

University of Wollongong - Research Online

Thesis Collection

Title: Solving very large distributed constraint satisfaction problems

Author: Peter Harvey

Year: 2009

Repository DOI:

Copyright Warning

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

You are reminded of the following: This work is copyright. Apart from any use permitted under the Copyright Act 1968, no part of this work may be reproduced by any process, nor may any other exclusive right be exercised, without the permission of the author. Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material.

Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

Unless otherwise indicated, the views expressed in this thesis are those of the author and do not necessarily represent the views of the University of Wollongong.

Research Online is the open access repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

2009

Solving very large distributed constraint satisfaction problems

Peter Harvey
University of Wollongong

Follow this and additional works at: <https://ro.uow.edu.au/theses>

University of Wollongong

Copyright Warning

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

You are reminded of the following: This work is copyright. Apart from any use permitted under the Copyright Act 1968, no part of this work may be reproduced by any process, nor may any other exclusive right be exercised, without the permission of the author. Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material.

Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

Unless otherwise indicated, the views expressed in this thesis are those of the author and do not necessarily represent the views of the University of Wollongong.

Recommended Citation

Harvey, Peter, Solving very large distributed constraint satisfaction problems, Doctor of Philosophy thesis, School of Computer Science and Software Engineering - Faculty of Informatics, University of Wollongong, 2009. <https://ro.uow.edu.au/theses/3161>

Research Online is the open access institutional repository for the University of Wollongong. For further information contact the UOW Library: research-pubs@uow.edu.au

NOTE

This online version of the thesis may have different page formatting and pagination from the paper copy held in the University of Wollongong Library.

UNIVERSITY OF WOLLONGONG

COPYRIGHT WARNING

You may print or download ONE copy of this document for the purpose of your own research or study. The University does not authorise you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site. You are reminded of the following:

Copyright owners are entitled to take legal action against persons who infringe their copyright. A reproduction of material that is protected by copyright may be a copyright infringement. A court may impose penalties and award damages in relation to offences and infringements relating to copyright material. Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

Solving Very Large Distributed Constraint Satisfaction Problems

A thesis submitted in partial fulfilment of the
requirements for the award of the degree

Doctor of Philosophy

from

University of Wollongong

by

Peter Harvey

Bachelor of Mathematics

Bachelor of Computer Science

School of Computer Science and Software Engineering

2009

CERTIFICATION

I, Peter A. Harvey, declare that this thesis, submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy, in the School of Computer Science and Software Engineering, University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. The document has not been submitted for qualifications at any other academic institution.

Peter A. Harvey
8 December 2009

Abstract

This thesis investigates issues with existing approaches to distributed constraint satisfaction, and proposes a solution in the form of a new algorithm. These issues are most evident when solving large distributed constraint satisfaction problems, hence the title of the thesis.

We will first survey existing algorithms for centralised constraint satisfaction, and describe how they have been modified to handle distributed constraint satisfaction. The method by which each algorithm achieves completeness will be investigated and analysed by application of a new theorem.

We will then present a new algorithm, Support-Based Distributed Search, developed explicitly for distributed constraint satisfaction rather than being derived from centralised algorithms. This algorithm is inspired by the inherent structure of human arguments and similar mechanisms we observe in real-world negotiations.

A number of modifications to this new algorithm are considered, and comparisons are made with existing algorithms, effectively demonstrating its place within the field. Empirical analysis is then conducted, and comparisons are made to state-of-the-art algorithms most able to handle large distributed constraint satisfaction problems.

Finally, it is argued that any future development in distributed constraint satisfaction will necessitate changes in the algorithms used to solve small ‘embedded’ constraint satisfaction problems. The impact on embedded constraint satisfaction problems is considered, with a brief presentation of an improved algorithm for hypertree decomposition.

Previously published work includes [HG03, HCG05, HCG06a, HCG06b, HCG06c].

This thesis is dedicated to
my dearest wife Emily,
my baby daughter Adelaide,
my parents Keith and Sandra,
and my siblings Sean and Danielle.
I love you. You mean the world to me.

I would like to thank
Professor Aditya Ghose for his guidance,
Chee Fon Chang for his fellowship,
Farzad Salim for his friendship,
and the partners and many friends
who helped me through these last years.

Two weddings, one divorce, and a beautiful baby...
who would have thought it would take this long?

Table of Contents

1	Introduction	1
1.1	Constraint Satisfaction Problem	2
1.2	Distributed Constraint Satisfaction Problem	3
1.3	Centralised vs Distributed Algorithms	4
1.4	Motivation	5
2	Analysis of Completeness Techniques	10
2.1	Common Proof Theorem	11
2.2	Chronological Backtracking	14
2.3	Chronological Backtracking with Reordering	17
2.4	Dynamic Backtracking	20
2.5	Weak Commitment Search	24
2.6	Asynchronous Backtracking	26
2.7	Asynchronous Backtracking with Dynamic Ordering	31
2.8	Asynchronous Weak Commitment Search	34
2.9	Breakout	37
2.10	Distributed Breakout	38
2.11	Total, Partial, Dynamic and Static Orders	39
2.12	Categorisation	40
3	Support-Based Distributed Search	43
3.1	Introduction	43
3.2	Representation	45
3.2.1	Assignments as Arguments	47
3.2.2	Interpretation of Arguments	49
3.3	Solving	52
3.3.1	Asynchronicity with Isgoods	56
3.3.2	Computation of Isgoods	57
3.3.3	Demonstration of Isgoods	59
3.3.4	Postponement of Isgoods	61
3.4	Results	65
3.4.1	Soundness of Algorithm	68
3.4.2	Completeness of Algorithm	69

4	Variations and Relations	71
4.1	Minimising Conflicts	71
4.2	Minimising Communication	74
4.3	Minimising Storage	76
4.4	Relation to Other Algorithms	79
4.4.1	Asynchronous Weak-Commitment Search (AWCS)	79
4.4.2	Distributed Breakout (DBO)	80
4.4.3	Asynchronous Backtracking With Dynamic Ordering (ABT-DO) . .	81
5	Empirical Analysis	82
5.1	Metrics	82
5.1.1	Total vs Non-Concurrent Measures	83
5.1.2	Constraint Checks	84
5.1.3	Nogood Checks	84
5.1.4	Bytes	84
5.1.5	Packets	85
5.1.6	Bytes Per Packet	85
5.1.7	CPU Time	85
5.1.8	Concurrent Checks and Concurrent Traffic	85
5.2	Implementation	86
5.2.1	ABT and ABT-DO Implementation Notes	86
5.2.2	AWCS Implementation Notes	87
5.2.3	SBDS Implementation Notes	88
5.2.4	SBDS Value Selection Heuristic	88
5.3	Problem Sets	90
5.4	Results for Smaller Problems	91
5.5	Results for Larger Problems	99
5.6	Discussion	105
5.6.1	Coordinated Approach	105
5.6.2	Non-Coordinated Approach	106
5.6.3	Unified Approach	107
5.7	Summary	108
6	Multiple Variables Per Agent	109
6.1	Introduction	109
6.2	Hypertree Decompositions	111
6.2.1	General Form	112
6.2.2	Normal Form	113

6.2.3	Reduced Normal Form	114
6.3	Algorithm	118
6.3.1	opt- k -decomp	118
6.3.2	red- k -decomp	120
6.4	Performance	126
6.5	Application Within DisCSP Agents	128
7	Conclusion	129
7.1	Summary	131
7.2	Future Work	132
A	Results	140

List of Figures

1.1	It can never be clear which agent should take the greatest burden of search.	6
1.2	Adding links will increase the amount of communication between agents.	7
1.3	Distributed problems should not be treated in isolation, but in a wider network.	8
2.1	CBT: Making an assignment involves taking from the head of statically ordered list of unassigned variables and appending to the tail of an ordered list of assigned variables	14
2.2	CBT: Backtracking an assignment is the exact inverse, taking from the tail of the assigned variables and appending to the head of an ordered list of unassigned variables	14
2.3	CBT-R: Making an assignment involves choosing a variable from an unordered list of unassigned variables and appending to the tail of an ordered list of assigned variables	18
2.4	CBT-R: Backtracking an assignment is the exact inverse, taking from the tail of the assigned variables and appending to an unordered list of unassigned variables	18
2.5	DBT: ‘Eliminating explanations’ tell us which values are unusable, based on current values for specific assigned variables.	20
2.6	DBT: When all values are eliminated, the last-assigned variable causing any of the eliminations is unassigned, and a new eliminating explanation constructed.	20
2.7	WCS: Variables are assigned and partitioned into ‘current consistent’ and ‘tentative inconsistent’. Tentatively-assigned variables are iteratively given consistent assignments.	24
2.8	WCS: When a tentatively-assigned variable cannot be given a consistent assignment, the current assignment is transformed into a nogood and all assignments become tentative.	24
2.9	ABT: Changes in assignment are broadcast from each agent to each lower-ranked agent.	27
2.10	ABT: Nogoods only contain assignments from higher-ranked agents, and are sent to the lowest-ranked agent.	27
2.11	ABT-DO: Changes in assignment are broadcast to all neighbours regardless of rank.	32

2.12	ABT: Changes in ordering only effect lower-ranked agents, and are sent to all of them.	32
2.13	AWCS: Changes in assignment are broadcast from each agent to each neighbour, regardless of priorities.	34
2.14	AWCS: Nogoods are sent to all involved agents, and the priority of the sender is raised above all its neighbours.	34
2.15	BO: Constraint violations are weighted and summed. Dynamic adjustments of weights allow a simple hill-climbing algorithm to escape local minima. .	37
3.1	Example constraint model and graph	44
3.2	Constraint model and graph for the ring-ordering problem	61
3.3	Constraint model and graph demonstrating cyclic behaviour in SBDS . . .	63
4.1	Constraint model and support graph for a simple 5-node problem	72
5.1	Average feasibility of problem instances for Problem Sets 1 and 2	92
5.2	Constraint checks for Problem Sets 1 and 2. Full results on pages 141 and 159.	92
5.3	Constraint checks for Problem Sets 1 and 2, broken down by feasibility. Full results on pages 142, 143, 160 and 161.	93
5.4	Nogood checks for Problem Sets 1 and 2. Full results on pages 144 and 162.	94
5.5	Network traffic for Problem Sets 1 and 2. Full results on pages 147 and 165.	95
5.6	Numbers of packets for Problem Sets 1 and 2. Full results on pages 150 and 168.	96
5.7	Packet sizes for Problem Sets 1 and 2. Full results on pages 153 and 171. .	97
5.8	Concurrent checks for Problem Sets 1 and 2. Full results on page 192. . . .	98
5.9	CPU time for Problem Sets 1 and 2. Full results on pages 153 and 171. . . .	99
5.10	Average feasibility of problem instances for Problem Set 3	99
5.11	Constraint checks for Problem Set 3. Full results on page 177.	101
5.12	Nogood checks for Problem Set 3. Full results on page 180.	101
5.13	Number of packets for Problem Set 3. Full results on page 186.	101
5.14	Packet sizes for Problem Set 3. Full results on page 189.	103
5.15	Network traffic for Problem Set 3. Full results on page 183.	103
5.16	CPU time for Problem Set 3. Full results on page 189.	103
5.17	Concurrent checks for Problem Set 3, broken down by feasibility. Full results on pages 193 and 194.	104
5.18	Concurrent traffic for Problem Set 3, broken down by feasibility. Full results on pages 193 and 194.	105

6.1	A demonstration of how hypertree decomposition forms a new acyclic problem from a cyclic problem.	111
6.2	Transforming a normal form hypertree decomposition to reduced normal form.	117
6.3	Graphs of the CPU time of opt- k -decomp, and comparisons to its best-case and worst-case complexity functions. Each point represents a single run of opt- k -decomp.	127
6.4	Graphs of the CPU time of red- k -decomp, and comparisons to its best-case and worst-case complexity functions. Each point represents a single run of red- k -decomp.	127
6.5	A comparison of CPU times for random CSP instances. Values are computed as the CPU time for opt- k -decomp divided by the CPU time for red- k -decomp.	128
A.1	Number of constraint checks for all instances in problem set 1	141
A.2	Number of constraint checks for feasible instances in problem set 1	142
A.3	Number of constraint checks for infeasible instances in problem set 1 . . .	143
A.4	Number of nogood checks for all instances in problem set 1	144
A.5	Number of nogood checks for feasible instances in problem set 1	145
A.6	Number of nogood checks for infeasible instances in problem set 1	146
A.7	Network traffic for all instances in problem set 1	147
A.8	Network traffic for feasible instances in problem set 1	148
A.9	Network traffic for infeasible instances in problem set 1	149
A.10	Number of packets for all instances in problem set 1	150
A.11	Number of packets for feasible instances in problem set 1	151
A.12	Number of packets for infeasible instances in problem set 1	152
A.13	Average packet size and CPU time for all instances in problem set 1	153
A.14	Average packet size and CPU time for feasible instances in problem set 1 .	154
A.15	Average packet size and CPU time for infeasible instances in problem set 1 .	155
A.16	Other measures of concurrency for all instances in problem set 1	156
A.17	Other measures of concurrency for feasible instances in problem set 1 . . .	157
A.18	Other measures of concurrency for infeasible instances in problem set 1 . .	158
A.19	Number of constraint checks for all instances in problem set 2	159
A.20	Number of constraint checks for feasible instances in problem set 2	160
A.21	Number of constraint checks for infeasible instances in problem set 2 . . .	161
A.22	Number of nogood checks for all instances in problem set 2	162
A.23	Number of nogood checks for feasible instances in problem set 2	163
A.24	Number of nogood checks for infeasible instances in problem set 2	164
A.25	Network traffic for all instances in problem set 2	165

A.26	Network traffic for feasible instances in problem set 2 .	166
A.27	Network traffic for infeasible instances in problem set 2 .	167
A.28	Number of packets for all instances in problem set 2 .	168
A.29	Number of packets for feasible instances in problem set 2 .	169
A.30	Number of packets for infeasible instances in problem set 2 .	170
A.31	Average packet size and CPU time for all instances in problem set 2 .	171
A.32	Average packet size and CPU time for feasible instances in problem set 2 .	172
A.33	Average packet size and CPU time for infeasible instances in problem set 2 .	173
A.34	Other measures of concurrency for all instances in problem set 2 .	174
A.35	Other measures of concurrency for feasible instances in problem set 2 .	175
A.36	Other measures of concurrency for infeasible instances in problem set 2 .	176
A.37	Number of constraint checks for all instances in problem set 3 .	177
A.38	Number of constraint checks for feasible instances in problem set 3 .	178
A.39	Number of constraint checks for infeasible instances in problem set 3 .	179
A.40	Number of nogood checks for all instances in problem set 3 .	180
A.41	Number of nogood checks for feasible instances in problem set 3 .	181
A.42	Number of nogood checks for infeasible instances in problem set 3 .	182
A.43	Network traffic for all instances in problem set 3 .	183
A.44	Network traffic for feasible instances in problem set 3 .	184
A.45	Network traffic for infeasible instances in problem set 3 .	185
A.46	Number of packets for all instances in problem set 3 .	186
A.47	Number of packets for feasible instances in problem set 3 .	187
A.48	Number of packets for infeasible instances in problem set 3 .	188
A.49	Average packet size and CPU time for all instances in problem set 3 .	189
A.50	Average packet size and CPU time for feasible instances in problem set 3 .	190
A.51	Average packet size and CPU time for infeasible instances in problem set 3 .	191
A.52	Other measures of concurrency for all instances in problem set 3 .	192
A.53	Other measures of concurrency for feasible instances in problem set 3 .	193
A.54	Other measures of concurrency for infeasible instances in problem set 3 .	194

List of Tables

2.1	Classification of algorithms according to their completeness mechanism . .	41
6.1	Complexity results for red- k -decomp	124
6.2	Results of red- k -decomp on successively larger ‘spider webs’ constraint graphs. ‘Cond 2’ and ‘Cond 3’ shows the number of k -vertices discarded due to con- ditions 2 and 3 of RNF. Run-times (in seconds) for <i>init-vertices</i> and <i>init-</i> <i>graph</i> are given as a combined value.	125

Terminology

Constraint satisfaction literature often uses the same term but with differing definitions. The following definitions will be used throughout this thesis.

complete	<p>The terms ‘complete’ and ‘incomplete’ will indicate whether concepts or methods cover all possibilities. An assignment is complete if and only if it provides values for all variables. An algorithm is complete if and only if it provides an answer for all problems. Note that we are using ‘complete’ in the general algorithmic sense, and not to indicate that a constraint satisfaction algorithm considers all possible assignments.</p> <p><i>Example:</i> A solution must be a complete assignment.</p> <p><i>Example:</i> Breakout is an incomplete algorithm.</p>
consistent	<p>The terms ‘consistent’ and ‘inconsistent’ will refer to simple tests that can be conducted with available information. The most common instance of this in constraint satisfaction is to say that a particular combination of values is consistent/inconsistent with the set of constraint. If necessary, an algorithm may redefine what it means to test an assignment for consistency.</p> <p><i>Example:</i> The assignment is first tested for consistency.</p> <p><i>Example:</i> The current assignment may still be inconsistent.</p>
feasible	<p>The terms ‘feasible’ and ‘infeasible’ will refer to more complex determinations made by an algorithm during its execution. This is most often used in constructive search algorithms once they prove, by exhaustive search, that a partial assignment of values to variables cannot be extended into a consistent assignment for all variables. Note that an assignment is feasible if and only if it is a subset of a complete consistent assignment.</p> <p><i>Example:</i> Nogoods record which assignments are infeasible.</p> <p><i>Example:</i> Let T be the set of all feasible assignments.</p>
solvable	<p>The terms ‘solvable’ and ‘unsolvable’ will refer to whether or not a constraint satisfaction problem has a solution. A solution is a complete, consistent assignment of values to variables. By definition, an unsolvable problem has no feasible assignments.</p> <p><i>Example:</i> If $E = \emptyset$, we can conclude the problem is unsolvable.</p> <p><i>Example:</i> Breakout search is only suitable for solvable problems.</p>

Formal Notation

Formulas, algorithms and proofs will attempt to use a consistent lettering and numbering scheme. When no additional information is provided, the following definitions should be assumed.

$\mathcal{V}, \mathcal{C}, \mathcal{D}$	The symbols \mathcal{V} , \mathcal{C} , and \mathcal{D} respectively refer to the variables, constraints, and domain of a given problem. If more than one problem exists we will subscript related symbols according. For example \mathcal{V}_1 , \mathcal{C}_1 and \mathcal{D}_1 .
V, C, D	The letters V , C , and D will refer to <i>subsets</i> of \mathcal{V} , \mathcal{C} , and \mathcal{D} respectively.
s, t	In most instances the letters s and t refer to assignments. An assignment is a function mapping some subset of \mathcal{V} to \mathcal{D} . They should not be assumed to be complete assignments (mapping <i>all</i> of \mathcal{V} to \mathcal{D}) unless explicitly stated.
\hat{s}	We use \hat{s} to denote the set of variables assigned values by s . Formally, if $s : V \rightarrow \mathcal{D}$ then $\hat{s} = V \subseteq \mathcal{V}$. Due to the nature of many constraint algorithms, we will assume that there exists an ‘order of assignment’ for \hat{s} , and will define the symbol for this order as needed.
$s \downarrow_V$	We use $s \downarrow_V$ to denote the assignment s projected on to some $V \subseteq \hat{s}$.
\mathcal{S}	The symbol \mathcal{S} refers to the set of partial assignments for a problem. Note that it does include all complete assignments $s : \mathcal{V} \rightarrow \mathcal{D}$, and the empty assignment $s = \emptyset$.
c	In most instances the letter c is used to refer to a constraint. A constraint is seen as a mapping from the set of assignments to a value T or F. If necessary an index such as i or j may be applied to differentiate between constraints. For example, $c_i, c_j \in \mathcal{C}$.
\hat{c}	We use \hat{c} to denote the scope (set of variables) of a constraint c .
$c(s)$	We use $c(s)$ to denote the evaluation of an assignment $s \downarrow_{\hat{c}}$ by the constraint c .
u, v, w	In most instances the letters u , v and w refer to variables. If necessary an index such as i or j may be applied to differentiate between variables. For example $v_i, v_j \in \mathcal{V}$.
d	In most instances this symbol is used to refer to a value. If necessary an index such as i or j may be applied to differentiate between values. For example $d_i, d_j \in \mathcal{D}$.

Pseudocode Notation

Algorithm pseudocode in this thesis will use some keywords and notation beyond the usual ‘if’, ‘for’, ‘while’ and ‘break’. These are presented below, along with a standardised interpretation for other common keywords such as ‘set’ and ‘let’.

algorithm, procedure	The label ‘algorithm’ is used to refer to the main function of a constraint satisfaction algorithm. Component functions such as backtracking and computing nogoods are labelled ‘procedure’ and are numbered accordingly.
when	<p>The term ‘when’ is used to model event-driven programming commonly found in distributed programs. It is assumed that the program pauses at the beginning of a ‘when’ block until one of the conditions is satisfied, and will not exit the ‘when’ block until none of the conditions are satisfied.</p> <p><i>Example:</i> when an assignment $v \rightarrow d$ is received from a neighbour</p> <p><i>Example:</i> when some random amount of time t has passed</p>
let	<p>The term ‘let’ is used to declare variables, often stating their intent and initial value. This is often also used to declare constants, or to define useful terms to simplify formulas.</p> <p><i>Example:</i> let V be a set of variables, initially empty</p> <p><i>Example:</i> let v be the variable most recently added to \hat{s}</p>
set	<p>The term ‘set’ is used to modify variables, describing the new value that they will take. This is most often used to modify functions, but also can be used in other circumstances.</p> <p><i>Example:</i> set $s(v)$ to a value consistent with the assignments in t</p> <p><i>Example:</i> set N to $N \cup \{n\}$</p>
unset	<p>The term ‘unset’ is used to give a variable <i>no</i> value. This is most often used to remove a particular mapping from a function. Note that ‘unset V’ is different from the ‘set V to \emptyset’. That is, if V is unset then $V \neq \emptyset$.</p> <p><i>Example:</i> unset $s(v)$, for all v appearing in \hat{c}</p> <p><i>Example:</i> unset the eliminating explanation $e(v, d)$</p>
'	The decoration ‘ $'$ ’ is used only in algorithm proofs, and not in algorithm bodies. It refers to the <i>next</i> value of a variable. For example, if s refers to the current variable-value assignment, then s' refers to the variable-value assignment after one step or iteration of the algorithm.