and T. Hogg. A New Look at the Easy-Hard-Easy Pattern of Combinatorial Search Difficulty. *Journal of Artificial Intelligence Research (JAIR)*, 7:47–66, 1997.

[144] J. P. Marques-Silva and K. A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, 1996.

[145] P. Martin and D. B. Shmoys. A New Approach to Computing Optimal Schedules for the Job-Shop Scheduling Problem. In *5th International Conference on Integer Programming and Combinatorial Optimization-IPCO'96. LNCS No. 1084*, pages 389–403, 1996.

[146] A. Mascis and D. Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498–517, 2002.

[147] B. Mazure, L. Sais, and É. Grégoire. Boosting Complete Techniques Thanks to Local Search Methods. *Annals of Mathematics and Artificial Intelligence*, 22(3-4):319–331, 1998.

[148] Q. McNemar. *Psychological Statistics*. John Wiley, New York, 4th edition, 1969.

[149] I. Meiri. Combining Qualitative and Quantitative Constraints in Temporal Reasoning. In *Proceedings of the 9th National Conference on Artificial Intelligence - AAAI'91*, pages 260–267, 1991.

[150] D. Metha and M. R. C. van Dongen. Static Value Ordering Heuristics for Constraint Satisfaction Problems. In M. R. C. van Dongen, editor, *Proceedings of the Second International Workshop on Constraint Propagation And Implementation*, pages 49–62, 2005.

[151] L. D. Michel and P. Van Hentenryck. Activity-Based Search for Black-Box Contraint-Programming Solvers. *CoRR*, abs/1105.6314, 2011.

[152] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[153] P. Morris. The Breakout Method for Escaping from Local Minima. In *Proceedings of the 11th National Conference on Artificial Intelligence - AAAI'93*, pages 40–45, 1993.

[154] T. E. Morton and D. W. Pentico. *Heuristic Scheduling Systems*. John Wiley and Sons, 1993.

[155] M. Moskewicz, C. Madigan, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the Design Automation Conference*, pages 530–535, 2001.

[156] T. Müller, R. Barták, and H. Rudová. Conflict-based statistics. In *J. Gottlieb, D. Landa Silva, N. Musliu, and E. Soubeiga, editors, EU/ME Workshop on Design and Evaluation of Advanced Hybrid Meta-Heuristics. University of Nottingham*, 2004.

[157] E. Nowicki and C. Smutnicki. A Fast Taboo Search Algorithm for the Job Shop Problem. *Manage. Sci.*, 42(6):797–813, 1996.

[158] E. Nowicki and C. Smutnicki. An Advanced Tabu Search Algorithm for the Job Shop Problem. *Journal of Scheduling*, 8(2):145–159, 2005.

[159] W. Nuijten. *Time and Resource Constraint Scheduling: A Constraint Satisfaction Approach*. PhD thesis, Eindhoven University of Technology, 1994.

[160] C. H. Papadimitriou and D. Wolfe. The Complexity of Facets Resolved. *Journal of Computer and System Sciences*, 37(1):2–13, 1988.

[161] C. Le Pape. Implementation of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint-Based Scheduling Systems. *Intelligent Systems Engineering*, 3:55–66, 1994.

[162] K. Pipatsrisawat and A. Darwiche. A Lightweight Component Caching Scheme for Satisfiability Solvers. In *Proceedings of Theory and Applications of Satisfiability Testing - SAT'07. LNCS No. 4501*, pages 294–299. Springer, 2007.

[163] P. Prosser. Hybrid Algorithms for the Constraint Satisfaction Problem. *Computational Intelligence*, 9:268–299, 1993.

[164] P. Prosser. Domain Filtering can Degrade Intelligent Backtracking Search. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence - IJCAI'93*, pages 262–267, 1993.

[165] P. Prosser, K. Stergiou, and T. Walsh. Singleton Consistencies. In *Proceedings of Principles and Practice of Constraint Programming - CP'00. LNCS No. 1894*, pages 353–368, 2000.

[166] W. H. M. Raaymakers and J. A. Hoogeveen. Scheduling multipurpose batch process industries with no-wait restrictions by simulated annealing. *European Journal of Operational Research*, 126(1):131 – 151, 2000.

[167] C. Rajendran. A No-Wait Flowshop Scheduling Heuristic to Minimize Makespan. *The Journal of the Operational Research Society*, 45(4):472–478, 1994.

[168] Y. Ran, N. Roos, and J. van den Herik. Approaches to find a nearminimal change solution for dynamic CSPs. In *4th International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems-CPAIOR'02*, pages 373–387, 2002.

[169] P. Refalo. Impact-Based Search Strategies for Constraint Programming. In *Proceedings of Principles and Practice of Constraint Programming - CP'04. LNCS No. 3258*, pages 557–571, 2004.

[170] J-C. Régin. A Filtering Algorithm for Constraints of Difference in CSPs. In *Proceedings of the 12th National Conference on Artificial Intelligence - AAAI'94*, pages 362–367, 1994.

[171] E. T. Richards and B. Richards. Nonsystematic Search and No-Good Learning. *Journal of Automated Reasoning*, 24(4):483–533, 2000.

[172] B. Roy and M. A. Sussman. Les problèmes d'ordonnancement avec contraintes disjonctive. Technical Report Note DS No.9 bis, SEMA, Paris, 1964.

[173] S. Russel and P. Norvig. *Artificial Intelligence: a modern approach*. Prentice Hall, New York, 1995.

[174] D. Sabin and E. C. Freuder. Contradicting Conventional Wisdom in Constraint Satisfaction. In *Proceedings of the 11th European Conference on Artificial Intelligence - ECAI'94*, pages 125–129, 1994.

[175] N. M. Sadeh. *Lookahead techniques for micro-opportunistic job-shop scheduling*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1991.

[176] N. M. Sadeh and M. S. Fox. Variable and Value Ordering Heuristics for the Job Shop Scheduling Constraint Satisfaction Problem. *Artificial Intelligence*, 86(1):1–41, 1996.

[177] S. Sahni and Y. Cho. Complexity of Scheduling Shops with No Wait in Process. *Mathematics of Operations Research*, 4(4):448–457, 1979.

[178] H. El Sakkout and M. Wallace. Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling. *Constraints*, 5(4):359–388, 2000.

[179] I. R. Savage. Contributions to the theory of rank order statistics-the two-sample case. *The Annals of Mathematical Statistics*, 27(3):590–615, 1956.

[180] T. Schiex and G. Verfaillie. Nogood Recording for Static and Dynamic Constraint Satisfaction Problem. In *Proceedings of the 5th International Conference on Tools with Artificial Intelligence, ICTAI'04)*, pages 48–55, 1993.

[181] C. J. Schuster. No-wait job shop scheduling: Tabu search and complexity of problems. *Mathamatical Methods of Operations Research*, 63:473–491, 2006.

[182] C. J. Schuster and J. M. Framinan. Approximative procedures for no-wait job shop scheduling. *Operations Research Letters*, 31(3):308–318, 2003.

[183] B. Selman and H. Kautz. Domain-independent extensions to GSAT : solving large structured satisfiability problems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence - IJCAI'93*, pages 290–295, 1993.

[184] B. Selman, H. J. Levesque, and D. G. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proceedings of the 10th National Conference on Artificial Intelligence - AAAI'92*, pages 440–446, 1992.

[185] D. Y. Sha and C-Y. Hsu. A new Particle Swarm Optimization for the Open Shop Scheduling Problem. *Computers & Operations Research*, 35(10): 3243–3261, 2008.

[186] P. Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In *Proceedings of Principles and Practice of Constraint Programming - CP'98. LNCS No. 1520*, pages 417–431, 1998.

[187] D. Sleeman, P. Langley, and T. M. Mitchell. Learning from solution paths: An approach to the credit assignment problem. *AI Magazine*, 3:48–52, 1982.

[188] B. M. Smith. The Brélaz Heuristic and Optimal Static Orderings. In *Proceedings of Principles and Practice of Constraint Programming - CP'99. LNCS No. 1713*, pages 405–418, 1999.

[189] B. M. Smith and S. A. Grant. Trying Harder to Fail First. In *Proceedings of the 13th European Conference on Artificial Intelligence - ECAI'98*, pages 249–253, 1998.

[190] S. F. Smith and C-C. Cheng. Slack-Based Heuristics for Constraint Satisfaction Scheduling. In *Proceedings of the 11th National Conference on Artificial Intelligence - AAAI'93*, pages 139–144, 1993.

[191] G. W. Snedecor and W. G. Cochran. *Statistical Methods*. Iowa State, Ames, 7th edition, 1969.

[192] N. Sörensson and N. Een. Minisat v1. 13-a sat solver with conflict-clause minimization. *SAT*, 2005:53, 2005.

[193] E. Stamatatos and K. Stergiou. Learning How to Propagate Using Random Probing. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems-CPAIOR'09. LNCS No. 5547*, pages 263–278, 2009.

[194] K. Stergiou and N. Samaras. Binary Encodings of Non-binary Constraint Satisfaction Problems: Algorithms and Experimental Results. *Journal of Artificial Intelligence Research (JAIR)*, 24:641–684, 2005.

[195] R. H. Storer, S. D. Wu, and R. Vaccari. New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling. *Management Science*, 38(10):1495–1509, 1992.

[196] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operations Research*, 64:278–285, 1993.

[197] N. Tamura, A. Taga, S. Kitagawa, and M. Banbara. Compiling Finite Linear CSP into SAT. In *Proceedings of Principles and Practice of Constraint Programming - CP'06. LNCS No. 4204*, pages 590–603, 2006.

[198] John Thornton. *Constraint Weighting Local Search for Constraint Satisfaction*. PhD thesis, Griffith University, Australia, 2000.

[199] D. A. D. Tompkins and H. H. Hoos. Warped Landscapes and Random Acts of SAT Solving. In *Proceedings of the Eight International Symposium on Artificial Intelligence and Mathematics-ISAIM04*, 2004.

[200] P. Torres and P. Lopez. On Not-First/Not-Last conditions in disjunctive scheduling. *European Journal of Operational Research*, 127(2):332–343, 2000.

[201] J. J. J. van den Broek. *MIP-based Approaches for Complex Planning Problems*. PhD thesis, Eindhoven University of Technology, Netherlands, 2009. URL `http://www.bsik-bricks.nl/documents/2009_PhD_van_den_Broek.pdf`.

[202] M. R. C. van Dongen, C. Lecoutre, and O. Roussel. CSP Solver Competition Benchmarks. `http://www.cril.univ-artois.fr/~lecoutre/research/benchmarks/benchmarks.html`, 2009.

[203] P. Van Hentenryck and L. Michel. *Constraint-based local search*. MIT Press, 2005.

[204] P. Van Hentenryck and T. L. Provost. Incremental Search in Constraint Logic Programming. *New Generation Comput.*, pages 257–276, 1991.

[205] P. Van Hentenryck, V. Saraswat, and Y. Deville. Design, implementation, and evaluation of the constraint language cc (FD). *The Journal of Logic Programming*, 37(1-3):139–164, 1998.

[206] M. Vázquez and L. D. Whitley. A Comparison of Genetic Algorithms for the Dynamic Job Shop Scheduling Problem. In *Genetic and Evolutionary Computation Conference - GECCO'00*, pages 1011–1018, 2000.

[207] N. R. Vempaty. Solving Constraint Satisfaction Problems Using Finite State Automata. In *Proceedings of the 10th National Conference on Artificial Intelligence - AAAI'92*, pages 453–458, 1992.

[208] G. Verfaillie and N. Jussien. Constraint solving in uncertain and dynamic environments: A survey. *Constraints*, 10(3):253–281, 2005.

[209] G. Verfaillie and T. Schiex. Solution Reuse in Dynamic Constraint Satisfaction Problems. In *Proceedings of the 12th National Conference on Artificial Intelligence - AAAI'94*, pages 307–312, 1994.

[210] P. Vilím. *Global Constraints in Scheduling*. PhD thesis, Charles University in Prague, Faculty of Mathematics and Physics, Department of Theoretical Computer Science and Logic, KTIML MFF, Universita Karlova, Malostranské náměstí 2/25, 118 00 Praha 1, Czech Republic, August 2007. URL `http://vilim.eu/petr/disertace.pdf`.

[211] P. Vilím. Filtering Algorithms for the Unary Resource Constraint. *Archives of Control Sciences*, 18(2), 2008.

[212] J. Vion. Hybridation de prouveurs CSP et apprentissage. In *Actes des troisièmes Journée Francophones de Progammation par Contraintes (JFPC'07)*, 2007.

[213] R. J. Wallace. Factor analytic studies of CSP heuristics. In *Proceedings of Principles and Practice of Constraint Programming - CP'05. LNCS No. 3709*, pages 712–726, 2005.

[214] R. J. Wallace. Analysis of heuristic synergies. In B. Hnich, M. Carlsson, F. Fages, and F. Rossi, editors, *Recent Advances in Constraints. Joint ERCIM/CologNet Workshop on Constraint Solving and Constraint Logic Programming - CSCLP 2005. Revised Selected and Invited Papers. LNAI No. 3978*, pages 73–87, Berlin, 2006. Springer.

[215] R. J. Wallace. Determining the basis for performance variations in CSP heuristics. In F. Azevedo, P. Barahona, F. Fages, and F. Rossi, editors, *Proceedings, 11th Annual ERCIM Workshop on Constraint Solving and Constraint Programming, Lisbon, Portugal, June 2006*, pages 145–158, 2006.

[216] R. J. Wallace. Heuristic Policy Analysis and Efficiency Assessment in Constraint Satisfaction Search. In *Proceedings of the 18th International Conference on Tools with Artificial Intelligence, ICTAI'06)*, pages 305–314, 2006.

[217] R. J. Wallace and E. C. Freuder. Stable Solutions for Dynamic Constraint Satisfaction Problems. In *Proceedings of Principles and Practice of Constraint Programming - CP'98. LNCS No. 1520*, pages 447–461, 1998.

[218] R. J. Wallace and D. Grimes. Experimental Studies of Variable Selection Strategies Based on Constraint Weights. *Journal of Algorithms*, 63:114–129, 2008.

[219] J. P. Walser. *Integer optimization by local search: a domain-independent approach*, volume 1637 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1999. URL `http://books.google.com/books?id=RX9-smOWstoC`.

[220] T. Walsh. Depth-bounded Discrepancy Search. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence - IJCAI'97*, pages 1388–1395, 1997.

[221] T. Walsh. Search in a Small World. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - IJCAI'99*, pages 1172–1177, 1999.

[222] T. Walsh. Stochastic Constraint Programming. In *Proceedings of the 15th European Conference on Artificial Intelligence - ECAI'02*, pages 111–115, 2002.

[223] D. Waltz. Understanding line drawings of scenes with shadows. In Patrick Winston, editor, *The Psychology of Computer Vision*, pages 19–91. McGraw-Hill, 1975.

[224] J-P. Watson and J. C. Beck. A Hybrid Constraint Programming / Local Search Approach to the Job-Shop Scheduling Problem. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems-CPAIOR'08. LNCS No. 5015*, pages 263–277, 2008.

[225] J-P. Watson, L. Barbulescu, A. E. Howe, and L. D. Whitley. Algorithm Performance and Problem Structure for Flow-shop Scheduling. In *Proceedings*

of the 16th National Conference on Artificial Intelligence - AAAI'99, pages 688–695, 1999.

[226] J-P. Watson, A. E. Howe, and L. D. Whitley. Deconstructing Nowicki and Smutnicki's -TSAB tabu search algorithm for the job-shop scheduling problem. *Computers & Operations Research*, 33:2623–2644, 2006.

[227] R. Williams, C. P. Gomes, and B. Selman. Backdoors To Typical Case Complexity. In G. Gottlob and T. Walsh, editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence - IJCAI'03*, pages 1173–1178, 2003.

[228] D. A. Wismer. Solution of the Flowshop-Scheduling Problem with No Intermediate Queues. *Operations Research*, 20(3):689–697, 1972.

[229] A. Wolf. Better Propagation for Non-preemptive Single-Resource Constraint Problems. In *Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming-CSCLP'05*, pages 201–215, 2005.

[230] A. Wolf. Constraint-Based Task Scheduling with Sequence Dependent Setup Times, Time Windows and Breaks. In S. Fischer, E. Maehle, and R. Reischuk, editors, *GI Jahrestagung*, volume 154 of *LNI*, pages 3205–3219. GI, 2009.

[231] H. Wu. Randomization and Restarts. Master's thesis, University of Waterloo, 2006.

[232] H. Wu and P. van Beek. On Universal Restart Strategies for Backtracking Search. In *Proceedings of Principles and Practice of Constraint Programming - CP'07. LNCS No. 4741*, pages 681–695, 2007.

[233] K. Xu and W. Li. Exact Phase Transitions in Random Constraint Satisfaction Problems. *CoRR*, cs.AI/0004005, 2000.

[234] T. Yamada and R. Nakano. A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems. In *Parallel Problem Solving from Nature 2 (PPSN-II)*, pages 283–292, 1992.

[235] B. Yang, J. Geunes, and W. J. O'Brien. Resource-constrained project scheduling: Past work and new directions. Technical report, Department of Industrial and Systems Engineering, University of Florida, 2001.

[236] A. Zanarini. *Exploiting Global Constraints for Search and Propagation*. PhD thesis, École Polytechnique de Montréal, 2010.

[237] Y. Zhan. Randomization and Restarts on a State-of-the-Art SAT solver – SATZ. Master's thesis, University of York, 2001.

[238] C. Y. Zhang, P. Li, Y. Rao, and Z. Guan. A very fast TS/SA algorithm for the job shop scheduling problem. *Computers & Operations Research*, 35: 282–294, 2008.

[239] J. Zhu, X. Li, and Q. Wang. Complete local search with limited memory algorithm for no-wait job shops to minimize makespan. *European Journal of Operational Research*, 198(2):378–386, 2009.

Appendix A

Full Experimental Analysis of Restarting Strategies: Benchmarks

The following problem descriptions are primarily taken from the benchmark website of the CSP solver competition[†].

- All Interval Series (*All Intervals*)

  The all-interval series problem is the task of finding a vector $s = (s_1, \ldots, s_n)$, such that s is a permutation of $\{0,1, \ldots, n\text{-}1\}$ and the interval vector $v = (|s_2 - s_1|, |s_3 - s_2|, \ldots, |s_n - s_{n-1}|)$ is a permutation of $\{1,2,\ldots,n\text{-}1\}$. See problem 07 at `http://www.csplib.org`.

- All Squares

  This problem involves finding the smallest square that the n squares $\{1\times1, \ldots, n \times n\}$ will fit into. This sequence starts 1, 3, 5, 7, 9, 11, 13, 15, 18, 21, 24, 27, 30, 33, 36, 39, 43, 47, 51, 54, 58, 63, 67, 71, . . . and is number A005842 of the Encyclopedia of Integer Sequences. It is a special case of problem 09 of `http://www.csplib.org`. The series tested here were generated by Hadrien Cambazard.

- Balanced Incomplete Block Desgins (*BIBD*)

[†]http://www.cril.univ-artois.fr/ lecoutre/research/benchmarks/benchmarks.html

Given a tuple of natural numbers (v, b, r, k, λ), a BIBD involves arranging v distinct objects into b blocks. Each block contains exactly k distinct objects, each object occurs in exactly r different blocks, and every two distinct objects occur together in exactly λ blocks (Frisch et al. [67]). See problem 28 at `http://www.csplib.org`.

- Binary decision diagrams A binary decision diagram (BDD) [36] is a method of compactly representing the solution set to a constraint on Boolean variables. It is a directed acyclic graph with two terminal nodes: the 0-terminal (representing a false assignment) and the 1-terminal (representing a true assignment). Each non-terminal node u is labeled with a Boolean variable b_i, and has a 0-successor (1-succesor) representing the assignment of the value 0 (1 *resp.*) to the variable.

  A BDD constraint is a random extensional constraint built from a binary decision diagram. These series have been generated by Kenil Cheng and Roland Yap in the context of the paper Cheng and Yap [45]. There are two sets of 35 instances each. The first set contains problems with many small BDD constraints while the latter set contains problems with few large BDD constraints.

- Boolean Problems

  These problems contain only Boolean variables and are satisfiability problems proposed in the context of the second Dimacs implementation challenge [113]. A description of the different problem sets can be found at `http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html`. A sample of 100 instances were randomly selected.

- Bounded Model Checking

  The bmc instances are bounded model checking problems which were converted from CNF to intensional CSP by Marc van Dongen. These instances are non-binary. The original instances may be found on SATLIB: `http://www.satlib.org/`. For each SATLIB instance, CSP instances were generated for different domain sizes. For example, the original instance

bmc-ibm-1.cnf was converted to instances bmc-ibm-1-2.xml, bmc-ibm-1-4.xml, bmc-ibm-1-8.xml and bmc-ibm-1-16.xml. The domain of the instance bmc-ibm-1-n.xml is given by $\{0,\ldots,n\text{-}1\}$. All instances corresponding to a given original SAT instance are equivalent in the sense that their solutions are the same. This was achieved by adding unary constraints to remove values greater than 1.

- Chessboard Coloration (*Chess Coloration*)

  The chessboard coloration problem is the task of coloring all squares of a chessboard composed of r rows and c columns. There are exactly n available colors and the four corners of any rectangle extracted from the chessboard must not be assigned the same color.

- Costas Array

  A Costas array can be regarded geometrically as a set of n points lying on the squares of a $n \times n$ checkerboard, such that each row or column contains only one point, and that all of the $n(n-1)/2$ displacement vectors between each pair of dots are distinct. This result in an ideal 'thumbtack' auto-ambiguity function, making the arrays useful in applications such as Sonar and Radar. A Costas array may be represented numerically as an nxn array of numbers, where each entry is either 1, for a point, or 0, for the absence of a point. When interpreted as binary matrices, these arrays of numbers have the property that, since each row and column has the constraint that it only has one point on it, they are therefore also permutation matrices. Thus, the Costas arrays for any given n are a subset of the permutation matrices of order n.

- Crosswords

  Given a grid and a dictionary, the problem is to fill the grid with the words contained in the dictionary. Here, three series of grids (*Herald*, *Puzzle*, *Vg*) and four dictionaries (*lex*, *ogd*, *uk*, *words*) have been used. *Herald* refers to crossword puzzles taken from the Herald Tribune (Spring, 1999), *Puzzle* refers to crossword puzzles mentioned in Ginsberg [76] and Ginsberg et al. [78], and *Vg* refers to blank grids.

Lex is a dictionary used in Stergiou and Samaras [194], *uk* corresponds to the UK cryptic solvers dictionary, words corresponds to the dictionary found in /usr/dict/words under Linux, and *ogd* corresponds to a french dictionary. *Lex* and *words* are small dictionaries whereas *uk* and *ogd* are large ones. The model used to represent the instances is the one identified by $m1$ in Beacham et al. [20]. However, for the *Vg* grids, all instances only involve constraints in extension as for these grids, putting two times the same word has been authorized. Hadrien Cambazard, Kostas Stergiou and Julian Ullmann contributed to converting these instances to XML format. A sample of 100 instances were randomly selected.

- Driver Log

  According to the planning competition website[†], this problem involves drivers that can walk between locations and trucks that can drive between locations. Walking requires traversal of different paths from those used for driving, and there is always one intermediate location on a footpath between two road junctions. The trucks can be loaded or unloaded with packages (with or without a driver present) and the objective is to transport packages between locations, ending up with a subset of the packages, the trucks and the drivers at specified destinations.

- Fischer

  This series correspond to Fischer Satisfiability Modulo Theory (SMT) instances from MathSat[‡] converted to CSP by Lucas Bordeaux. The original instances can be found at the SMT-LIB website[§] (in the QF_IDL family of benchmarks). The translation is a straightforward encoding of the SMT syntax in which Boolean combinations of arithmetic constraints are decomposed into primitive constraints, using reification where appropriate. Here the domains have been artificially bounded, whereas in SMT theorem prov-

[†] `http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume20/long03a-html/node39.html`

[‡] `http://mathsat.fbk.eu/`

[§] `http://www.smtlib.org`

ing should be done over the unbounded integers. A sample of 100 instances were randomly selected.

- Frequency Assignment Problem with Polarization (*Fapp*)

  The Frequency Assignment Problem with Polarization constraints (denoted FAPP) is the optimization problem retained for the ROADEF'2001 challenge (a description is available here `challenge.roadef.org/2001/files/fapp_roadef01_rev2_msword_en.ps.gz`) . It is one extended subject of the CALMA European project (Combinatorial ALgorithms for Military Applications). In this problem, one can find some constraints concerning the distance between frequencies and depending on their polarizations. For such constraints, a progressive relaxation is authorized: a relaxation level is possible between 0 (no relaxation) and 10 (the maximum relaxation). To obtain a decision problem, one has just to set one of these relaxation levels. In other words, 11 CSP instances can be defined from any original FAPP instance. A sample of 100 instances were randomly selected.

- Golomb Ruler

  The Golomb Ruler problem is the task of putting n marks on a ruler of length m such that the distance between any two pairs of marks is distinct. See problem 06 at `http://www.csplib.org`.

- Graph Coloring (*Coloring*)

  The graph coloring problem can be stated as follows: given a graph $G = (V, E)$, the objective is to find the minimum number of colors k such that there exists a mapping r from V to $\{1 \ldots k\}$ with $r(i) \neq r(j)$ for every edge (i, j) of G. The decision problem associated with this optimization problem involves determining if the graph can be colored when using a given number of colors k. All CSP instances given here have been built from resources that can be found at `http://mat.gsia.cmu.edu/COLOR04/`. A sample of 100 instances were randomly selected.

- Langford

The (generalized version of the) problem is to arrange k sets of numbers ranging from 1 to n, so that each appearance of the number m is m numbers on from the last. See problem 24 at `http://www.csplib.org`.

- Multi-Knapsack

  The basic knapsack problem involves a set of n items, with each item having an associated profit p_j and weight w_j. The objective is to pick some of the items, with maximal total profit, while obeying that the maximum total weight of the chosen items must not exceed W. Generally, these coefficients are scaled to become integers, and they are almost always assumed to be positive. The multi-dimensional knapsack problem (multi-knapsack problem) is where there are additional constraints restricting certain combinations of the items.

  The problems in optimization format are available at the operations research library website[†]. In satisfaction format there are constraints of two types, the first involves finding the combination of items which gives the optimal value for the problem:

$$\sum_{i=1}^{n}(c_i * x_i) = Opt \tag{A.1}$$

$$\forall j \sum_{i=1}^{n}(cj_i * x_i) \leq k \tag{A.2}$$

  where x_i refers to the n Boolean variables, c_i their associated constants ($>$ 0), cj_i are the associated constants in the additional j capacity constraints (if a variable doesn't contribute to the dimensional capacity its weight, cj_i, is set to 0 in the constraint).

- Primes

  The Primes instances are non-binary intensional instances which were created by Marc van Dongen. All instances are satisfiable. The domains of

[†]http://people.brunel.ac.uk/ mastjjb/jeb/orlib/mknapinfo.html

the variables consist of prime numbers and all constraints are linear equations. The coefficients and constants in the equations are also prime numbers. These instances are interesting because solving them using Gausian elimination is polynomial, assuming that the basic arithmetic operartions have a time complexity of $O(1)$. In reality this assumption does not hold and the choise of prime numbers in the equations gives rise to large intermediate coefficients in the equations, making the basic operations more time consuming. A sample of 50 instances were randomly selected.

- Pseudo-Boolean (*PseudoBool*)

  These instances correspond to the Pseudo-Boolean instances that were used for testing solvers submitted to the Pseudo Boolean Evaluation 2006 (`http://www.cril.univ-artois.fr/PB06/archive.html`). A sample of 100 instances were randomly selected.

- Pseudo-Boolean with global constraints (*Pseudo-Glb*)

  These instances were also used for testing solvers submitted to the Pseudo Boolean Evaluation 2006, and are similar to the other Pseudo-Boolean problems with the exception that all these instances involve the *weightedSum* global constraint. A sample of 100 instances were randomly selected.

- Quasi-group completion problem / Quasi-group with holes (*Qcp/Qwh*)

  The Quasi-group Completion problem (QCP) is the task of determining whether the remaining entries of the partial Latin square can be filled in such a way that we obtain a complete Latin square, ie. a full multiplication table of a quasi-group. The Quasi-group With Holes problem (QWH) is a variant of the QCP where instances are generated in such a way that they are guaranteed to be satisfiable. The randomly selected sample of 100 instances used here were taken from 8 series generated by Radoslaw Szymanek for the 2005 CSP Solver Competition.

- Radar

  The radar surveillance problem involves determining a radar surveillance plan for a geographical area, divided into hexagonal cells (Walser [219],

discussed in Chapter 6 on "Covering and assignment"). There are a number of radio stations in fixed cells. Each cell must be monitored by at least three radio stations.

- Ramsey

  The Ramsey problem is the task of coloring, using k colors, the edges of a complete graph involving n nodes in such a way that there is no monochromatic triangle in the graph (i.e., in any triangle at most two edges have the same color). See problem 17 at `http://www.csplib.org`.

- Random Extensional Problems (*Random*)

  A sample of 100 instances were randomly selected. This sample contains problems generated according to different models, in particular models B, D (both described in [81]), and RB as described in Xu and Li [233].

  Some quasi-random CSP instances were also included, in particular instances of type "composed" and "geometric". The composed problems are generated such that each instance is composed of a main (under-constrained) fragment and some auxiliary fragments, each of which being grafted to the main one by introducing some binary constraints.

  The geometric instances are generated as follows. Instead of a density parameter, a "distance" parameter, dst, is used such that $dst \leq \sqrt{2}$. For each variable, two coordinates are chosen at random so the associated point lies in the unit square. Then for each variable pair, (x,y), if the distance between their associated points is less than or equal to dst, the arc (x,y) is added to the constraint graph. Constraint relations are created in the same way as they are for homogeneous random CSP instances.

- Radio Link Frequency Assignment Problem (*RLFAP*)

  The Radio Link Frequency Assignment Problem (RLFAP) is the task of assigning frequencies to a number of radio links in such a manner as to simultaneously satisfy a large number of constraints and use as few distinct frequencies as possible.

- Scheduling

  A sample of 100 scheduling instances were randomly selected from the following scheduling problem sets:

  - Cabinet

    This is an assignment scheduling problem where some tasks are processed on one of a number of machines. Depending on the machine chosen, a task may take a different amount of time to be processed. The objective is to assign all the tasks to machines in order to minimize the overall makespan. This is equivalent to minimizing the maximum makespan of each machine. In these examples, the problem is presented as a satisfaction one, and involves the global constraint *element*. This series has was contributed to the solver competition by Leonid Ioffe and James Little, with the participation of Marc van Dongen.

  - Classical job shop scheduling (cjss)

    These problems are derived from classical job-shop scheduling problems by multiplying the number of jobs (and thus the number of tasks) and the capacity of the resources by the given factor. The makespan value is also specified. This series has been submitted by Naoyuki Tamura to the 2008 CSP solver competition and involves the global constraint cumulative.

  - Resource constrained project scheduling (rcpsp)

    This is the Resource Constrained Project Scheduling Problem, which involves assigning jobs or tasks to a resource or set of resources with limited capacity in order to meet some predefined objective (Yang et al. [235]). These satisfaction instances have been generated from data/instances of Baptiste and Le Pape by Hadrien Cambazard with the help of Sophie Demassey.

  - Job-shop scheduling with super solutions (super-jsp)

    These series of instances have been built by converting original problems into the corresponding ”super-solutions” problems (Hebrard et al.

[103]). That is, any solution of the resulting problem is a (1,0)-super solution of the original one. Observe that it requires instances with large domains and a lot of solutions to produce interesting problems. These series have been converted and generated by Emmanuel Hebrard in the context of the 2008 CSP Solver Competition.

- Schurr's Lemma

  The Schurr's lemma problem is the task of putting n balls labelled from 1 to n into 3 boxes so that for any triple of balls (x, y, z) with $x + y = z$, not all are in the same box. See problem 15 at `http://www.csplib.org`.

- Social Golfer

  This is a series of instances proposed by Daniel le Berre and Ines Lynce for the 2006 CSP Solver Competition. The original problem can be described by enumerating four constraints as follows: a golf club has 32 members; each member plays golf once a week; golfers always play in groups of four; and no golfer plays in the same group as any other golfer twice. It can be easily generalized. An instance to this generalized problem is then characterized by a triple w-p-g, where w is the number of weeks, p is the number of players per group and g is the number of groups. The encoding used here is the one proposed by Walser and is available at `http://www.csplib.org`.

- Timetabling (*Patat*)

  These instances come from the 2007/2008 timetabling competition `http://www.cs.qub.ac.uk/itc2007/` as well as from the previous one in 2002. They involve the global $allDifferent$ constraint as well as intensional constraints. These series have been converted and generated by Emmanuel Hebrard in the context of the 2008 CSP Solver Competition.

- Traveling Salesman

  The Travelling Salesman problem is the task of finding a tour of minimal length that traverses each city (only once) for a given set of cities. The prob-

lems used are two series of 15 ternary instances (all satisfiable), which were generated by Radoslaw Szymanek for the 2005 competition.

- Two-Dimensional Strip Packing (*Tdsp*)

  The Two-Dimensional Strip Packing problem involves fitting all given rectangles into a container ("strip") whose width is prescribed, such that the overall height is minimized (Hopper [107]). This series was submitted by Naoyuki Tamura to the 2008 CSP solver competition.