Francisco Azevedo
Pedro Barahona
François Fages
Francesca Rossi (Eds.)

LNAI 4651

Recent Advances in Constraints

11th Annual ERCIM International Workshop
on Constraint Solving and Constraint Logic Programming, CSCLP 2006
Caparica, Portugal, June 2006
Revised Selected and Invited Papers

Springer

Francisco Azevedo Pedro Barahona
François Fages Francesca Rossi (Eds.)

Recent Advances in Constraints

11th Annual ERCIM International Workshop
on Constraint Solving and
Contraint Logic Programming, CSCLP 2006
Caparica, Portugal, June 26-28, 2006
Revised Selected and Invited Papers

Springer

Series Editors

Jaime G. Carbonell, Carnegie Mellon University, Pittsburgh, PA, USA
Jörg Siekmann, University of Saarland, Saarbrücken, Germany

Volume Editors

Francisco Azevedo
Universidade Nova de Lisboa
2829-516 Caparica, Portugal
E-mail: fa@di.fct.unl.pt

Pedro Barahona
Universidade Nova de Lisboa
2825 Caparica, Portugal
E-mail: pb@di.fct.unl.pt

François Fages
INRIA Rocquencourt - Projet CONTRAINTES
BP 105, 78153 Le Chesnay Cedex, France
E-mail: Francois.Fages@inria.fr

Francesca Rossi
University of Padova
35131 Padova, Italy
E-mail: frossi@math.unipd.it

Library of Congress Control Number: 2007931597

CR Subject Classification (1998): I.2.3, F.3.1-2, F.4.1, D.3.3, F.2.2, G.1.6, I.2.8

LNCS Sublibrary: SL 7 – Artificial Intelligence

ISSN 0302-9743
ISBN-10 3-540-73816-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-73816-9 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12096343 06/3180 5 4 3 2 1 0

Lecture Notes in Artificial Intelligence 4651

Edited by J. G. Carbonell and J. Siekmann

Subseries of Lecture Notes in Computer Science

Preface

Constraints are a natural way to represent knowledge, and constraint programming is a declarative programming paradigm successfully used to solve many difficult combinatorial problems. Examples of application domains where such problems naturally arise, and where constraint programming has made a valuable contribution, are scheduling, production planning, communication networks, robotics, and bioinformatics.

This volume contains the extended and reviewed versions of a selection of papers presented at the 11th International Workshop on Constraint Solving and Constraint Logic Programming (CSCLP 2006), that was held during June 26–28, 2006 at the New University of Lisbon, Portugal. It also contains papers that were submitted in response to the open call that followed the workshop. Both types of papers were reviewed independently by three experts in the specific topics.

The papers in this volume present original research results, as well as applications, in many aspects of constraint solving and constraint logic programming. Research topics that can be found in the papers are symmetry breaking, privacy, distributed forward checking, quantified CSPs, bipolar preferences, first-order constraints, microstructure, constraint handling rules, acyclic clustered problems, as well as the analysis of application domains such as disjunctive resource problems and stochastic inventory control. Moreover, the volume also contains a tutorial on hybrid algorithms.

The editors would like to take the opportunity to thank all the authors who submitted a paper to this volume, as well as the reviewers for their helpful work. This volume was made possible thanks to the support of the European Research Consortium for Informatics and Mathematics (ERCIM), the Association for Constraint Programming (ACP), the Portuguese Foundation for Science and Technology (FCT), the Department of Computer Science of the New University of Lisbon, and its Centre for Artificial Intelligence (CENTRIA).

We hope that the present volume is useful to everyone interested in the recent advances and new trends in constraint programming, constraint solving, problem modelling, and applications.

March 2007

F. Azevedo
P. Barahona
F. Fages
F. Rossi

Organization

CSCLP 2006 was organized by the ERCIM Working Group on Constraints.

Organizing and Program Committee

Francisco Azevedo	CENTRIA, FCT/UNL, Portugal
Pedro Barahona	CENTRIA, FCT/UNL, Portugal
François Fages	INRIA Rocquencourt, France
Francesca Rossi	University of Padova, Italy

Additional Reviewers

R. Barták
M. Basharu
M. Benedetti
T. Benoist
D. Le Berre
S. Bistarelli
K. Cheng
K. Djelloul
T. Frühwirth
B. Hnich
N. Jussien
A. Kovacs
A. Lallouet
M. Maher
F. Manyà
M. Meister
P. Meseguer
M. Moffitt
E. Monfroy
B. O'Sullivan
B. Peintner
S. Prestwich
J.-F. Puget
S. Soliman
J. Vautard
M. Wallace
R. Wallace
R. Yap
N. Yorke-Smith
P. Zoeteweij

Sponsoring Institutions

ERCIM Working Group on Constraints
Association for Constraint Programming
Universidade Nova de Lisboa
Centro de Inteligência Artificial, Portugal
Fundação para a Ciência e Tecnologia, Portugal

Lecture Notes in Artificial Intelligence (LNAI)

Vol. 4660: S. Džeroski, J. Todoroski (Eds.), Computational Discovery of Scientific Knowledge. X, 327 pages. 2007.

Vol. 4651: F. Azevedo, P. Barahona, F. Fages, F. Rossi (Eds.), Recent Advances in Constraints. VIII, 185 pages. 2007.

Vol. 4632: R. Alhajj, H. Gao, X. Li, J. Li, O.R. Zaïane (Eds.), Advanced Data Mining and Applications. XV, 634 pages. 2007.

Vol. 4617: V. Torra, Y. Narukawa, Y. Yoshida (Eds.), Modeling Decisions for Artificial Intelligence. XII, 502 pages. 2007.

Vol. 4612: I. Miguel, W. Ruml (Eds.), Abstraction, Reformulation, and Approximation. XI, 418 pages. 2007.

Vol. 4604: U. Priss, S. Polovina, R. Hill (Eds.), Conceptual Structures: Knowledge Architectures for Smart Applications. XII, 514 pages. 2007.

Vol. 4603: F. Pfenning (Ed.), Automated Deduction – CADE-21. XII, 522 pages. 2007.

Vol. 4597: P. Perner (Ed.), Advances in Data Mining. XI, 353 pages. 2007.

Vol. 4594: R. Bellazzi, A. Abu-Hanna, J. Hunter (Eds.), Artificial Intelligence in Medicine. XVI, 509 pages. 2007.

Vol. 4585: M. Kryszkiewicz, J.F. Peters, H. Rybinski, A. Skowron (Eds.), Rough Sets and Intelligent Systems Paradigms. XIX, 836 pages. 2007.

Vol. 4578: F. Masulli, S. Mitra, G. Pasi (Eds.), Applications of Fuzzy Sets Theory. XVIII, 693 pages. 2007.

Vol. 4573: M. Kauers, M. Kerber, R. Miner, W. Windsteiger (Eds.), Towards Mechanized Mathematical Assistants. XIII, 407 pages. 2007.

Vol. 4571: P. Perner (Ed.), Machine Learning and Data Mining in Pattern Recognition. XIV, 913 pages. 2007.

Vol. 4570: H.G. Okuno, M. Ali (Eds.), New Trends in Applied Artificial Intelligence. XXI, 1194 pages. 2007.

Vol. 4565: D.D. Schmorrow, L.M. Reeves (Eds.), Foundations of Augmented Cognition. XIX, 450 pages. 2007.

Vol. 4562: D. Harris (Ed.), Engineering Psychology and Cognitive Ergonomics. XXIII, 879 pages. 2007.

Vol. 4548: N. Olivetti (Ed.), Automated Reasoning with Analytic Tableaux and Related Methods. X, 245 pages. 2007.

Vol. 4539: N.H. Bshouty, C. Gentile (Eds.), Learning Theory. XII, 634 pages. 2007.

Vol. 4529: P. Melin, O. Castillo, L.T. Aguilar, J. Kacprzyk, W. Pedrycz (Eds.), Foundations of Fuzzy Logic and Soft Computing. XIX, 830 pages. 2007.

Vol. 4511: C. Conati, K. McCoy, G. Paliouras (Eds.), User Modeling 2007. XVI, 487 pages. 2007.

Vol. 4509: Z. Kobti, D. Wu (Eds.), Advances in Artificial Intelligence. XII, 552 pages. 2007.

Vol. 4496: N.T. Nguyen, A. Grzech, R.J. Howlett, L.C. Jain (Eds.), Agent and Multi-Agent Systems: Technologies and Applications. XXI, 1046 pages. 2007.

Vol. 4483: C. Baral, G. Brewka, J. Schlipf (Eds.), Logic Programming and Nonmonotonic Reasoning. IX, 327 pages. 2007.

Vol. 4482: A. An, J. Stefanowski, S. Ramanna, C.J. Butz, W. Pedrycz, G. Wang (Eds.), Rough Sets, Fuzzy Sets, Data Mining and Granular Computing. XIV, 585 pages. 2007.

Vol. 4481: J. Yao, P. Lingras, W.-Z. Wu, M. Szczuka, N.J. Cercone, D. Ślęzak (Eds.), Rough Sets and Knowledge Technology. XIV, 576 pages. 2007.

Vol. 4476: V. Gorodetsky, C. Zhang, V.A. Skormin, L. Cao (Eds.), Autonomous Intelligent Systems: Multi-Agents and Data Mining. XIII, 323 pages. 2007.

Vol. 4455: S. Muggleton, R. Otero, A. Tamaddoni-Nezhad (Eds.), Inductive Logic Programming. XII, 456 pages. 2007.

Vol. 4452: M. Fasli, O. Shehory (Eds.), Agent-Mediated Electronic Commerce. VIII, 249 pages. 2007.

Vol. 4451: T.S. Huang, A. Nijholt, M. Pantic, A. Pentland (Eds.), Artifical Intelligence for Human Computing. XVI, 359 pages. 2007.

Vol. 4438: L. Maicher, A. Sigel, L.M. Garshol (Eds.), Leveraging the Semantics of Topic Maps. X, 257 pages. 2007.

Vol. 4429: R. Lu, J.H. Siekmann, C. Ullrich (Eds.), Cognitive Systems. X, 161 pages. 2007.

Vol. 4426: Z.-H. Zhou, H. Li, Q. Yang (Eds.), Advances in Knowledge Discovery and Data Mining. XXV, 1161 pages. 2007.

Vol. 4411: R.H. Bordini, M. Dastani, J. Dix, A.E.F. Seghrouchni (Eds.), Programming Multi-Agent Systems. XIV, 249 pages. 2007.

Vol. 4410: A. Branco (Ed.), Anaphora: Analysis, Algorithms and Applications. X, 191 pages. 2007.

Vol. 4399: T. Kovacs, X. Llorà, K. Takadama, P.L. Lanzi, W. Stolzmann, S.W. Wilson (Eds.), Learning Classifier Systems. XII, 345 pages. 2007.

Vol. 4390: S.O. Kuznetsov, S. Schmidt (Eds.), Formal Concept Analysis. X, 329 pages. 2007.

Vol. 4389: D. Weyns, H.V.D. Parunak, F. Michel (Eds.), Environments for Multi-Agent Systems III. X, 273 pages. 2007.

Vol. 4384: T. Washio, K. Satoh, H. Takeda, A. Inokuchi (Eds.), New Frontiers in Artificial Intelligence. IX, 401 pages. 2007.

Vol. 4371: K. Inoue, K. Satoh, F. Toni (Eds.), Computational Logic in Multi-Agent Systems. X, 315 pages. 2007.

Vol. 4369: M. Umeda, A. Wolf, O. Bartenstein, U. Geske, D. Seipel, O. Takata (Eds.), Declarative Programming for Knowledge Management. X, 229 pages. 2006.

Vol. 4342: H. de Swart, E. Orłowska, G. Schmidt, M. Roubens (Eds.), Theory and Applications of Relational Structures as Knowledge Instruments II. X, 373 pages. 2006.

Vol. 4335: S.A. Brueckner, S. Hassas, M. Jelasity, D. Yamins (Eds.), Engineering Self-Organising Systems. XII, 212 pages. 2007.

Vol. 4334: B. Beckert, R. Hähnle, P.H. Schmitt (Eds.), Verification of Object-Oriented Software. XXIX, 658 pages. 2007.

Vol. 4333: U. Reimer, D. Karagiannis (Eds.), Practical Aspects of Knowledge Management. XII, 338 pages. 2006.

Vol. 4327: M. Baldoni, U. Endriss (Eds.), Declarative Agent Languages and Technologies IV. VIII, 257 pages. 2006.

Vol. 4314: C. Freksa, M. Kohlhase, K. Schill (Eds.), KI 2006: Advances in Artificial Intelligence. XII, 458 pages. 2007.

Vol. 4304: A. Sattar, B.-h. Kang (Eds.), AI 2006: Advances in Artificial Intelligence. XXVII, 1303 pages. 2006.

Vol. 4303: A. Hoffmann, B.-h. Kang, D. Richards, S. Tsumoto (Eds.), Advances in Knowledge Acquisition and Management. XI, 259 pages. 2006.

Vol. 4293: A. Gelbukh, C.A. Reyes-Garcia (Eds.), MICAI 2006: Advances in Artificial Intelligence. XXVIII, 1232 pages. 2006.

Vol. 4289: M. Ackermann, B. Berendt, M. Grobelnik, A. Hotho, D. Mladenič, G. Semeraro, M. Spiliopoulou, G. Stumme, V. Svátek, M. van Someren (Eds.), Semantics, Web and Mining. X, 197 pages. 2006.

Vol. 4285: Y. Matsumoto, R.W. Sproat, K.-F. Wong, M. Zhang (Eds.), Computer Processing of Oriental Languages. XVII, 544 pages. 2006.

Vol. 4274: Q. Huo, B. Ma, E.-S. Chng, H. Li (Eds.), Chinese Spoken Language Processing. XXIV, 805 pages. 2006.

Vol. 4265: L. Todorovski, N. Lavrač, K.P. Jantke (Eds.), Discovery Science. XIV, 384 pages. 2006.

Vol. 4264: J.L. Balcázar, P.M. Long, F. Stephan (Eds.), Algorithmic Learning Theory. XIII, 393 pages. 2006.

Vol. 4259: S. Greco, Y. Hata, S. Hirano, M. Inuiguchi, S. Miyamoto, H.S. Nguyen, R. Słowiński (Eds.), Rough Sets and Current Trends in Computing. XXII, 951 pages. 2006.

Vol. 4253: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part III. XXXII, 1301 pages. 2006.

Vol. 4252: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part II. XXXIII, 1335 pages. 2006.

Vol. 4251: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part I. LXVI, 1297 pages. 2006.

Vol. 4248: S. Staab, V. Svátek (Eds.), Managing Knowledge in a World of Networks. XIV, 400 pages. 2006.

Vol. 4246: M. Hermann, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning. XIII, 588 pages. 2006.

Vol. 4223: L. Wang, L. Jiao, G. Shi, X. Li, J. Liu (Eds.), Fuzzy Systems and Knowledge Discovery. XXVIII, 1335 pages. 2006.

Vol. 4213: J. Fürnkranz, T. Scheffer, M. Spiliopoulou (Eds.), Knowledge Discovery in Databases: PKDD 2006. XXII, 660 pages. 2006.

Vol. 4212: J. Fürnkranz, T. Scheffer, M. Spiliopoulou (Eds.), Machine Learning: ECML 2006. XXIII, 851 pages. 2006.

Vol. 4211: P. Vogt, Y. Sugita, E. Tuci, C.L. Nehaniv (Eds.), Symbol Grounding and Beyond. VIII, 237 pages. 2006.

Vol. 4203: F. Esposito, Z.W. Raś, D. Malerba, G. Semeraro (Eds.), Foundations of Intelligent Systems. XVIII, 767 pages. 2006.

Vol. 4201: Y. Sakakibara, S. Kobayashi, K. Sato, T. Nishino, E. Tomita (Eds.), Grammatical Inference: Algorithms and Applications. XII, 359 pages. 2006.

Vol. 4200: I.F.C. Smith (Ed.), Intelligent Computing in Engineering and Architecture. XIII, 692 pages. 2006.

Vol. 4198: O. Nasraoui, O. Zaïane, M. Spiliopoulou, B. Mobasher, B. Masand, P.S. Yu (Eds.), Advances in Web Mining and Web Usage Analysis. IX, 177 pages. 2006.

Vol. 4196: K. Fischer, I.J. Timm, E. André, N. Zhong (Eds.), Multiagent System Technologies. X, 185 pages. 2006.

Vol. 4188: P. Sojka, I. Kopeček, K. Pala (Eds.), Text, Speech and Dialogue. XV, 721 pages. 2006.

Vol. 4183: J. Euzenat, J. Domingue (Eds.), Artificial Intelligence: Methodology, Systems, and Applications. XIII, 291 pages. 2006.

Vol. 4180: M. Kohlhase, OMDoc – An Open Markup Format for Mathematical Documents [version 1.2]. XIX, 428 pages. 2006.

Vol. 4177: R. Marín, E. Onaindía, A. Bugarín, J. Santos (Eds.), Current Topics in Artificial Intelligence. XV, 482 pages. 2006.

Vol. 4160: M. Fisher, W. van der Hoek, B. Konev, A. Lisitsa (Eds.), Logics in Artificial Intelligence. XII, 516 pages. 2006.

Vol. 4155: O. Stock, M. Schaerf (Eds.), Reasoning, Action and Interaction in AI Theories and Systems. XVIII, 343 pages. 2006.

Vol. 4149: M. Klusch, M. Rovatsos, T.R. Payne (Eds.), Cooperative Information Agents X. XII, 477 pages. 2006.

Table of Contents

Tutorial

Technical Papers

Hybrid Algorithms in Constraint Programming

Mark Wallace

Monash University, Faculty of Information Technology,
Building 63, Clayton, Vic. 3800, Australia
mark.wallace@infotech.monash.edu.au
http://www.infotech.monash.edu.au/

Abstract. This paper surveys hybrid algorithms from a constraint programming perspective. It introduces techniques used within a constructive search framework, such as propagation and linear relaxation, as well as techniques used in combination with search by repair.

Keywords: constraint programming, hybrid algorithms, search.

1 Introduction

1.1 Tribes

There are three research communities exploring combinatorial optimisation problems. Within each community there is strong debate and ideas are shared naturally. Between the communities, however, there is a lack of common background and limited cross-fertilisation.

We belong to one of those communities: the CP community.[1] The other two are Mathematical Programming (MP) and Local Search and meta-heuristics (LS). Currently LS seems to be the largest of the three. It has become clear that such a separation hampers progress towards our common goal, and there should be one larger community - whose name is a point of contention - which should include us all.

Hybrid algorithms lie at the boundary between CP, MP and LS. We will explore some of the techniques used in MP and LS, and show how they can be used in conjunction with CP techniques to build better algorithms. We will not here be investigating the "frontiers of research" in these communities. However it is my belief that CP can contribute right now at these frontiers. Hybrid techniques are not peripheral to the research of any of these communities. They are the key to real progress in all three.

1.2 Overview

Firstly we explore the mapping of problems to algorithms, the requirement for problem decomposition, and the need for linking solvers and solver cooperation.

[1] There are also, of course, many people in the CP community who are not exploring combinatorial optimisation problems.

F. Azevedo et al. (Eds.): CSCLP 2006, LNAI 4651, pp. 1–32, 2007.

Different ways of linking solvers will be discussed, and some of their benefits and applications.

Secondly we will investigate different kinds of search, coming from the different communities, and see how they can be used separately, and together.

The paper is presented from a CP viewpoint, aimed at a CP audience. However, the objective is to lower the barrier to exploiting hybrid techniques, encouraging research at the boundaries of CP, MP and LS, and finally to help bring these communities together.

2 Hybrid Constraint Solving

2.1 The Conceptual Model and the Design Model

To solve a problem we start with a problem-oriented *conceptual* model. The syntax of conceptual models is targeted to clarity, expressive power and ease of use for people interested in solving the problem.

The conceptual model is mapped down to a *design* model which is machine-oriented [Ger01]. The design model specifies the algorithm(s) which will be used to solve the problem at a level that can be interpreted by currently implemented programming platforms, like ECLiPSe [AW06].

The CP community usually separates the *model* from the *search strategy*. The design model in our terminology includes both a model with variables and constraints and a search strategy. The variables and constraints in the design model are chosen so as to be easy to solve, and to fit with the search routine. Moreover with each constraint in the design model solving methods are specified which associate a behaviour with the constraint. Though the variables and constraints in the design model of a problem may be quite different from those in its conceptual model, they are logically equivalent - they represent the same set of solutions. In principle, the conceptual modeling language could be a subset of the design modeling language.

Real problems are complex and, especially, they involve different kinds of constraints and variables. For example a "workforce scheduling" problem [AAH95] typically involves the following decision variables:

- For each task, one or more *employees* assigned to the task.
- For each task, a *start time*
- For each (group of) employee(s), a *route* that takes them from task to task.
- For each (group of) employee(s), *shift start, end, and break times*

This is in sharp contrast to typical CSP puzzles and benchmarks, such as graph colouring, where all the variables are of the same "kind" and sometimes even have the same initial domains.

The constraints and data in real problems are also diverse. The workforce scheduling problem includes:

- *Location and distance* constraints on and between tasks
- *Skills* data and constraints on and between employees and tasks
- *Time* constraints on tasks and employee shifts

Naturally there are many more constraints in real workforce scheduling problems on vehicles, road speeds, equipment, team composition and so on.

The consequence is that the algorithm(s) needed to solve real problems are typically *hybrid*. The skills constraints are best solved by a different sub-algorithm from the routing constraints, for example.

2.2 Mapping from the Conceptual to the Design Model

To map a problem description to an algorithm, it is often necessary to decompose the whole problem into parts that can be efficiently solved. The challenge is to be able to glue the subproblem solutions together into a consistent solution to the whole problem. Moreover, for optimisation problems, it is not enough to find the optimal solution to each subproblem. Glueing these "local" optima together does not necessarily yield a "global" optimum.

For these reasons we need to ensure that the subproblem algorithms cooperate with each other so as to produce solutions that are both consistent with each other and, as near optimal as possible. The design of *hybrid algorithms* that meet these criteria is the topic of this section.

In principle we can map a conceptual model to a design model by

- Associating a behaviour, or a constraint solver, with each problem constraint
- Adding a search algorithm to make up for the incompleteness of the constraint solvers

In practice the design model produced by any such mapping is strongly influenced by the particular choice of conceptual model. The "wrong" conceptual model could make it very hard to produce an efficient algorithm to solve the problem.

For this reason we must map a given conceptual model to an efficient design model in two steps:

- Transform the conceptual model into another one that is more suitable for mapping
- Add constraint behaviour and search, to yield an efficient design model

The first step - transforming the conceptual model - is an art rather than a science. It involves four kinds of transformations:

- Decomposition - separating the constraints and variables into subproblems
- Transformation - rewriting the constraints and variables into a form more suitable for certain kinds of solving and search
- Tightening - the addition of new constraints whose behaviour will enhance efficient problem solving
- Linking - the addition of constraints and variables that will keep the separate subproblem solvers "in step" during problem solving

The decomposition is germane to our concerns. It is therefore worth discussing briefly here. Firstly, we note that the decomposition covers the original problem

(of course), but it is *not* a partition: otherwise the subproblems would have no link whatsoever between them.

Therefore some subproblems share some variables.[2] Each subproblem solver can then make changes to a shared variable, which can be used by the other solver. Sometimes constraints are shared by different subproblems. In this case the same constraint is handled multiple times by different solvers, possibly yielding different and complementary information within each solver. When the constraints in different subproblems are transformed in different ways, the result is that the same constraint may appear several times in several different forms in the transformed conceptual model. We shall see later a model with a resource constraint that is written three different ways for three different solvers.

We shall now move on to examine design models for a variety of hybrid algorithms.

3 Constraint Solvers

In this section we discuss different constraint solvers, and constraint behaviours. We investigate what kinds of information can be passed between them, in different hybrid algorithms, and how their cooperative behaviour can be controlled.

The solvers, and constraint behaviours, we will cover are

- Finite domain (FD) constraint propagation and solving
- Global constraints and their behaviour
- Interval constraints, and bounds propagation
- Linear constraint solving
- Propositional clause (SAT) solving
- Set constraint solving
- One-way constraints (or "invariants")

Referring back to the three research communities, we can relate these solvers to the CP and MP communities. Accordingly this work lies on the border of CP and MP. The hybrids on the border of CP and LS will be explored in the next section.

3.1 Modelling Requirements for Constraint Solvers

Supposing a constraint C appears in the conceptual model and it is to be handled by a particular solver, say FD. Then the design model must express a number of requirements.

Firstly it must associate an initial domain which each of the variables in em C. For the solver FD, the initial domains are discrete, and finite. For other solvers,

[2] This is a simplification. For example mathematical decomposition techniques such as Lagrangian relaxation and column generation use more sophisticated techniques than shared variables to relate the subproblems and the master problem. These will be discussed in sections 4.2 and 5.1 below.

such as SAT or linear, they must be initialised as booleans, or reals as required by the solvers.

Secondly the constraint should be explicitly associated with a particular solver. It may also be associated with more than one solver - and this is the subject of section 4 below.

Thirdly its activation conditions must be specified. A constraint can be activated by simply sending it to the solver, so that it is automatically handled whenever the solver is invoked, or by explicitly introducing conditions under which it should be woken. For an almost linear constraint, for example, which includes a few non-linear expressions, the constraint might only be woken and sent to the linear solver once all its expressions have become linear. The most common use of explicit waking is for propagation constraints, discussed in the following subsection.

No further specification about the constraints themselves need be given in the design model. Details about what information is passed to the constraint before solving it, and what information is extraced from the constraint after solving it is a property of the solver, rather than the individual constraint.

In principle, when there is a single search routine with a current state, the solvers communicate two key types of information to the state:

1. Satisfiability or inconsistency
2. Variable values

Other information can be extracted from the different solvers by explicit requests expressed in the design model. Variable values are also, typically, passed from the search state to the solvers. Thus when one solver, say FD, instantiates a variable, this information is made available to all the other solvers.

3.2 Constraints Which Propagate Domain Information

Information Exported. We first consider finite domains and global FD constraints. The relevant issues for hybrid algorithms are

- what information can be extracted from the solver
- under what conditions *all* the information has been extracted from the solver: i.e. the extracted information entails the constraint, so can we be sure that a state which appears to be feasible for the other subproblems and their solvers is also guaranteed to be consistent with the FD solver.

The answers are as follows:

- Information that can be extracted from the solver includes upper and lower bounds on the variables, domain size and if necessary precise information about which values are in the domain of each variable. The solver also reports inconsistency whenever a domain becomes empty.
- All the information has been extracted from a constraint when it is entailed by the current (visible) domain information. An FD solver can sometimes

detect when this is the case and "kill" the constraint. Until then, the constraint is still "active". Therefore active constraints are ones which, to the FD solver's knowledge, are not yet entailed by the domains of the variables. Some FD solvers don't guarantee to detect this entailment until all the variables have been instantiated. Many FD solvers support *reified* constraints, which have a boolean variable to flag entailment or disentailment (inconsistency with the current variable domains).

The domain, inconsistency and entailment information are all logical consequences of the FD constraints and input variable domains. For this reason, no matter what other constraints in other subproblems are imposed on the variables, this information is still correct. In any solution to the whole problem, the values of the FD variables must belong to the propagated domains. Inconsistency of the subproblem, implies inconsistency of the whole problem. If the variable domains entail the subproblem constraints, then they are still entailed when the constraints from the rest of the problem are considered.

Global Constraints. Notice that global constraints are often themselves implemented by hybrid techniques, even though the information imported and exported is restricted to the above. For example a feasible assignment may be recorded internally, so as to support a quick consistency check, or to speed up the propagation algorithm. An interesting case of hybridisation is the use of continuous variables in global constraints. The classic example is a global constraint for scheduling, where resource variables are FD, but the task start time variables could be continuous. As far as I know the hybrid discrete/continuous scheduling constraint is not yet available in any CP system.[3]

Interval Constraints. For interval constraint solvers only upper and lower bounds, and constraint entailment are accessible. The problem with continuous constraints is that they are not necessarily instantiated during search. Since continuous variables can take infinitely many different values, search methods that try instantiating variables to all their different possible values don't necessarily terminate. Instead search methods for continuous values can only tighten the variable's bounds, until the remaining interval associated with the variable becomes "sufficiently" small.

Not all values within these small intervals are guaranteed to satisfy all the constraints. Indeed there are common cases where, actually, there are no feasible solutions, even though the final intervals appear prima facie compatible. One vivid example is Wilkinson's problem (quoted in [Van98]). It has two constraints: $\prod_{i=1}^{20}(X+i)+P\times X^{19}=0$ and $X\in[-20.4..-9.4]$. When $P=0$ the constraints have 11 solutions ($X=-10\ldots X=-20$), but when P differs infinitesimally from 0 (viz. $P=2^{-23}$), it has no solutions!

For these reasons "answers" returned by search routines which associate small intervals with continuous variables are typically conjoined with a set of undecided constraints, which must be satisfied in any solution.

[3] CP scheduling will be covered in more detail later.

3.3 Linear Constraints

Underlying Principles. A linear constraint solver can only handle a very restricted set of constraints. These are linear numeric constraints that can be expressed in the form $Expr \geq Number$ or $Expr \leq Number$. The expression on the left hand side is a sum of *linear terms*, which take the form *Coefficient* $\times$ *Variable*. The coefficients, and the number on the right hand side, are either integers or real numbers [Wil99].

Linear constraint solvers are designed not only to find feasible solutions, but also to optimise against a cost function in the form of another linear expression.

In the examples in this chapter we shall typically write the linear constraints in the form $Expr \geq Number$, and assume that the optimisation direction is *minimisation*.

Whilst much less expressive than CP constraints, they have a very important property: any set of linear constraints, over real variables, can be tested for *global* consistency in polynomial time. This means we can throw *all* the linear constraint of a problem into the linear solver and immediately determine whether they are consistent.

By adding just one more kind of constraint, an *integrality* constraint that requires a variable to take only integer values, we can now express any problem in the class *NP*. (Of course the consistency problem for mixed linear and integrality constraints - termed MIP, for "Mixed Integer Programming" - is NP-hard).

The primary information returned by the linear solver is consistency or inconsistency among the set of linear constraints. However for building cooperative solvers we will seek more than this.

Firstly the solver can also export an optimal solution - assuming throughout this section that the cost function is linear. In general there may be many optimal solutions, but even from a single optimum we now have a known optimal value for the cost function. No matter how many other constraints there may be in other solvers, the optimal value cannot improve when they are considered, it can only get worse. Thus the linear solver returns a bound on the cost function.

Linear constraints are special because if S_1 and S_2 are two solutions (two complete assignment that satisfy all the linear constraints), then any assignment that lies on the line between S_1 and S_2 is also feasible. For example if $X = 1$, $Y = 1$ is a solution, and so is $X = 4$ and $Y = 7$, then we can be sure that $X = 2$ and $Y = 3$ is a solution, and so is $X = 3$ and $Y = 5$. Moreover since the cost function is linear, the cost of any solution on the line between $S1$ and $S2$ has a cost between the cost of $S1$ and the cost of $S2$.

These properties have some important consequences. Supposing $Expr \geq Number$ is a constraint, and that at the optimal solution the value of $Expr$ is strictly greater than $Number$. Then the problem has the same optimal value even if this constraint is dropped (or "relaxed"). Otherwise you could draw a line between a new optimal solution and the old one, on which all points are feasible for the relaxed problem. Moreover the cost must decrease continuously towards the new optimum solution. Therefore at the point where this line crosses the line $Expr = Number$ (i.e. at the first point where the solution is also feasible for the