Alan M. Frisch (Ed.)

Modelling and Reformulating Constraint Satisfaction Problems:

Towards Systematisation and Automation

Second International Workshop
Kinsale, Ireland, September 2003
Proceedings

Held in conjunction with the
Ninth International Conference on
Principles and Practice of
Constraint Programming (CP 2003)

Foreword

Many organisations have scheduling, assignment, supply chain and other problems that could be solved with a constraint programming toolkit. Although the solution of these problems is of vital importance, constraint programming toolkits are not widely used because there is insufficient expertise available to model problems as constraint programs.

This so-called modelling bottleneck could be reduced by the development of a general, principled understanding of modelling that in the future could guide the manual or automatic formulation of models and the choice among alternative models.

Researchers and practitioners have developed effective models for a wide range of problems. The time has come to form generalisations from these case studies that can be used to guide modelling in the future. These generalisations could then be systematised for use by a non-expert and be codified in textbooks in much the same way that data structuring expertise is. Ultimately this modelling expertise could be embedded in automated modelling tools. Progress on any of these fronts would bring the proven power of constraint programming to a wider user base.

This "Second International Workshop of Modelling and Reformulating Constraint Satisfaction Problem: Towards Systematisation and Automation" was convened to provide a forum for researchers who share these goals.

This volume contains the thirteen contributed papers presented as talks or posters at the workshop as well as an abstract of the talk presented by the invited speaker.

I wish to thank all the authors who submitted papers to this workshop; the members of the programme committee; the invited speaker, Pascal Van Hentenryck; the CP'2003 Tutorial and Workshop Chair, Christian Bessière; and two people who assisted with the typesetting of these proceedings, Chris Jefferson and Peter Nightingale.

August 2003

Alan M. Frisch
Programme Chair

Programme Committee

Additional Reviewers

Table of Contents

I Invited Paper

Combinatorial Modelling for Combinatorial Optimization (Abstract) 1
Pascal Van Hentenryck

II Contributed Papers

Towards Automatic Modelling of Constraint Satisfaction Problems: A System Based on Compositional Refinement 2
Adam Bakewell, Alan M. Frisch, Ian Miguel

An Empirical Study of Mutual Routing-Scheduling Reformulation 18
J. Christopher Beck, Patrick Prosser, Evgeny Selensky

Automated Reformulation of Specifications by Safe Delay of Constraints . 33
Marco Cadoli, Toni Mancini

Semi-Automatic Modeling by Constraint Acquisition 48
Remi Coletta, Christian Bessiere, Barry O'Sullivan, Eugene C. Freuder, Sarah O'Connell, Joel Quinqueton

Introducing ESRA, a Relational Language for Modelling Combinatorial Problems 63
Pierre Flener, Justin Pearson, Magnus Ågren

Encoding Connect-4 Using Quantified Boolean Formulae 78
Ian P Gent, Andrew G D Rowley

Why Channel? Multiple Viewpoints for Branching Heuristics 94
Brahim Hnich, Toby Walsh

Preference Constraints: New Global Soft Constraints Dedicated to Preference Binary Relations 104
Rémy-Robert Joseph, Peter Chan, Michael Hiroux, Georges Weil

A Regular Language Membership Constraint for Sequences of Variables .. 110
Gilles Pesant

Boolean and Pseudo-Boolean Models for Scheduling 120
Steven Prestwich and Colin Quirke

Bridge Problem as Vehicle Routing 132
Evgeny Selensky

Search Strategies for Optimization: Modelling the SONET Problem 143
Barbara M. Smith

A Generalisation of the Backtracking Algorithm 158
M.R.C. van Dongen

Combinatorial Modeling for Combinatorial Optimization (Abstract)

Pascal Van Hentenryck

Brown University, Box 1910, Providence, RI 02912

One of the main strengths of constraint programming is what I call *combinatorial modelling*, i.e. the ability to model combinatorial substructures of an application explicitly. Besides providing modeling concepts closer to the application at hand, this fundamental feature of constraint programming makes it possible to design specialized filtering algorithms for these combinatorial structures, as well as dedicated search procedures which embody branching strategies and heuristics tailored to the substructures. As a consequence, constraint programming strikes an excellent tradeoff between specificity and generality, by exposing the problem structure while promoting reuse through a library of combinatorial structures.

This talk argues that combinatorial modeling is not limited to constraint programming and benefits other combinatorial optimization paradigms in similar ways, although the technical details may differ considerably. It reviews recent developments in mathematical programming and local search which highlight the benefits of combinatorial modeling and its ubiquity. It also discuss the implications of this generalization for traditional modeling topics such as symmetries, reformulation, and hybridization.

Towards Automatic Modelling of Constraint Satisfaction Problems: A System Based on Compositional Refinement

Adam Bakewell, Alan M. Frisch, and Ian Miguel

AI Group, Dept. Computer Science, University of York, UK
{ajb,frisch,ianm}@cs.york.ac.uk

Abstract. Many and diverse search problems have been solved with great success using constraint programming. However, in order to apply constraint programming tools to a particular domain, the problem must be *modelled* as a constraint satisfaction or optimisation problem. Since constraints provide a rich language, typically many alternative models exist. Formulating a *good* model therefore requires a great deal of expertise on the part of the modeller. This paper describes a prototype automated modelling system that *refines* an abstract specification of a problem into a set of alternative constraint programs. Refinement is compositional: alternative constraint programs are generated by composing refinements of the components of the specification. Since the high-level specification language is significantly closer than a constraint program to the way in which problems are commonly specified in natural language, the modelling bottleneck is substantially reduced.

1 Introduction

Modelling combinatorial problems in a constraint programming language can be automated. In particular, the task of transforming a combinatorial problem in an abstract, high-level specification language into an effective program (specified in an existing constraint programming language) for solving the problem can be automated. This is our conjecture and it is the aim of this paper to provide initial support for it.

The successful development of an automated modelling system would have significant practical consequences. To solve complex—and, often, important—problems with existing constraint satisfaction toolkits requires considerable expertise in modelling the problem with the low-level constructs provided by the toolkit. Solving the problem with the use of an automated modelling system would require significantly less expertise; one would need only to specify the problem in a high-level language. By bringing this language closer to the way that humans conceptualise problems and express them in natural language, we reduce the expertise required for specifying the problem.

Our belief is that modelling expertise can be systematised, catalogued and, ultimately, formalised and embedded in an automated system. Not only would the automated system be useful, but a catalogue of systematic modelling techniques would be of great benefit to the constraint modelling community. Such a catalogue could be consulted to find appropriate modelling techniques and to find what is known about the tradeoffs among alternatives. This idea is detailed and illustrated under the label of "constraint patterns" [9].

We speculate that in some respects an automated modelling system could outperform a human expert. The automated system could embody the modelling techniques that are employed by many experts, thus having at its disposal more techniques than any single expert. The automated system could be systematic in considering combinations of alternatives, whereas a human might overlook some. Also, a human might be overwhelmed by the size and complexity of a specification. We believe that a system could be highly-capable of combining existing modelling techniques in novel and complex ways. However, it is not our goal to produce a system that could discover new modelling techniques.

We divide our pursuit of producing an automated modelling system into two stages. The goal of the first stage, the *competence stage*, is to develop a system that generates a set of feasible models for a given specification. This set should contain all the effective models that would be produced by a human expert modeller. The set might also contain ineffective models and effective models that might not be produced by a human expert. The goal of the second stage of research, the *performance stage*, is to endow the system with some ability to evaluate the models it generates. Thus, the system would not generate models that are obviously ineffective and, as human experts can do, it might suggest that some of the alternatives are likely to be more effective than others.

This paper contributes to the goals of the competence stage. We believe that the key to the competence stage is a fundamental conjecture, *model compositionality*: the effective models for a combinatorial problem can be constructed by composing models for each component of the problem specification.[1] The contribution of this paper is to demonstrate, by example, that refinement can be performed compositionally; that is, a set of alternative refinements of a specification can be generated by composing refinements of the components of the specification. In particular, we present a set of compositional refinement rules that can transform a specification in a high-level source language into alternative models expressed in a target language whose constructs are similar to those provided by current constraint programming languages. The alternative models employ different structures for representing the decision variables and some employ redundant structures linked by channelling constraints, which are also generated automatically.

We have implemented in Haskell an automated refinement system that incorporates a set of refinement rules that generalises and extends those presented here as well as a β-reduction rule to simplify λ-expressions. The tool automatically produces models for problems like BIBD—for which it generates 8 models—and the SONET problem [7] for which it generates 64 models.

[1] Of course, the ways that models compose may be highly complex and uncovering them may be difficult.

2 A Brief Introduction to Our Specification Language

This section briefly introduces the specification language in which we specify problems to be refined. Since this paper focuses on refinement rather than language design, we present only those features of the language that are necessary to proceed with the presentation of refinement. Here we merely present the language; later we discuss how the demands of refinement have driven us to this design.

Throughout this and the following two sections, the *Balanced Incomplete Block Design* (BIBD) problem (problem 28 at `www.csplib.org`) is used as an example. Despite its relative simplicity, this problem has important practical applications, such as cryptography and experimental design. Here it illustrates our language. The following is an English definition of the BIBD problem (from CSPLib).

> Given positive integers, nv, nb, k, r and l, arrange nv distinct objects in nb blocks such that each block contains k distinct objects, each object occurs in exactly r different blocks and every two distinct objects occur together in exactly l blocks.

With this problem in mind, we now turn our attention to our specification language. A problem specification is a triple consisting of a list of *declarations*, a set of *decision variables* and a set of *constraints*. We write this triple as

declarations `find` *decision variables* `such that` *constraints*

The list of declarations is used to define *parameters* and *constants*. The parameters to a problem specify the instance of the problem class. The value of a parameter is provided by the user in specifying the problem instance to be solved. The BIBD problem has parameters nv, nb, r, k, and l. The decision variables denote the combinatorial objects that are to be found by the solver. The goal in the BIBD problem is to find a relation, which we shall call $BIBD$, between the blocks and the objects.

The specification language is strongly typed and every expression has a type. The five parameters in the BIBD problem are of type *natural*. The type of relation $BIBD$ is the powerset of $B \times V$, written $\wp(B \times V)$, where B and V are primitive types representing the blocks and objects. B and V are constant types and therefore must be declared in the list of declarations. Putting these pieces together, the specification of the BIBD is of the form

`given` nv, nb, r, k, l : nat
`letting` B be type-of-size nb, V be type-of-size nv
`find` $BIBD : \wp(B \times V)$
`such that` *constraints*

We see here that *declarations* is a list of intermingled `given` statements, which declare the parameters, and `letting` statements, which declare the constants (which may themselves be parameterised). B and V are declared to be primitive types. As in many constraint specification languages, our language allows a type (or domain) to be specified by enumerating the members. Unlike other languages, a type can also be specified by giving its size and not enumerating its members.

The specification language supports fully-compositional type constructors. So, for example, a decision variable may be of type integer, of type set of integer, of type set of set of integer, and so forth. More formally, if $\tau_1, \ldots, \tau_n$ are types then so are $\tau_1 \times \cdots \times \tau_n$ (a cross product), $\wp(\tau_1)$ (the powerset of τ_1), $\wp_n(\tau_1)$ (the set of all n-element subsets of τ_1), $\tau_1 \rightarrow \tau_2$ (a partial function from τ_1 to τ_2), partition τ_1 (a partitioning of τ_1), and regpart τ_1 n (a partitioning of τ_1 into subsets of size n).

Constraints in this specification language are built from the declared parameters, constants and decision variables using operators commonly found in mathematics and in other constraint specification languages. The language also includes universally quantified variables ranging over any specified finite type. Each subexpression in a constraint is typed. The *constraints* component of the BIBD problem can be expressed as

$$\forall b : B \cdot |BIBD(b, \_)| = k \tag{1}$$

$$\forall v : V \cdot |BIBD(\_, v)| = r \tag{2}$$

$$\forall \{v, v'\} : \wp(V) \cdot |BIBD(\_, v) \cap BIBD(\_, v')| = l \tag{3}$$

Here $\{v, v'\} : \wp(V)$ is a two-element set containing objects of type V, and $BIBD(b, \_)$ is the projection of $BIBD$ onto b—that is, $\{v : V | BIBD(b, v)\}$.

3 Introducing Refinement Rules

This section introduces a set of *refinement rules* that refine specifications into constraint programs. Each rule specifies, by means of a refinement operator ρ, the refinement of a specification in our specification language into a constraint language similar to a constraint program. Refinement requires a number of mechanisms, which are introduced incrementally through a sequence of rule sets of increasing power. The rule sets are distinguished by subscripting the ρ operator.

The refinement rules in our implementation are formulated to handle relations of arbitrary arity. To simplify the presentation, the rules presented in this paper operate on relations of only a fixed arity.

3.1 Building One Model at a Time

Let us now present domain independent refinement rules using the refinement of the BIBD problem as an example. The rule sets given in this sub-section are simplified in that they assume that a single model is being produced using one type of target-level object. The subsequent sub-section considers how to produce a set of alternative models containing different types of object.

Refinement to a 01 Matrix Model A binary relation variable $R : \wp(D_1 \times D_2)$ can be represented by a 2-dimensional 01 matrix that is indexed by D_1 and D_2, which is named R_{M01}.

RELATION1: $\rho_1(R : \wp(D_1 \times D_2)) = \lambda d_1, d_2.R_{M01}[d_1, d_2] : D_1 \times D_2 \rightarrow 01$

The relationship between the two structures is:

$$\forall d_1 : D_1, d_2 : D_2 \cdot R(d_1, d_2) \leftrightarrow R_{M01}[d_1, d_2] = 1 \qquad (4)$$

We have two rules to refine a projection onto either argument of a binary relation. For any expression ϕ of type $\wp(D_1 \times D_2)$, any x of type D_1 and any x' of type D_2:

PROJECTION1A: $\rho_1(\phi(x, \_)) = \lambda d_2.(\rho_1(\phi)(x, d_2)) : D_2 \rightarrow 01$
PROJECTION1B: $\rho_1(\phi(\_, x')) = \lambda d_1.(\rho_1(\phi)(d_1, x')) : D_1 \rightarrow 01$

The Application of the RELATION1 and PROJECTION1A rules to $BIBD(b, \_)$,a sub-expression of constraint (1), produces:

$$\rho_1(BIBD(b, \_)) = \lambda v.BIBD_{M01}[b, v]$$

Another rule refines the cardinality of a relational expression ϕ:

CARDINALITY1: $\rho_1(|\phi|) = \sum_{d:D} \rho_1(\phi)(d) : \text{nat}$, where $D \rightarrow 01$ is the type of $\rho_1(\phi)$

Three more rules are required to complete the example refinement:

EQUALITY1: $\rho_1(\phi = \phi') = (\rho_1(\phi) = \rho_1(\phi')) : 01$.
CONST/PARAM1: $\rho_1(k : D) = k : D$, where k is any constant or parameter or quantified variable.
QUANTIFIER1: $\rho_1(\forall d : D \cdot \phi) = \forall d : D \cdot \rho_1(\phi)$

Using the given rules, constraints (1) and (2) can now be refined:

$$\rho_1(\forall b : B \cdot |BIBD(b, \_)| = k) = \forall b : B \cdot \sum_{v:V} BIBD_{M01}[b, v] = k$$
$$\rho_1(\forall v : V \cdot |BIBD(\_, v)| = r) = \forall v : V \cdot \sum_{b:B} BIBD_{M01}[b, v] = r$$

To refine constraint (3), we need one further rule for the intersection of two expressions ϕ and ϕ', both of type $\wp(D)$.

INTERSECTION1: $\rho_1(\phi \cap \phi') = (\lambda d.(\rho_1(\phi)(d) \wedge \rho_1(\phi')(d))) : D \rightarrow 01$

The refinement of constraint (3) completes the refinement of the specification of the BIBD problem to a 01 matrix model:

$$\rho_1(\forall \{v, v'\} : \wp(V) \cdot |BIBD(\_, v) \cap BIBD(\_, v')| = l) =$$
$$\forall \{v, v'\} : \wp(V) \cdot \sum_{b \in B} BIBD_{M01}[b, v] \wedge BIBD_{M01}[b, v'] = l$$

Refinement to 1-d Matrix of Sets Indexed by First Argument The previous subsection showed how the well-known 01 matrix model of the BIBD problem can be generated by refining the relational description. There are, however, several other alternatives. For example, another way to represent the BIBD relation is with a 1-dimensional matrix of set variables. Here we specify a refinement operator, ρ_2, to generate this alternative.

For every relation $R : \wp(D_1 \times D_2)$, R_{MD_1} denotes a 1-dimensional matrix indexed by D_1 whose elements are subsets of D_2.

Relation2**:** $\rho_2(R : \wp(D_1 \times D_2)) = \lambda d.R_{MD_1}[d] : D_1 \to \wp(D_2)$

The relationship between the two structures is:

$$\forall \langle d_1, d_2\rangle : D_1 \times D_2 \cdot R(d_1, d_2) \leftrightarrow d_2 \in R_{MD_1}[d_1] \tag{5}$$

The refinement rules for the other constructs are identical to those for ρ_1, except that we replace Projection1A, Cardinality1 and Intersection1 with

Projection2A**:** $\rho_2(\phi(x, \_)) = \rho_2(\phi)(x) : \wp(D_2)$, where $\rho_2(\phi)$ has type $D_1 \to \wp(D_2)$ and x has type D_1.
Cardinality2**:** $\rho_2(|\phi|) = |\rho_2(\phi)| :$ nat, where $\rho_2(\phi)$ has type $\wp(D)$.
Intersection2**:** $\rho_2(\phi \cap \phi') = (\rho_2(\phi) \cap \rho_2(\phi')) : D$, where $\rho_2(\phi), \rho_2(\phi')$ have type $\wp(D)$.

In Cardinality2 we have assumed that set variables have a built-in cardinality operator. There is no Projection2B rule as there is no direct way to implement it with this model of the *BIBD* relation.

Constraint (1) can now be refined as follows:

$$\rho_2(\forall b : B \cdot |BIBD(b, \_)| = k) = \forall b : B \cdot |BIBD_{MB}[b]| = k \tag{6}$$

Constraints (2) and (3) cannot be refined because there is no Projection2B to refine a projection onto the second argument.

Refinement to 1-d Matrix of Sets Indexed by Second Argument Of course, we can also model the *BIBD* relation by a 1-dimensional matrix indexed by the second argument of the relation. Here we specify a refinement operator, ρ_3, to generate this alternative.

For every relation $R : \wp(D_1 \times D_2)$, R_{MD_2} denotes a 1-dimensional matrix indexed by D_2 whose elements are subsets of D_2.

Relation3**:** $\rho_3(R : \wp(D_1 \times D_2)) = \lambda d.R_{MD_2}[d] : D_2 \to \wp(D_1)$

The relationship between the two structures is:

$$\forall \langle d_1, d_2\rangle : D_1 \times D_2 \cdot R(d_1, d_2) \leftrightarrow d_1 \in R_{MD_2}[d_2] \tag{7}$$

The refinement rules for the other constructs are identical to those for ρ_2, except that we replace Projection2A with

PROJECTION3B: $\rho_3(\phi(\_, x)) = \rho_3(\phi)(x) : \wp(D_1)$, where $\rho_3(\phi)$ has type $\wp(D_1 \times D_2)$, and x has type D_2.

This time there is no PROJECTION3A rule as there is no direct way to implement it with this model of the $BIBD$ relation.

Constraints (2) and (3) can now be refined as follows:

$$\rho_3(\forall v : V \cdot |BIBD(\_, v)| = r) = \forall v : V \cdot |BIBD_{MV}[b]| = r \tag{8}$$

$$\begin{aligned} &\rho_3(\forall \{v, v'\} : \wp(V) \cdot |BIBD(\_, v) \cap BIBD(\_, v')| = l) = \\ &\quad \forall \{v, v'\} : \wp(V) \cdot |BIBD_{MV}[v] \cap BIBD_{MV}[v']| = l \end{aligned} \tag{9}$$

However, constraint (1) cannot be refined into this representation. The solution to this problem is to use ρ_2 to model constraints (1), use ρ_3 to model constraints (2) and (3), and introduce *channelling* constraints [1] to connect the two representations. This necessitates the use of a more powerful rule mechanism that enables a single rule to generate multiple representations simultaneously. We now turn our attention to this.

3.2 Building All Models

Good CP models are often *hybrids* that contain multiple representations of an object of interest. This is beneficial when some constraints are easily stated on one representation, and other constraints are easily stated on an alternative representation. As in the refinement of the BIBD problem in the previous section, constraint (1) might be refined using R_{MB}, while constraints (2) and (3) might be refined using R_{MV}. In order to produce such models, a refinement system must:

1. have rules that support each alternative representation,
2. select, according to type, the correct refinement in the context of a partially refined expression, and
3. add channelling constraints to maintain consistency between different representations.

Point 1 is achieved by defining the ρ_4 operator to produce a *set* of typed refined expressions. For example, a single relation refinement rule can produce the three refinements that were previously generated by the RELATION1, RELATION2 and RELATION3 rules:

RELATION4: $\rho_4(R : \wp(D_1 \times D_2)) = \{\, \lambda d_1, d_2 . R_{M01}[d_1, d_2] : D_1 \times D_2 \rightarrow 01,$
$\lambda d . R_{MD_1}[d] : D_1 \rightarrow \wp(D_2),$
$\lambda d . R_{MD_2}[d] : D_2 \rightarrow \wp(D_1)\}$

That is, a binary relation variable is refined to a set of three typed expressions, in this case a 01 matrix and two one-dimensional matrices of sets.

Point 2 is achieved by selecting from the refined expressions by type. For example, consider the following rule for projection of a relation onto its first argument:

PROJECTION4A: $\rho_4(\phi(v, \_)) =$
$$\{\lambda d_2.(X(v, d_2)) : D_2 \to 01 | X : D_1 \times D_2 \to 01 \in \rho_4(\phi)\} \quad \cup$$
$$\{X(v) : \wp(D_2) | X : D_1 \to \wp(D_2) \in \rho_4(\phi)\}$$

The set of expressions produced by this rule contains at most one element per element of the set of refinements of ϕ. Each expression is generated according to the type of the corresponding refinement of ϕ, as dictated by the conditions.

As a further example, the following rule refines the intersection of two binary relations:

INTERSECTION4: $\rho_4(\phi \cap \phi') =$
$$\begin{aligned}
&\{(\lambda d_1, d_2.X(d_1, d_2) \wedge X'(d_1, d_2)) : D_1 \times D_2 \to 01 \\
&\qquad | \ (X : D_1 \times D_2 \to 01 \in \rho_4(\phi)) \wedge (X' : D_1 \times D_2 \to 01 \in \rho_4(\phi'))\} \quad \cup \\
&\{(\lambda d_1.X(d_1) \cap X'(d_1)) : D_1 \to \wp(D_2) \\
&\qquad | \ (X : D_1 \to \wp(D_2) \in \rho_4(\phi)) \wedge (X' : D_1 \to \wp(D_2) \in \rho_4(\phi'))\} \quad \cup \\
&\{(\lambda d_2.X(d_2) \cap X'(d_2)) : D_2 \to \wp(D_1) \\
&\qquad | \ (X : D_2 \to \wp(D_1) \in \rho_4(\phi)) \wedge (X' : D_2 \to \wp(D_1) \in \rho_4(\phi'))\}
\end{aligned}$$

In this case, the set of expressions produced contains at most one element per pair of elements in the cross product of the refinements of ϕ and ϕ' where both elements have the same type.

Using rules of this form, a set of alternative models is generated by taking the cross product of the sets of refinements of each object level constraint. Reconsider the model represented by constraints (6), (8) and (9). The presence of two representations of the BIBD relation in this model underlines the importance of point 3 above.

Point 3 is achieved by adding channelling constraints to maintain consistency between these two representations. Recall that the object and target-level representations are linked using axioms (5) and (7). In this case, the axioms produce:

$$\forall \langle b, v\rangle : B \times V \cdot BIBD(b, v) \leftrightarrow v \in BIBD_{MB}[b] \tag{10}$$
$$\forall \langle b, v\rangle : B \times V \cdot BIBD(b, v) \leftrightarrow b \in BIBD_{MV}[v] \tag{11}$$

Using constraints (10) and (11), it is simple to construct channelling constraints to connect the two target-level objects:

$$\forall \langle b, v\rangle : B \times V \cdot v \in BIBD_{MB}[b] \leftrightarrow b \in BIBD_{MV}[v]$$

In order to illustrate this set-based refinement mechanism, reconsider the refinement of constraint (1). We proceed in an inside-out manner to show how the refined objects are formed and composed. First, RELATION4 is used to refine the BIBD relation itself into a set of three target-level objects:

$$\begin{aligned}
\rho_4(BIBD) = \{ \ &\lambda d_1, d_2.BIBD_{M01}[d_1, d_2] : B \times V \to 01, \\
&\lambda d.BIBD_{MB}[d] : B \to \wp(V), \\
&\lambda d.BIBD_{MV}[d] : V \to \wp(B)\}
\end{aligned}$$

Note how the conditions of Projection4a capture successfully the situation described in Section 3.1: only two of these three objects are permitted to be projected onto $b : B$. Hence, the unsupported refinement of projecting $BIBD_{MV} : V \rightarrow \wp(B)$ onto an element of B is not made.

$$\begin{aligned}\rho_4(BIBD(b, \_)) = \{&\lambda v.BIBD_{M01}[b, v] : V \rightarrow 01,\\ &BIBD_{MB}[b] : \wp(V)\}\end{aligned}$$

The set-based version of the cardinality rule is introduced before continuing:

Cardinality4: $\rho_4(|\phi|) = \{\sum_{d \in D} X(d) : \text{nat} \mid X : D \rightarrow 01 \in \rho_4(\phi)\} \;\cup$
$\{|X| : \text{nat} \mid X : \wp(D) \in \rho_4(\phi)\}$

A larger sub-expression of constraint (1) can now be refined:

$$\begin{aligned}\rho_4(|BIBD(b, \_)|) = \{ &\textstyle\sum_{v \in V} BIBD_{M01}[b, v] : \text{nat},\\ &|BIBD_{MB}[b]| : \text{nat}\}\end{aligned}$$

The Equality4, Const/Param4, and Quantifier4 rules follow a similar pattern, leading to the full refinement of constraint (1):

$$\begin{aligned}\rho_4(\forall b : B \cdot |BIBD(b, \_)| = k) = \{ &\forall b : B \cdot \textstyle\sum_{v \in V} BIBD_{M01}[b, v] = k : 01,\\ &\forall b : B \cdot |BIBD_{MB}[b]| = k : 01\}\end{aligned}$$

This example has shown how alternative refinements of abstract expressions can be composed via dynamic type checking.

4 Identifying and Breaking Symmetry

Experienced modellers know that the effective solution of many problems requires identifying and breaking many of the symmetries that are present in a model of the problem. We claim that symmetries in models arise from (at least) two sources: some symmetries may be present in the original statement of the problem and other symmetries may be introduced by the refinement process. Though it is important to deal with both types of symmetry, dealing with the latter type is central to the study of refinement.

Refinement introduces symmetries when it creates a model that distinguishes between objects that are not distinguished in the original problem formulation. For example, the original statement of the BIBD problem refers to a set B of blocks and a set V of objects, but does not distinguish among the elements of each set; no particular block or object is named. In the 01 matrix model given in Section 3.1, the blocks and objects serve as indices of the two dimensional matrix $BIBD_{M01}$. Indices of a matrix are distinguished; one index refers to the first row/column of a matrix, another refers to the second row/column, and so forth. Hence $BIBD_{M01}$ has row and column symmetry: its rows can be interchanged and its columns can be interchanged [2].[2]

[2] More precisely, an assignment of values to the variables of the matrix is a solution to the problem if and only if it is a solution after swapping the values assigned to any two rows/columns.

We now show how our refinement rules can be enhanced to to recognise the symmetries that they introduce and to incorporate appropriate symmetry-breaking techniques into the models they produce. In the case of the BIBD example, it is relatively simple to infer that B and V each contain indistinguishable elements and hence a refined matrix indexed by B or V (or both) will have index symmetry. The declarations of B and V as types do not name their element; hence there is no way that an individual element of either B or V can be distinguished by the problem constraints.

We enhance our refinement operator so that with each alternative refinement that it generates it associates a set of declarations, a set of annotations and a set of constraints that are to be added to the model. For the purpose of symmetry breaking, the refinement operator produces annotations that identify the symmetry that is to be broken. A later stage of refinement can then use these annotations to break the symmetry, either by adding symmetry-breaking constraints to the model or by using a search technique that dynamically prunes symmetric branches of the search space. Hence, our new refinement operator ρ_5, generates a set of (up to) 4-tuples, where each 4-tuple is of the form

– e `letting` *declarations* `annotate` *annotations* `such that` *constraints*

Here e is the refined expression (as in previous rules), *declarations* is the set of constant declarations to add to the model, and *annotations* and *constraints* are the set of annotations and constraints to add to the model. For the present purpose of symmetry detection we add only annotations. However, in the more complex example considered in the next section, all three facilities are necessary.

As an example, consider the Relation5 rule:

```
RELATION5: $\rho_5(R : \wp(D_1 \times D_2)) =$
    $\{\ \lambda d_1, d_2.R_{M01}[d_1, d_2] : D_1 \times D_2 \rightarrow 01$
            annotate $\{\text{symIndex}(R_{M01}, D_1, \text{equivClasses}(D_1)),$
                     $\text{symIndex}(R_{M01}, D_2, \text{equivClasses}(D_2))\},$
      $\lambda d.R_{MD_1}[d] : D_1 \rightarrow \wp(D_2)$
            annotate $\{\text{symIndex}(R_{MD_1}, D_1, \text{equivClasses}(D_1))\},$
      $\lambda d.R_{MD_2}[d] : D_2 \rightarrow \wp(D_1)$
            annotate $\{\text{symIndex}(R_{MD_2}, D_2, \text{equivClasses}(D_2))\}\}$
```

As per Relation4 (see Section 3.2), when Relation5 is applied to a binary relation, it produces a set of three alternative refinements: a two-dimensional 0/1 matrix, and two one-dimensional matrices of sets. Since this refinement has the potential to introduce symmetry, the rule checks whether symmetry has been introduced by checking whether one or more equivalence classes can be formed from the elements of the index set(s) of the matrix. If so, the model is annotated to reflect the fact that a matrix has symmetry in one or more of its indices. A decision can then be taken as to how to break this symmetry. If more than one matrix in a model is indexed by the same symmetrical set, symmetry must be broken consistently in all cases [2]. It should be clear that the symmetry declaration is local to an element of the set of refinements of R: R_{MD_2} cannot introduce symmetry on the indices of D_1, whereas R_{M01} can.

We shall not elaborate further on the mechanisms for dealing with symmetry, but instead close by noting that refining a specification that includes indistinguishable objects inevitably introduces symmetry. In its basic conception, an instance of the finite domain constraint satisfaction problem consists of a set of *named* variables, each of which is associated with a finite domain of *named* values. Hence, refining an abstract specification with indistinguishable objects must name, and thus distinguish, the objects, thereby introducing symmetry.

5 Refining Compositional Types

We now describe the refinement of a more complex problem, Social Golfers [6] (Problem 10 at www.csplib.org), which further illustrates compositional refinement and symmetry detection. An English definition of the problem is as follows:

> Schedule p golfers into groups of size s for each of w weeks, in such a way that any two golfers play in the same group at most once.

The schedule can be represented by a set S, of fixed size w, of regular partitions, so this problem can be expressed as follows:

```
given $p, w, s$:nat
letting $G$ be type-of-size $p$
find $S : \wp_w(\text{regpart}\ G\ s)$
such that $\forall \{R, R'\} \subseteq S \cdot \forall g \in R, g' \in R' \cdot |g \cap g'| \leq 1$
```

where $R : \wp(\text{regpart}\ G\ s)$, and $R' : \wp(\text{regpart}\ G\ s)$ are regular partitions in S; $\{R, R'\}$ is a fixed-size subset of S; and $g : \wp_s(G)$ and $g' : \wp_s(G)$ are groups in a regular partition. Hence, any two groups have at most one player in common.

This problem is specified in terms of a sized relation, the refinement of which has not been discussed previously. Initially, therefore, consider the following refinement rule in the style of ρ_5, which assumes that the arguments of the sized relation have atomic types. For every sized relation $R : \wp_n(D)$, R_{MI} denotes a 1-dimensional matrix whose elements are of type D and that is indexed by I, a locally introduced type of size n:

```
SizedReln5: $\rho_5(R : \wp_n(D)) =$
    $\{\ \lambda i.R_{MI}[i] : I \rightarrow D$
            letting $I$ be type-of-size $n$
            annotate symIndex$(R_{MI}, I)\}$
```

Note that, via I, we have distinguished the index of each element of R, whereas the original relation representation did not. This introduces a symmetry which is recorded in the annotations of this refinement. It is also possible to refine R to a 01 matrix, with constraints to ensure that a total of n tuples are admitted. For brevity, these rules are omitted, since the above refinement is the most straightforward when the arguments of the relation have molecular types, as per $S : \wp_w(\text{regpart}\ G\ s)$ from the Social Golfers example.

General compositional refinement, which must support arbitrarily nested molecular type expressions, requires a recursive application of the refinement rules to ensure that the input specification is fully refined. This approach is embodied by our new refinement operator, ρ_6. The general version of the sized relation refinement rule provides an example. We saw how SizedReln5 refines a relation $R : \wp_n(D)$ to a one-dimensional matrix $R_{MI} : I \rightarrow D$, where I is a locally introduced type of size n. The generalised rule refines a relation $R : \wp_n(\tau)$, where τ is an arbitrary type expression, to a one-dimensional matrix $R_{MI} : I \rightarrow \rho_6(\tau)$. Instead of refining each element of R_{MI} individually, which could lead to an exponential number of refinements of R, we refine an arbitrary element of τ and ensure that every element of R_{MI} is refined identically:

```
SizedReln6: $\rho_6(R : \wp_n(\tau)) =$
    $\{\ \lambda i.R_{MI}[i] : I \rightarrow \tau'$
            letting $I$ be type-of-size $n, \forall i : I \cdot dcls[R_{MI}[i]/e']$
            annotate symIndex$(R_{MI}, I), \forall i : I \cdot anns_1[R_{MI}[i]/e']$
            such that $\forall i : I \cdot cstr[R_{MI}[i]/e']$
        $|$   $(e' : \tau'$ letting $dcls$ annotate $anns_1$ such that $cstr) \in \rho_6(e : \tau)\}$
```

More concretely, to refine τ we introduce a variable e to stand for an arbitrary element of τ and refine it to give $e' : \tau'$. The declarations, annotations and constraints associated with e' are lifted to hold for all elements of τ', with the appropriate element of R_{MI} substituted for e' (where $exp[x/y]$ denotes the substitution of x for all occurrences of y in exp). Refining an element of an atomic type yields an unchanged type and no declarations, annotations or constraints.

The regular partition type is also central to the specification of the Golfers problem. For every regular partition $R : regpart\ \tau\ n$, let $R_f : \tau \rightarrow P$ denote a function from τ to a locally declared type P (for partition identifiers). Since R_f is not atomic, it too must be refined. The refinement of R_f is denoted R'_f.

```
RegPartition6: $\rho_6(R : regpart\ \tau\ n) =$
    $\{\ R'_f$
            letting $P$ be type-of-size $|\tau|/n$, $dcls_1$, $dcls_2$
            annotate $anns_1$, $anns_2$
            such that $C \wedge cstr_1 \wedge cstr_2$
        $|$   $R'_f$ letting $dcls_1$ annotate $anns_1$ such that $cstr_1$
                $\in \rho_6(R_f : \tau \rightarrow P) \wedge$
            $C$ letting $dcls_2$ annotate $anns_2$ such that $cstr_2$
                $\in \rho_6(\forall x : P \cdot |R'^{-1}_f(x)| = n)\}$
```

That is, if R is a regular partition of τ into disjoint subsets of size n, its refinement is R'_f, the refinement of a total function R_f, mapping τ into the new type P. To ensure that R'_f accurately represents R, we introduce and refine the constraint $\forall x : P \cdot |R'^{-1}_f(x)| = n$ (i.e. all partitions have size n, following refinement).

Now let us consider an example illustrating these two rules. If RegPartition6 is applied to an element $R \in S$ in the Social Golfers problem, and we assume that the function representing the partition is further refined to a one-dimensional matrix, R_{MP} of sets of golfers indexed by P, the result is as follows. For reasons of space, refinement rules for functions are omitted. Note, however, that the symmetry introduced by indexing the matrix R_{MP} by P, a locally introduced type with indistinguishable elements, is detected and recorded.

$$\begin{aligned}\rho_6(R : \mathit{regpart}\, G\, s) = \{\ & R_{MP} : P \rightarrow \wp(G) \\ & \texttt{letting}\, P\, \texttt{be}\ \text{type-of-size}\ |G|/s \\ & \texttt{annotate}\, \text{symIndex}(R_{MP}, P) \\ & \texttt{such that}\, \forall x : P \cdot |R_{MP}[x]| = s \wedge \\ & \forall \{x, x'\} : \wp(P) \cdot R_{MP}[x] \cap R_{MP}[x'] = \emptyset\}\end{aligned} \tag{12}$$

Now, if SizedReln6, and recursively RegPartition6, are applied to the example, S becomes a matrix S_{MW}, indexed by weeks in W, of matrices representing the weekly schedules.

$$\begin{aligned}\rho_6(\ & S : \wp_w(\text{regpart}\, G\, s)) = \{ \\ & \lambda i.S_{MW}[i] : W \rightarrow (P \rightarrow \wp(G)) \\ & \texttt{letting}\, W\, \texttt{be}\ \text{type-of-size}\ w, P\, \texttt{be}\ \text{type-of-size}\ |G|/s \\ & \texttt{annotate}\, \text{symIndex}(S_{MW}, W), \forall i : W \cdot \text{symIndex}(S_{MW}[i], P) \\ & \texttt{such that}\, \forall i : W \cdot \forall p : P \cdot |S_{MW}[i][p]| = s \wedge \\ & \forall i : W \cdot \forall \{x, x'\} : \wp(P) \cdot S[i][x] \cap S[i][x'] = \emptyset\}\end{aligned} \tag{13}$$

We now introduce a new universal quantifier rule for the case where the bound variable is a member of an ordinary set expression.

ForallElem6: $\rho_6(\forall x \in Y \cdot C) =$

$$\begin{aligned}\{\ & \forall x' \in Y' \cdot C' \\ & \quad \texttt{letting}\ \mathit{dcls_1}, \mathit{dcls_2} \\ & \quad \texttt{annotate}\ \mathit{anns_1}, \mathit{anns_2} \\ & \quad \texttt{such that}\ \mathit{cstr_1} \wedge \mathit{cstr_2} \\ |\ & \quad x' \in Y'\ \texttt{letting}\ \mathit{dcls_1}\ \texttt{annotate}\ \mathit{anns_1}\ \texttt{such that}\ \mathit{cstr_1} \in \rho_6(x \in Y) \wedge \\ & \quad C'\ \texttt{letting}\ \mathit{dcls_2}\ \texttt{annotate}\ \mathit{anns_2}\ \texttt{such that}\ \mathit{cstr_2} \in \rho_6(C[x'/x])\}\end{aligned}$$

That is, we refine a finitely universally quantified variable x by refining the binding expression $x \in Y$, then substituting x', a refinement of x, for x in the constraint C, then refining C. The substitution of x' for x prior to the refinement of C ensures that each occurrence of x in C is refined in the same way. The definitions of the Element6 rules are omitted for brevity.

Applying ForallElem6 to our example results in the following constraint:

$$\begin{aligned}\rho_6(\ & \forall g \in R, g' \in R' \cdot |g \cap g'| \leq 1) = \{ \\ & \forall g \in \{R_{MP}[p] \mid p : P\}, g' \in \{R'_{MP}[p] \mid p : P\} \cdot |g \cap g'| \leq 1 \\ & \texttt{letting}\, P\, \texttt{be}\ \text{type-of-size}\ |G|/s \\ & \texttt{annotate}\, \text{symIndex}(R_{MP}, P) \\ & \texttt{such that}\, \forall p : P \cdot |R_{MP}[p]| = s \wedge \forall \{x, x'\} : \wp(P) \cdot R_{MP}[x] \cap R_{MP}[x'] = \emptyset \\ & \wedge \forall p : P \cdot |R'_{MP}[p]| = s \wedge \forall \{x, x'\} : \wp(P) \cdot R'_{MP}[x] \cap R'_{MP}[x'] = \emptyset\}\end{aligned} \tag{14}$$

A similar rule, FORALLSUBSET6, refines universally quantified expressions where the binding expression is a subset. Hence, assuming rules for refining cardinality and intersection expressions similar to those presented in the previous sections, the Social Golfers specification can be refined via the application of FORALLSUBSET6:

```
given $p, w, s$:nat
letting
    $G$ be type-of-size $p$, $W$ be type-of-size $w$, $P$ be type-of-size $|G|/s$,
annotate symIndex($S_{MW}, W$), $\forall i : W \cdot$ symIndex($S_{MW}[i], P$)
find $S_{MW} : W \rightarrow (P \rightarrow \wp(G))$
such that
    $\forall \{R_{MP}, R'_{MP}\} \subseteq \{S_{MW}[i] \mid i : W\} \cdot$
        $\forall g \in \{R_{MP}[p] \mid p : P\}, g' \in \{R'_{MP}[p] \mid p : P\} \cdot |g \cap g'| \leq 1 \wedge$
    $\forall i : W \cdot \forall p : P \cdot |S[i][p]| = s \wedge$
    $\forall i : W \cdot \forall \{x, x'\} : \wp(P) \cdot S[i][x] \cap S[i][x'] = \emptyset$
```

Having refined the binding expression of the outer universal quantifier, which causes R and R' to be refined to the matrices R_{MP} and R'_{MP}, the refinement of the remainder is much simpler. This is because R_{MP} is substituted for R (and similarly for R'_{MP}) beforehand. Several other refinements are possible from the input specification, according to how S and the regular partition are refined. These are not given for reasons of space. The model developed here is close to that developed by hand by Stefano Novello[3].

6 Discussion of our Specification Language

Now that we have seen how our compositional refinement system operates, we can consider how the demands of effective refinement have driven the design of the specification language. In general these demands have led to a language that is more expressive and contains constructs that are more abstract than previous specification languages such as OPL [8] and ESRA [3], each of which was designed to be more abstract than its predecessors.

One demand is that the specification language must be powerful enough to enable important distinctions to be drawn. Consider, for example the specifications of the BIBD problem and the Social Golfers problem. The decision variables in these two problems are, respectively, a relation and a set of partitions (that is, a set of sets of sets). Though each of these decision variables can be refined to a matrix of decision variables, the matrices have different symmetries. As our specification language supports both relations and sets of partitions, specifications can distinguish between these two types, thus providing the refinement system with the opportunity to choose the appropriate symmetry breaking. Many languages, such as OPL, do not support sets or relations, so both problem specifications must employ matrices. Though ESRA is a more abstract

[3] Available at http://www.icparc.ic.ac.uk/eclipse/examples/

language, which supports sets and relations, it does not support sets of sets, so a natural specification of both problems would have decision variables of type relation. Therefore, in any of these languages the identification of symmetry is the responsibility of the user, not the compiler or refinement system.

Effective refinement also demands that the specification language must provide sufficient abstraction so that a specification need not make unnecessary distinctions, such as distinguishing between objects that should be indistinguishable. The consequence of using such a language is that fewer symmetries are contained in the initial specification, but more are introduced by refinement, which we believe can be identified and broken automatically. This point is clearly exemplified in our treatment of the BIBD problem. The initial specification does not name the elements of B or V; names are introduced by the refinement process, which simultaneously recognises that this introduces symmetry into the model. We know of no constraint specification language that supports sets of unnamed objects. Without this facility, the job of recognising the interchangeability of the arbitrarily-named objects is the responsibility of the user.

7 Conclusion

This paper has provided substantial support for the conjecture that constraint satisfaction models can be constructed by compositional refinement of an abstract specification. This is an important claim; if models cannot be constructed compositionally, then it is hard to see how one could construct an automated refinement system that handled a broad class of specifications. This is also a significant result; because the decisions involved in constructing a model are not independent, it is difficult to identify the manner in which models can be composed and to capture this in a set of rules.

Before safely concluding that refinement is compositional it would be necessary to show that a wide range of models appearing in the literature can be constructed by compositional rules. Doing this would require developing a more expressive specification language, identifying a broader range of structures that are employed by expert modellers, and formulating a more extensive set of reformulation rules.

A set of broad-coverage refinement rules would lie at the heart of an automated system for the competence stage of modelling. But such a system would need much more. As discussed in Section 4, the system would need to identify symmetries in the models it generates. Once a model is produced, a system should be able to improve it through certain transformations, just as human expert modellers do. Such transformations include adding constraints that are implied by those that are already in the model but which could potentially increase pruning of the search space; removing redundant constraints (those that result in no extra pruning of the search space); replacing common subexpressions with an auxiliary variable to increase pruning [5]; and replacing constraints with their logical equivalents. We have investigated transformations such as these, resulting in the development of an automated system called CGRASS [4]. However, we have only considered applying the transformations to problem instances. The difficult challenge is formulate corresponding transformations that can be applied to problem classes.

Acknowledgements This work is partially supported by UK-EPSRC grant no. GR/N16129. We thank Brahim Hnich and Toby Walsh for useful discussions.

References

1. B.M.W. Cheng, K.M.F. Choi, J.H. Lee, and J.C.K. Wu. Increasing constraint propagation by redundant modelling. *Constraints*, 4(2):167–192, 1999.
2. P. Flener, A.M. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, and T. Walsh. Breaking row and column symmetries in matrix models. In P. van Hentenryck, editor, *Proc. 8th International Conference on Principles and Practice of Constraint Programming*, pages 462–476, 2002.
3. P. Flener, J. Pearson, and M. Ågren. The syntax, semantics, and type system of ESRA. Technical Report ASTRA, Uppsala University, 2003.
4. A.M. Frisch, I. Miguel, and T. Walsh. CGRASS: A system for transforming constraint satisfaction problems. In B. O'Sullivan, editor, *Proc. Joint Workshop of the ERCIM/CologNet area on Constraint Solving and Constraint Logic Programming (LNAI 2627)*, pages 15–30, 2002.
5. W. Harvey and P. J. Stuckey. Improving linear constraint propagation by changing constraint representation. *Constraints*, 8(2):173–207, 2003.
6. M. Sellmann and W. Harvey. Heuristic constraint propagation. In *Proc. 4th International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CPAIOR)*, pages 191–204, 2002.
7. H.D. Sherali and J.C. Smith. Improving discrete model representations via symmetry considerations. In *Proc. International Symposium on Mathematical Programming*, 2000.
8. P. van Hentenryck. *The OPL Optimization Programming Language*. The MIT Press, 1999.
9. T. Walsh. Constraint patterns. In F. Rossi, editor, *Proc. 9th International Conference on Principles and Practice of Constraint Programming.* Springer, 2003.

An Empirical Study of Mutual Routing-Scheduling Reformulation*

J. Christopher Beck[1], Patrick Prosser[2], and Evgeny Selensky[2]

[1] Cork Constraint Computation Center, University College Cork, Ireland.
c.beck@4c.ucc.ie

[2] Department of Computing Science, University of Glasgow, Scotland.
pat/evgeny@dcs.gla.ac.uk

Abstract. Despite a number of similarities, vehicle routing problems and scheduling problems are typically solved with different techniques. In this paper, we undertake a systematic study of problem characteristics that differ between vehicle routing and scheduling problems in order to identify those that are important for the performance of typical vehicle routing and scheduling techniques. In particular, we find that the addition of temporal constraints among visits or the addition of tight vehicle specialization constraints significantly improves the performance of scheduling techniques relative to vehicle routing techniques.

1 Introduction

It is a long standing belief in AI that finding the right problem representation is a key component of solving a problem [11, 12]. The real world dictates the necessity of making compromises in formulating a problem. There will be aspects of the real problem that do not fit easily into a "pure" version of a known combinatorial optimization problem. The decision must then be made to ignore the problem components that cannot be modeled in a pure way or to add "impure" or customized model components. The drawback of the former approach is that it is not clear that solutions to the problem model are solutions to the real problem. The latter approach also presents challenges, due to the fact that we do not yet understand combinatorial problem solving well enough to be able to predict the effect of adding a customized model component. It can be that what looks like a minor addition to the model significantly changes the performance of standard problem solving techniques.

This paper directly addresses the relationship between problem characteristics and search performance. In previous work [19], it was shown that scheduling technology performs poorly on reformulated vehicle routing problems while vehicle routing technology performs poorly on reformulated scheduling problems. Here, we investigate the reasons behind this results. We perform an in-depth study by systematically transforming pure vehicle routing problems (VRP) such

* This work was supported by EPSRC research grant GR/M90641.

that they become more like scheduling problems and, symmetrically, by transforming pure scheduling problems so that they look more like VRPs. We then solve these problems with routing and with scheduling technologies, to discover what problem characteristics influence the problem solving techniques.

This paper is organized as follows. We start by defining vehicle routing problems and scheduling problems, and show how we can reformulate VRPs as scheduling problems, and vice versa. We then identify five characteristics that we believe differentiate routing problems from scheduling problems. We describe the design of our experiments, to investigate the effect of these problem characteristics on the relative performance of routing and scheduling technology. The results of these experiments are reported and followed by a conclusion.

2 Problem Definitions: VRP & JSP

In the capacitated vehicle routing problems with time windows, m identical vehicles initially located at a depot are to deliver discrete quantities of goods to n customers. Each customer has a demand for goods and each vehicle has a capacity. A vehicle can make only one tour starting at the depot, visiting a subset of customers and returning to the depot. Time windows define an interval for each customer within which the visit must be made. A solution is a set of tours for a subset of vehicles such that all customers are served only once and time window and capacity constraints are respected. The objective is to minimize distance traveled, and sometimes additionally to reduce the number of vehicles used. The problem is NP-hard [15].

An $n \times m$ job shop scheduling problem (JSP) consists of n jobs and m resources. Each job is a set of m completely ordered activities, where each activity has a duration for which it must execute and a resource which it must execute on. The total ordering defines a set of precedence constraints, meaning that no activity can begin execution until the activity that immediately precedes it in the complete ordering has finished execution. Each of the m activities in a single job requires exclusive use of one of the m resources defined in the problem. No activities that require the same resource can overlap in their execution and once an activity is started it must be executed for its entire duration (i.e. no preemption is allowed). The job shop scheduling decision problem is to decide if all activities can be scheduled, given for each job a release date of 0 and a due date of the desired makespan D, while respecting the resource and precedence constraints. The job shop scheduling decision problem is NP-complete [15].

While not part of the basic JSP definition, transition times and resource alternatives have been reported in the scheduling literature [13]. In the case of a transition time, there is a temporal constraint specifying a minimum time that must expire between pairs of activities executed on the same resource. When alternative resources are represented, each activity has a set of resources that it can be performed on (and this extension of the JSP is known as the flexible job shop problem).

3 Mutual Reformulations

In [5] two reformulations are presented: a reformulation of VRP as a flexible shop problem with setups, and symmetrically a reformulation of JSP as a a VRP.

We reformulate the VRP into a scheduling problem as follows. Each vehicle is represented as a resource, and each customer visit as an activity. The distance between a pair of visits corresponds to a transition time between respective activities. Each activity can be performed on any resource, and is constrained to start execution within the time window defined in the original VRP. Each activity has a demand for a secondary resource, corresponding to a visit's demand within a vehicle. For each resource R there are two special activities $Start_R$ and End_R. Activities $Start_R$ and End_R must be performed on resource R. $Start_R$ must be the first activity performed on R and End_R the last. The transition time between $Start_R$ and any other activity A_i corresponds to the distance between the depot and the i^{th} visit. Similarly the transition time between End_R and A_i corresponds to the distance between the depot and the i^{th} visit. The processing time of $Start_R$ and End_R is zero. We associate a consumable secondary resource with every (primary) resource to model the capacity of vehicles. Consequently a sequence of activities on a resource corresponds to a vehicle's tour in the VRP. In the resultant scheduling problem each job consists of only one activity, each activity can be performed on any resource, and there are transition times between each pair of activities. The problem is then to minimize the sum of transition times on all machines and maybe also to minimize the number of resources used.

We reformulate a scheduling problem into a VRP as follows. We have for each resource a vehicle, and for each activity a customer visit. The visits have a duration the same as that of the corresponding activities. Each visit can be made only by the vehicles corresponding to the set of resources for the activity. Any ordering between activities in a job results in precedence constraints between visits. Transition times between activities correspond to travel distances between visits. The deadline D imposes time windows on visits. Assuming we have m resources, and therefore m vehicles, we have $2m$ dummy visits corresponding to the departing and returning visits to the depot. A vehicle's tour corresponds to a schedule on a resource. For the $n \times m$ JSP we have a VRP with $m(n+2)$ visits and m vehicles. Each visit can be performed only by one vehicle. Since there are no transition times in the JSP, there are no travel distances between visits, but visits have durations corresponding to those of the activities. There are precedence constraints between those visits corresponding to activities in a job. The decision problem is then to find an ordering of visits on vehicles that respects the precedence constraints and time windows.

4 Problem Characteristics

The performance difference between VRP and scheduling techniques must be due to the characteristics on which the problems differ. In this section, we identify five main characteristics that we believe are sufficient to explain the performance differences between the VRP and scheduling techniques.

Alternative Resources Perhaps the most obvious difference between standard VRP and scheduling problems is the number of alternative resources that may be used for each operation. In vehicle routing there are typically many vehicles that can be used to perform a visit. For example, the standard Solomon benchmarks [21] have twenty five identical vehicles. In contrast, scheduling problems tend to have very few alternative resources and, as noted, in the pure JSP there are no resource alternatives. This has been reflected in the scheduling literature. For example, Focacci et al. [13] experiment on problems with up to three alternatives while Davenport & Beck [9] use problem instances with at most eight alternatives. We expect few resource alternatives to favour scheduling techniques while many should improve the performance of VRP techniques.

Temporal Constraints In pure VRPs each visit is independent, i.e. there are no constraints requiring a visit to have some temporal relation with other visits. In contrast, scheduling problems typically have long chains or complex directed acyclic graphs of temporal relationships among activities. As a consequence temporal reasoning has been widely explored in the literature [7] and is an extremely important component of many scheduling algorithms [16]. We therefore predict that scheduling technology should improve on problems where there are complex temporal constraints.

The Ratio of Operation Duration to Transition Time In pure VRPs a visit has no duration. In contrast, in pure JSP the transition time between operations is zero. The modeling abstraction for scheduling is obviously opposite to the VRP: the transition time between activities on the same resource is so small that it can be ignored. These two problem models are extremes. In both the literature and in the real world there are VRPs with visit duration and scheduling problems with transition times and costs. However, even then the ratio of operation duration to transition time tends to be different. In VRPs the ratio is very small while in scheduling it is often very large. This difference is reflected in solution techniques. Many VRP techniques (e.g., the savings heuristic [8]) look exclusively at transition times when searching, while many of the strong constraint propagation [18, 17] and heuristic search techniques [20, 2] in scheduling ignore transition time. Overall, we can expect that the smaller the ratio of operation duration to transition time, the better the VRP techniques should perform relative to the scheduling techniques.

Optimization Criterion In a VRP the standard optimization criterion is to minimize the total distance traveled by each vehicle. In scheduling, a common criterion is the minimization of makespan: the time between the start of the first operation and the end of the last operation. While many criteria have been studied (e.g., tardiness [1], earliness and tardiness [3], flow time, transition times [13]) makespan has received a great deal of attention in the literature even though its relevance has been questioned in practical applications [4], [14]. We can expect the performance of the scheduling techniques should be favoured when using the minimization of makespan as the criterion.

Temporal Slack Although it seems difficult to make specific predictions as to how temporal slack, i.e., the difference between the time window for an operation and the operation's duration, can affect the performance of the search techniques in both routing and scheduling, we think it worthwhile to experiment with it as well. Slack can be important while solving both routing and scheduling problems, e.g., there are state-of-the-art global constraint propagation algorithms and efficient search heuristics based on temporal slack [20].

These characteristics may be sufficient to explain the performance differences between VRP and scheduling techniques, because they are sufficient to transform one problem model to the other. E.g., starting with a pure VRP, if we reduce the alternative resources to a singleton for each task, add chains of temporal constraints, reduce the transition time to zero while increasing task duration, and change the optimization criterion to makespan minimization, we have a pure JSP problem.

5 Experimental Design

Our goal is to empirically determine which of the above five problem characteristics actually have an effect on the relative performance of scheduling techniques compared to VRP techniques.

We perform two symmetric sets of experiments. In the first set of experiments, we take pure VRP instances and examine the effect of varying one problem characteristic at a time. We generate instances of VRPs. We then solve each problem twice: first we model the problem as a VRP and solve it using routing technology; second, we model the VRP as a scheduling problem (as described in sect. 2) and solve it with scheduling technology. We then measure the relative performance of both technologies. For each problem characteristic, therefore, we begin with a set (or ensemble) of pure VRP instances. After that we generate additional problem sets by varying the parameter in question. Intuitively, as the parameter value deviates more from the VRP, the ensemble should look increasingly like scheduling problems.

In a symmetric set of experiments, we take pure JSP instances and again vary one problem parameter at a time. We generate pure and transformed JSPs and, as for the VRPs above, apply both scheduling and routing technologies to each instance and measure their relative performance.

In both series of experiments, the size of an ensemble of instances is 100.

5.1 Problems and Their Parameters

VRP instances To create the initial set of pure VRP instances, the geographic coordinates of each visit are randomly drawn with replacement from a data file containing all postal codes of locations in the city of Glasgow, within a 5 km radius of the city centre. These postal codes can be directly translated to coordinates with an accuracy of ten meters. Because we are using city coordinates, all distances are Manhattan distance. The other parameters for generating VRPs are listed below.

JSP instances Each of the instances has 10 machines and 10 jobs, i.e., 10 totally ordered sets of operations. In every instance of the initial set, a single machine is specified for each operation. There are no transition times between operations on a given machine. The other parameters for generating JSPs are displayed below.

n the number of operations:
VRP: the number of customers. In all problems $n = 100$.
JSP: the number of activities. In all problems $n = 100$.

m the number of resources:
VRP: the number of vehicles. In all problems $m = 25$.
JSP: the number of machines. In all problems $m = 10$.

p resource specialization. The parameter p represents the proportion of the pool of resources (vehicles or machines) that can perform each operation. For operation i, the number of possible resources m_i is calculated as:

$$m_i = \lceil p.m \rceil \quad (1)$$

Small values of p correspond to few resource alternatives and therefore correspond more to scheduling problems.
VRP: the default value is $p = 1.0$ (any vehicle can do any visit).
JSP: the default value is $p = 0.1$ (each operation has a single specified machine).

pc The (integer) number of precedence constraints. This parameter specifies the number of pairs of visits whose sequence is constrained such that visit i must end before the start of visit j. Note that i and j are not necessarily on the same resource. Large values of pc correspond to many precedence constraints characteristic of scheduling problems.
VRP: the default value is $pc = 0$.
JSP: the default value is $pc = 450$ (10 jobs with 10 operations each).

ρ The ratio of visit duration to travel time for the VRP or the ratio of activity duration to transition time for the JSP. This parameter is used to set V, the speed of the vehicles or the speed of transitions on machines:

$$V = \frac{\rho.\overline{d}}{\overline{\tau}} \quad (2)$$

where $\overline{\tau}$ is the average duration of visits/activities, and $\overline{d}$ is the average distance/transition between visits/activities. Large ρ values are typical of scheduling problems while small ρ values correspond to VRPs.
VRP: the default value is $\rho = 1.0$.
JSP: the default value is $\rho = \infty$ (transition time equals zero).

σ Normalized slack.

$$\sigma = \frac{le_i - \tau_i - es_i}{le_i - es_i}, \tag{3}$$

where τ_i is the duration of operation i, and le_i and es_i are the latest end and earliest start of operation i. It follows from this definition that $\sigma \in (0, 1)$ and that the larger σ becomes, the wider the time window is for the same τ_i. Operation durations are measured in minutes. The working day is 24 hours.
VRP: the default value is $\rho = 0.9$.
JSP: the default value is $\rho = 0.986$ (time windows are as long as the working day).

c The optimization criterion:
VRP: the default value is $c =$ *minimize total travel time*.
JSP: the default value is $c =$ *minimize makespan*.

Pure VRP and JSP problems are generated with the default parameters, specified above. Problems are tested for solubility, and are rejected if not found to be soluble within 5 minutes of CPU time.

5.2 Solution Technology

In our experiments we use the ILOG optimization suite. The scheduling library (ILOG Scheduler 5.2) has global constraint propagation [1, 16, 18] within constructive tree search as its core technology, while the vehicle routing library (ILOG Dispatcher 3.2) focuses on local search [10].

As part of the routing technique, we apply first accept neighbourhood search enhanced by the guided local search (GLS) metaheuristic [22] with a penalty factor of 0.4. To construct a neighbourhood of a solution, we use the standard move operators (*TwoOpt*, *OrOpt*, *Relocate*, *Cross* and *Exchange*). Search starts from the first solution found by the savings heuristic or uses the first solution found by the scheduling technique. The scheduling technique is constructive depth first search with standard slack based heuristics, global constraint propagation, and edge finding [1].

5.3 Evaluation Criteria

Our primary evaluation criterion is λ, the ratio of the cost of the best solution found by the scheduling technique to that found by the VRP technique in a given CPU time limit. Specifically:

$$\lambda = \frac{C_{sched}}{C_{rout}}, \tag{4}$$

where C_{sched} and C_{rout} are the cost of the best solutions found by the scheduling and routing techniques in the given amount of CPU time. Because we are minimizing, routing technology is outperforming scheduling technology when $\lambda > 1$.

When $\lambda < 1$, scheduling technology dominates. And, finally, when $\lambda = 1$ both technologies perform equally.

Apart from λ, we also check if technologies are capable of finding solutions within a specified time limit. The CPU time bound is chosen to be 10 minutes because our preliminary tests showed that there was no significant improvement in solutions for both solving technologies after 10 minutes.

6 The Empirical Results

6.1 Base cases: pure VRPs and JSPs

In this section we present the experimental results obtained by scheduling and routing technologies on two base problem sets, pure VRPs and pure JSPs.

In Fig. 1 we display two scatters of λ, one for the base VRP instances and the other for the base JSP instances. Clearly, the routing technology dominates for the pure VRP instances ($\lambda > 1$) and the scheduling technology dominates for the pure JSPs ($\lambda < 1$). In the experiment with pure JSPs the routing technology wasn't able to come up with a first solution independently because the first solution construction heuristics are sensitive to impurities in the VRPs such as resource allocation constraints and temporal relationships between activities [6]. Therefore in this and all subsequent experiments with JSPs both technologies started search from common first scheduling solutions.

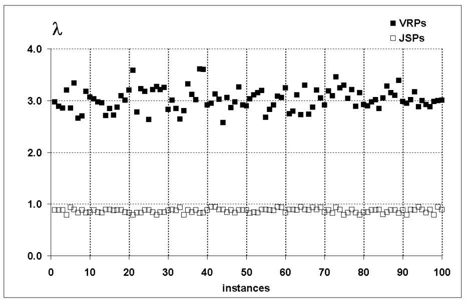


Fig. 1. Base case: λ's for 100 pure JSPs and 100 pure VRPs. The default parameters for the VRPs are $p = 1.0$, $\rho = 1.0$, $pc = 0$, $\sigma = 0.9$, both technologies start search independently. The default parameters for the JSPs are $p = 0.1$, $\rho = \infty$, $pc = 450$, $\sigma = 0.986$; search starts from common initial solutions found by the scheduling technique.

We now study the influence of the following: variation in the number of alternative resources, the amount of precedence constraints, the influence of operation duration versus transition times, optimization criteria, and finally slack.

6.2 Alternative Resources (p)

VRP instances To investigate the influence of resource alternatives we create a series of problem sets by varying the vehicle specialization p, such that $p \in \{0.05, 0.1, 0.25, 0.5, 1.0\}$. With $m = 25$ this corresponds to VRPs with 1, 3, 6, 13, and 25 possible vehicles per visit respectively. For a given p to get the corresponding numbers of vehicle alternatives, we take a visit and randomly select without replacement so many vehicles of the fleet. All other parameters take their default values (i.e. $pc = 0$, $\rho = 1.0$, $\sigma = 0.9$, and c minimizing total travel time).

In Fig. 2 we present λ with respect to p, expressed as a percentage. Two contours are presented, one for λ for first solution found (λ_{first}) by both techniques, and λ for the best solution found (λ_{best}) by both techniques. We see that as specialization of the fleet increases (i.e. p decreasing) the performance of the routing technology degrades, and ultimately when we have one vehicle per visit the technologies perform approximately equally. Both contours exclude any instance that could not be solved by the routing technology in 10 minutes.

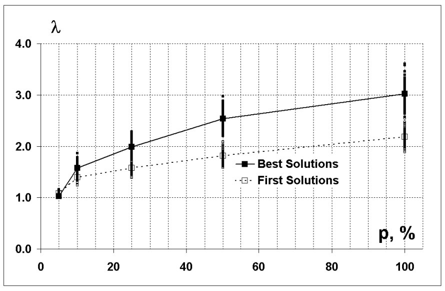


Fig. 2. VRPs: What happens when we specialize vehicles? p varies, $pc = 0$, $\rho = 1.0$, $\sigma = 0.9$, $c = \textit{min travel time}$. Search starts from independently found first solutions.

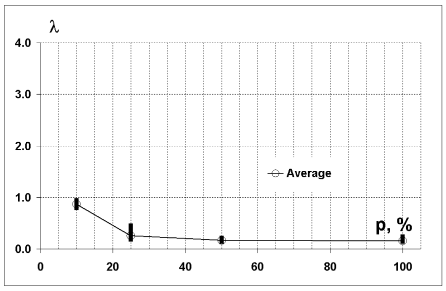


Fig. 3. JSPs: What happens when we vary machine specialization? p varies, $pc = 450$, $\rho = \infty$, $\sigma = 0.986$, $c = \textit{min makespan}$. Search starts from common first scheduling solutions.

JSP instances We generate sets of JSPs with a varying percentage of resources allowed for each operation such that $p \in \{0.1, 0.25, 0.5, 1.0\}$. This corresponds to each operation having 1, 3, 5 and 10 alternative resources. All other parameters take their default values ($pc = 450$, $\rho = \infty$, $\sigma = 0.986$, $c =$ minimizing the makespan).

In Fig. 3 we show how λ depends on the number of alternative resources allowed for an operation. We could expect that as p grows and, consequently, problems become more VRP-like, the routing technology should be favoured. However, we see the opposite effect. The JSP turns out to be not so amenable to the routing technology because, on the one hand, there are many temporal constraints leading to the rejection of many moves and, on the other, a solution neighbourhood for the problem instances of this size is very large.

6.3 Precedence Constraints (*pc*)

VRP instances To study the influence of precedence constraints we vary *pc*, such that $pc \in \{0, 50, 450, 1200\}$. Each precedence constraint imposes an ordering for two given visits, without forcing these two visits to use the same vehicle. This corresponds to varying numbers and lengths of totally ordered precedence chains such that 5 chains each involving 5 visits are equivalent to a *pc* of 50, 10 chains with 10 visits each to a *pc* of 450, 4 chains with 25 visits each to a *pc* of 1200.

Experiments show that as we introduce precedence constraints the VRP technology fails to produce initial solutions, but the scheduling technology appears to be robust. If both techniques start with a scheduling solution, the VRP technology dominates the scheduling technology (when $pc = 0$, $\lambda \approx 1.2$) but this becomes less significant as we increase the number of precedence constraints (when $pc = 1200$, $\lambda \approx 1.1$).

JSP instances To experiment with varying numbers of precedence constraints, apart from the pure 10×10 JSPs with a *pc* of 450, we generate additional problem sets such that $pc \in \{0, 40, 126, 450\}$. With respect to the number *pc* of precedences between operations in isolation, removing precedence constraints correspond to making the instances more like the VRP.

Experiments show that as we remove precedence constraints and both technologies start from common initial scheduling solutions, they tend to perform equally ($\lambda \approx 1$). It is an expected result because if we remove *all* precedences from a JSP, the problem becomes tractable. Consequently, when $pc = 0$, the first solution found by the scheduling technique is an optimal solution for any instance.

6.4 Operation Duration vs Transition Time (ρ)

VRP instances VRP instances were generated with values of ρ (visit duration to transition times) as follows: $\rho \in \{0.25, 0.5, 1.0, 2.0, 2.25, 3.0\}$. All visit durations were set at 20 minutes, consequently problems have a vehicle speed ranging from 12km/h to 72km/h.

In [5] it was argued that higher ρ's will tend to favour scheduling techniques because large values of ρ correspond to small transition times compared to operation durations, typical of scheduling problems. Our results demonstrate, however, that in isolation this parameter does not bring about such a performance difference and routing technology continues to outperform the scheduling technique (when $\rho = 0.25$, $\lambda \approx 2.9$ and, when $\rho = 3.0$, $\lambda \approx 3.2$). This is not what we expected.

JSP instances We now generate JSP instances with 10 machines, 10 jobs and with varying values of ρ (activity duration to transition times) as follows: $\rho \in \{0.5, 1.0, 3.0, 10.0, \infty\}$. All operation durations were set at 20 minutes. A ρ of ∞ corresponds to pure JSPs with infinitely fast transitions between operations on a machine.

We observe that decreasing ρ for the JSPs marginally improves the relative performance of the routing technique (when $\rho = \infty$, $\lambda \approx 0.87$ and, when $\rho = 0.5$, $\lambda \approx 1.04$).

6.5 Optimization Criterion (c)

VRP instances We now solve pure VRPs using two different optimization criteria: minimize total travel (i.e. from a scheduling perspective, minimize the sum of setups), and minimize makespan (complete all the visits as early as possible). In Fig. 4 we have a scatter plot, on the left λ_{best} and λ_{first} using the minimization of travel, and on the right λ_{best} and λ_{first} where the criterion is minimizing makespan. We see a large difference in the behaviour of the solving technologies. When minimizing travel (scatter on the left) the routing technology is consistently 2 to 3 times better than the scheduling technology. When we switch to minimizing makespan (scatter on the right) the scheduling and routing technologies perform equally (i.e. λ_{best} and λ_{first} are around 1). These results are particularly dramatic given that all the other parameters are that of a standard VRP: the only difference is the optimization criterion.

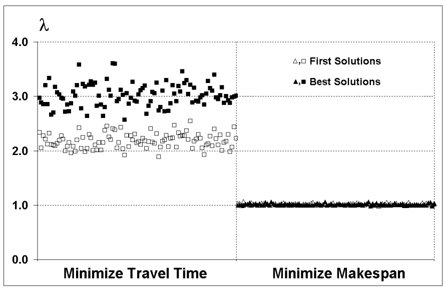


Fig. 4. VRPs: What happens when we swap optimization for pure VRPs? On the left, *minimize travel time*, and on the right *minimize makespan*. Search starts from independently found first solutions.

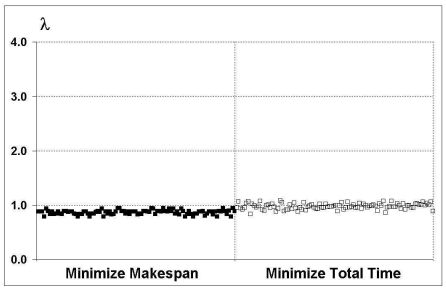


Fig. 5. JSPs: What happens when we swap optimization for pure JSPs? On the left, *minimize makespan*, and on the right *minimize total time*. Search starts from common first scheduling solutions.

Why is there such a difference when we change optimization criterion? One answer might be the underlying techniques used in the scheduling technology. When minimizing a maximum function (such as makespan), the upper bound can be directly propagated on the completion time of each operation/visit, and the domains of possible start times can be effectively tightened. However, when a sum function (e.g., sum of transition times) is being optimized, the respective constraint has to be considered at each stage of search simultaneously with resource constraints [1] thus making it harder to achieve large domain reductions. The scheduling technology we used is built to exploit the powerful constraint propagation that is possible using such criterion as makespan, whereas the routing technology is not. We might therefore expect when we have a VRP, and the goal is to complete all visits as soon as possible, that we might be better solving it as a scheduling problem.

JSP instances We now experiment with two different optimization criteria for pure JSPs, makespan and total processing time. Total processing time minimization can be thought of as minimizing the sum of makespans over all machines.

In Fig. 5 we have a scatter plot with the minimization of the makespan on the left and the minimization of total processing time on the right. When minimizing makespan (scatter on the left) the routing technology is consistently dominated by the scheduling technology (the mean λ is 0.868). When we switch to minimizing total time (scatter on the right) the routing technology starts to compete with the scheduling technology (the mean λ is 0.977).

These results again demonstrate the importance of the optimization criterion for the performance of the routing and scheduling techniques. Note that even though the values of λ have not changed significantly for the JSPs, the change in favour of the routing technique does occur in the area of zero transition cost (i.e., where arcs in the routing graph have a zero cost)! Because the cost function calculation in the routing technique is entirely arc-based, we expect this change to be even greater for mixed JSPs with non-zero transition costs.

6.6 Temporal slack (σ)

VRP instances To experiment with slack, we generate VRPs with normalized slack $\sigma \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$. All other parameters take default values. As σ increases, time windows get larger with respect to visit durations.

Experiments demonstrate that, as slack increases, routing technology improves relative to scheduling technology (from a λ of approximately 2.7 when $\sigma = 0.5$ to a λ of 3.0 when $\sigma = 0.9$). Conversely, when slack decreases it appears that scheduling technology is improving, yet continues to be dominated by the routing technology. We might conclude from this that as slack decreases, and problems become more temporally constrained, we might incorporate scheduling technology into the routing technology.

JSP instances We generate additional problem sets such that $\sigma \in \{0.65, 0.75, 0.85, 0.95, 0.986\}$. All other parameters take default values.

It turns out that tightening slack for JSPs leads to a λ of 1.0 (when $\sigma = 0.986$, $\lambda \approx 0.87$ and, when $\sigma = 0.65$, $\lambda \approx 1.0$). This is because the first solution found by the scheduling technique tends to be optimal as we are making the

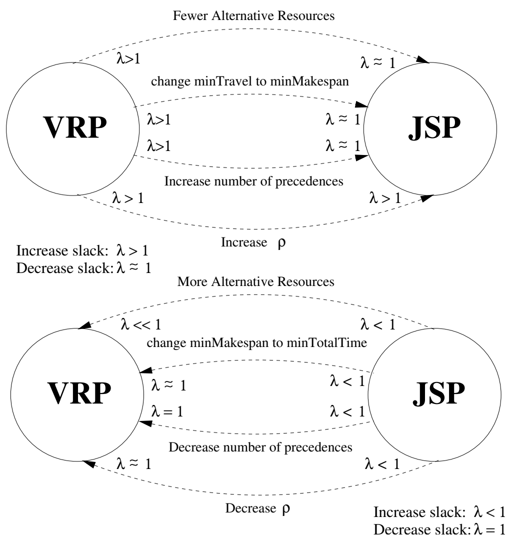


Fig. 6. Influence of various problem characteristics on the performance of VRP and JSP technologies. Arrows represent reformulation direction, arrow labels show the varied parameter. $\lambda > 1$ means routing technology outperforms scheduling technology, when $\lambda < 1$ scheduling technology is a winner, $\lambda \approx 1$ shows that both technologies perform equally.

problems more temporally constrained. Consequently, tightening slack in JSPs makes both techniques perform equally when search starts from common initial scheduling solutions. Conversely, increasing σ leads to the relative improvement of scheduling technology against routing technology.

6.7 Summary of Results

We have varied five different parameters, each in isolation, in two symmetric sets of experiments. In the first series of experiments, we started from pure VRPs and transformed them systematically so that they became more like JSPs. In the second series, we started from pure JSPs and moved towards VRPs.

We can transform a JSP into a VRP or vise versa by simultaneously varying the number of alternative resources, the number of temporal constraints between operations, the optimization criterion and operation duration vs. transition time at the same time. Varying slack does not contribute to this problem transformation.

Fig. 6 pictorially summarizes our study. The pure VRP and pure JSP domains are shown as circles. Problem transformations are depicted by means of arrows labeled by the corresponding problem parameters. We also show the response to each parameter change in the relative quality λ of the solving techniques.

The most prominent effects on the performance of the solving techniques were brought about by varying:

the number of alternative resources p The removal of alternative vehicles in the VRPs favours the scheduling technology but the addition of alternative machines to the JSPs proves even less amenable to the routing technique than pure JSPs;

the number of temporal constraints between activities pc An increase in pc in VRPs favours the scheduling technique, and, symmetrically, a decrease in pc in the JSP improves the performance of the routing technique;

optimization criterion c Changing c from total travel minimization to makespan minimization in VRPs notably favours the scheduling technology. Symmetrically, changing c from makespan minimization to total processing time minimization in JSPs favours the routing technique.

7 Conclusions

We have considered the vehicle routing and job shop scheduling problems and carried out an empirical study to develop an understanding of how their characteristics contribute to the performance of standard solving technologies. We have identified five such parameters: vehicle specialization, complex temporal relationships between operations, the optimization criterion, operation duration to travel time ratio, and temporal slack.

In the experiments, we started from pure VRPs and, separately, from pure JSPs and varied those parameters, each in isolation, to assess their influence on the performance of the solving techniques. In our study we used existing VRP and scheduling technology found in commercially available tools. This may appear naive. However, we believe that this is a critical feature of our study. The current research literature boasts a number of powerful approaches for solving combinatorial optimization problems (e.g., constraint programming, mixed integer programming, SAT solvers, etc.). These approaches, however, require significant skill and experience to apply to their full potential. In this study, therefore, we were not interested in the algorithmic customizations we could make to scheduling or VRP technology to allow it to solve "impure" problems better. Rather we were interested in the problem characteristics and their impact on existing solution techniques. One of our long term goals is to automate the process of problem analysis to be able to suggest appropriate solution technology. The identification of the relationship between problem characteristics and search performance is an important step in this direction.

References

1. Ph. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems.* Kluwer Academic Publishers, 2001.
2. J. C. Beck and M. S. Fox. Dynamic problem structure analysis as a basis for constraint-directed scheduling heuristics. *Artificial Intelligence*, 117(1):31–81, 2000.

3. J. C. Beck and P. Refalo. A hybrid approach to scheduling with earliness and tardiness costs. *Annals of Operations Research*, 2002. in press.
4. J.C. Beck, A.J Davenport, and M.S. Fox. Five pitfalls of empirical scheduling research. In *3rd Int. Conference on Principles and Practice of Constraint Programming (CP'97)*, 1997.
5. J.C. Beck, P. Prosser, and E. Selensky. On the reformulation of vehicle routing problems and scheduling problems. In *LNAI 2371, Proceedings of the Symposium on Abstraction, Reformulation and Approximation (SARA 2002)*, pages 282–289, 2002.
6. J.C. Beck, P. Prosser, and E. Selensky. Vehicle routing and job shop scheduling: What's the difference? In *Proceedings of the 13th International Conference on Artificial Intelligence Planning and Scheduling*, 2003.
7. A. Cesta, A. Oddi, and S.F. Smith. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8(1):109–136, 2000.
8. G. Clarke and J.W. Wright. Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research*, 46:93–100, 1964.
9. A.J. Davenport and J.C. Beck. An investigation into two approaches for constraint directed resource allocation and scheduling. In *INFORMS*, 1999.
10. B. DeBacker, V. Furnon, P. Shaw, P. Kilby, and P. Prosser. Solving vehicle routing problems using constraint programming and metaheuritics. *Journal of Heuristics*, 6:5001–523, 2000.
11. E. Fink. Systematic approach to the design of representation-changing algorithms. In *Symposium on Abstraction, Reformulation and Approximation (SARA)*, 1995.
12. E. Fink. How to solve it automatically: Selection among problem-solving methods. In *4th Int. Conference on Artificial Intelligence Planning Systems*, 1998.
13. F. Focacci, P. Laborie, and W. Nuijten. Solving scheduling problems with setup times and alternative resources. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling*, 2000.
14. M.S. Fox. *Constraint-Directed Search: A Case Study of Job Shop Scheduling*. PhD thesis, Carnegy Mellon University, Intelligent Systems Laboratory, The Robotics Institute, Pittsburgh, PA, 1983.
15. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.
16. P. Laborie. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. In *Proceedings of the 6th European Conference on Planning (ECP01)*, 2001.
17. C. Le Pape. Implementation of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint-Based Scheduling Systems. *Intelligent Systems Engineering*, 3(2):55–66, 1994.
18. W. P. M. Nuijten. *Time and resource constrained scheduling: a constraint satisfaction approach*. PhD thesis, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1994.
19. E. Selensky. On mutual reformulation of shop scheduling and vehicle routing. In *Proceedings of the 20th UK PLANSIG*, 2001.
20. S. Smith and C. Cheng. Slack based heuristics for constraint satisfaction scheduling. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 139–144, 1993.
21. M. Solomon. Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints. *Operations Research*, 35:254–365, 1987.
22. C. Voudouris and E.P.K. Tsang. Guided Local Search. *European Journal of Operational Research*, 113(2):80–110, 1998.

Automated Reformulation of Specifications by Safe Delay of Constraints

Marco Cadoli and Toni Mancini

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
cadoli|tmancini@dis.uniroma1.it

Abstract. In this paper we propose a form of reasoning on problem *specifications*, with the goal of reformulating them so that they are more efficiently solvable. The reformulation technique highlights constraints that can be safely "delayed", and solved afterwards. Our main contribution is the characterization (with soundness proof) of safe-delay constraints with respect to a criterion on the specification, thus obtaining a mechanism for the automated reformulation of specifications applicable to a great variety of problems, e.g., graph coloring and job shop scheduling. This is an advancement with respect to the forms of reasoning done by state-of-the-art-systems, which typically just detect linearity of specifications. Another contribution is a preliminary experimentation on the effectiveness of the proposed technique, which reveals promising time savings.

1 Introduction

Several systems and languages for the solution of constraint problems adopt a clear separation between the *specification* of a problem, e.g., graph three-coloring, and its *instance*, e.g., *a* graph. On top of that, they use a two-level architecture for finding solutions: the specification is *instantiated* against the instance, and then an appropriate solver is invoked. Examples of systems of such kind are AMPL [10], OPL [17], DLV [8], SMODELS [16], and NP-SPEC [4].

The major benefit of this separation is the decoupling of the solver from the specification. Ideally, the programmer can focus only on the specification of the problem, without committing *a priori* to a specific solver. In fact, some systems, e.g., AMPL, are able to translate –at the request of the user– a specification in various formats, suitable for different solvers.

Some systems go one step further, and offer a limited form of *reasoning* on the specification, with the goal of choosing the most appropriate solver for a problem. As an example, the OPL system checks whether a specification contains only linear constraints and objective function, and in this case invokes an integer linear programming solver (typically very efficient); otherwise, it uses a constraint programming solver.

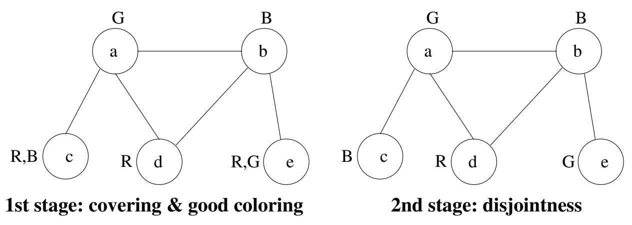


Fig. 1. Delaying the disjointness constraint in 3-coloring

Our long-term goal is to propose a more sophisticated architecture, in which reasoning on the specification is more complex. In particular, we propose to *reformulate* the specification so that the later stages of computation, i.e., instantiation and solution, are more efficient. We get inspiration from the relational database technology, since it is well-known that reformulating queries –independently on the database– may result in greater efficiency. As an example, making selections as soon as possible is a simple heuristic that typically allows to decrease the number of accesses to disk (cf., e.g., [1]).

Reformulation is quite difficult in general: a specification is essentially a formula in second-order logic, and it is well-known that the equivalence problem is undecidable already in the first-order case [3]. For this reason we limit our attention on restricted forms of reformulation, and, more specifically, in this paper we focus on a reformulation that selects constraints that can be safely "delayed", and solved afterwards.

The NP-complete graph k-coloring problem offers a simple example of a constraint of this kind. The problem amounts to find an assignment of nodes to k colors such that:

– Each node has at least one color (*covering*);
– Each node has at most one color (*disjointness*);
– Adjacent nodes have different colors (*good coloring*).

For each instance of the problem, if we obtain a solution neglecting the disjointness constraint, we can *always* choose for each node one of its colors in an arbitrary way in a later stage (cf. Fig. 1). We call a constraint with this property a *safe-delay constraint.* It is interesting to note that the standard DIMACS formulation in SAT of k-coloring omits the disjointness constraint.

Of course not all constraints are safe-delay: as an example, both the covering and the good coloring constraints are not. Intuitively, identifying the set of constraints of a specification which are safe-delay may lead to several advantages:

– The instantiation phase will typically be faster, since safe-delay constraints are not taken into account. As an example, let's assume we want to use a SAT solver (after instantiation) for the solution of k-coloring on a graph with n nodes and e edges. The SAT instance encoding the k-coloring instance –in the obvious way, cf., e.g., [12]– has $n \cdot k$ propositional variables, and a number of clauses which is n, $n \cdot k \cdot (k-1)/2$, and $e \cdot k$ for covering, disjointness, and

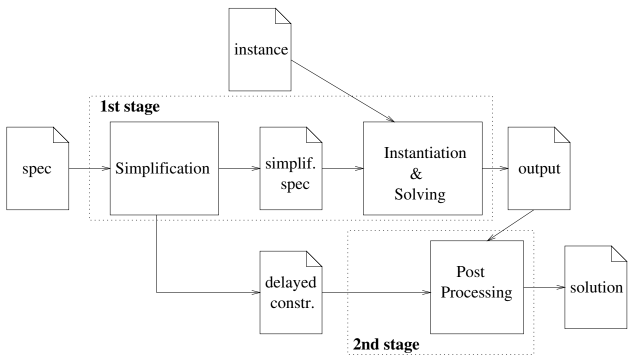


Fig. 2. Reformulation architecture

good coloring, respectively. If we delay disjointness, $n \cdot k \cdot (k-1)/2$ clauses must not be generated.

- Solving the simplified problem, i.e., the one without disjointness, might be easier than the original formulation for some classes of solvers. In our (even if preliminary) experiments, using a SAT solver, we obtained a fairly consistent (in some cases, more than one order of magnitude) speed-up for hard instances of various problems, e.g., graph coloring and job shop scheduling. On top of that, we implicitly obtain several good solutions. The approach seems promising for some classes of instances even when state-of-the-art solvers for integer linear programming like CPLEX are used (cf. Sec. 5).
- Ad hoc efficient methods for solving delayed constraints may exist. In fact, the problem of choosing only one color for the nodes with more than one color is $O(n)$.

The architecture we propose is illustrated in Fig. 2 and can be applied to any system which separates the instance from the specification. It is in some sense similar to the well-known *divide and conquer* technique, but rather than dividing the instance, we divide the constraints. In general, the first stage will be more computationally expensive than the second one, which, in our proposal, will always be doable in polynomial time.

The goal of this paper is to understand in which cases a constraint is safe-delay. Our main contribution is the characterization of safe-delay constraints with respect to a criterion on the specification. This allows us to obtain a mechanism for the automated reformulation of a specification (shown in Sec. 3) that can be applied to a great variety of problems, including the so-called *functional* ones. Another contribution is an experimentation on the effectiveness of the proposed reformulation. A preliminary discussion on the adopted experimental methodology is given in Sec. 4. The experiments are shown in Sec. 5, and have been done on both benchmark and randomly generated instances, using both a SAT solver and state-of-the-art linear and constraint programming solvers. Finally, conclusions and related work are described in Sec. 6.

2 Preliminaries

In this paper, we use *existential second-order logic* (ESO) for the specification of problems, which allows to represent all search problems in the complexity class NP [9]. The use of ESO as a modelling language for problem specifications is common in the database literature, but unusual in constraint programming, therefore few comments are in order. Constraint modelling systems like those mentioned in Sec. 1 have a richer syntax and more complex constructs, and we plan to eventually move from ESO to such languages. For the moment, we claim that studying the simplified scenario is a mandatory starting point for more complex investigations, and that our results can serve as a basis for reformulating specifications written in higher-level languages. In Sec. 4 we discuss further our choice. Coherently with all state-of-the-art systems, we represent an instance of a problem by means of a *relational database*. All constants appearing in a database are *uninterpreted*, i.e., they don't have a specific meaning.

In the following, σ denotes a fixed set of relational symbols not including equality "=", and $S_1, \ldots, S_h$ denote variables ranging over relational symbols distinct from those in $\sigma \cup \{=\}$. By Fagin's theorem [9], any collection $\mathbf{D}$ of finite databases over σ is recognizable in NP time if and only if it is defined by an ESO formula of the kind:

$$\exists S_1, \ldots, S_h \ \phi, \tag{1}$$

where $S_1, \ldots, S_h$ are relational variables of various arities and ϕ is a function-free first-order formula containing occurrences of relational symbols from $\sigma \cup \{S_1, \ldots, S_h\} \cup \{=\}$. The symbol "=" is always interpreted in the obvious way, i.e., as "identity". A database D is in $\mathbf{D}$ if and only if there is a list of relations $\Sigma_1, \ldots, \Sigma_h$ (matching the list of relational variables $S_1, \ldots, S_h$) which, along with D, satisfies formula (1), i.e., such that $(D, \Sigma_1, \ldots, \Sigma_h) \models \phi$. The tuples of $\Sigma_1, \ldots, \Sigma_h$ must take elements from the *Herbrand universe* of D, i.e., the set of constant symbols occurring in it.

Example 1. In the "three-coloring" NP-complete decision problem (cf. [13, Prob. GT4, p. 191]) the input is a graph, and the question is whether it is possible to give each of its nodes one out of three colors (red, green, and blue), in such a way that adjacent nodes (not including self-loops) are never colored the same way. The question can be easily specified as an ESO formula ψ over a binary relation *edge*:

$$\begin{aligned}
\exists RGB \quad & \forall X \ \ R(X) \vee G(X) \vee B(X) \ \wedge && (2)\\
& \forall X \ \ R(X) \rightarrow \neg G(X) \ \wedge && (3)\\
& \forall X \ \ R(X) \rightarrow \neg B(X) \ \wedge && (4)\\
& \forall X \ \ B(X) \rightarrow \neg G(X) \ \wedge && (5)\\
& \forall XY \ X \neq Y \wedge R(X) \wedge R(Y) \rightarrow \neg edge(X,Y) \wedge && (6)\\
& \forall XY \ X \neq Y \wedge G(X) \wedge G(Y) \rightarrow \neg edge(X,Y) \wedge && (7)\\
& \forall XY \ X \neq Y \wedge B(X) \wedge B(Y) \rightarrow \neg edge(X,Y), && (8)
\end{aligned}$$

where clauses (2), (3-5), and (6-8) represent the covering, disjointness, and good coloring constraints, respectively. Referring to the graph in Fig. 1, the Herbrand universe is the set $\{a, b, c, d, e\}$, the input database has only one relation, i.e., *edge*, which has five tuples (one for each edge). Formula ψ is satisfied if R, G, and B are assigned to, e.g., the following relations (cf. Fig. 1, right):

$$\mathrm{R}\begin{array}{|c|}\hline \mathrm{d}\\\hline\end{array}\quad \mathrm{G}\begin{array}{|c|}\hline \mathrm{a}\\ \mathrm{e}\\\hline\end{array}\quad \mathrm{B}\begin{array}{|c|}\hline \mathrm{b}\\ \mathrm{c}\\\hline\end{array}$$

In what follows, existentially quantified predicates (like R, G, and B) will be called *guessed*, and the set of tuples from the Herbrand universe they take will be called their *extension* and denoted with $ext()$. As an example, $ext(G) = \{a, e\}$ in the previous example. The symbol $ext()$ will be used also for any first-order formula with one free variable. An interpretation will be sometimes denoted as the aggregate of several extensions.

3 Reformulation

In this section we show sufficient conditions for constraints of a specification to be safe-delay. We refer to the architecture of Fig. 2, with some general assumptions:

1. As shown in Fig. 1, the output of the first stage of computation may –implicitly– contain *several* solutions. In the second stage we do not want to compute all of them, but just to arbitrarily select one.
2. The second stage of computation can only *shrink* the extension of a guessed predicate. Fig. 3 represents the extensions of the red predicate in the first (R^*) and second (R) stages of Fig. 1 ($ext(B)$ and $ext(G)$ are unchanged). This assumption is coherent with the way most algorithms for constraint satisfaction operate: each variable has an associated *finite domain*, from which values are progressively eliminated, until a satisfying assignment is found.

Identification of safe-delay constraints requires reasoning on the whole specification, taking into account relations between guessed and database predicates. For the sake of simplicity, we will initially focus our attention on *a single monadic* guessed predicate, trying to figure out which constraints concerning it can be delayed. Afterwards, we extend our results to *sets* of monadic guessed predicates, then to *binary* predicates.

3.1 Single monadic predicate

We refer to the 3-coloring specification of Example 1, focusing on the guessed predicate R, and trying to find an intuitive explanation for the fact that clauses (3-4) can be delayed. We immediately note that clauses in the specification can be partitioned into three subsets:

NO: R does not occur in them, i.e., (5,7,8);

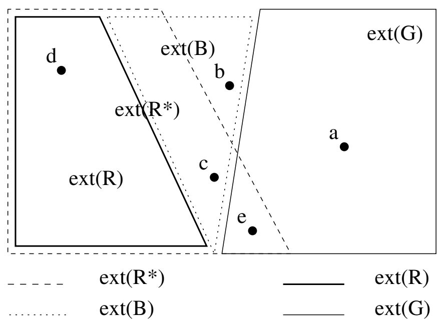


Fig. 3. Extensions for the 3-coloring spec

NEG: R occurs negatively in them, i.e., (3,4,6);
POS: R occurs positively in them, i.e., (2).

Neither **NO** nor **NEG** clauses can be violated by shrinking the extension of R. Such constraints will be called *safe-forget* for R, because if we decide to process (and satisfy) them in the first stage, they can be safely ignored in the second one. We note that this is just a possibility, and we are not obliged to do that: as an example, clauses (3-4) will *not* be processed in the first stage.

Although in general **POS** clauses are not safe-forget –because shrinking the extension of R can violate them– we now show that clause (2) it is. In fact, if we equivalently rewrite clauses (2) and (3-4), respectively, as follows:

$$\forall X\ \neg B(X) \land \neg G(X) \rightarrow R(X) \qquad (2)'$$
$$\forall X\quad R(X) \rightarrow \neg B(X) \land \neg G(X), \qquad (3-4)'$$

we note that clause $(2)'$ sets a lower bound for the extension of R, and clauses (3-4)$'$ set an upper bound for it; both the lower and the upper bound are $ext(\neg B(X) \land \neg G(X))$. If we use –in the first stage– clauses (2,5-8) for computing $ext(R^*)$, then –in the second stage– we can safely define $ext(R)$ as $ext(R^*) \cap ext(\neg B(X) \land \neg G(X))$, and no constraint will be violated (cf. Fig. 3). The next theorem shows that is not by chance that the antecedent of $(2)'$ is semantically related to the consequence of (3-4)$'$.

Theorem 1. *Let Φ be an ESO formula of the form:*

$$\exists S_1, \ldots, S_h, R \quad \Xi \quad \land \quad \forall X\ \alpha(X) \rightarrow R(X) \quad \land \quad \forall X\ R(X) \rightarrow \beta(X), \qquad (9)$$

where Ξ is a conjunction of clauses, both α and β are arbitrary formulae in which R does not occur and X is the only free variable, and it holds that:

Hyp 1: *R either does not occur or occurs negatively in Ξ;*
Hyp 2: $\models \forall X\ \alpha(X) \rightarrow \beta(X)$.

Let Φ^s be:

$$\exists S_1, \ldots, S_h, R^* \quad \Xi^* \quad \land \forall X\ \alpha(X) \rightarrow R^*(X),$$

where R^* is a new predicate symbol, and $\varXi^*$ is $\varXi$ with R replaced by R^*. Let $\varPhi^d$ be:

$$\forall X \; R(X) \leftrightarrow R^*(X) \wedge \beta(X).$$

For each database D and each list M^s of extensions for $(S_1, \ldots, S_h, R^*)$ such that $(D, M^s) \models \varPhi^s$, then $(D, M^s - ext(R^*), ext(R)) \models \varPhi$, where $ext(R)$ is the extension of R as defined by M^s and $\varPhi^d$.

Referring to Fig. 2, $\varPhi$ is the spec, D is the instance, $\varPhi^s$ is the "simplified spec", and $\forall X \; R(X) \rightarrow \beta(X)$ is the "delayed constraint". Solving $\varPhi^s$ against D produces –if the instance is satisfiable– a list of extensions M^s (the "output"). Evaluating $\varPhi^d$ against M^s corresponds to the "PostProcessing" phase in the second stage; since the last stage amounts to the evaluation of a first-order formula against a fixed database, it can be done in logarithmic space (thus in polynomial time).

In other words, the theorem says that, for each satisfiable instance D of the simplified specification $\varPhi^s$, each solution M^s of $\varPhi^s$ can be translated, via $\varPhi^d$, to a solution of the original specification $\varPhi$; we can also say that $\varXi \wedge \forall X \; \alpha(X) \rightarrow R(X)$ is safe-forget, and $\forall X \; R(X) \rightarrow \beta(X)$ is safe-delay.

Referring to the specification of Example 1, $\varXi$ is the conjunction of clauses (5-8), and $\alpha(X)$ and $\beta(X)$ are both $\neg B(X) \wedge \neg G(X)$, cf. clauses (3-4)$'$. Fig. 3 represents possible extensions of the red predicate in the first (R^*) and second (R) stages, for the instance of Fig. 1.

Proofs are omitted for lack of space. As evidence for the soundness of Theorem 1, we claim that Fig. 4 (left) shows that, if **Hyp 2** holds, then the constraint $\forall X \; \alpha(X) \rightarrow R(X)$ can never be violated in the second stage.

We are guaranteed that the two-stage process preserves at least one solution of $\varPhi$ by the following proposition.

Proposition 1. *Let D, $\varPhi$, $\varPhi^s$ and $\varPhi^d$ as in Theorem 1. For each database D, if $\varPhi$ is satisfiable, $\varPhi^s$ and $\varPhi^d$ are satisfiable.*

To substantiate the reasonableness of the two hypotheses of Theorem 1, we play the devil's advocate and add to the specification of Example 1 the constraint

$$\forall X \; edge(X, X) \rightarrow R(X), \tag{10}$$

saying that self-loops must be red. We immediately notice that now clauses (3-4) are not safe-delay: intuitively, after the first stage, nodes may be red either because of (2) or because of (10), and (3-4) are not enough to set the correct color for a node. Now, if –on top of (5-8)– $\varXi$ contains also the constraint (10), **Hyp 1** is clearly not satisfied. Analogously, if (10) is used to build $\alpha(X)$, then $\alpha(X)$ becomes $edge(X, X) \vee (\neg B(X) \wedge \neg G(X))$, and **Hyp 2** is not satisfied. Fig. 4, right, gives further evidence that the constraint $\forall X \; \alpha(X) \rightarrow R(X)$ can be violated if $ext(R)$ is computed using $\varPhi^d$ and $ext(\alpha)$ is not a subset of $ext(\beta)$.

Some further comments about Theorem 1 are in order:

- Ξ does not need to be a conjunction of clauses, but can be any formula such that, from any structure M s.t. $M \models \Xi$, shrinking $ext(R)$ and keeping everything else fixed we obtain another model of Ξ. As an example, Ξ may contain the conjunct $\exists X\ R(X) \rightarrow G(X)$.
- Although **Hyp 2** calls for a tautology check –which is not decidable in general– we will see in what follows that many specifications satisfy it *by design.*

3.2 Set of monadic predicates

Theorem 1 can be applied recursively to the specification Φ^s, by focusing on a different guessed predicate, in order to obtain a new simplified specification $(\Phi^s)^s$ and new delayed constraints $(\Phi^s)^d$. Since, by Proposition 1, satisfiability of such formulae is preserved, it is afterwards possible to translate, via $(\Phi^s)^d$, each solution of $(\Phi^s)^s$ to a solution of Φ^s, and then, via Φ^d, to a solution of Φ. The procedure REFORMULATE deals with the general case of a set of guessed predicates: if the input specification Φ is satisfiable, it returns a simplified specification $\overline{\Phi^s}$ and a list of delayed constraints $\overline{\Phi^d}$. Algorithm SOLVEBYDELAYING gets any solution of $\overline{\Phi^s}$ and translates it, via the evaluation of formulae in the list $\overline{\Phi^d}$ –with LIFO policy– to a solution of Φ.

```
**Algorithm** SOLVEBYDELAYING
**Input** a spec $\Phi$, a database $D$
**Output** a solution of $\langle D, \Phi \rangle$, if
    satisfiable; ‘unsatisfiable’ o.w.;
**begin**
  $\langle \overline{\Phi^s}, \overline{\Phi^d} \rangle$ = REFORMULATE($\Phi$);
  **if** ($\langle \overline{\Phi^s}, D \rangle$ is satisfiable) **then**
  **begin**
    **let** $M$ be a solution of $\langle \overline{\Phi^s}, D \rangle$;
    **while** ($\overline{\Phi^d}$ is not empty) **do**
    **begin**
      **Constraint** $d = \overline{\Phi^d}$**.pop**();
      $M = M \cup$ solution of $d$;
      // cf. Theorem 1
    **end**;
    **return** $M$;
  **end**;
  **else return** ‘unsatisfiable’;
**end**;
```

```
**Procedure** REFORMULATE
**Input** a specification $\Phi$
**Output** $\langle \overline{\Phi^s}, \overline{\Phi^d} \rangle$, where $\overline{\Phi^s}$ a simplified spec,
    and $\overline{\Phi^d}$ a stack of delayed constraints
**begin**
  **Stack** $\overline{\Phi^d}$ = the empty stack;
  $\overline{\Phi^s} = \Phi$;
  **for each** monadic guessed pred. $R$ in $\overline{\Phi^s}$ **do**
  **begin**
    partition constr. in $\overline{\Phi^s}$ according to Thm 1, in:
    $\langle \Xi;\ \forall X\, \alpha(X) \rightarrow R(X);\ \forall X\, R(X) \rightarrow \beta(X) \rangle$
    **if** (prev. step is possible with $\forall X\, \beta(X) \neq$ TRUE)
    **then begin**
      $\overline{\Phi^d}$**.push**(‘$\forall X\, R(X) \leftrightarrow R^*(X) \wedge \beta(X)$’);
      $\overline{\Phi^s} = \Xi^* \wedge \forall X\ \alpha(X) \rightarrow R^*(X)$;
    **end**;
  **end**;
  **return** $\langle \overline{\Phi^s}, \overline{\Phi^d} \rangle$;
**end**;
```

As an example, we evaluate the procedure REFORMULATE on the specification of Example 1, by focusing on the guessed predicates in the order R, G, B. The output is the following simplified specification $\overline{\Phi^s}$:

$$\exists R^* G^* B\ \ \forall X\ \ R^*(X) \vee G^*(X) \vee B(X)\ \ \wedge$$

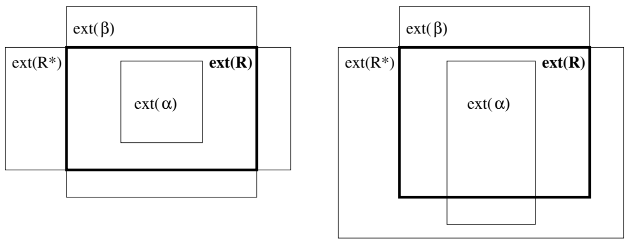


Fig. 4. Extensions with and without **Hyp 2**

$$\begin{aligned}&\forall XY\ X \neq Y \wedge R^*(X) \wedge R^*(Y) \rightarrow \neg edge(X,Y)\ \wedge\\&\forall XY\ X \neq Y \wedge G^*(X) \wedge G^*(Y) \rightarrow \neg edge(X,Y)\ \wedge\\&\forall XY\ X \neq Y \wedge B(X) \wedge B(Y) \rightarrow \neg edge(X,Y),\end{aligned}$$

and the following list $\overline{\Phi^d}$ of delayed constraints:

$$\forall X\ \ R(X) \leftrightarrow R^*(X) \wedge \neg G(X) \wedge \neg B(X); \tag{11}$$

$$\forall X\ \ G(X) \leftrightarrow G^*(X) \wedge \neg B(X). \tag{12}$$

Note that the check that $\forall X\ \beta(X)$ is not a tautology prevents the (useless) delayed constraint $\forall X\ B(X) \leftrightarrow B^*(X)$ to be pushed in $\overline{\Phi^d}$.

From any solution of $\overline{\Phi^s}$, a solution of Φ is obtained in the **while** loop of SolveByDelaying by reconstructing first of all the extension for G by formula (12), and then the extension for R by formula (11) (synthesized, respectively, in the second and first iteration of the algorithm). Since each constraint in the stack $\overline{\Phi^d}$ is first-order, the whole **while** loop is doable in logarithmic space.

We observe that the procedure Reformulate is intrinsically non-deterministic, because of the partition that must be applied to the constraints.

3.3 Binary predicates

The specification of the k-coloring problem using a binary predicate Col –the first argument being the node and the second the color– is as follows (constraints represent, respectively, covering, disjointness, and good coloring).

$$\begin{aligned}&\exists Col\ \forall X\ \exists Y\ Col(X,Y) \wedge\\&\qquad \forall XYZ\ Col(X,Y) \wedge Col(X,Z) \rightarrow Y = Z \wedge\\&\qquad \forall XYZ\ X \neq Y \wedge Col(X,Z) \wedge Col(Y,Z) \rightarrow \neg edge(X,Y).\end{aligned} \tag{13}$$

Since the number of colors is finite, it is always possible to unfold the above constraints with respect to the second argument of Col. As an example, if $k = 3$, we obtain –up to an appropriate renaming of the Col predicate– the specification of Example 1.

The above considerations imply that we can use the architecture of Fig. 2 for a large class of specifications.

Definition 1. *A* safe-delay functional spec *is an ESO formula of the form:*

$$\exists P \quad \Xi \quad \wedge \quad \forall X\, \exists Y\; P(X,Y) \quad \wedge \quad \forall XYZ\; P(X,Y) \wedge P(X,Z) \rightarrow Y = Z,$$

where Ξ is a conjunction of clauses in which P either does not occur or occurs negatively.

The term "functional" originates from covering and disjointness constraints, which imply a search space which is a (total) function from a finite domain to a finite codomain. In particular, the disjointness constraints are safe-delay, while the covering and the remaining ones, i.e., Ξ, are safe-forget. Formally, soundness of the architecture on safe-delay functional formulae is guaranteed by Theorem 1 and algorithm SolveByDelaying.

Safe-delay functional specifications are quite common; apart from graph coloring, notable examples (formal details are omitted for lack of space) are Bin packing [15] (which goal is to pack n objects of given sizes into k bins of capacity c) and Job shop scheduling [13, Prob. SS18]: there are n jobs, m tasks, and p processors. Jobs are ordered collections of tasks and each task has an integer-valued length and the processor that performs it. Each processor can perform a task at the time, and the tasks belonging to the same job must be performed in their order. Finally, there is a global deadline D that has to be met by all jobs. The disjointness constraint imposes at most one starting time for each task: thus, by applying Theorem 1 for delaying it, we allow a task to have multiple starting times, i.e., the task does not overlap with any other task at any of its start times. In the PostProcessing stage we can arbitrarily choose one of them to obtain a solution of the original problem.

4 Methodological discussion

In this section we make a discussion on the methodology we adopted in this work, in particular the use of ESO as a modelling language, and the choice of the solvers for the preliminary experimentation.

Using ESO for specifying problems wipes out many aspects of state-of-the-art languages which are somehow difficult to take into account (e.g., numbers, arithmetics, constructs for functions, etc.), thus simplifying the task of finding criteria for reformulating problem specifications. However, it must be observed that ESO, even if somewhat limited, is not too far away from the modelling languages provided by some commercial systems. An example of such a language is AMPL, which admits only linear constraints: in this case, the reformulation technique described in Theorem 1 can often be straightforwardly applied; for instance, a reasonable specification of the k-coloring problem in such a language is the following:

```
param n_nodes;   param n_colors integer, > 0;
set NODES := 1..n_nodes;   set EDGES within NODES cross NODES;
set COLORS := 1..n_colors;
var Coloring {NODES,COLORS} binary;   # Coloring as a 2-ary predicate
```

```
s.t. CoveringAndDisjointness {x in NODES}:
    sum {c in COLORS} Coloring[x,c] = 1;   # nodes have exactly one color
s.t. GoodColoring {(x,y) in EDGES, c in COLORS}:
    Coloring[x,c] + Coloring[y,c] <= 1;   # nodes linked by an edge have diff. colors
```

The above specification is similar to (13), and the reformulated spec can be obtained by rewriting the "CoveringAndDisjointness" constraint in the following way:

```
s.t. Covering {x in NODES}
    sum {c in COLORS} Coloring[x,c] >= 1;
```

As for languages that admit non linear constraints, e.g., OPL, it is possible to write a different specification using integer variables for the colors and inequality of colors between adjacent nodes. In this case it is not possible to separate the disjointness constraint from the other ones, since it is implicit in the definition of the domains.

For what concerns the experimentation, it must be noted that a specification written in ESO naturally leads to a translation into a SAT instance. For this reason, we have chosen to use a SAT solver to start the experimentation of the proposed technique. Anyway, since it is well-known that state-of-the-art linear and constraint programming systems may perform much better than SAT on some problems, we repeated the experimentation using the commercial systems CPLEX and SOLVER. Thus, SAT-based experiments have to be considered as a starting point and an encouraging evidence of the reasonableness of the proposed approach, by showing that consistent speed-ups in the solving process can be obtained by a mere reformulation of a pure declarative specification. Actually, this evidence has been partially confirmed by the experiments done using the linear integer programming solver CPLEX: also in this case, several classes of instances benefit from our reformulation technique.

5 Experimental results

We made an experimentation of our reformulation techniques on 3-coloring (randomly generated instances), k-coloring (instances taken from the DIMACS benchmark repository `ftp://dimacs.rutgers.edu/pub/challenge`), and job shop scheduling (benchmark instances taken from the OR library at `www.ms.ic.ac.uk/info.html`).

We solved each instance both with and without delaying the disjointness constraints, using both the DPLL-based SAT system SATZ [14] (and an ad hoc program [4] for the instantiation stage), and the state-of-the-art constraint and linear programming system OPL [17], obviously using it as a pure modeling language, and omitting search procedures. For what concerns the latter system, we wrote both a linear and a non-linear specification for the above problems, and applied our reformulation technique to the linear ones (cf. Sec. 4)

Experiments were executed on an Intel 2.4 GHz Xeon bi-processor computer. The size of instances was chosen so that our machine is able to solve (most of) them in more than few seconds, and less than one hour. In this way, both instantiation and postprocessing, i.e., evaluation of delayed constraints, times are negligible, and comparison can be done only on solving time.

Summing up, we solved several thousands of instances. For what concerns the SAT experimentation, it is worth noting that in all instances the SAT time without disjointness is less than or equal to the time with disjointness. As far as CPLEX is concerned, we found that several (but not all) instances benefitted from delaying constraints, especially those for which the linear specification is more efficient than the non-linear one (cf. the following paragraphs).

In what follows, we refer to the *saving percentage*, defined as the ratio

$$(\mathit{time\_with\_disjointness} - \mathit{time\_without\_disjointness})\,/\,\mathit{time\_with\_disjointness}.$$

3-coloring We solved the problem on 3,500 randomly generated graph instances with 430 nodes each. The number of edges varies, and covers the phase transition region [6]: the ratio (# of directed edges/# of nodes) varies between 2 and 6. The average solving time (150 instances for each fixed number of edges) varies between fractions of a second and a minute. The saving percentage in the SAT-based experiments varies between 15% and 50%, for both positive and negative instances, the hardest instances being at 30%. Experimentation with CPLEX and SOLVER is planned.

k-coloring Results of our SAT-based experiments are shown in Table 1 (n is the number of nodes and e of the edges). The saving percentage varies between 9.0% and 59%.

On the other hand, when applying the reformulation technique to OPL, we observed two major evidences:

1. It is not the case that a specification (linear or non-linear) is always more efficient than the other one. In particular, the ratio between the solving time of CPLEX and SOLVER is highly variable, and the linear specification can be much more efficient than the non-linear one, especially for *negative* instances.
2. By focusing on the class of instances for which the linear specification is more efficient than the non-linear one, we found out several instances in which delaying the disjointness constraint leads to appreciable time savings, as Table 1 shows.

Job shop scheduling We considered two instances known as FT06 (36 tasks, 6 jobs, 6 processors, solvable with deadline 55) and LA02 (50 tasks, 10 jobs, 5 processors, solvable with deadline 655). SAT solving times are listed in Table 2 for different values for the deadline. As it can be observed, the saving is quite consistent.

For what concerns the experiments in OPL, CPLEX seems not to be affected much by delaying constraints (or even affected negatively), and anyway it is slower than SOLVER.

Summing up, delaying of constraints seems to be always effective when using a SAT solver. As far as CPLEX is concerned, we have mixed evidence. We plan

Table 1. Solving times (seconds) for k-coloring

	Col-	Solv-	SAT			CPLEX			SOLVER
Instance	ors	able?	W/ disj.	W/o disj.	% sav.	W/ disj.	W/o disj.	% sav.	W/ disj.
le450_5d	5	Y	6.0	5.2	13.3	540.8	628.1	−16.1	1.2
le450_15a	13	N	–	–	–	8.6	6.3	26.7	–
le450_25a	21	N	–	–	–	83.6	17.2	79.4	–
DSJC125.9	21	N	–	–	–	1408.4	936.7	33.5	–
DSJC250.1	9	Y	1.6	0.9	43.8	–	–	–	12.5
DSJR500.1	11	N	–	–	–	2.2	1.3	40.9	297.4
DSJR500.1	12	Y	–	–	–	18.4	18.1	1.6	0.5
queen8_8	9	Y	1.5	1.2	20.0	–	2411.2	>33.0	1.9
queen9_9	10	Y	32.1	25.3	21.2	–	–	–	134.8
queen10_10	15	Y	1.3	0.9	30.8	3.3	3.1	6.1	0.9
queen11_11	13	Y	22.9	17.7	22.7	–	125.1	>96.5	41.7
queen12_12	15	Y	1.3	0.8	38.5	463.7	228.3	50.8	9.7
fpsol2.i.2	21	N	–	–	–	21.8	13.6	37.6	–

Table 2. Solving times (seconds) for job shop scheduling

	Dead-	Solv-	SAT			CPLEX			SOLVER
Instance	line	able?	W/ disj.	W/o disj.	% sav.	W/ disj.	W/o disj.	% sav.	W/ disj.
FT06	100	Y	–	4.6	∼100	117.4	–	$\sim -\infty$	1.1
FT06	65	Y	3.7	1.7	52.8	–	–	–	1.2
FT06	55	Y	4.0	1.6	60.7	–	–	–	1.5
FT06	54	N	16.1	4.3	73.6	–	–	–	–
FT06	52	N	2.0	1.0	48.0	–	–	–	7.2
LA02	1200	Y	4.6	2.1	53.9	31.2	–	$\sim -\infty$	1.5
LA02	1000	Y	2.9	1.5	48.8	–	–	–	1.6
LA02	960	Y	22.2	1.8	91.8	–	–	–	–
LA02	860	Y	740.0	8.5	98.8	–	–	–	–
LA02	840	Y	–	15.1	∼100	–	–	–	–

(‘–’ means that the solver did not terminate in one hour)

to extend the experimentation to problems in which SAT performs worse than other solvers.

6 Conclusions, related and future work

In this paper we have shown a simple reformulation architecture and proven its soundness for a large class of problems. The reformulation allows to delay the solution of some constraints, which results in faster solving. In this way, we have shown that reasoning on a specification can be very effective.

Related work Several researchers addressed the issue of reformulation of a problem *after the instantiation phase*. As an example, in [11] it is shown that abstracting problems by simplifying constraints is useful for finding more efficient reformulations of the original problem; abstraction may require backtracking for finding solutions of the original problem. In our work, we focus on reformulation of the specification, and the approach is backtracking-free: once the first stage is completed, a solution will surely be found by evaluating the delayed constraints.

Analogously, in [19] it is shown how to translate an instantiated CSP into its Boolean form, which is useful for finding different reformulations. Finally, in [7] it is shown how to generate a conjunctive decomposition of an instantiated CSP, by localizing independent subproblems.

Other papers investigate the best way to encode an instance of a problem into a format adequate for a specific solver. As an example, many different ways for encoding graph coloring or permutation problems into SAT have been figured out, cf., e.g., [12, 18]. Conversely, in our work we take a specification-oriented approach.

Finally, we point out that a logic-based approach has also been successfully adopted in the ‘80s to study the query optimization problem for relational DBs. Analogously to the approach presented in this paper, the query optimization problem has been attacked relying on the query (i.e., the specification) only, without considering the database (i.e., the instance), and it was firstly studied in a formal way using first-order logic (cf., e.g., [2, 5]). In a later stage, the theoretical framework has been translated into rules for the automated rewriting of queries expressed in languages and systems used in real world.

Future work In this paper we have focused on a form of reformulation which partitions the first-order part of a specification. This basic idea can be generalized, as an example by evaluating in both stages of the computation a constraint (as an example (10)), in order to allow reformulation for a larger class of specifications. Even more generally, the second stage may amount to the evaluation of a second-order formula. In the future, we plan –with a more extensive experimentation– to check whether such generalizations are effective in practice. It is also interesting to check how delaying of constraints behave when used with other techniques, e.g., symmetry breaking.

Moreover, we wish to extend our results to the so-called *permutation* problems, i.e., problems which, on top of the constraints of Definition 1 have also the following constraints:

$$\forall X\ \exists Y\ P(Y,X) \quad \wedge \quad \forall XYZ\ P(Y,X) \quad \wedge \quad P(Z,X) \rightarrow Y = Z.$$

However, this kind of specifications do not belong to the class studied in this paper, and a generalization of Theorem 1 is needed to support them. Permutation problems arise frequently in theory and practice: examples are the *n-queens*, the *Hamiltonian circuit*, and the *code generation for parallel assignments* (resp. [13, Prob. GT39, PO6]) problems.

Finally, it is our goal to rephrase the theoretical results into rules for automatically reformulating problem specifications given in much more complex languages, e.g. AMPL and OPL, which have higher-level built-in constructs, such as integers and functions.

Acknowledgements This research has been supported by MIUR (Italian Ministry for Instruction, University, and Research) under the FIRB project ASTRO (Automazione dell'Ingegneria del Software basata su Conoscenza).

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison Wesley Publ. Co., Reading, Massachussetts, 1995.
2. S. Y. Aho, A.V. and J. Ullman. Equivalences among relational expressions. *SIAM Journal on Computing*, 2(8):218–246, 1979.
3. E. Börger, E. Gräedel, and Y. Gurevich. *The Classical Decision Problem.* Perspectives in Mathematical Logic. Springer, 1997.
4. M. Cadoli and A. Schaerf. Compiling problem specifications into SAT. In *Proceedings of the European Symposium On Programming (ESOP 2001)*, volume 2028 of *LNCS*, pages 387–401. Springer, 2001.
5. A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc. 9th ACM Symp. Theory of Computing (STOC'77)*, pages 77–90. Boulder, CO, USA, 1977.
6. P. Cheeseman, B. Kanefski, and W. M. Taylor. Where the really hard problem are. In *Proc. of IJCAI'91*, pages 163–169, 1991.
7. B. Y. Choueiry and G. Noubir. On the computation of local interchangeability in discrete constraint satisfaction problems. In *Proc. of AAAI'98*, pages 326–333, 1998.
8. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR system `dlv`: Progress report, Comparisons and Benchmarks. In *Proc. of KR'98*, pages 406–417, 1998.
9. R. Fagin. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In R. M. Karp, editor, *Complexity of Computation*, pages 43–74. AMS, 1974.
10. R. Fourer, D. M. Gay, and B. W. Kernigham. *AMPL: A Modeling Language for Mathematical Programming.* International Thomson Publishing, 1993.
11. E. C. Freuder and D. Sabin. Interchangeability supports abstraction and reformulation for multi-dimensional constraint satisfaction. In *Proc. of AAAI'97*, pages 191–196, 1997.
12. A. M. Frisch and T. J. Peugniez. Solving non-boolean satisfiability problems with stochastic local search. In *Proc. of IJCAI 2001*, pages 282–290, 2001.
13. M. R. Garey and D. S. Johnson. *Computers and Intractability—A guide to NP-completeness.* W.H. Freeman and Company, San Francisco, 1979.
14. C. M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proc. of IJCAI'97*, pages 366–371, 1997.
15. S. Martello and P. Toth. *Knapsack Problems: algorithms and computer implementation.* John Wiley & Sons Ltd, 1990.
16. I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. of Mathematics and Artificial Intelligence*, 25(3,4):241–273, 1999.
17. P. Van Hentenryck. *The OPL Optimization Programming Language.* The MIT Press, 1999.
18. T. Walsh. Permutation problems and channelling constraints. In *Proc. of LPAR 2001*, number 2250 in LNCS, pages 377–391. Springer, 2001.
19. R. Weigel and C. Bliek. On reformulation of constraint satisfaction problems. In *Proc. of ECAI'98*, pages 254–258, 1998.

Semi-Automatic Modeling by Constraint Acquisition [⋆]

Remi Coletta[1], Christian Bessiere[1], Barry O'Sullivan[2],
Eugene C. Freuder[2], Sarah O'Connell[2], and Joel Quinqueton[1]

[1] LIRMM-CNRS (UMR 5506), 161 rue Ada 34392 Montpellier Cedex 5, France
{coletta,bessiere,jq}@lirmm.fr
[2] Cork Constraint Computation Centre, University College Cork, Ireland
{b.osullivan,e.freuder,s.oconnell}@4c.ucc.ie

Abstract. Constraint programming is a technology which is now widely used to solve combinatorial problems in industrial applications. However, using it requires considerable knowledge and expertise in the field of constraint reasoning. This paper introduces a framework for automatically learning constraint networks from sets of instances that are either acceptable solutions or non-desirable assignments of the problem we would like to express. Such an approach has the potential to be of assistance to a novice who is trying to articulate her constraints. By restricting the language of constraints used to build the network, this could also assist an expert to develop an efficient model of a given problem. This paper provides a theoretical framework for a research agenda in the area of interactive constraint acquisition, automated modelling and automated constraint programming.

1 Introduction

Over the last 30 years, considerable progress has been made in the field of Constraint Programming (CP), providing a powerful paradigm for solving complex problems. Applications in many areas such as resource allocation, scheduling, planning and design have been reported in the literature [16]. However, the use of CP still remains limited to specialists in the field. Modelling a problem in the constraint formalism requires significant expertise in constraint programming. This precludes novices from being able to use CP on complex problems without the help of an expert. This has a negative effect on the uptake of constraint technology in the real-world by non-experts [5].

In addition, in many practical applications humans find it difficult to articulate their constraints. While the human user can recognize examples of where their constraints should be satisfied or violated, they cannot articulate the constraints themselves. However, by presenting examples of what is acceptable, the human user can be assisted in developing a model of the set of constraints she is trying to articulate. This can be regarded as an instance of constraint acquisition. One of the goals of our work is to assist the, possibly novice, human user by providing semi-automatic methods for acquiring the user's constraints.

[⋆] The collaboration between LIRMM and the Cork Constraint Computation Centre is supported by a Ulysses Travel Grant from Enterprise Ireland, the Royal Irish Academy and CNRS (Grant Number FR/2003/022). This work has also received support from Science Foundation Ireland under Grant 00/PI.1/C075.

Furthermore, even if the user has sufficient experience in CP to encode her problem, a poor model can negate the utility of a good solver based on state-of-the-art filtering techniques. For example, in order to provide support for modelling, some solvers provide facilities for defining constraints extensionally (i.e., by enumerating the set of allowed tuples). Such facilities considerably extend the expressiveness and ease-of-use of the constraints language, thus facilitating the definition of complex relationships between variables. However, a disadvantage of modelling constraints extensionally is that the constraints lose any useful semantics they may have which can have a negative impact on the inference and propagation capabilities of a solver. As a result, the resolution performance of the solver can be significantly deteriorated in the parts of the problem where such constraints are used. Therefore, another goal of our work is to facilitate the expert user who wishes to reformulate her problem (or a part of it that is suspected of slowing down the resolution). Given sets of accepted/forbidden instantiations of the (sub)problem (that can be generated automatically from the initial formulation), the expert will be able, for instance, to test whether an optimised constraint library associated with her solver is able to model the (sub)problem in a way which lends itself to being efficiently solved.

However, constraint acquisition is not only important in an interactive situation involving a human user. Often we may wish to acquire a constraint model from a large set of data. For example, given a large database of tuples defining buyer behaviour in a variety of markets, for a variety of buyer profiles, for a variety of products, we may wish to acquire a constraint network which describes the data in this database. While the nature of the interaction with the source of training data is different, the constraint acquisition problem is fundamentally the same.

This paper makes the following contributions:

1. We present an approach to semi-automatically acquiring models of constraint satisfaction problems from examples;
2. We present an algorithm for acquiring such models, we prove its correctness and report on some empirical results which demonstrate some properties of our approach;
3. Finally, we give some insights into our research agenda in this area.

The remainder of this paper is organised as follows. Section 2 presents an overview of the related work in this area, Section 3 provides some preliminary definitions on constraint networks. Section 4 briefly presents the machine learning techniques that can be used for our problem. In Section 5, we formulate our problem as a learning problem. Section 6 presents the technique in detail, and proves some properties of the approach that are guaranteed. In Section 7, some of the issues that the approach raises are presented, and their possible effects on the learning process are illustrated by some preliminary experiments. A brief overview of the main topics in our research agenda are outlined in Section 8. Some concluding remarks are made in Section 9.

2 Related Work

Recently, researchers have become more interested in techniques for solving problems where users have difficulties articulating constraints. Padmanabhuni *et al.* have proposed a framework for learning constraints [12]. In [14], the goal of Rossi and Sperduti is not exactly to help the user learning a constraint network, but to help her learning the valuations of the tuples in a semi-ring constraint network where the constraint structures are already given. Freuder and Wallace have considered suggestion strategies for applications where a user cannot articulate all constraints in advance, but can articulate additional constraints when confronted with something which is unacceptable [3]. Freuder and O'Sullivan have focused on generating constraints which model tradeoffs between variables in problems which have become over-constrained during an interactive configuration session [2]. Version spaces have been reported by O'Connell *et al* for acquiring single constraints, with a focus on acquisition from humans where dialog length is a critical factor [9, 10]. The focus of their work was interactive acquisition of constraints from users of differing abilities. The motivation for their work comes from the field of design [11]. Page and Frisch have studied the issue of generalising constraints and PAC-learning of constrained atoms in an inductive logic programming context [13].

Furthermore, a considerable amount of work has been done in constraint reasoning to identify polynomial classes of constraint networks [1, 15]. Unfortunately, these polynomial classes are weak in terms of expressivity and are capable of encoding only small number of interesting real problems. These polynomial classes are based on static properties, and do not adapt themselves to the kind of problem the user wants to solve. We, therefore, have the problem of finding a problem which fits a polynomial class, and not the reverse. The approach proposed in this paper intends to be more flexible. It does not require that we define *a priori* classes in which the expressivity of a problem is to be tested, but lets the expert define it by herself. In this context we can imagine an interactive scenario in which the user tries more and more expressive libraries of constraints until her problem can be adequately expressed. While the manner in which the constraint language (library) is chosen is critical, it is outside the scope of this paper. However, we could use a case-based reasoning approach selecting libraries depending on the type of problem being tackled [5].

3 Preliminaries

Definition 1 (Constraint Network) *A constraint network is defined as a triplet* $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ *where:*

- $\mathcal{X} = \{X_1, \ldots, X_n\}$ *is a set of variables.*
- $\mathcal{D} = \{D_{X_1}, \ldots, D_{X_n}\}$ *is the set of their domains: each variable* X_i *takes its values in the domain* D_{X_i}.
- $\mathcal{C} = (C_1, \ldots, C_m)$ *is a sequence of constraints on* $\mathcal{X}$ *and* $\mathcal{D}$*, where a constraint* C_i *is defined by the sequence* $var(C_i)$ *of variables it involves, and the relation* $rel(C_i)$ *specifying the allowed tuples on* $var(C_i)$.

We regard the constraints as a sequence to simplify the forthcoming notations.

Definition 2 (Instance) *Let* $Y = \{Y_1, \cdots, Y_k\}$ *be a subset of* $\mathcal{X}$*. An* instance e_Y *on* Y *is a tuple* $(v_1, \ldots, v_k) \in D_{Y_1} \times \cdots \times D_{Y_k}$*. This instance is partial if* $Y \neq \mathcal{X}$*, complete otherwise (noted e). An instance* e_Y *on* Y violates *the constraint* C_i *iff* $var(C_i) \subseteq Y$ *and* $e_Y[var(C_i)] \notin rel(C_i)$.

Definition 3 (Solution) *A complete instance on the set $\mathcal{X}$ of variables is a* solution *of the constraint network $N = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ iff it does not violate any constraint. Otherwise it is a* non solution. *$Sol(N)$ denotes the set of solutions of N.*

4 The Fundamental Problem

As a starting point, we assume that the user knows the set of variables of her problem and their domains of possible values. She is also assumed to be able to classify an instance as positive (a solution) or negative (non-solution). Therefore, the available data are the set $\mathcal{X}$ of the variables of the problem, their domains $\mathcal{D}$, a subset E^+ of the solutions of the problem, and a set E^- of non-solutions.

In addition to the "assisting the expert" perspective, the aim is to code the problem efficiently, using only efficient constraint relations between these variables; i.e. a library of constraints with efficient propagation features is assumed to be given. Indications can also be given revealing the possible location of the constraints, by defining variables between which constraints must be found (learned), or by restricting ourselves to binary constraints only. These semantic and structural limitations define the inductive bias:

Definition 4 (Bias) *Given a set $\mathcal{X}$ of variables and the set $\mathcal{D}$ of their domains, a* bias $\mathcal{B}$ *on $(\mathcal{X}, \mathcal{D})$ is a sequence $(B_1, \ldots, B_m)$ of local biases, where a local bias B_i is defined by a sequence $var(B_i) \subseteq \mathcal{X}$ of variables, and a set $L(B_i)$ of possible relations on $var(B_i)$.*

The set $L(B_i)$ of relations allowed on a set of variables $var(B_i)$ can be any library of constraints of arity $|var(B_i)|$.

Definition 5 (Membership of a Bias) *Given a set $\mathcal{X}$ of variables and the set $\mathcal{D}$ of their domains, a sequence of constraints $\mathcal{C} = (C_1, \ldots, C_m)$* belongs *to the bias $\mathcal{B} = (B_1, \ldots, B_m)$ on $(\mathcal{X}, \mathcal{D})$ if $\forall C_i \in \mathcal{C}, var(C_i) = var(B_i)$ and $rel(C_i) \in L(B_i)$. We note $\mathcal{C} \in \mathcal{B}$.*

The problem consists in looking for a sequence of constraints $\mathcal{C}$ belonging to a given bias $\mathcal{B}$, and whose solution set is a superset of E^+ containing no element of E^-.

Definition 6 (Constraint Acquisition Problem) *Given a set of variables $\mathcal{X}$, their domains $\mathcal{D}$, two sets E^+ and E^- of instances on $\mathcal{X}$, and a bias $\mathcal{B}$ on $(\mathcal{X}, \mathcal{D})$, the* constraint acquisition problem *consists in finding a sequence of constraints $\mathcal{C}$ such that:*

$\mathcal{C} \in \mathcal{B}$,
$\forall e^- \in E^-$, e^- is a non solution of $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, and,
$\forall e^+ \in E^+$, e^+ is a solution of $(\mathcal{X}, \mathcal{D}, \mathcal{C})$.

If the sets E^+ and E^-, called the *training data*, are provided by an interaction with the user, then the acquisition problem can be regarded as the modelling phase for the user's problem. Otherwise, it can be regarded as an assistance to the expert for an automatic reformulation of her problem.

We can point out that if $E^+ \cup E^- = D_{X_1} \times \cdots \times D_{X_n}$, and $\mathcal{B}$ is a bias on $(\mathcal{X}, \mathcal{D})$ containing $n(n-1)/2$ local biases such that for each pair of variables (X_i, X_j), $\exists B_i \in \mathcal{B}$

with $var(B_i) = (X_i, X_j)$, and $L(B_i) = \mathcal{P}(D_{X_i} \times D_{X_j})$,[1] then the constraint acquisition problem answers the representability problem of a relation $\rho = E^+$ with a binary constraint network [8].

5 Constraint Acquisition as Concept Learning

Concept induction is a well known paradigm in Machine Learning. The underlying problem can be described the following way: given a set H of hypotheses, two training data sets (E^+ of positive and E^- of negative instances), find an hypothesis h consistent with this training data, i.e., which rejects all the negative instances and accepts all the positive instances. The concept providing the training data is called the target concept.

In our context, this concept is the unknown network that we are looking for that consistently captures all the information given by the user in the training set. So, in our vocabulary:

- An hypothesis h is a sequence of constraints,
- H is the set of possible sequences of constraints belonging to $\mathcal{B}$,
- The target concept is the sequence of constraints $\mathcal{C}$ we are looking for,
- A positive instance is a solution of $(\mathcal{X}, \mathcal{D}, \mathcal{C})$, a negative one is a non-solution of $(\mathcal{X}, \mathcal{D}, \mathcal{C})$.

There are many techniques from the field of Machine Learning, from decision trees to neural networks or genetic algorithms. We propose here a method based on version spaces [7], which has several nice properties amongst which the most interesting from our perspective are: they provide two approximations of the target concept, an upper bound and a lower bound; their computation is incremental with respect to the training data; and the result does not depend on the order of the instances in the training set (commutativity). This last property is essential in an interactive acquisition process.

We briefly present version spaces, which rely on the partial-order based on inclusion in the set H of hypotheses.

Definition 7 (Generalisation relation $\leq_G$) *Given $(\mathcal{X}, \mathcal{D})$ a set of variables and their domains, an hypothesis h_1 is less general than or equal to an hypothesis h_2 (noted $h_1 \leq_G h_2$) iff the set of solutions of $(\mathcal{X}, \mathcal{D}, h_1)$ is a subset of this of $(\mathcal{X}, \mathcal{D}, h_2)$.*

A version space does not only provide one consistent hypothesis, but the whole subset of H consistent with the training data:

Definition 8 (Version Space) *Given $(\mathcal{X}, \mathcal{D})$ a set of variables and their domains, E^+ and E^- two training data sets , and H a set of hypotheses, the* version space *is the set:*

$$V = \{h \in H / E^+ \subseteq Sol(\mathcal{X}, \mathcal{D}, h), E^- \cap Sol(\mathcal{X}, \mathcal{D}, h) = \emptyset\}$$

Because of its nice property of incrementality with respect to the training data, a version space is learned by incrementally processing the training instances of E^+ and E^-. In addition, due to the $\leq_G$ partial order, a version space V is completely characterised by two boundaries: the specific boundary S of maximally specific (minimal) elements of V (according to $\leq_G$), and the general boundary G of maximally general (maximal) elements.

[1] E being a set, $\mathcal{P}(E)$ is the set of subsets of E.

Property 1 *Given a version space V, and its boundaries S and G, $\forall h \in V, \exists s \in S$ and $\exists g \in G / s \leq_G h \leq_G g$.*

In the general case, V is exponential in the size of the data. So, thanks to Property 1, the constraint acquisition problem is restricted to computing the bounds S and G of the version space consistent with (E^+, E^-).

Given a set of hypotheses H on $(\mathcal{X}, \mathcal{D})$, and the training data (E^+, E^-), if there does not exist any $h \in H$ consistent with (E^+, E^-), then the version space acquisition will finish in a state where there exists $s \in S$ and $g \in G$ such that $s \neq g$ and $g \leq_G s$. This is called the *collapsing* state of the version space.

6 Learning the Constraint Version Space

In this section, we describe the process of learning the version space corresponding to the constraint acquisition problem on $(\mathcal{X}, \mathcal{D})$ with the two training sets (E^+, E^-) of solutions and non-solutions, and bias $\mathcal{B}$ on $(\mathcal{X}, \mathcal{D})$.

Let us first define the concepts that will be used at the single constraint level. We can project the generalisation relation $\leq_G$ at the constraint level. To be completely consistent with version space theory, we define $L^H(B_i) = L(B_i) \cup \{\bot, \top\}$, where $\bot$ is the empty relation $\bot$, and $\top$ the universal relation. Note that without loss of generality, the universal relation can be stated as belonging to any library of constraints. Thus, $\leq_g$ is a partial order on $L^H(B_i)$ such that $\forall r_1, r_2 \in L^H(B_i), r_1 \leq_g r_2 \Leftrightarrow r_1 \subseteq r_2$.

Given $L_1 \subseteq L^H(B_i)$ and $L_2 \subseteq L^H(B_i)$, we note that $L_1 \leq_g L_2$ iff $\forall r_1 \in L_1, \forall r_2 \in L_2, r_1 \leq_g r_2$.

$$\begin{tikzcd}
 & \top & \\
\neq \arrow[ur] & \leq \arrow[u] & \geq \arrow[ul] \\
< \arrow[u] \arrow[ur] & > \arrow[ul] \arrow[ur] & = \arrow[u] \arrow[ul] \\
 & \bot \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

Fig. 1. $L^H(B_i)$

Example 1 Let $L(B_i) = \{<, \leq, =, \geq, >, \neq\}$ be a given local bias. Fig. 1 shows the set $(L^H(B_i), \leq_g)$, which in this case is a lattice.

Restricting ourselves to each constraint individually, we introduce a local version space for each local bias. Because $\leq_g$ is, like $\leq_G$, a partial order, each local version space inherits Property 1. Thus, each local version space is completely characterized by its own local specific and general boundaries.

Definition 9 (Local boundaries) *$L(S_i)$ (resp. $L(G_i)$) is the set of relations of $L^H(B_i)$ which appear in an element of S (resp. G):*

$$L(S_i) = \{r \in L^H(B_i) / \exists s \in S : (var(Bi), r) \in s\}$$

$$L(G_i) = \{r \in L^H(B_i) / \exists g \in G : (var(Bi), r) \in g\}$$

S_i and G_i are the corresponding sets of constraints:

$$S_i = \{(var(B_i), r)\},\ \textit{where } r \in L(S_i); \qquad G_i = \{(var(B_i), r)\},\ \textit{where } r \in L(G_i)$$

6.1 The learning process

We are now ready to describe the CONACQ algorithm (Algorithm 1), which takes as input two training sets E^+, E^-, and returns the corresponding version space V on the bias $\mathcal{B}$. We present step by step the different scenarios that can occur when a training instance is processed.

Instances from $\boldsymbol{E}^+$ A positive instance e^+ must be a solution of all the networks $(\mathcal{X}, \mathcal{D}, h)$ for which $h \in V$. So,

$$\forall h \in V, \forall C_i \in h, e^+[var(C_i)] \in rel(C_i)$$

Projecting onto the local version spaces of each local bias B_i, we obtain the following property:

Property 2 (Projection property of S_i's) *Each local specific boundary S_i must accept all the positives instances. $L(S_i)$ is thus the set of maximally specific relations (minimal w.r.t. $\leq_g$) of $L(B_i)$ that accept all E^+:*

$$L(S_i) = min_{\leq_g}\{r \in L^H(B_i) / \forall e^+ \in E^+, e^+[var(B_i)] \in r\}$$

Corollary 1 *The specific boundary S is the Cartesian product of the local specific boundaries S_i's, i.e., the set of hypotheses, where each constraint takes its relation from $L(Si)$:*

$$S = \bigtimes_{i \in 1..m} S_i$$

From Property 2, when a positive instance e^+ is presented, each local bias B_i can be processed individually (line 2 of Algorithm CONACQ). If the specific boundary of a constraint already accepts this positive instance, it is skipped (line 3), else the boundary goes up to the most specific relations of the local version space (i.e., the relations of $L^H(B_i)$ between $L(S_i)$ and $L(G_i)$) that accept e^+ (line 4). If no such relation exists, this means that no hypothesis can accept this positive instance. Then, the algorithm terminates since a collapsing state has been encountered (line 5).

Instances from $\boldsymbol{E}^-$ A negative instance e^- must be a non solution for all the networks $(\mathcal{X}, \mathcal{D}, h)$ where $h \in V$. So,

$$\forall h \in V, \exists C_i \in h / e^-[var(C_i)] \notin rel(C_i)$$

Since at least one violated constraint is sufficient for an instance to be regarded as negative, instead of all satisfied constraints necessary in the case of a positive instance, G does not have the projection property exhibited by S:[2]

[2] If this was not the case the constraint defined on $rel(B_i)$ would be sufficient to reject all negative instance of E^-.

$$L(G_i) \neq max_{\leq_g}\{r \in L^H(B_i)/\forall e^- \in E^-, e^-[var(B_i)] \notin r\}$$

We can only say that $\forall e^- \in E^-, \exists i/\forall r \in L(G_i), e^-[var(B_i)] \notin r$. However, the cause of the rejection (which constraint(s) has been violated) may not be obvious. Furthermore, storing only the local general boundaries G_i's is not sufficient to express this uncertainty.

The traditional approach to version space learning involves storing the set of all possible global boundaries G. However, this can require exponential space and time [4]. In order to ensure our algorithm remains polynomial, we do not store this set, but encode each negative instance e^- as a clause, Cl. Each constraint that could possibly be involved in the rejection of a negative example e^- will be encoded as a meta-variable in the clause Cl. Semantically, the clause Cl represents the disjunction of the possible explanations for the rejection of this negative instance. In other words, it encodes a disjunction of the constraints that could have been responsible for the inconsistency in the instance.

When a negative instance e^- is presented, a new clause, initially empty, is built by processing each local bias B_i one-by-one (lines 12-14). Those biases whose specific boundary, $L(S_i)$, already accepts the negative instance, e^-, are skipped (line 15). The reason being that S_i is the maximally specific boundary of the local version space for B_i and, by definition, we know that at each step of the learning process the constraint defined on $rel(B_i)$ cannot be involved in the rejection of e^-, since this has already been deemed acceptable by at least one positive example.

For all the other constraints, a subset of which is responsible for the rejection e^-, we compute A_i, the subset of maximally specific relations (w.r.t. $\leq_g$) between $L(S_i)$ and $L(G_i)$ which accept $e^-[var(B_i)]$, i.e. the least upper bound that B_i must not take if it is proven to be a contributor to the rejection of e^- (line 16). Depending on this set of relations we have two alternative courses of action to consider. Firstly, if the set A_i is empty it means that all possible relations for the constraint defined on $rel(B_i)$ already reject e^-. Therefore, every hypothesis in the version space is consistent with e^- so there is nothing to do for e^-; we are then ready to process the next instance (line 17). Secondly, if A_i is not empty, we add the meta-variable $(L(G_i) <_g A_i)$ to the clause Cl (line 18). The semantic of this meta-variable is *"if B_i is involved in the rejection of e^-, then G_i must be made more specific than A_i"*.

To reject e^-, a sequence of constraints h must satisfy at least one meta-variable in the clause Cl encoding e^-. We will denote this as $h \models Cl$. The set of clauses, characterizing all the set E^-, is denoted as $\mathcal{K}$. Below we will summarise the maintenance of these clauses for clarity.

Maintenance of the set of clauses

i Maintenance of clauses when positive instances are added.
If a clause Cl contains the meta-variable $(L(G_i) <_g A_i)$ and if after the processing of positive instances, $L(S_i) \cap A_i$ becomes not empty, then the relations in $L(S_i) \cap A_i$ are no longer a possible explanation for the rejection of the negative instance denoted by Cl. Let A'_i be the set of relations of A_i that are not in $L(S_i)$. If A'_i is not empty, the new explanation becomes $(L(G_i) <_g A'_i)$. Otherwise the meta-variable $(L(G_i) <_g A_i)$ is erased from Cl (lines 6-9).

Algorithm 1: The CONACQ Algorithm

```
    $\mathcal{K} \leftarrow \emptyset$
    **foreach** $B_i$ **do** $L(S_i) \leftarrow \{\bot\};\ L(G_i) \leftarrow \{\top\}$
    **foreach** *training instance e* **do**
1     **if** $e \in E^+$ **then**
2       **foreach** $B_i$ **do**
3         **if** $\exists r \in L(S_i)/e[var(B_i)] \notin r$ **then**
4           $L(S_i) \leftarrow min_{\leq_g}\{r/S_i \leq_g r \leq_g G_i$ and $e[var(B_i)] \in r\}$
5           **if** $L(S_i) = \emptyset$ **then** "collapsing"
6           **foreach** $Cl/(L(G_i) <_g A_i) \in Cl$ *and* $A_i \cap L(S_i) \neq \emptyset$ **do**
7             $Cl \leftarrow Cl \setminus (L(G_i) <_g A_i)$
8             $A'_i \leftarrow A_i \setminus L(S_i)$
9             **if** $A'_i \neq \emptyset$ **then** $Cl \leftarrow Cl \cup (L(G_i) <_g A'_i)$
10            **if** $Cl = \emptyset$ **then** "collapsing"
11            **if** $Cl = \{(L(G_i) <_g A_i)\}$ **then**
                $G_i \leftarrow max_{\leq_g}\{r/S_i \leq_g r \leq_g G_i$ and $r <_g A_i\}$ ; $\mathcal{K} \leftarrow \mathcal{K} \setminus Cl$

12    **if** $e \in E^-$ **then**
13      $Cl \leftarrow \emptyset$ ; $reject \leftarrow false$
14      **foreach** $B_i$ ***while*** $\neg reject$ **do**
15        **if** $\exists r \in L(S_i)/e[var(B_i)] \notin r$ **then**
16          $A_i \leftarrow min_{\leq_g}\{r/S_i \leq_g r \leq_g G_i$ and $e[var(B_i)] \in r\}$
17          **if** $A_i = \emptyset$ **then** $reject \leftarrow true$
18          **else** $Cl \leftarrow Cl \cup (L(G_i) <_g A_i)$
        **if** $\neg reject$ **then**
19        **if** $Cl = \emptyset$ **then** "collapsing"
20        **if** $Cl = \{(L(G_i) <_g A_i)\}$ **then**
            $G_i \leftarrow max_{\leq_g}\{r/S_i \leq_g r \leq_g G_i$ and $r <_g A_i\}$ ; $\mathcal{K} \leftarrow \mathcal{K} \setminus Cl$
21        **else if** $\nexists Cl'/Cl' \subseteq Cl$ **then** $\mathcal{K} \leftarrow \mathcal{K} \cup Cl$
22        **foreach** $Cl''/Cl \subset Cl''$ **do** $\mathcal{K} \leftarrow \mathcal{K} \setminus Cl''$
```

ii Empty clause implies that the version space has collapsed.

When a clause Cl, encoding a negative instance e^-, is empty, either during its construction (line 19), or following the addition of some positive instances (line 10), it implies that there does not exist a possible explanation for rejecting the negative instance e^-, then the algorithm collapses.

iii The one-meta-variable clauses.

The one-meta-variable clauses represent the only possible explanation for the rejection of the coded negative instances. They must be reported on the local version space then removed from $\mathcal{K}$. Let $(L(G_i) <_g A_i)$ be a such clause: $L(G_i)$ must be made more specific than A_i.(lines 11 and 20).

iv Subsumption of negative instances.

A negative instance e_1^- (encoded by Cl_1) subsumes e_2^- (encoded by Cl_2) iff $Cl_1 \subseteq Cl_2$. Indeed, if Cl_1 is satisfied, then Cl_2 too. It is thus useless to store Cl_2. Lines 21-22 of Algorithm CONACQ maintain this minimality of the base of clauses $\mathcal{K}$.

Example 2 In Figure 2, we describe the learning process on a single constraint C_{12} and where B_{12} is the bias of Figure 1. Initially $L(S_{12}) = \{\bot\}$, and $L(G_{12}) = \{\top\}$. When the positive instance $e_1^+ = (1,1)$ is received, $L(S_{12})$ goes up in $L^H(B_{12})$ to the most specific relations accepting this tuple (a), namely $\{=\}$. We know now that the final relation of constraint C_{12} will be more general than or equal to "$=$". When the negative instance $e_2^- = (2,1)$ is received, it is necessary to restrict $L(G_{12})$ such that it will reject this tuple (b). At this step: $L(S_{12}) = \{=\}$, and $L(G_{12}) = \{\leq\}$. The negative instance $e_3^- = (0,3)$ forbids the relation "$\leq$" (c). Then $L(S_{12}) = L(G_{12}) = \{=\}$, that is a local convergence.

E	X_1	X_2
e_1^+	1	1
e_2^-	2	1
e_3^-	0	3

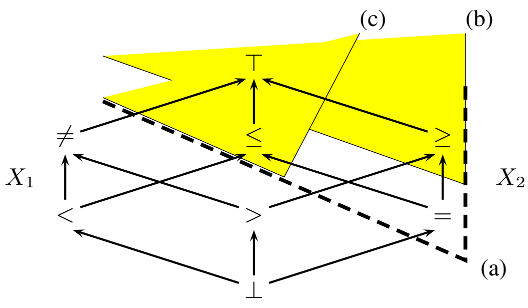
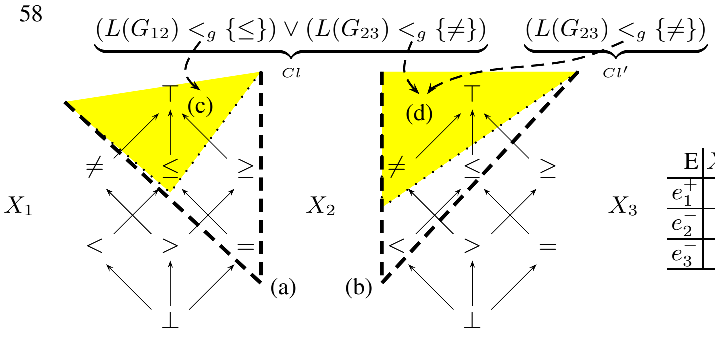


Fig. 2. A local example

Example 3 In Figure 3, we present an example of our algorithm using a constraint network involving three variables X_1, X_2 and X_3. The biases used are $\{B_{12}, B_{23}\}$, where $var(B_{12}) = (X_1, X_2)$, $var(B_{23}) = (X_2, X_3)$ and $L^H(B_{12}) = L^H(B_{23})$, which for the purposes of this example we will assume to be the same as that presented as $L^H(B_i)$ in Figure 1. When processing the positive example $e_1^+ = (2,2,5)$, due to the projection property of S, S_{12} **and** S_{23} go up (i.e., $L(S_{12}) \leftarrow \{=\}$ and $L(S_{23}) \leftarrow \{<\}$) (a) and (b).

However, since G does not have this projection property: when $e_2^- = (1,3,2)$ is received, either C_{12} must reject the tuple $(1,3)$ **or** C_{23} must reject $(3,2)$. Therefore, we build the clause $Cl = (rel(C_{12}) <_g^L \{\leq\}) \vee (rel(C_{23}) <_g^L \{\neq,\})$ to store these alternatives (c) or (d). When the negative instance $e_3^- = (1,1,0)$ is received, we know that the constraint C_{12} is not involved in its rejection, because $e_3^-[var(C_{12})] = (1,1)$ is an allowed tuple of $L(S_{12}) = \{=\}$. The only explanation for the rejection of e_3^- is $Cl' = (rel(C_{23}) <_g^L \{\neq,\})$ (d). Note than Cl' subsumes Cl (i.e., $Cl' \subseteq Cl$). The explanation of the rejection of e_3^- (d) is also a valid one for e_2^-: Cl becomes subsumed by Cl', and thus is discarded (the explanation (c) is discarded too). After these three instances, the CONACQ result is $L(S_{12}) = \{=\}$, $L(G_{12}) = \{\top\}$, C_{23}: $L(S_{23}) = \{<\}$ and $L(G_{23}) = \{\leq\}$.



$(L(G_{12}) <_g \{\leq\}) \vee (L(G_{23}) <_g \{\neq\})$
cl
$(L(G_{23}) <_g \{\neq\})$
cl'
$\top$
(c)
(d)
$\neq$
$\leq$
$\geq$
$<$
$>$
$=$
$\bot$
X_1
X_2
X_3
(a)
(b)

E	X_1	X_2	X_3
e_1^+	2	2	5
e_2^-	1	3	2
e_3^-	1	1	0

Fig. 3. A global example

6.2 Correctness of Algorithm CONACQ

The theorem of representability [7] says that if cn the target network belongs to H (the hypothesis space) and if S and G are the minimal and maximal boundaries of the version space wrt the generalisation order $\leq_G$, then at every step of training, cn is such that $\exists s \in S, \exists g \in G / s \leq_G cn \leq_G g$.

To ensure the correctness of Algorithm CONACQ, we have only to prove that S and G are the minimal and maximal boundaries of V. We just give a sketch of the proof. For S, this is easy to prove because of the projection property of S. The global S is the union of the local S_i's. And by construction, each local S_i is the minimal boundary of V on $var(S_i)$. For G, this is more complex because G is not equivalent to the union of its local projections. Remember that because of its space complexity, we do not store G in extension but only its clause characterisation $\mathcal{K}$. To be in G, an hypothesis h must reject all negative instances, i.e., $h \models \mathcal{K}$. It is an implicant[3] of $\mathcal{K}$. In addition, h must be as general as possible wrt $\leq_G$, which means that it satisfies as few meta-variables as possible. It is thus a prime implicant of $\mathcal{K}$:

$$G = \{g \in \bigtimes_{i \in 1..m} G_i / g \models^\star \mathcal{K}\}$$

Now, an hypothesis $h \in G$ must be such that $E^- \cap Sol(\mathcal{X}, \mathcal{D}, h) = \emptyset$, and such that $\nexists h' \neq h / h \leq_G h'$ and $E^- \cap Sol(\mathcal{X}, \mathcal{D}, h') = \emptyset$. Suppose there exists such a h'. Since it rejects all the negative instances, we have $h' \models \mathcal{K}$ by construction of $\mathcal{K}$. And then, h would not be a prime implicant, which contradicts the assumption.

Since our S and G have the required properties, the following corollary applies:

Corollary 2 ([7]) *Given a target network cn and a space H of hypotheses, the version space collapses iff $cn \notin H$, which means that the selected bias is not sufficient.*

7 Experiments and Observations

We report here on some preliminary experiments to evaluate the learning capabilities of our approach. Rather than focusing on techniques for minimising the number of inter-

[3] A term i is an implicant of a set of clauses B iff $i \models B$. It is a prime implicant, noted $i \models^\star B$, iff for each other implicant i', $i' \not\subset i$.

actions, our focus here is on studying a number of properties of the CONACQ algorithm which provide motivation for our research agenda.

We performed experiments with a simulated teacher, which plays the role of the user, and a simulated learner that uses the algorithm presented earlier. The teacher has the knowledge of a randomly generated (target) network, represented by the triple $< 50, 8, C >$, defining a problem involving 50 variables with domains $\{1, ..8\}$, and a number C of constraints. Each constraint is randomly chosen from the bias $\{<, =, >, \leq, \neq, \geq\}$. The teacher provides the learner with solutions and non solutions. The learner acquires a version space for the problem using CONACQ algorithm.

7.1 Experiment 1: Effect of the order of the instances

In this following experiment, we assess aspects of the runtime characteristics of the CONACQ algorithm. In particular, we study computing time and the size of the version space, while varying the order in which examples are presented. Instances from a set E of size 100 are given by the teacher to the learner based on a $< 50, 8, 50 >$ network. The set E contains 10 positive and 90 negative instances.

Table 1 presents the time needed by the learner to acquire the version space, V, for the example set while varying the arrival time of the 10 positive instances. The positive instances were presented at the beginning (a), middle (b), and end (c) of the interaction between teacher and learner.

Table 1. Effect of the timing of the introduction of positive instances

Introduction time for positives	0 (a)	50 (b)	90 (c)
Computing time (in sec.)	3.3	5.1	8.6
$log(\|V\|)$	2,234	2,234	2,234

We observe that *"the sooner, the better"* seems to be the good strategy for the introduction of positive instances. Indeed, the specific bound S rises quickly in the space of hypotheses with positive instances, reducing the size of the version space. Because of that, the CPU time needed is also reduced when positive instances arrive at the beginning. But we can see that the final size of the version space is not affected by the order of the instances. This is due to the commutativity property of version spaces, discussed in Section 5.

7.2 Experiment 2: Utility of partial instances

In some cases, the user can reject an instance while justifying it by a negative sub-instance. For example, in a real-estate setting the customer (teacher) might reject an apartment citing the reason that *"this living-room is too small for me"*. The estate agent (learner) knows that the violation is due to the variables defining the living room, which can being very helpful for handling negative examples. The utility of such justified rejections can be measured by providing our learner with partial instances. In the following experiment (Table 2), the teacher provides the learner with 90 partial negative instances (after 10 complete positive ones) in the training data. We consider partial instances involving 2, 5, 10 variables, and report the size of the version space and of the set of clauses after 100 instances have been given.

Table 2. Effect of the partial instances

Nb of variables involved in instances of E^-	50	10	5	2
$log(\|V\|)$	2,234	2,233	2,225	2,144
$\|K\|$ (10^4 meta-variables)	7.6	6.1	3.2	0

We observe that partial instances speed up the process of convergence of the version space. The smaller these partial instances are, the more helpful they are. This opens a promising way of helping the learning process: asking the user to justify why she rejects some instances can assist in reducing the length of the dialog with the teacher. This is a critical issue if we are learning in an interactive setting from a human user.

7.3 Experiment 3: Non-representable constraints and version space collapse

We have seen that our algorithm learns a network expressed using a given bias. An overly restrictive bias leads to version space collapse since it is unlikely to be capable of expressing the target network. This happens because some of the constraints of the target network are non-representable in the given bias. In the following experiment, (Table 3), we analyse the effect on the speed of version space collapse by varying the proportion of non-representable constraints in the target network. For each proportion of non-representable constraints we learned 100 different $< 50, 8, 50 >$ target networks. Non-representable constraints, generated randomly, were not members of our bias $\{<, =, >, \leq, \neq, \geq\}$. The table presents the number of times the version space collapsed after 1,000 instances where presented. Note that the focus of this experiment is related to automatic reformulation rather than interactive modelling.

Table 3. Effect of non-representable constraints

Ratio of non representable constraints	10%	25%	50%
% of collapsing	47	78	100

We observe that the more non-representable constraints in the target network, the faster the version space collapses. This is to be expected since version spaces are sensitive to noise (errors) in the training data. However, an interesting observation was made during this analysis. On a number of constraint networks containing very few non-representable constraints we observed that the version space did not collapse, even after millions of instances were presented. This seems to violate Corollary 2. However, what occured in these situations was that the non-representable constraints in the target *become* representable by propagation of other constraints. As an example, consider three variables X_1, X_2, and X_3 linked by the following constraints in the target network: $X_1 = X_2$, $X_2 = X_3$, and $(X_1 = X_3) \vee (X_1 = X_3 + 5)$. Obviously, the constraint between X_1 and X_3 is not representable in the bias $\{<, =, >, \leq, \neq, \geq\}$, and every network containing it is expected to cause the version space to collapse. However, this is not the case in this example, because we can restrict the constraint between X_1 and X_3 to $X_1 = X_3$ by transitivity. This transitive closure constraint is representable.

It is worth noting that if we had used a technique that learned each constraint in the network separately, the version space for our problem would have collapsed had we attempted to acquire the constraint between X_1 and X_3.

7.4 Observation: Implicit constraints

The general phenomenon of constraints that can be inferred by other constraints can have another effect, which is to prevent the version space from converging to the smallest possible on each constraint taken separately. Consider again an example with three variables X_1, X_2 and X_3, linked in the target network by the constraints $X_1 = X_2$, $X_2 = X_3$, and $X_1 = X_3$. Furthermore, consider what occurs at some point in the learning process when the version space local to the constraint defined on (X_1, X_3) contains both $X_1 = X_3$ and $X_1 \leq X_3$. If the training data contains only complete instances, it is impossible to converge to the constraint $X_1 = X_3$ because every negative instance that would permit us to discard $X_1 \leq X_3$ from the version space (e.g., $((X_1, 1), (X_2, 2), (X_3, 2))$) will also be rejected by $X_1 = X_2$ or by $X_2 = X_3$. Thus, the boundary G will never determine that culpability lies with $X_1 \leq X_3$.

Applying some levels of local consistency seems to be a promising approach to improving the reduction of the version space, by adding implicit constraints to the learned network. In the previous example, path-consistency would be enough to deduce that the only candidate between X_1 and X_3 is the constraint $X_1 = X_3$. This phenomenon can cause the VS to keep a size bigger than it should have.

8 Aspects of our Research Agenda

In this paper we have presented an approach to acquiring models of constraint satisfaction problems from examples. There is significant scope for research in this area. Here we give some insights into some aspects of our research in this area.

Standard version space learning algorithms are senstive to noise in the training data and, as a consequence, are brittle to false positives and negatives provided to the algorithm. However, some recent work from the machine learning community gives us a basis for making our approach more robust to such errors [6].

Another issue, and one which is particularly interesting, is that of implicit constraints and redundancy which has already been discussed in Section 7. When we deal with partial instances, this will have some interesting implications, such as the effect that the order in which examples are provided has on the representability of a particular problem in the given constraint language.

Finally, we are considering the effect that various models of interaction can have on the speed with which we can learn the target problem, particularly from the perspective of minimising the number of interactions with the user. Some preliminary results have already been reported on this issue [10, 11].

9 Conclusion

We have proposed an original method to learn constraint networks from instances that should or should not be solutions. The technique used is based on version spaces, a machine learning paradigm that has good properties (e.g., incrementality, commutativity, wrt the training data) that will be essential in a process interacting with a user. Even if this paper is mainly a description of the general process of learning a constraint network from instances, we can easily foresee the many applications it could have, in assisting a novice in modelling her problem, or helping an expert to test whether a given library of constraints with good computational properties can encode her problem.

We have presented preliminary experiments that show that our approach raises several important issues, such as the speed of the learning process, or the question of the implicit constraints. Based on these experiments, and the framework in general, we have given an insight into some of the very interesting research issues which are raised by our work.

References

1. E.C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, Jan. 1982.
2. E.C. Freuder and B. O'Sullivan. Generating tradeoffs for interactive constraint-based configuration. In Toby Walsh, editor, *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming*, pages 590–594, November 2001.
3. E.C. Freuder and R.J. Wallace. Suggestion strategies for constraint-based matchmaker agents. In *Principles and Practice of Constraint Programming - CP98*, pages 192–204, October 1998.
4. Haym Hirsh. Polynomial-time learning with version spaces. In *National Conference on Artificial Intelligence*, pages 117–122, 1992.
5. J. Little, C. Gebruers, D. Bridge, and E.C. Freuder. Capturing constraint programming experience: A case-based approach. In *CP-02 Workshop on Reformulating Constraint Satisfaction Problems*, 2002.
6. F.A. Marginean. Soft learning: A bridge between data mining and machine learning. In *Proceedings of the 4th International Conference on Recent Advances in Soft Computing (RASC-2002)*, pages 108–115, December 2002.
7. T. Mitchell. Concept learning and the general-to-specific ordering. In *Machine Learning*, chapter 2, pages 20–51. McGraw Hill, 1997.
8. U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7(95-132), 1974.
9. S. O'Connell, B. O'Sullivan, and E.C. Freuder. Query generation for interactive constraint acquisition. In *Proceedings of the 4th International Conference on Recent Advances in Soft Computing (RASC-2002)*, pages 295–300, December 2002.
10. S. O'Connell, B. O'Sullivan, and E.C. Freuder. Strategies for interactive constraint acquisition. In *Proceedings of the CP-2002 Workshop on User-Interaction in Constraint Satisfaction*, pages 62–76, September 2002.
11. B. O'Sullivan, S. O'Connell, and E.C. Freuder. Interactive constraint acquisition for concurrent engineering. In *Proceedings of the 7th International Conference on Concurrent Enterprising – ICE-2003*, 2003.
12. S. Padmanabhuni, J.-H. You, and Ghose A. A framework for learning constraints. In *Proceedings of the PRICAI Workshop on Induction of Complex Representations*, August 1996.
13. C.D. Page and A.M. Frisch. Generalization and learnability: A study of constrained atoms. In S.H.. Muggleton, editor, *Inductive Logic Programming*, pages 29–61. Academic Press, 1992.
14. F. Rossi and A. Sperduti. Learning solution preferences in constraint problems. *Journal of experimental and theoretical computer science*, 10, 1998.
15. P. van Beek and R. Dechter. On the minimality and global consistency of row-convex constraint networks. *Journal of the ACM*, 42(3):543–561, 1995.
16. M. Wallace. Practical applications of constraint programming. *Constraints*, 1(1–2):139–168, 1996.

Introducing ESRA, a Relational Language for Modelling Combinatorial Problems

Pierre Flener, Justin Pearson, and Magnus Ågren [⋆] [⋆⋆]

Department of Information Technology
Uppsala University, Box 337, S – 751 05 Uppsala, Sweden
{pierref,justin,agren}@it.uu.se

Abstract. Current-generation constraint programming languages are considered by many, especially in industry, to be too low-level, difficult, and large. We argue that solver-independent, high-level relational constraint modelling leads to a simpler and smaller language, to more concise, intuitive, and analysable models, as well as to more efficient and effective model formulation, maintenance, reformulation, and verification, and all this without sacrificing the possibility of efficient solving, so that even lazy or less competent modellers can be well assisted. Towards this, we propose the ESRA relational constraint modelling language, showcase its elegance on some well-known problems, and outline a compilation philosophy for such languages.

1 Introduction

Current-generation constraint programming languages are considered by many, especially in industry, to be too low-level, difficult, and large. Consequently, their solvers are not in as widespread use as they ought to be, and constraint programming is still fairly unknown in many application domains, such as molecular biology. In order to unleash the proven powers of constraint technology and make it available to a wider range of problem modellers, a higher-level, simpler, and smaller modelling notation is needed.

In our opinion, even recent commercial languages such as OPL [20] do not go far enough in that direction. Many common modelling patterns have not been captured in special constructs. They have to be painstakingly spelled out each time, at a high risk for errors, often using low-level devices such as reification.

There is much to be learned from formal methods and semantic modelling. In recent years, modelling languages based on some logic with sets and relations have gained popularity in formal methods, witness the B [1] and Z [18] specification languages, the ALLOY [11] object modelling language, and the *Object Constraint Language* (OCL) [23] of the *Unified Modelling Language* (UML) [16]. In semantic data modelling this had been long advocated; most notably via entity-relationship-attribute (ERA) diagrams.

[⋆] The authors' names are ordered according to the Swedish alphabet.
[⋆⋆] An extended abstract of this paper appears in the pre-proceedings of LOPSTR'03.

Sets and set expressions recently started appearing as modelling devices in some constraint programming languages. Set variables are often implemented by the set interval representation [8]. In the absence of such an explicit set concept, modellers usually painstakingly represent a set variable as a sequence of 0/1 integer variables, as long as the domain of the set.

Relations have not received much attention yet in constraint programming languages, except the particular case of a total function via the concept of array. Indeed, a total function f can be represented in many ways, say as a 1D array of variables over the range of f, indexed by its domain, or as a 2D array of 0/1 variables, indexed by the domain and range of f, or even with some redundancy. Other than retrieving the (unique) image under a total function of a domain element, there has been no support for relational expressions.

Matrix modelling [5] has been advocated as one way of capturing common modelling patterns. Alternatively, it has been argued [6, 10] that functions, and hence relations, should be supported by an abstract datatype (ADT). It is then *the compiler* that must (help the modeller) choose a suitable representation, say in a contemporary constraint programming language, for each instance of the ADT, using empirically or theoretically gained modelling insights. We here claim, as in [4], that a suitable first-order relational calculus is a good basis for a high-level ADT-based constraint modelling language. It gives rise to very natural and easy-to-maintain models of combinatorial problems. Even in the (temporary) absence of a corresponding high-level search language, this generality does not necessarily come at a loss in solving efficiency, as high-level relational models are devoid of representation details so that the results of analysis can be exploited.

Our aims here are only to justify and present our new language, called ESRA, to illustrate its elegance and the flexibility of its models by some examples, and to argue that it can be compiled into efficient models in lower-level constraint programming languages. The syntax, denotational semantics, and type system of the proposed language are discussed in full detail in an online appendix [7] and a prototype of the advocated compiler is currently under implementation.

The rest of this paper is organised as follows. In Section 2, we present our relational language for modelling combinatorial problems and deploy it on three real-life problems, before discussing its compilation. This allows us to list, in Section 3, the benefits of relational modelling. Finally, in Section 4, we conclude as well as discuss related and future work.

2 Relational Modelling with ESRA

In Section 2.1, we justify the design decisions behind our new ESRA modelling language. Then, in Section 2.2, we introduce its concepts, syntax, type system, and semantics. Next, in Section 2.3, we deploy ESRA on three real-life problems. Finally, in Section 2.4, we discuss the design of our prototype compiler for ESRA.

2.1 Design Decisions

The key design decisions for our new relational constraint modelling language — called ESRA for *Executable Symbolism for Relational Algebra* — were as follows.

We want to capture common modelling idioms in a new abstract datatype for relations, so as to design a high-level and simple language. The constructs of the language must be orthogonal, so as to keep the language small. Computational completeness is not aimed at, as long as the language is useful for elegantly modelling a large number of combinatorial problems.

We focus on *finite*, discrete domains. Relations are built from such domains and sets are viewed as unary relations. Theoretical difficulties are sidestepped by supporting only bounded quantification, but no negation and no sets of sets.

The language has an ASCII syntax, mimicking mathematical and logical notation as closely as possible, as well as a LaTeX-based syntax, especially used for pretty-printing models in that notation.

2.2 Concepts, Syntax, Type System, and Semantics of ESRA

For reasons of space, we only give an informal semantics. The interested reader is invited to consult [7] for a complete description of the language. Code examples are provided out of the semantic context of any particular problem statement, just to illustrate the syntax, but a suggested reading in plain English is always provided. In Section 2.3, we will actually start from plain English problem statements and show how they can be modelled in ESRA. Code excerpts are always given in the pretty-printed form, but we indicate the ASCII notation for every symbol where it necessarily differs. An ESRA model starts with a sequence of declarations of named *domains* (or types) as well as named *constants* and *decision variables* that are tied to domains. Then comes the *objective*, which is to find values for the decision variables within their domains so that some *constraints* are satisfied and possibly some *cost function* takes an optimal value.

The Type System. A *primitive domain* is a finite, extensionally given set of new names or integers, comma-separated and enclosed as usual in curly braces. An integer domain can also be given intensionally as a finite integer interval, by separating its lower and upper bounds with '$\ldots$' (denoted in ASCII by '`..`'), without using curly braces. When these bounds coincide, the corresponding singleton domain $n \ldots n$ or $\{n\}$ can be abbreviated to n. Context always determines whether an integer n designates itself or the singleton domain $\{n\}$. A domain can also be given intensionally using set comprehension notation.

The only *predefined* primitive domains are the sets $\mathbb{N}$ (denoted in ASCII by '`nat`') and $\mathbb{Z}$ (denoted in ASCII by '`int`'), which are $0 \ldots \mathrm{sup}$ and $\mathrm{inf} \ldots \mathrm{sup}$ respectively, where the predefined constant identifiers 'inf' and 'sup' stand for the smallest negative and largest positive representable integers respectively. *User-defined* primitive domains are declared after the 'dom' keyword and initialised in-line, using the '=' symbol, or at run-time, via a datafile, otherwise interactively.

Example 1. The declaration

$$\mathrm{dom}\ \mathit{Varieties}, \mathit{Blocks}$$

declares two domains called *Varieties* and *Blocks* that are to be initialised at run-time. Similarly, the declaration

$$\mathrm{dom}\ \mathit{Players} = 1 \ldots g * s,\ \ \mathit{Weeks} = 1 \ldots w,\ \ \mathit{Groups} = 1 \ldots g$$

where g, s, w are integer-constant identifiers (assumed previously declared, in a way shown below), declares integer domains called *Players*, *Weeks*, and *Groups* that are initialised in-line. Finally, the declaration

$$\text{dom } Even = \{i \mid i \in 0 \ldots 100 \mid i \ \% \ 2 = 0\}$$

declares and initialises the domain *Even* of all even natural numbers up to 100.

The usual binary infix $\times$ domain constructor (denoted in ASCII by '`#`') allows the construction of Cartesian products, so that relations can be declared of this *constructed domain*. Consider the relation domain $A \times B$; then A and B must be domains, designating the participant sets of any relation in $A \times B$.

In order to capture frequently occurring multiplicity constraints on relations, we offer a parameterised binary infix $\times$ domain constructor. Consider the relation domain $A \ {}^{M_1}\!\times^{M_2} \ B$. The conditions on A and B are as above. The optional M_1 and M_2, called *multiplicities*, must be integer sets and have the following semantics: for every element a of A, the number of elements of B related to a must be in M_1, while for every element b of B, the number of elements of A related to b must be in M_2.[1] An omitted multiplicity stands for $\mathbb{N}$.

Example 2. The domain '*Varieties* ${}^{r}\!\times^{k}$ *Blocks*' designates the set of all relations in *Varieties* $\times$ *Blocks* where every variety occurs in exactly r blocks and every block contains exactly k varieties. These are also two examples where an integer abbreviates the singleton domain containing it.

In the absence of such facilities for relations and their multiplicities, a relation domain would have to be declared using arrays, say. This may constitute a premature commitment to a concrete data structure, as the modeller may not know yet, especially prior to experimentation, which particular (array-based) representation of a relation decision variable will lead to the most efficient solving. The problem constraints, including the multiplicities, would have to be enforced further down in the model, based on the chosen representation. If the experiments revealed that another representation should be tried, the modeller would have to first painstakingly rewrite the declaration of the decision variable as well as all the constraints on it. Our ADT view of relations overcomes this flaw; it is now *the compiler* that must (help the modeller) choose a suitable representation for each instance of the ADT by using empirically or theoretically gained modelling insights. Furthermore, multiplicities need not become counting constraints, but are succinctly and conveniently captured in the declaration.

We view sets as unary relations. So $A \ M$, where A is a domain and M an integer set, constructs the domain of all subsets of A whose cardinality is in M.

For total and partial functions the left-hand multiplicity M_1 is $1 \ldots 1$ and $0 \ldots 1$ respectively. In order to dispense with these left-hand multiplicities for total and partial functions, we offer the usual $\longrightarrow$ and $\not\longrightarrow$ (denoted in ASCII

[1] Note that our syntax is the opposite of the UML one, say, where the multiplicities are written in the other order, with the *same* semantics. That convention can however *not* be usefully upgraded to Cartesian products of arity higher than 2.

by '`->`' and '`+>`') domain constructors respectively, as shorthands. They may still have right-hand multiplicities though.

For injections, surjections, and bijections, the right-hand multiplicity M_2 is $0 \dots 1$, $1 \dots \sup$, and $1 \dots 1$ respectively. Rather than elevating these particular cases of functions to first-class concepts with an invented specific syntax in ESRA, we prefer keeping our language lean and close to mathematical notation.

Example 3. The domain '$(\mathit{Players} \times \mathit{Weeks}) \longrightarrow^{s*w} \mathit{Groups}$' designates the set of all total functions from $\mathit{Players} \times \mathit{Weeks}$ into *Groups* such that every group is related to exactly sw player-week pairs. Note the nesting in this domain: the Cartesian product $\mathit{Players} \times \mathit{Weeks}$ is the left-hand argument of the outer Cartesian product.

We provide no support for multisets and sequences. Note that a *multiset* can be modelled as a total function from its domain into $\mathbb{N}$, giving the multiplicity of each element. Similarly, a *sequence* of length n can be modelled as a total function from $1 \dots n$ into its domain, telling which element is at each position. This does *not* mean that the representation of multisets and sequences is fixed (to the one of total functions), because, as we shall see in Section 2.4, the relations (and thus total functions) of a model need not have the same representation.

Modelling the Instance Data and Decision Variables. All declarations are strongly typed in ESRA. All identifier declarations denote variables that are universally quantified over the entire model, with the constants expected to be ground before search begins while the decision variables can still be unbound.

Like the user-defined primitive domains, constants help describe the instance data of a problem. A constant identifier is declared after the 'cst' keyword and is tied to its domain by '$\in$' (denoted in ASCII by '`in`'), meaning set membership, or by ':' (which is often used in mathematics and logic for '$\subseteq$'), meaning set inclusion. Constants are initialised in-line, using the '$=$' symbol, or at run-time, via a datafile, otherwise interactively. Run-time initialisation provides a neat separation of problem models and problem instances.

Example 4. The declaration 'cst $r, k, \lambda \in \mathbb{N}$' declares three natural number constants that are to be initialised at run-time. As already seen in Example 2, the availability of total functions makes arrays unnecessary. The declaration

$$\text{cst } \mathit{CrewSize} : \mathit{Guests} \longrightarrow \mathbb{N},\ \mathit{SpareCapacity} : \mathit{Hosts} \longrightarrow \mathbb{N}$$

declares that the given crew sizes of the guests as well as the given spare capacities of the hosts are natural numbers, to be provided at run-time.

A decision-variable identifier is declared after the 'var' keyword and is tied to its domain by ':' or '$\in$'.

Example 5. The declaration 'var $\mathit{BIBD} : \mathit{Varieties} \ {}^{r}\!\times^{k} \ \mathit{Blocks}$' declares a relation called *BIBD* of the domain of Example 2. Replacing the : by $\in$ would rather declare a particular *pair* of that domain.

Modelling the Cost Function and the Constraints. *Expressions* and first-order logic *formulas* are constructed in the usual way.

For *numeric expressions* the arguments are either integers or identifiers of the domain $\mathbb{N}$ or $\mathbb{Z}$, including the predefined constants 'inf' and 'sup'. Usual unary ($-$, 'abs' for absolute value, and 'card' for the cardinality of a set expression), binary infix ($+$, $-$, $*$, $/$ for integer quotient, and $\%$ for integer remainder), and aggregate ($\sum$, denoted in ASCII by '`sum`') arithmetic operators are available. A sum is indexed by local variables ranging over finite sets, which may be filtered on-the-fly by a condition given after the '$|$' symbol (read 'such that').

Sets obey the same rules as domains. So, for *set expressions*, the arguments are either (intensionally or extensionally) given sets or set identifiers, including the predefined sets $\mathbb{N}$ and $\mathbb{Z}$. Only the binary infix domain constructor $\times$ and its specialisations $\longrightarrow$ and $\not\longrightarrow$ are available as operators.

Finally *function expressions* are built by applying a function identifier to an argument tuple. We have found no use yet for any other operators (but see the discussion of future work in Section 4).

Example 6. The numeric expression

$$\sum_{g \in Guests \ | \ Schedule(g,p)=h} \mathit{CrewSize}(g)$$

denotes the sum of the crew sizes of all the guests that are scheduled to visit host h at period p, assuming this expression is within the scope of the local variables h and p. The nested function expression $\mathit{CrewSize}(g)$ stands for the size of the crew of guest g, which is a natural number according to Example 4.

Atoms are built from numeric expressions with the usual comparison predicates, such as the binary infix $=$, $\neq$, and $\leq$ (denoted in ASCII by `=`, `!=`, and `=<` respectively). Atoms also include the predefined 'true' and 'false', as well as references to the elements of a relation. We have found no use yet for any other predicates. Note that '$\in$' is unnecessary as $x \in S$ is equivalent to $S(x)$.

Example 7. The atom $\mathit{BIBD}(v_1, i)$ stands for the truth value of variety v_1 being related to block i in the BIBD relation of Example 5.

Formulas are built from atoms. The usual binary infix connectives ($\wedge$, $\vee$, $\Rightarrow$, $\Leftarrow$, and $\Leftrightarrow$, denoted in ASCII by '`/\`', '`\/`', '`=>`', '`<=`', and '`<=>`' respectively) and quantifiers ($\forall$ and $\exists$, denoted in ASCII by '`forall`' and '`exists`' respectively) are available. A quantified formula is indexed by local variables ranging over finite sets, which may be filtered on-the-fly by a condition given after the '$|$' symbol (read 'such that'). As we provide a rich (enough) set of predicates, models can be formulated positively, making the negation connective unnecessary. The usual typing and precedence rules for operators and connectives apply. All binary operators associate to the left.

Example 8. The formula

$$\forall(p \in \mathit{Periods},\ h \in \mathit{Hosts}) \left(\sum_{g \in Guests \ | \ Schedule(g,p)=h} \mathit{CrewSize}(g) \right) \leq \mathit{SpareCapacity}(h)$$

constrains the spare capacity of any host boat h not to be exceeded at any period p by the sum of the crew sizes of all the guest boats that are scheduled to visit host boat h at period p.

A generalisation of the $\exists$ quantifier turns out to be very useful. We define

$$\mathrm{count}(\mathit{Multiplicity})(x \in \mathit{Set} \mid \mathit{Condition})$$

to hold if and only if the cardinality of the set comprehension $\{x \in \mathit{Set} \mid \mathit{Condition}\}$ is in the integer set *Multiplicity*. So '$\exists(x \in \mathit{Set} \mid \mathit{Condition})$' is actually syntactic sugar for '$\mathrm{count}(1 \ldots \mathrm{sup})(x \in \mathit{Set} \mid \mathit{Condition})$'.

Example 9. The formula

$$\forall(v_1 < v_2 \in \mathit{Varieties})\ \mathrm{count}(\lambda)(i \in \mathit{Blocks} \mid \mathit{BIBD}(v_1, i) \land \mathit{BIBD}(v_2, i))$$

says that each pair of ordered varieties v_1 and v_2 occurs together in exactly λ blocks, via the *BIBD* relation. Regarding the excerpt '$v_1 < v_2 \in \mathit{Varieties}$', note that multiple local variables can be quantified at the same time, and that a condition on them may then be pushed forward in the usual way.

Example 10. Recalling from Ex. 3 that *Schedule* returns groups, the formula

$$\forall(p_1 < p_2 \in \mathit{Players})\ \mathrm{count}(0 \ldots 1)(v \in \mathit{Weeks} \mid \mathit{Schedule}(p_1, v) = \mathit{Schedule}(p_2, v))$$

says that there is at most one week where any two ordered players p_1 and p_2 are scheduled to play in the same group.

A *cost function* is a numeric expression that has to be optimised. The *constraints* on the decision variables of a model are a conjunction of formulas, using $\land$ as the connective. The *objective* of a model is either to solve its constraints:

$$\text{solve } \mathit{Constraints}$$

or to minimise the value of its cost function subject to its constraints:

$$\text{minimise } \mathit{CostFunction} \text{ such that } \mathit{Constraints}$$

or similarly for maximising. A *model* consists of a sequence of domain, constant, and decision-variable declarations followed by an objective, without separators.

Example 11. Putting together code fragments from Examples 1, 4, 5, and 9, we obtain the model of Figure 2 two pages ahead, discussed in Section 2.3.

The grammar of ESRA is described in Figure 1. For brevity and ease of reading, we have omitted most syntactic sugar options as well as the rules for identifiers, names, and numbers. The notation $\langle nt \rangle^{s^*}$ stands for a sequence of zero or more occurrences of the non-terminal $\langle nt \rangle$, separated by symbol s. Similarly, $\langle nt \rangle^{s^+}$ stands for one or more occurrences of $\langle nt \rangle$, separated by s. The type rules ensure that the equality predicates $=$ and $\neq$ are only applied to expressions of the same type, that the other comparison predicates, such as $\leq$, are only applied to numeric expressions, and so on. Only one feature of the language has not been described yet, namely projections. We prefer doing so in the semantic context of the Progressive Party problem in Section 2.3.

```
⟨Model⟩ ::= ⟨Decl⟩^+ ⟨Objective⟩

⟨Decl⟩ ::= ⟨DomDecl⟩ | ⟨CstDecl⟩ | ⟨VarDecl⟩

⟨DomDecl⟩ ::= dom ⟨Id⟩ [ = ⟨Set⟩ ]

⟨CstDecl⟩ ::= cst ⟨Id⟩ [ = ⟨Tuple⟩ | ⟨Set⟩ ] ( in | : ) ⟨SetExpr⟩

⟨VarDecl⟩ ::= var ⟨Id⟩ ( in | : ) ⟨SetExpr⟩ ⟨ProjClause⟩^/\*

⟨ProjClause⟩ ::= where ⟨Id⟩( ( ⟨Set⟩ | _ ),^+ ) : ⟨SetExpr⟩

⟨Objective⟩ ::= solve ⟨Formula⟩
             |  ( minimise | maximise ) ⟨NumExpr⟩ such that ⟨Formula⟩

⟨Expr⟩ ::= ⟨Id⟩ | ⟨Name⟩ | ⟨Tuple⟩ | ⟨NumExpr⟩ | ⟨SetExpr⟩ | ⟨FuncAppl⟩ | ( ⟨Expr⟩ )

⟨NumExpr⟩ ::= ⟨Id⟩ | ⟨Int⟩ | ⟨Nat⟩ | inf | sup | ⟨FuncAppl⟩
           |  ⟨NumExpr⟩ ( + | - | * | / | % ) ⟨NumExpr⟩
           |  ( - | abs ) ⟨NumExpr⟩
           |  card ⟨SetExpr⟩
           |  sum ( ⟨QuantExpr⟩ ) ( ⟨NumExpr⟩ )

⟨SetExpr⟩ ::= ⟨Set⟩ | ⟨SetExpr⟩ [⟨Set⟩]
           |  ⟨SetExpr⟩ ( [[⟨Set⟩]#[⟨Set⟩]] | # ) ⟨SetExpr⟩
           |  ⟨SetExpr⟩ ( [->[⟨Set⟩]] | -> | [+>[⟨Set⟩]] | +> ) ⟨SetExpr⟩

⟨Set⟩ ::= ⟨Id⟩ | int | nat
       |  { ⟨Tuple⟩,^* } | { ⟨ComprExpr⟩ }
       |  ⟨NumExpr⟩..⟨NumExpr⟩ | ⟨NumExpr⟩

⟨ComprExpr⟩ ::= ⟨Expr⟩ | ( ⟨IdTuple⟩^&+ in ⟨SetExpr⟩ )^/\+ [ | ⟨Formula⟩ ]

⟨FuncAppl⟩ ::= ⟨Id⟩ ⟨Tuple⟩

⟨Tuple⟩ ::= (⟨Expr⟩,^+) | ⟨Expr⟩

⟨Formula⟩ ::= true | false | ⟨RelAppl⟩
           |  ⟨Formula⟩ ( /\ | \/ | => | <= | <=> ) ⟨Formula⟩
           |  ⟨Expr⟩ ( < | =< | = | >= | > | != ) ⟨Expr⟩
           |  forall ( ⟨QuantExpr⟩ ) ( ⟨Formula⟩ )
           |  count ( ⟨Set⟩ ) ( ⟨QuantExpr⟩ )

⟨RelAppl⟩ ::= ⟨Id⟩ ⟨Tuple⟩

⟨QuantExpr⟩ ::= ( ( ⟨RelQvars⟩ | ⟨IdTuple⟩^&+ ) in ⟨SetExpr⟩ ),^+ [ | ⟨Formula⟩ ]

⟨RelQvars⟩ ::= ⟨Expr⟩ ( < | =< | = | >= | > | != ) ⟨Expr⟩

⟨IdTuple⟩ ::= ⟨Id⟩ | ( ⟨Id⟩,^+ )
```

Fig. 1. The grammar of ESRA

dom *Varieties*, *Blocks*
cst $r, k, \lambda \in \mathbb{N}$
var *BIBD* : *Varieties* ${}^{r}\!\times^{k}$ *Blocks*
solve
  $\forall(v_1 < v_2 \in \textit{Varieties})\ \text{count}(\lambda)(i \in \textit{Blocks} \mid \textit{BIBD}(v_1, i) \wedge \textit{BIBD}(v_2, i))$

Fig. 2. A pretty-printed ESRA model for BIBDs

```
dom Varieties, Blocks
cst r, k, lambda in nat
var BIBD : Varieties [r#k] Blocks
solve
   forall (v1 < v2 in Varieties)
      count (lambda) (i in Blocks | BIBD(v1,i) /\ BIBD(v2,i))
```

Fig. 3. An ESRA model for BIBDs

2.3 Examples

We now showcase the elegance and flexibility of our language on three real-life problems, namely Balanced Incomplete Block Designs, the Social Golfers problem, and the Progressive Party problem.

Balanced Incomplete Block Designs. Let V be any set of v elements, called *varieties*. A *balanced incomplete block design* (BIBD) is a multiset of b subsets of V, called *blocks*, each of size k (constraint C_1), such that each pair of distinct varieties occurs together in exactly λ blocks (C_2), with $2 \leq k < v$. Implied constraints are that each variety occurs in the same number of blocks (C_3), namely $r = \lambda(v-1)/(k-1)$, as well as that $bk = vr$ and $\lambda < r$. A BIBD is thus parameterised by a 5-tuple $\langle v, b, r, k, \lambda \rangle$ of parameters, not all of which are independent. Originally intended for the design of statistical experiments, BIBDs also have applications in cryptography and other domains. See Problem 28 at www.csplib.org for more information.

The instance data can be declared as the two domains *Varieties* and *Blocks*, of implicit sizes v and b respectively, as well as the three natural-number constants r, k, and λ, as in Examples 1 and 4. A unique decision variable, *BIBD*, can then be declared using the relational domain in Example 5, thereby immediately taking care of the constraints C_1 and C_3. The remaining constraint C_2 can be modelled as in Example 9. Figure 2 shows the resulting pretty-printed ESRA model, while Figure 3 shows it in ASCII notation.

The Social Golfers Problem. In a golf club, there are n players, each of whom play golf once a week (constraint C_1) and always in g groups of size s (C_2), hence $n = gs$. The objective is to determine whether there is a schedule of w weeks of play for these golfers, such that there is at most one week where any two distinct players are scheduled to play in the same group (C_3). An implied constraint is that every group occurs exactly sw times across the schedule (C_4). See Problem 10 at www.csplib.org for more information.

cst $g, s, w \in \mathbb{N}$
dom $\mathit{Players} = 1 \ldots g * s,\ \mathit{Weeks} = 1 \ldots w,\ \mathit{Groups} = 1 \ldots g$
var $\mathit{Schedule} : (\mathit{Players} \times \mathit{Weeks}) \longrightarrow^{s*w} \mathit{Groups}$
solve
$\quad \forall(p_1 < p_2 \in \mathit{Players})\ \mathrm{count}(0 \ldots 1)(v \in \mathit{Weeks} \mid \mathit{Schedule}(p_1, v) = \mathit{Schedule}(p_2, v))$
$\quad \wedge\ \forall(h \in \mathit{Groups}, v \in \mathit{Weeks})\ \mathrm{count}(s)(p \in \mathit{Players} \mid \mathit{Schedule}(p, v) = h)$

Fig. 4. A pretty-printed ESRA model for the Social Golfers problem

The instance data can be declared as the three natural-number constants g, s, and w, via 'cst $g, s, w \in \mathbb{N}$', as well as the three domains *Players*, *Weeks*, and *Blocks*, as in Example 1. A unique decision variable, *Schedule*, can then be declared using the functional domain in Example 3, thereby immediately taking care of the constraints C_1 (because of the totality of the function) and C_4. The constraint C_3 can be modelled as in Example 10. The constraint C_2 can be stated using the count quantifier, as seen in the pretty-printed ESRA model of Figure 4.

The Progressive Party Problem. The problem is to timetable a party at a yacht club. Certain boats are designated as hosts, while the crews of the remaining boats are designated as guests. The crew of a host boat remains on board throughout the party to act as hosts, while the crew of a guest boat together visits host boats over a number of periods. The spare capacity of any host boat is not to be exceeded at any period by the sum of the crew sizes of all the guest boats that are scheduled to visit it then (constraint C_1). Any guest crew can visit any host boat in at most one period (C_2). Any two distinct guest crews can visit the same host boat in at most one period (C_3). See Problem 13 at www.csplib.org for more information.

The instance data can be declared as the three domains *Guests*, *Hosts*, and *Periods*, via 'dom *Guests*, *Hosts*, *Periods*', as well as the two constant functions *SpareCapacity* and *CrewSize*, as in Example 4. A unique decision variable, *Schedule*, can then be declared via 'var $\mathit{Schedule} : (\mathit{Guests} \times \mathit{Periods}) \longrightarrow \mathit{Hosts}$'. The constraint C_1 can now be modelled as in Example 8. The constraint C_2 could be enforced as follows:

$$\forall(g \in \mathit{Guests}, h \in \mathit{Hosts})\ \mathrm{count}(0 \ldots 1)(p \in \mathit{Periods} \mid \mathit{Schedule}(g, p) = h)$$

but the same effect can be achieved more succinctly by introducing the *projection* of the *Schedule* function on the guests:

$$\text{where } \mathit{Schedule}(\mathit{Guests}, \_) : \mathit{Periods} \longrightarrow^{0 \ldots 1} \mathit{Hosts}$$

This *projection clause* has to be adjoined to the declaration above of the *Schedule* decision variable, and is automatically compiled into the more complex constraint above. Finally, the constraint C_3 can be captured as follows:

$$\forall(g_1 < g_2 \in \mathit{Guests})\ \mathrm{count}(0 \ldots 1)(p \in \mathit{Periods} \mid \mathit{Schedule}(g_1, p) = \mathit{Schedule}(g_2, p))$$

Figure 5 shows the resulting pretty-printed ESRA model.

$$
\begin{array}{l}
\text{dom } \mathit{Guests},\ \mathit{Hosts},\ \mathit{Periods}\\
\text{cst } \mathit{SpareCapacity} : \mathit{Hosts} \longrightarrow \mathbb{N},\ \mathit{CrewSize} : \mathit{Guests} \longrightarrow \mathbb{N}\\
\text{var } \mathit{Schedule} : (\mathit{Guests} \times \mathit{Periods}) \longrightarrow \mathit{Hosts}\\
\quad \text{where } \mathit{Schedule}(\mathit{Guests}, \_) : \mathit{Periods} \longrightarrow^{0\ldots1} \mathit{Hosts}\\
\text{solve}\\
\quad \forall(p \in \mathit{Periods},\ h \in \mathit{Hosts}) \left(\sum_{g \in \mathit{Guests}\ \mid\ \mathit{Schedule}(g,p)=h} \mathit{CrewSize}(g) \right) \leq \mathit{SpareCapacity}(h)\\
\quad \wedge\\
\quad \forall(g_1 < g_2 \in \mathit{Guests})\ \text{count}(0\ldots1)(p \in \mathit{Periods} \mid \mathit{Schedule}(g_1,p) = \mathit{Schedule}(g_2,p))
\end{array}
$$

Fig. 5. A pretty-printed ESRA model for the Progressive Party problem

2.4 Compiling Relational Models

A prototype compiler for ESRA is currently under development. It is being written in OCAML (www.ocaml.org) and compiles an ESRA model into a SICStus Prolog [3] finite-domain constraint program. This choice of the target language is motivated by its excellent collection of global constraints and by our collaboration with its developers on designing new global constraints. We have several statements of interest for developing compilers of ESRA into other target languages. We already have an ESRA-to-OPL compiler for a restriction of ESRA to functions [24, 10]. The ESRA language is so high-level that it is very small compared to such target languages, especially in the number of necessary primitive constraints. The full panoply of features of these target languages can, and must, be deployed during compilation. In particular, the implementation of decision-variable indices is well-understood.

In order to bootstrap this prototype quickly, we chose the initial simplistic strategy of representing *every* relational variable by a table of 0/1 integer variables, indexed by its participating sets. This compiler is thus deterministic.

The plan is then to add alternatives to this unique representation rule, depending on the multiplicities and other constraints on the relation, achieving a *non-deterministic compiler*. The modeller is then invited to experiment with her (real-life) instances and the resulting compiled programs, so as to determine which one is the 'best'. If the compiler is provided with those instances, then it can be extended to automate such experiments and rankings.

Eventually, more intelligence will be built into the compiler via *heuristics* (such as those of [10]) for the compiler to rank the resulting compiled programs by decreasing likelihood of efficiency, without any recourse to experiments. Indeed, depending on the multiplicities and other constraints on a relation, certain representations thereof can be shown to be better than others, under certain assumptions on the solver, and this either theoretically (see, e.g., [22] for bijections and [10] for injections) or empirically (see, e.g., [17] for bijections).

Our ultimate aim is of course to design an actual *solver for relational constraints*, without going through compilation. Work in this direction has begun.

3 Benefits of Relational Modelling

In our experience, and as observable in Section 2.3, a relational constraint modelling language leads to more *concise and intuitive models*, as well as to more *efficient and effective model formulation and verification.* Due to ESRA being a *smaller language* than conventional constraint languages, we believe it is easier to learn and master, making it a good candidate for a teaching medium. All this could entail a better dissemination of constraint technology.

Relational languages seem a good trade-off between generality and specificity, enabling *efficient solving* despite more generality. Relations are a *single*, powerful concept for elegantly modelling many aspects of combinatorial problems. Also, there are *not too many* different, and even *standard*, ways of representing relations and relational expressions. Known and future modelling insights, such as those in [10, 17, 22], can be built into the compiler(s), so that even lazy or less competent modellers can benefit from them. Modelling is unencumbered by early if not uninformed commitments to representation choices. Low-level modelling devices such as reification and higher-order constraints can be encapsulated as implementation devices. The number of decision variables being reduced, there is even hope that directly solving the constraints at the high relational level can be faster than solving their compiled lower-level counterparts. All this illustrates that more generality need not mean poorer performance.

Relational models are more amenable to *maintenance* when the combinatorial problem changes, because most of the tedium is taken care of by the compiler, so that even lazy or less competent modellers are well assisted. Model maintenance at the relational level reduces to adapting to the new problem, with all representation (and solving) issues left to the compiler. Little work is involved here when a multiplicity change entails a preferable representation change for a relation. Maintenance can even be necessary when the statistical distribution of the problem instances that are to be solved changes [15]. If information on the new distribution is given to the compiler, a simple recompilation will take care of the maintenance.

Relational models are at a more suitable level for possibly automated model *reformulation*, such as via the inference and selection of suitable *implied constraints*, with again the compiler assisting in the more mundane aspects. In the BIBD and Social Golfers examples, we have observed that multiplicities provide a nice framework for discovering and stating some implied constraints because the language makes the modeller think about making these multiplicities explicit, even if they were not in the original problem formulation.

Relational models are more amenable to *constraint analysis.* Detected properties as well as properties consciously introduced during compilation into lower-level programs, such as symmetry or bijectiveness, can then be taken into account during compilation, especially using tractability results [21].

There would be further benefits to a relational modelling language if it were adopted as a *standard front-end language* for solvers. Indeed, models and instance data would then be solver-independent and could be shared between solvers. This would facilitate fair and homogeneous solver comparisons, say via new standard benchmarks, as well as foster competition in fine-tuning the compilers.

4 Conclusion

Summary. We have argued that solver-independent, high-level relational constraint modelling leads to a simpler and smaller language; to more concise, intuitive, and analysable models; as well as to more efficient and effective model formulation, maintenance, reformulation, and verification; and all this without sacrificing the possibility of efficient solving, so that even lazy or less competent modellers can be well assisted. Towards this, we have proposed the ESRA relational modelling language, showcased its elegance on some well-known problems, and outlined a compilation philosophy for such languages.

Related Work. We have here generalised and re-engineered our work [6, 24, 10] on a predecessor of ESRA, now called Functional-ESRA, that only supports function variables, by pursuing the plan outlined in [4].

This research owes a lot to previous work on relational modelling in formal methods and on ERA-style semantic data modelling, especially to the ALLOY object modelling language [11], which itself gained much from the Z specification notation [18] (and learned from UML/OCL how not to do it). Contrary to ERA modelling, we do not distinguish between attributes and relations.

In constraint programming, the commercial OPL [20] stands out as a medium-level modelling language and actually gave the impetus to design ESRA: consult [4] for a comparison of elegant ESRA models with more awkward (published) OPL counterparts that do not provide all the benefits of Section 3. Experimental higher-level constraint modelling languages have been proposed, such as ALICE [13], $CLP(Fun(D))$ [9], EACL [19], NCL [25], and NP-SPEC [2]. Our ESRA shares with them the quest for a practical declarative modelling language based on a strongly-typed fuller first-order logic than Horn clauses, possibly with functions or relations, while dispensing with such hard-to-properly-implement and rarely-necessary (for constraint modelling) luxuries as recursion, negation, and unbounded quantification. However, ESRA goes beyond them, by advocating an abstract-datatype view of relations, so that their representations need not be fixed in advance, as well as an elegant notation for multiplicity constraints. We lack the space here for a deeper comparison with these languages.

Future Work. Most of our future work has already been listed in Sections 2.4 and 3 about the compiler design and long-term benefits of relational modelling, such as the generation of implied constraints and the breaking of symmetries.

We have argued that our ESRA language is very small. This is mostly because we have not yet identified the need for any other operators or predicates. An exception to this is the need for *transitive closure relation constructors*. We have not yet fully worked out the details, but aim at modelling the well-known Travelling Salesperson (TSP) problem as in Figure 6, where the transitive closure of the bijection *Next* on *Cities* is denoted by $Next^*$. This general mechanism avoids the introduction of an *ad hoc* 'circuit' constraint as in ALICE [13]. As we

dom *Cities*
cst *Distance* : $(\mathit{Cities} \times \mathit{Cities}) \longrightarrow \mathbb{N}$
var *Next* : $\mathit{Cities} \longrightarrow^{1} \mathit{Cities}$
minimise $\sum_{c \in \mathit{Cities}} \mathit{Distance}(c, \mathit{Next}(c))$
such that $\forall (c_1 \& c_2 \in \mathit{Cities})\ \mathit{Next}^*(c_1) = c_2$

Fig. 6. A pretty-printed ESRA model for the Travelling Salesperson problem

do not aim at a complete modelling language, we can be very conservative in what missing features shall be added to ESRA when they are identified.

In [14], a type system is derived for binary relations that can be used as an input to specialised filtering algorithms. This kind of analysis can be integrated into the *relational solver* we have in mind. Also, a *graphical language* could be developed for the data modelling, including the multiplicity constraints on relations, so that only the cost function and the constraints would need to be textually expressed. Finally, a *search language*, such as SALSA [12] or the one of OPL [20], but at the level of relational modelling, should be adjoined to the constraint modelling language proposed here, so that more expert modellers can express their own search heuristics.

Acknowledgements. This work is partially supported by grant 221-99-369 of VR, the Swedish Research Council, and by institutional grant IG2001-67 of STINT, the Swedish Foundation for International Cooperation in Research and Higher Education. We thank Nicolas Beldiceanu, Mats Carlsson, Maria Fox, Brahim Hnich, Daniel Jackson, François Laburthe, Derek Long, Gerrit Renker, Christian Schulte, and Mark Wallace for stimulating discussions, as well as the constructive reviewers of a previous version of this paper.

References

1. J.-R. Abrial. *The B-Book: Assigning Programs to Meanings.* CUP, 1996.
2. M. Cadoli, L. Palopoli, A. Schaerf, and D. Vasile. NP-SPEC: An executable specification language for solving all problems in NP. In: G. Gupta (ed), *Proc. of PADL'99*, pp. 16–30. LNCS 1551. Springer-Verlag, 1999.
3. M. Carlsson, G. Ottosson, and B. Carlson. An open-ended finite domain constraint solver. In *Proc. of PLILP'97*, pp. 191–206. LNCS 1292. Springer-Verlag, 1997.
4. P. Flener. Towards relational modelling of combinatorial optimisation problems. In: Ch. Bessière (ed), *Proc. of the IJCAI'01 Workshop on Modelling and Solving Problems with Constraints*, 2001. At www.lirmm.fr/~bessiere/ws_ijcai01/.
5. P. Flener, A.M. Frisch, B. Hnich, Z. Kızıltan, I. Miguel, and T. Walsh. Matrix modelling: Exploiting common patterns in constraint programming. In *Proc. of the Int'l Workshop on Reformulating CSPs*, held at CP'02. At www-users.cs.york.ac.uk/~frisch/Reformulation/02/.
6. P. Flener, B. Hnich, and Z. Kızıltan. Compiling high-level type constructors in constraint programming. In: I.V. Ramakrishnan (ed), *Proc. of PADL'01*, pp. 229–244. LNCS 1990. Springer-Verlag, 2001.

7. P. Flener, J. Pearson, and M. Ågren. *The Syntax, Semantics, and Type System of* ESRA. At www.it.uu.se/research/group/astra/, April 2003.
8. C. Gervet. Interval propagation to reason about sets: Definition and implementation of a practical language. *Constraints* 1(3):191–244, 1997.
9. T.J. Hickey. Functional constraints in CLP languages. In: F. Benhamou and A. Colmerauer (eds), *Constraint Logic Programming: Selected Research*, pp. 355–381. The MIT Press, 1993.
10. B. Hnich. *Function Variables for Constraint Programming.* PhD Thesis, Department of Information Science, Uppsala University, 2003. At publications.uu.se/theses/91-506-1650-1/.
11. D. Jackson, I. Shlyakhter, and M. Sridharan. A micromodularity mechanism. In *Proc. of FSE/ESEC'01*, *Software Engineering Notes* 26(5):62–73, 2001.
12. F. Laburthe and Y. Caseau. SALSA: A language for search algorithms. *Constraints* 7:255–288, 2002.
13. J.-L. Laurière. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence* 10(1):29–127, 1978.
14. D. Lesaint. Inferring constraint types in constraint programming. In: P. Van Hentenryck (ed), *Proc. of CP'02*, pp. 462–476. LNCS 2470. Springer-Verlag, 2002.
15. S. Minton. Automatically configuring constraint satisfaction programs: A case study. *Constraints* 1(1–2):7–43, 1996.
16. J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual.* Addison-Wesley, 1999.
17. B.M. Smith. Modelling a permutation problem. Research Report 18, School of Computing, University of Leeds, UK, 2000.
18. J.M. Spivey. *The* Z *Notation: A Reference Manual* (second edition). Prentice, 1992.
19. E. Tsang, P. Mills, R. Williams, J. Ford, and J. Borrett. A computer-aided constraint programming system. In: J. Little (ed), *Proc. of PACLP'99*, pp. 81–93. The Practical Application Company, 1999.
20. P. Van Hentenryck. *The* OPL *Optimization Programming Language.* The MIT Press, 1999.
21. P. Van Hentenryck, P. Flener, J. Pearson, and M. Ågren. Tractable symmetry breaking for CSPs with interchangeable values. In *Proc. of IJCAI'03*, Morgan Kaufmann Publishers, 2003.
22. T. Walsh. Permutation problems and channelling constraints. In *Proc. of LPAR'01*, pp. 377–391. LNCS 2250. Springer-Verlag, 2001.
23. J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modeling with* UML. Addison-Wesley, 1999.
24. S. Wrang. *Implementation of the* ESRA *Constraint Modelling Language.* Master's Thesis in Computing Science 223, Department of Information Technology, Uppsala University, Sweden, 2002. At ftp.csd.uu.se/pub/papers/masters-theses/.
25. J. Zhou. Introduction to the constraint language NCL. *J. of Logic Programming* 45(1–3):71–103, 2000.

Encoding Connect-4 Using Quantified Boolean Formulae

Ian P Gent and Andrew G D Rowley

University of St. Andrews, Fife, Scotland
{ipg, agdr}@dcs.st-and.ac.uk

Abstract. The game of Connect-4 has been solved, as a win for the first player. Since games are one domain of application of QBF (Quantified Boolean Formulae), it is natural to ask if QBF solvers can prove this result. So, Toby Walsh has proposed solving the game of Connect-4 as a challenge problem for QBF solvers and researchers. In this report we take the zeroth step towards this challenge, by showing how the general game of Connect-c on a $w \times h$ grid can be encoded as a QBF. We present in detail the sets of QBF clauses used, but do not prove them correct. For the main Connect-4 problem, our encoding contains 18687 variables, 70946 clauses, and 21 alternations of quantifiers. A key part of our encoding are variables capturing the notion of a player "cheating". A win for the first player is any normal win, or any game in which the second player cheats. Without these, a player could win by, for example, making four moves simultaneously. In terms of the QBF, these are existential variables which are made equivalent to an expression involving universal variables. Our generator and selected instances are being submitted to QBFLib[1].

1 Introduction

Quantified Boolean Formulae (QBF) can be seen as games between a player setting values of existential variables, and one setting values of universal variables. The existential player wants to make the underlying SAT formula true, while the universal player wants to make it false. This is not just an analogy. Indeed any QBF instance can be made into a game, although not necessarily an enjoyable one.[2] Thus, two player games encoded as QBF instances are a natural domain of application of QBF.

In his invited talk at SAT-03, Toby Walsh suggested that solving the game of Connect-4 would be a productive challenge for QBF researchers [2]. Since it has already been solved using direct techniques (it is a win for the first player), the challenge is not in advancing the understanding of games. Rather, it is in the fact that since we know the problem can be solved exhaustively, it provides a good test case for QBF solvers. If we cannot solve it using reasonably general QBF techniques, that suggests that those techniques are quite limited. On the other hand, if we can solve it, the problem is large enough that it is likely that a

[1] http://www.qbflib.org/

[2] A Java implementation by Andrew Pickering makes the Game of QBF playable over the web [1].

number of advances in QBF solving will have been made. Even if it turns out to be easy to solve, there are harder games and other applications of QBF which will benefit from the experience of solving Connect-4.

Before anyone can solve Connect-4 as a QBF, somebody has to encode it. That is the initial step that we take in this paper.

Our encoding has two aspects that may be of value to others in encoding problems in QBF. First, we introduce "cheat variables". These solve the problem that the existential and universal players of the game of QBF are not obliged to obey the rules of Connect-4: cheat variables become true if a player cheats, and give the win to the opposite player. Second, we have "gameover" variables, which stop the game continuing when a player wins, and are intended to avoid unnecessary search when a game is won.

2 The game of Connect-c on a w by h board

Connect-4, sometimes known as "Four in a row" is played on a vertical grid of 7 squares across by 6 high, giving a total of 42 squares. There are two players, red and white, who each have 21 pieces. The players take it in turns to place their pieces in one of the 7 columns, where it falls to the bottom-most unoccupied square in the column. Only a total of 6 pieces can be played in any column. The players must play a piece on their turn and must only play one piece. The first player to get 4 pieces in a horizontal, vertical, or diagonal line wins, while if neither player achieves this the game is drawn. The red player goes first.

Connect-4 is available from toyshops under a variety of names, and a number of sites on the web allow online play [3]. It has been solved and it is known that the first player can always win [4].

We consider a slight generalisation of the game, Connect-c, where the players must connect c pieces instead of 4. We also allow variation in the board size of w squares across by h high. We force $w \geq c$ and $h \geq c$ to make the game interesting: if either w or h are less than c the game loses a dimension of play since diagonal lines and horizontal or vertical lines respectively become impossible to create. Also, $c \geq 2$ must be enforced otherwise the first player wins by playing the first piece. In fact $c = 2$ is trivial as the first player always wins on the second move; it is observed that Connect-c on a c dimensional board is always trivial. We restrict ourselves to a 2-dimensional board here.

3 Quantified Boolean Formulae

We provide some very brief details of QBF's, and refer the reader elsewhere for a more detailed introduction [5].

A Quantified Boolean Formula is of the form QB where Q is a sequence of quantifiers $Q_1v_1...Q_nv_n$, where each Q_i quantifies ($\exists$ or $\forall$) a variable v_i and each variable occurs exactly once in the sequence Q. B is a Boolean formula in conjunctive normal form (CNF), a conjunction of clauses where each clause is a disjunction of literals. Each literal is a variable and a sign. The literal is said to be negative if negated and positive otherwise. Universals and existentials are those variables quantified by a universal or existential quantifier respectively. A QBF of the form $\exists v_1 Q_2 v_2 ... Q_n v_n B$ is true if either $Q_2 v_2 ... Q_n v_n B[v_1 := T]$ or $Q_2 v_2 ... Q_n v_n B[v_1 := F]$ is true, where T represents true and F represents false. Similarly, a QBF of the form $\forall v_1 Q_2 v_2 ... Q_n v_n B$ is true if both $Q_2 v_2 ... Q_n v_n B[v_1 := T]$ and $Q_2 v_2 ... Q_n v_n B[v_1 := F]$ are true.

4 Connect-c as a QBF

In this section we present our encoding of Connect-c on a w wide by h high board. We will encode Connect-4 using a QBF in conjunctive normal form (CNF). QBF is still PSPACE-complete in CNF, and most current solvers work only on CNF instances. Before moving to the excruciating detail, we describe some general principles. These may be useful to those developing encodings for other problems.

Not surprisingly, the key variables we use encode the possible moves for each player. As there are up to w possible moves for each player at each turn, we introduce w^2h boolean variables.[3] The variable $redmove_{p,z}$ (respectively $whitemove_{p,z}$) is true when red (respectively white) plays a piece in column p at move z, $1 \leq z \leq h \times w$.

Encoding a game into a game-like logical system leads to some possible confusions, certainly enough to confuse us at various points in developing the encoding. The most crucial of these is that neither red nor white, viewed as the existential or universal player, need respect the set of clauses if there is a trivial way to "win". The red player wins by getting the formula to be true, and the white player by making it false. But many clauses encode rules of the game, not choices of the players. Without detailed care, this leads to the situation where one or other player can win by gratuitous cheating. For example, we encode that the white (universal) player must make a move at its first turn. Naively, this can be done by writing a clause $whitemove_{1,2} \vee \ldots \vee whitemove_{w,2}$. This gives white the winning strategy of cheating by not playing a move, falsifying the clause and thus the whole QBF. A little thought suggests the idea of adding an existential variable $whitecheat_{0,2}$, quantified later than the universal variables, to be true if this happens. We declare that red wins if $whitecheat_{0,2}$ is true, and revise the clause to $whitecheat_{0,2} \vee whitemove_{1,2} \vee \ldots \vee whitemove_{w,2}$. Unfortunately, now we have guaranteed the red player an equally fatuous winning strategy. However conscientiously white plays, red declares that white has cheated, setting $whitecheat_{0,2}$ true even though (say) $whitemove_{3,2}$ is also true. We seem to have a situation where either one player or the other has a trivial winning strategy. Of course neither player is cheating within the rules of QBF, but neither is prepared to accept the rules of Connect-4.

The solution to this problem is to introduce clauses forcing *equivalence* between the "cheat variables" and their intended meaning. As well as the clause $whitecheat_{0,2} \vee whitemove_{1,2} \vee \ldots \vee whitemove_{w,2}$, we therefore add w binary clauses, $\neg whitecheat_{0,2} \vee \neg whitemove_{p,z}$. In combination, these mean that

[3] One could use a logarithmic number of variables for each move. Doing so would complicate the encoding, and conventional wisdom in SAT is that logarithmic encodings are often ineffective.

$whitecheat_{0,2}$ has the intended meaning. If white tries to set the first clause false by making no move, red declares that white has cheated. But red can only make this claim when white has indeed made no move, or one of the binary clauses will become false. Note that the set of clauses, taken together, encodes $whitecheat_{0,2} \equiv \{whitemove_{1,2} \vee \ldots \vee whitemove_{w,2}\}$, but as we are working in CNF we cannot add this directly and instead encode it by one large clause and many binary clauses.

This pattern is ubiquitous, and absolutely dominates our encoding. In SAT, we are used to encoding some property by simply writing a clause for it. Here, we must be more subtle. Starting from the clause we want, we introduce a "cheat variable" to be true when the clause is false. We guard the original clause with the cheat variable, and then introduce new clauses to ensure that when the cheat variable is true the clause is indeed false.

In many cases, such as the example above, the term "cheat" is fair, while in others we simply need a variable to express some property for reference elsewhere, e.g. that red or white has a line of 4. In general, we therefore use the more neutral term "indicator variables". But for all indicator variables we see the same pattern: a clause with an indicator variable guarding it, and then many with the negated indicator forcing it to have the intended meaning. Note that indicator variables are all existential, even though many encode properties of universal variables. If they were universal variables, the universal player could win by setting them inconsistently with their intended meaning, thereby falsifying a clause and winning. They are quantified inside all variables they depend on, and as far out as possible given that constraint. That is, an indicator variable is quantified existentially following all the variables involved in the formula it indicates.

One set of indicator variables are true when the game is over at a given move. These play two roles. First, we use them to encode correctly the achievement of a winning line of 4. A winning line only counts if the game is not over yet: if one player has won, then any winning lines achieved later are irrelevant. The *gameover* variables have another important role. We guard all clauses by stating that they only apply if the game is not over at the relevant move. This counteracts a disadvantage of encoding in QBF: that all later variables have to be set even if a game is already won. By making clauses irrelevant if a game is over, we make sure that *any* assignment to later variables is valid and yields the same result. We hope this will make search easier.

Throughout our encoding we do not try to formalise deductions such as that there is only one column playable at the last move, or that the game cannot be over after the first $c - 1$ moves by each player. While this would reduce the size of the encoding, it would complicate its presentation and understanding. Furthermore, we suspect that most of these trivial deductions will be made by propagation in search, although it may well be that there is a set of implied clauses that will greatly improve search.

4.1 Indicator Variables

We now describe in detail the indicator variables we introduce. As red variables are existential, *redcheat* variables could be omitted in favour of direct clauses. However, we include them and similar variables, making our encoding pleasingly symmetric in red and white. While leading to slightly more variables, the performance penalty should be minor as the superfluous red indicator variables will be set by propagation.

In the case of cheating and of constructing a line, we number arbitrarily the ways of doing this, introducing a variable for each type of cheating and line.

– *redwin*, *whitewin* and *draw* - true iff red, white or nobody (respectively) wins.
– $gameover_z$ - true if the game is over at move z, i.e. the result is determined solely by moves up to $z - 1$ or earlier.
– $redwin_z$ and $whitewin_z$ - true iff red or white respectively wins at move z.
– $redcheat_z$ and $whitecheat_z$ - true iff red or white respectively cheats at move z. As there are many ways to cheat, we will make *redcheat* and *whitecheat* equivalent to the disjunction of the following variables.
– $redcheat_{f,z}$ and $whitecheat_{f,z}$ - true iff red or white respectively cheat in fashion f at move z. The fashions are numbered from 0 to $w(w - 1)/2$. Fashion 0 is for playing no counter on a turn, while the other $w(w - 1)/2$ enumerate the ways of playing counters in two different columns on the same turn.
– $rline_{n,z}$ and $whiteline_{n,z}$ - true iff red or white respectively has a line of c of type n at move z.
– $red_{x,y,z}$ and $white_{x,y,z}$ - true iff there is a red or white piece (respectively) in row x, column y after move z has been made.
– $occupied_{x,y,z}$ - true iff there is a piece in row x, column y before move z has been made ($1 \leq z \leq hw+1$). To simplify our encoding, we include a notional row 0, occupied with uncoloured pieces before move 1.
– $redflip_{x,y,z}$ and $whiteflip_{x,y,z}$ - true iff a red or white piece respectively appeared in row x column y at move z.

4.2 Quantifiers

We need to quantify the move variables $redmove_{,}$ and $whitemove_{,}$ in the correct way. We would like to know if there is a way for red to win given any move that white makes. We therefore quantify the $redmove_{,}$ variables existentially and the $whitemove_{,}$ variables universally. In addition to this, each player makes their move in turn starting with the red player; we do not want to allow white to play a piece at move n that is independent of move $n - 1$. This means that we must alternate the quantification between red's moves at move $n-1$ and white's moves at move n. The quantifiers of the key variables are therefore as follows:

$$\begin{array}{ll} \exists & (redmove_{1,1}, \ldots, redmove_{w,1}) \\ \forall & (whitemove_{1,2}, \ldots, whitemove_{w,2}) \end{array}$$

$$\begin{array}{ll} \ldots & \\ \exists & (redmove_{1,h\times w-1},\ldots,redmove_{w,h\times w-1}) \\ \forall & (whitemove_{1,h\times w},\ldots,whitemove_{w,h\times w}) \end{array}$$

In most cases, an indicator variable should be quantified so it can only be set after values have been assigned to the relevant move variables. This is done by inserting an existential quantifier for the indicator variable after the quantifier for the last relevant move variable. The exception are the *redwin*, *whitewin* and *draw* variables. These are there to specify the outcome that we require before the game has begun. While their truth value depends on variables quantified later, they must be quantified at the outermost level. They express properties of all assignments to variables, rather than indicating status of a particular partial assignment.

This leaves the quantification scheme as shown in Fig. 1.

$$\begin{array}{ll} \exists & redwin, whitewin, draw, gameover_1, occupied_{0,1,1},\ldots,occupied_{h+1,w,1} \\ \exists & redmove_{1,1},\ldots,redmove_{w,1} \\ \exists & \{\text{indicator variables for red move 1}\} \\ \forall & whitemove_{1,2},\ldots,whitemove_{w,2} \\ \exists & \{\text{indicator variables for white move 2}\} \\ \ldots & \\ \exists & redmove_{1,h\times w-1},\ldots,redmove_{w,h\times w-1} \\ \exists & \{\text{indicator variables for red move } h\times w-1\} \\ \forall & whitemove_{1,h\times w},\ldots,whitemove_{w,h\times w} \\ \exists & \{\text{indicator variables for white move } h\times w\} \end{array}$$

Fig. 1. Quantification scheme for encoding Connect-c on a $w \times h$ board. Note that the *only* universally quantified variables are $whitemove_{,}$. The list of existential indicator variables after move z are (for all appropriate x, y, n): $gameover_{z+1}$, $redwin_z$, $whitewin_z$, $redcheat_z$, $whitecheat_z$, $redcheat_{n,z}$, $whitecheat_{n,z}$, $rline_{n,z}$, $whiteline_{n,z}$, $red_{x,y,z}$, $white_{x,y,z}$, $occupied_{x,y,z+1}$, $redflip_{x,y,z}$, $whiteflip_{x,y,z}$. The scheme shown is where $h\times w$ is even. If both of h and w are odd the last move quantification is existential on $redmove_{,}$

4.3 Clauses

Our encoding contains 20 sets of clauses. Of course we do not expect a casual reader to study them in detail, but we wish to provide them in detail so that they are documented beyond code in a Perl script, and would encourage other researchers to do the same.

We explain the intention behind each set before detailing the clauses. In many cases there are red- and white-oriented versions of clauses, and we use a horizontal line to separate the red from white clauses. In most cases many versions of clauses are necessary, and we indicate this by one or more conjunctions. As the number of moves is hw, this can be odd or even, and this leads to an asymmetry between red and white, as red makes $\lceil \frac{1}{2}hw \rceil$ moves while white makes $\lfloor \frac{1}{2}hw \rfloor$.

1. The first set of clauses express that, unless the game is over, each player must make a move at their turn or be declared to have cheated. We also include in these clauses that if the game is over, then there is no need for any piece to be played. We express this with the following sets of clauses.

$$\bigwedge_{z=1}^{\lceil \frac{1}{2}hw \rceil} (gameover_{2z-1} \vee redcheat_{0,2z-1} \vee \bigvee_{p=1}^{w} redmove_{p,2z-1})$$

$$\bigwedge_{z=1}^{\lceil \frac{1}{2}hw \rceil} \bigwedge_{p=1}^{w} (gameover_{2z-1} \vee \neg redcheat_{0,2z-1} \vee \neg redmove_{p,2z-1})$$

$$\bigwedge_{z=1}^{\lfloor \frac{1}{2}hw \rfloor} (gameover_{2z} \vee whitecheat_{0,2z} \vee \bigvee_{p=1}^{w} whitemove_{p,2z})$$

$$\bigwedge_{z=1}^{\lfloor \frac{1}{2}hw \rfloor} \bigwedge_{p=1}^{w} (gameover_{2z} \vee \neg whitecheat_{0,2z} \vee \neg whitemove_{p,2z})$$

2. This set of clauses expresses that, unless the game is over, each player must make only one move at their turn or be declared to have cheated. There are $\frac{1}{2}w(w-1)$ ways of playing two moves simultaneously. We use the function $f(w,p,p')$ to yield a unique number between 1 and $\frac{1}{2}w(w-1)$ for each possible way of cheating for red or white respectively: this is implemented in code by a simple counter.

$$\bigwedge_{z=1}^{\lceil \frac{1}{2}hw \rceil} \bigwedge_{p=1}^{w-1} \bigwedge_{p'=p+1}^{w} (gameover_{2z-1} \vee redcheat_{f(w,p,p'),2z-1} \vee \neg redmove_{p,2z-1} \vee \neg redmove_{p',2z-1})$$

$$\bigwedge_{z=1}^{\lceil \frac{1}{2}hw \rceil} \bigwedge_{p=1}^{w-1} \bigwedge_{p'=p+1}^{w} (gameover_{2z-1} \vee \neg redcheat_{f(w,p,p'),2z-1} \vee redmove_{p,2z-1})$$

$$\bigwedge_{z=1}^{\lceil \frac{1}{2}hw \rceil} \bigwedge_{p=1}^{w-1} \bigwedge_{p'=p+1}^{w} (gameover_{2z-1} \vee \neg redcheat_{f(w,p,p'),2z-1} \vee redmove_{p',2z-1})$$

$$\bigwedge_{z=1}^{\lfloor \frac{1}{2}hw \rfloor} \bigwedge_{p=1}^{w-1} \bigwedge_{p'=p+1}^{w} (gameover_{2z} \vee whitecheat_{f(w,p,p'),2z} \vee \neg whitemove_{p,2z} \vee \neg whitemove_{p',2z})$$

$$\bigwedge_{z=1}^{\lfloor \frac{1}{2}hw \rfloor} \bigwedge_{p=1}^{w-1} \bigwedge_{p'=p+1}^{w} (gameover_{2z} \vee \neg whitecheat_{f(w,p,p'),2z} \vee whitemove_{p,2z})$$

$$\bigwedge_{z=1}^{\lfloor\frac{1}{2}hw\rfloor}\bigwedge_{p=1}^{w-1}\bigwedge_{p'=p+1}^{w}(gameover_{2z}\vee\neg whitecheat_{f(w,p,p'),2z}\vee whitemove_{p',2z})$$

3. We now specify that a player has cheated at move z if and only if the player has cheated in some fashion at move z.

$$\bigwedge_{z=1}^{\lceil\frac{1}{2}hw\rceil}(gameover_{2z-1}\vee\neg redcheat_{2z-1}\vee\bigvee_{f=0}^{\frac{1}{2}w(w-1)}redcheat_{f,2z-1})$$

$$\bigwedge_{z=1}^{\lceil\frac{1}{2}hw\rceil}\bigwedge_{f=0}^{\frac{1}{2}w(w-1)}(gameover_{2z-1}\vee redcheat_{2z-1}\vee\neg redcheat_{f,2z-1})$$

$$\bigwedge_{z=1}^{\lfloor\frac{1}{2}hw\rfloor}(gameover_{2z}\vee\neg whitecheat_{2z}\vee\bigvee_{f=0}^{\frac{1}{2}w(w-1)}whitecheat_{f,2z})$$

$$\bigwedge_{z=1}^{\lfloor\frac{1}{2}hw\rfloor}\bigwedge_{f=0}^{\frac{1}{2}w(w-1)}(gameover_{2z}\vee whitecheat_{2z}\vee\neg whitecheat_{f,2z})$$

4. The following clauses specify that only one of a red and white piece can occupy any space at any time.

$$\bigwedge_{z=1}^{hw}\bigwedge_{x=1}^{h}\bigwedge_{y=1}^{w}(gameover_z\vee\neg red_{x,y,z}\vee\neg white_{x,y,z})$$

5. These clauses express that a red or white respectively in a position at move z means that there is a piece in the position at move $z+1$. Additionally, a red or white in a position means a red or white respectively in the same position on the next move.

$$\bigwedge_{z=1}^{hw}\bigwedge_{x=1}^{h+1}\bigwedge_{y=1}^{w}(gameover_z\vee\neg red_{x,y,z}\vee occupied_{x,y,z+1})$$

$$\bigwedge_{z=1}^{hw}\bigwedge_{x=1}^{h+1}\bigwedge_{y=1}^{w}(gameover_z\vee\neg white_{x,y,z}\vee occupied_{x,y,z+1})$$

$$\bigwedge_{z=1}^{hw}\bigwedge_{x=1}^{h+1}\bigwedge_{y=1}^{w}(gameover_z\vee\neg occupied_{x,y,z+1}\vee red_{x,y,z}\vee white_{x,y,z})$$

$$\bigwedge_{z=1}^{hw-1}\bigwedge_{x=1}^{h}\bigwedge_{y=1}^{w}(gameover_z\vee\neg red_{x,y,z}\vee red_{x,y,z+1})$$

$$\bigwedge_{z=1}^{hw-1}\bigwedge_{x=1}^{h}\bigwedge_{y=1}^{w}(gameover_z\vee\neg white_{x,y,z}\vee white_{x,y,z+1})$$

6. If the game is not over at move z and there is a piece in row x, column y and no piece above it, then if a red or white is played in column y then there is now a red or white respectively in the empty square. This is represented by the following clauses.

$$\bigwedge_{z=1}^{\lceil\frac{1}{2}hw\rceil}\bigwedge_{x=0}^{h}\bigwedge_{y=1}^{w}(\mathit{gameover}_{2z-1} \vee \neg \mathit{occupied}_{x,y,2z-1} \vee \mathit{occupied}_{x+1,y,2z-1} \vee \neg \mathit{redmove}_{y,2z-1} \vee \mathit{redflip}_{x+1,y,2z-1})$$

$$\bigwedge_{z=1}^{\lceil\frac{1}{2}hw\rceil}\bigwedge_{x=0}^{h}\bigwedge_{y=1}^{w}(\mathit{gameover}_{2z-1} \vee \neg \mathit{redflip}_{x+1,y,2z-1} \vee \mathit{occupied}_{x,y,2z-1})$$

$$\bigwedge_{z=1}^{\lceil\frac{1}{2}hw\rceil}\bigwedge_{x=0}^{h}\bigwedge_{y=1}^{w}(\mathit{gameover}_{2z-1} \vee \neg \mathit{redflip}_{x+1,y,2z-1} \vee \neg \mathit{occupied}_{x+1,y,2z-1})$$

$$\bigwedge_{z=1}^{\lceil\frac{1}{2}hw\rceil}\bigwedge_{x=0}^{h}\bigwedge_{y=1}^{w}(\mathit{gameover}_{2z-1} \vee \neg \mathit{redflip}_{x+1,y,2z-1} \vee \mathit{redmove}_{y,2z-1})$$

$$\bigwedge_{z=1}^{\lceil\frac{1}{2}hw\rceil}\bigwedge_{x=0}^{h}\bigwedge_{y=1}^{w}(\mathit{gameover}_{2z-1} \vee \neg \mathit{redflip}_{x+1,y,2z-1} \vee \mathit{red}_{x+1,y,2z-1})$$

$$\bigwedge_{z=1}^{\lfloor\frac{1}{2}hw\rfloor}\bigwedge_{x=0}^{h}\bigwedge_{y=1}^{w}(\mathit{gameover}_{2z} \vee \neg \mathit{occupied}_{x,y,2z} \vee \mathit{occupied}_{x+1,y,2z} \vee \neg \mathit{whitemove}_{y,2z} \vee \mathit{whiteflip}_{x+1,y,2z})$$

$$\bigwedge_{z=1}^{\lfloor\frac{1}{2}hw\rfloor}\bigwedge_{x=0}^{h}\bigwedge_{y=1}^{w}(\mathit{gameover}_{2z} \vee \neg \mathit{whiteflip}_{x+1,y,2z} \vee \mathit{occupied}_{x,y,2z})$$

$$\bigwedge_{z=1}^{\lfloor\frac{1}{2}hw\rfloor}\bigwedge_{x=0}^{h}\bigwedge_{y=1}^{w}(\mathit{gameover}_{2z} \vee \neg \mathit{whiteflip}_{x+1,y,2z} \vee \neg \mathit{occupied}_{x+1,y,2z})$$

$$\bigwedge_{z=1}^{\lfloor\frac{1}{2}hw\rfloor}\bigwedge_{x=0}^{h}\bigwedge_{y=1}^{w}(\mathit{gameover}_{2z} \vee \neg \mathit{whiteflip}_{x+1,y,2z} \vee \mathit{whitemove}_{y,2z})$$

$$\bigwedge_{z=1}^{\lfloor\frac{1}{2}hw\rfloor}\bigwedge_{x=0}^{h}\bigwedge_{y=1}^{w}(\mathit{gameover}_{2z} \vee \neg \mathit{whiteflip}_{x+1,y,2z} \vee \mathit{white}_{x+1,y,2z})$$

7. If there is a red or white in a position at the next move, then there was either one there at this move, or a red or white respectively was added at this move. This is specified as follows.

$$\bigwedge_{z=1}^{hw-1}\bigwedge_{x=0}^{h}\bigwedge_{y=1}^{w}(\mathit{gameover}_{z} \vee \neg \mathit{red}_{x,y,z+1} \vee \mathit{red}_{x,y,z} \vee \mathit{redflip}_{x,y,z})$$

$$\bigwedge_{z=1}^{hw-1} \bigwedge_{x=0}^{h} \bigwedge_{y=1}^{w} (gameover_z \vee \neg white_{x,y,z+1} \vee white_{x,y,z} \vee whiteflip_{x,y,z})$$

8. We must populate the notional row zero so that the pieces have something to hold them up. The following clauses do this.

$$\bigwedge_{z=1}^{hw} \bigwedge_{y=1}^{w} (occupied_{0,y,z})$$

9. This set of clauses states that if there is a piece in some position in row x that has a space above it and neither red nor white play in that space then there is no piece in the space on the next move.

$$\bigwedge_{z=1}^{\lceil \frac{1}{2}hw \rceil} \bigwedge_{x=0}^{h-1} \bigwedge_{y=1}^{w} (gameover_{2z-1} \vee \neg occupied_{x,y,2z-1} \vee occupied_{x+1,y,2z-1} \vee redmove_{y,2z-1} \vee \neg occupied_{x+1,y,2z})$$

$$\bigwedge_{z=1}^{\lfloor \frac{1}{2}hw \rfloor} \bigwedge_{x=0}^{h-1} \bigwedge_{y=1}^{w} (gameover_{2z} \vee \neg occupied_{x,y,2z} \vee occupied_{x+1,y,2z} \vee whitemove_{y,2z} \vee \neg occupied_{x+1,y,2z+1})$$

10. If there is no piece in a position at a move then there is no red and no white in the position above. This is represented as follows.

$$\bigwedge_{z=1}^{hw} \bigwedge_{x=1}^{h} \bigwedge_{y=1}^{w} (gameover_z \vee occupied_{x,y,z} \vee \neg red_{x+1,y,z})$$

$$\bigwedge_{z=1}^{hw} \bigwedge_{x=1}^{h} \bigwedge_{y=1}^{w} (gameover_z \vee occupied_{x,y,z} \vee \neg white_{x+1,y,z})$$

11. We must disallow the placing of pieces in the row above the top of the board. There is also no piece in the bottom row on the first move. To express this we add the following clauses.

$$\bigwedge_{z=1}^{hw} \bigwedge_{y=1}^{w} (\neg occupied_{h+1,y,z})$$

$$\bigwedge_{y=1}^{w} (\neg occupied_{1,y,1})$$

12. The game over state must be propagated so that if the game is over at a move it is over at the next move as well. If the game is not over at a move, and someone wins at that move, the game is over at the next move. The game is not over on the first move. These constraints are stated as follows.

$$\bigwedge_{z=1}^{hw-1} (\neg gameover_z \vee gameover_{z+1})$$
$$\bigwedge_{z=1}^{hw-1} (gameover_z \vee \neg redwin_z \vee gameover_{z+1})$$
$$\bigwedge_{z=1}^{hw-1} (gameover_z \vee \neg whitewin_z \vee gameover_{z+1})$$
$$\bigwedge_{z=1}^{hw-1} (\neg gameover_{z+1} \vee gameover_z \vee redwin_z \vee whitewin_z)$$
$$(\neg gameover_1)$$
$$(\neg redwin_1)$$
$$(\neg whitewin_1)$$

13. The following clauses represent the facts that if the game is not over at the end then it is a draw, and every game must be either a win for red, a win for white or a draw.
$$(gameover_{hw+1} \vee draw)$$
$$(\neg draw \vee \neg gameover_{hw+1})$$
$$(redwin \vee whitewin \vee draw)$$
$$(\neg redwin \vee \neg whitewin)$$
$$(\neg redwin \vee \neg draw)$$
$$(\neg whitewin \vee \neg draw)$$

14. The fact that red or white wins if red or white respectively wins at any intermediate move is encoded as follows.
$$\bigwedge_{z=1}^{hw} (gameover_z \vee \neg redwin_z \vee redwin)$$
$$(redwin \vee \bigvee_{z=1}^{hw} redwin_z)$$

$$\bigwedge_{z=1}^{hw} (gameover_z \vee \neg whitewin_z \vee whitewin)$$
$$(whitewin \vee \bigvee_{z=1}^{hw} whitewin_z)$$

15. The next set of clauses encode that red or white has a line if there are c reds or whites respectively in a line. $l_v = w(h - c + 1)$ is the number of vertical lines, $l_h = h(w - c + 1)$ is the number of horizontal lines and $l_d = (w - c + 1)(h - c + 1)$ is number of diagonal lines in each direction.

$$\bigwedge_{z=1}^{hw}\bigwedge_{y=1}^{w}\bigwedge_{x'=1}^{h-c+1}(gameover_z \vee rline_{w(x'-1)+y,z} \vee \bigvee_{x=x'}^{x'+c}\neg red_{x,y,z})$$

$$\bigwedge_{z=1}^{hw}\bigwedge_{y=1}^{w}\bigwedge_{x'=1}^{h-c+1}\bigwedge_{x=x'}^{x'+c}(gameover_z \vee \neg rline_{w(x'-1)+y,z} \vee red_{x,y,z})$$

$$\bigwedge_{z=1}^{hw}\bigwedge_{x=1}^{h}\bigwedge_{y'=1}^{w-c+1}(gameover_z \vee rline_{l_v+h(y'-1)+x,z} \vee \bigvee_{y=y'}^{y'+c}\neg red_{x,y,z})$$

$$\bigwedge_{z=1}^{hw}\bigwedge_{x=1}^{h}\bigwedge_{y'=1}^{w-c+1}\bigwedge_{y=y'}^{y'+c}(gameover_z \vee \neg rline_{l_v+h(y'-1)+x,z} \vee red_{x,y,z})$$

$$\bigwedge_{z=1}^{hw}\bigwedge_{y=1}^{w-c+1}\bigwedge_{x=1}^{h-c+1}(gameover_z \vee rline_{l_v+l_h+(w-c+1)(x-1)+y,z} \vee \bigvee_{d=0}^{c-1}\neg red_{x+d,y+d,z})$$

$$\bigwedge_{z=1}^{hw}\bigwedge_{y=1}^{w-c+1}\bigwedge_{x=1}^{h-c+1}\bigwedge_{d=0}^{c-1}(gameover_z \vee \neg rline_{l_v+l_h+(w-c+1)(x-1)+y,z} \vee red_{x+d,y+d,z})$$

$$\bigwedge_{z=1}^{hw}\bigwedge_{y=1}^{w-c+1}\bigwedge_{x=1}^{h-c+1}(gameover_z \vee rline_{l_v+l_h+l_d)+(w-c+1)(x-1)+y,z} \vee \bigvee_{d=0}^{c-1}\neg red_{x+d,y+c-1-d,z})$$

$$\bigwedge_{z=1}^{hw}\bigwedge_{y=1}^{w-c+1}\bigwedge_{x=1}^{h-c+1}\bigwedge_{d=0}^{c-1}(gameover_z \vee \neg rline_{l_v+l_h+l_d+(w-c+1)(x-1)+y,z} \vee red_{x+d,y+c-1-d,z})$$

$$\bigwedge_{z=1}^{hw}\bigwedge_{y=1}^{w}\bigwedge_{x'=1}^{h-c+1}(gameover_z \vee whiteline_{w(x'-1)+y,z} \vee \bigvee_{x=x'}^{x'+c}\neg white_{x,y,z})$$

$$\bigwedge_{z=1}^{hw}\bigwedge_{y=1}^{w}\bigwedge_{x'=1}^{h-c+1}\bigwedge_{x=x'}^{x'+c}(gameover_z \vee \neg whiteline_{w(x'-1)+y,z} \vee white_{x,y,z})$$

$$\bigwedge_{z=1}^{hw}\bigwedge_{x=1}^{h}\bigwedge_{y'=1}^{w-c+1}(gameover_z \vee whiteline_{l_v+h(y'-1)+x,z} \vee \bigvee_{y=y'}^{y'+c}\neg white_{x,y,z})$$

$$\bigwedge_{z=1}^{hw}\bigwedge_{x=1}^{h}\bigwedge_{y'=1}^{w-c+1}\bigwedge_{y=y'}^{y'+c}(gameover_z \vee \neg whiteline_{l_v+h(y'-1)+x,z} \vee white_{x,y,z})$$

$$\bigwedge_{z=1}^{hw} \bigwedge_{y=1}^{w-c+1} \bigwedge_{x=1}^{h-c+1} (gameover_z \vee whiteline_{l_v+l_h+(w-c+1)(x-1)+y,z} \vee \bigvee_{d=0}^{c-1} \neg white_{x+d,y+d,z})$$

$$\bigwedge_{z=1}^{hw} \bigwedge_{y=1}^{w-c+1} \bigwedge_{x=1}^{h-c+1} \bigwedge_{d=0}^{c-1} (gameover_z \vee \neg whiteline_{l_v+l_h+(w-c+1)(x-1)+y,z} \vee white_{x+d,y+d,z})$$

$$\bigwedge_{z=1}^{hw} \bigwedge_{y=1}^{w-c+1} \bigwedge_{x=1}^{h-c+1} (gameover_z \vee whiteline_{l_v+l_h+l_d+(w-c+1)(x-1)+y,z} \vee \bigvee_{d=0}^{c-1} \neg white_{x+d,y+c-1-d,z})$$

$$\bigwedge_{z=1}^{hw} \bigwedge_{y=1}^{w-c+1} \bigwedge_{x=1}^{h-c+1} \bigwedge_{d=0}^{c-1} (gameover_z \vee \neg whiteline_{l_v+l_h+l_d+(w-c+1)(x-1)+y,z} \vee white_{x+d,y+c-1-d,z})$$

16. If the game is not over and red has not cheated, then red has won at this move iff either white cheated or red obtained a line at this move. Similarly for white.

$$\bigwedge_{z=1}^{hw} (gameover_z \vee \neg redwin_z \vee redcheat_z \vee whitecheat_z \vee \bigvee_{f=0}^{l_v+l_h+2l_d} rline_{f,z})$$

$$\bigwedge_{z=1}^{hw} \bigwedge_{f=0}^{l_v+l_h+2l_d} (gameover_z \vee redwin_z \vee redcheat_z \vee \neg rline_{f,z})$$

$$\bigwedge_{z=1}^{hw} (gameover_z \vee redwin_z \vee redcheat_z \vee \neg whitecheat_z)$$

$$\bigwedge_{z=1}^{hw} (gameover_z \vee \neg whitewin_z \vee whitecheat_z \vee redcheat_z \vee \bigvee_{f=0}^{l_v+l_h+2l_d} whiteline_{f,z})$$

$$\bigwedge_{z=1}^{hw} \bigwedge_{f=0}^{l_v+l_h+2l_d} (gameover_z \vee whitewin_z \vee whitecheat_z \vee \neg whiteline_{f,z})$$

$$\bigwedge_{z=1}^{hw} (gameover_z \vee whitewin_z \vee whitecheat_z \vee \neg redcheat_z)$$

17. With the following clauses, we represent the fact that neither red or white can win by cheating and that red or white cannot cheat when it is not their turn.

$$\bigwedge_{z=1}^{hw} (gameover_z \vee \neg redcheat_z \vee \neg redwin_z)$$
$$\bigwedge_{z=1}^{hw} (gameover_z \vee \neg whitecheat_z \vee \neg whitewin_z)$$
$$\bigwedge_{z=1}^{\lfloor \frac{1}{2}hw \rfloor} (\neg redcheat_{2z})$$
$$\bigwedge_{z=1}^{\lceil \frac{1}{2}hw \rceil} (\neg whitecheat_{2z-1})$$

18. These clauses dictate that if the game is already over, neither player can win.

$$\bigwedge_{z=1}^{hw} (\neg gameover_z \vee \neg redwin_z)$$
$$\bigwedge_{z=1}^{hw} (\neg gameover_z \vee \neg whitewin_z)$$

19. This clause decides on which player we would like to win the game.

$$(redwin)$$

20. Finally, we include clauses to break symmetry. If the width is even, we just say that the first move is on the left hand side of the board.

$$(gameover_1 \vee redcheat_{0,1} \vee \bigvee_{y=1}^{\frac{1}{2}w} redmove_{y,1})$$

For odd width, symmetry remains as long as both players play in the centre. So we state that the first move which is *not* in the centre column is played on the left hand side of the board. This must happen by the $h + 1^{th}$ move at the latest, so we add the following $h + 1$ clauses.

$$\bigwedge_{z=1}^{\lceil \frac{1}{2}(h+1) \rceil} (gameover_{2z-1} \vee redcheat_{0,2z-1} \vee \bigvee_{z'=1}^{z-1} \neg redmove_{\frac{1}{2}(w+1),2z'-1} \vee$$

$$\bigvee_{z'=1}^{z-1} \neg whitemove_{\frac{1}{2}(w+1),2z'} \vee \bigvee_{y=1}^{\frac{1}{2}(w+1)} redmove_{y,z})$$

$$\bigwedge_{z=1}^{\lfloor \frac{1}{2}(h+1) \rfloor} (gameover_{2z} \vee whitecheat_{0,2z} \vee \bigvee_{z'=1}^{z} \neg redmove_{\frac{1}{2}(w+1),2z'-1} \vee$$

$$\bigvee_{z'=1}^{z-1} \neg whitemove_{\frac{1}{2}(w+1),2z'} \vee \bigvee_{y=1}^{\frac{1}{2}(w+1)} whitemove_{y,z})$$

5 Experiments

Here we describe some brief details of some basic experiments on the Connect-c encoding. These are not intended as a serious attack on solving the problem, but as indicative of the performance of a QBF solver using standard techniques and without any heuristics or propagation rules tuned for Connect-4. For all experiments, we used a local implementation of conflict and solution directed backjumping for QBF[6] using the Watched Clauses data structure[7]. The experiments were performed on an Intel Celeron 1.70 Ghz computer with 128MB RAM. The time-out was 3600 seconds.

Table 1. Sample Run Times on Connect-c problems. c is the number of pieces that must be in a line for a win, w is the width of the board, h is the height of the board and the time is the run time in seconds. '-' denotes a time-out on this problem. The standard Connect-4 problem is 4/7/6, and was unsolved.

c w h	Solution	Time (s)	c w h	Solution	Time (s)	c w h	Solution	Time (s)
2 2 2	T	0.00						
2 3 3	T	0.03	3 3 3	F	0.07			
2 4 4	T	0.15	3 4 4	T	8.10	4 4 4	F	3.08
2 5 5	T	4.23	3 5 5	T	451.41	4 5 5	F	703.92
2 6 6	T	22.80	3 6 6		-	4 6 6		-
2 7 6	T	99.11	3 7 6		-	4 7 6		-
2 7 7	T	137.59	3 7 7		-	4 7 7		-

Clearly, the dominating factor is the size of the board, and run time increases very fast in this. Increasing c does generally increase run time, in some cases by a factor of 40 or so. However, the increase from $c = 3$ to 4 was not so dramatic, and in one case run time actually reduced. The rapidly increasing run time for $c = 2$ is disappointing. Red is guaranteed to be able to win on its second move, irrespective of where it plays its first move on any of the board sizes tested. This suggests that either our encoding necessarily leads to additional search, or that our solver is not recognising that a simple winning strategy exists. On looking in closer detail, the reason for the result is that red does not necessarily play a piece to make a line on his second move when the choice is presented; often the first piece is played in the left-most column and the second in the right-most. While disappointing, this does at least show that there are significant improvements available, either in modified encodings or improved solvers, and these may carry over to the more interesting cases.

6 Conclusions

We should make two caveats about our work. First, we have not proven correctness of our encoding. Second, this is just one encoding of the game. The importance of representation is fundamental throughout artificial intelligence, and in both constraint satisfaction and in SAT increasing attention is being paid to the difference a good encoding makes. We have no reason for suggesting that our encoding will be particularly effective compared to other possible encodings of Connect-4.

Finally, we would like to encourage other researchers encoding complicated problems into QBF to report their work in detail as well as submitting instances to QBFLib. As well as giving others a chance to spot mistakes, it will enable the community to learn from each other's encoding tricks.

Acknowledgements

The first author is supported by a Royal Society of Edinburgh SEELLD Support Fellowship and the second by EPSRC grant GR/55382. The authors are members of the APES research group and thank other members. We especially thank Toby Walsh for taking up the first author's suggestion of posing the Connect-4 problem as a challenge for QBF.

References

1. Pickering, A.: The game of qbf. Undergraduate project (2001) http://www-hons.dcs.st-and.ac.uk/ProjectMuseum/SHProjects/tour.php.
2. Walsh, T.: Challenges for SAT and QBF. Presentation at SAT 2003 (2003) http://www.4c.ucc.ie/~tw/sat2003.ppt.
3. Bridges, S.: Four in a row. Game playable on the web (1999) http://www.geocities.com/ResearchTriangle/System/3517/C4/C4Conv.html.
4. Allis, V.: A knowledge-based approach of connect-four. Master's thesis, Vrije Universiteit (1988) ftp://ftp.cs.vu.nl/pub/victor/connect4.ps.Z.
5. Cadoli, M., Giovanardi, A., Schaerf, M.: An algorithm to evaluate quantified Boolean formulae. In: AAAI-98. (1998) 262–267. http://www.dis.uniroma1.it/pub/AI/papers/cado-giov-scha-98.ps.gz.
6. Guinchiglia, E., Narizzano, M., Tacchella, A.: Backjumping for quantified Boolean logic satisfiability. In: IJCAI-01. (2001) 275–281 http://www.mrg.dist.unige.it/~enrico/ftp/01ijcai.ps.gz.
7. Gent, I., Giunchiglia, E., Narizzano, M., Rowley, A., Tacchella, A.: Efficient data structures for QBF solvers. In: SAT. (2003) http://www.dcs.st-and.ac.uk/~apes/papers/sat03.ps.

Why Channel?
Multiple Viewpoints for Branching Heuristics

Brahim Hnich and Toby Walsh

Cork Constraint Computation Center, University College Cork, Ireland.
{brahim,tw}@4c.ucc.ie

Abstract. When modelling a problem, there are often alternative viewpoints that can be taken. It can even be advantageous to use multiple viewpoints, and to have constraints which channel between them to maintain consistency. Multiple viewpoints often make it much easier to post the different problem constraints, as well as improve the amount of constraint propagation. In this paper, we demonstrate another reason for using multiple viewpoints: branching heuristics can be more effective when they look at multiple viewpoints.

1 Introduction

Constraint programming is a highly successful technology for solving a wide variety of combinatorial problems like resource allocation, transportation, and scheduling. However, its uptake is hindered by the difficulty of modelling problems successfully as constraint programs. One key modelling decision is which viewpoint or viewpoints to use. For example, in modelling a sports tournament scheduling problem, do we take the viewpoint in which the games are the variables, and the times are their values? Or do we take the dual viewpoint in which the times are the variables, and the games are their values? Or do we take both viewpoints, and have channelling constraints to maintain consistency between the two viewpoints?

There are a number of reasons we might consider multiple viewpoints, even though using more than one viewpoint introduces additional overheads. First, different constraints may be easier to post in the different viewpoints. Second, propagation may be improved. For example, one viewpoint may be linear and so can be solved using ILP. A third possibility, proposed by Geelen in [1] and explored in detail here, is that branching heuristics may profit from looking at more than one viewpoint. The results given in [1] are promising but are limited to a small number of experiments on n-queens problems. It is therefore timely to perform a more extensive experimental study on a range of challenging problems.

2 Permutation problems

Our analysis will focus on permutation problems. A *permutation problem* is a constraint satisfaction problem in which each decision variable takes an unique value, and there is the same number of values as variables. In a permutation problem, we can easily transpose the roles of the variables and the values to give a *dual* model which is also a permutation problem. Each variable in the primal becomes a value in the dual, and vice versa. We shall also consider *multiple permutation problems* in which the variables divide into a number of (possibly overlapping) sets, each of which is a permutation problem.

It is possible to combine multiple viewpoints by using *channelling constraints* to maintain consistency between the different viewpoints. This approach is called "redundant modelling" by Cheng *et al.* [2] and was specifically suggested for permutation problems in [1]. In a permutation problem, the channelling constraints are of the form: $X_i = j$ iff $D_j = i$. Many constraint toolkits support channelling of this kind with efficient global constraints. For example, ILOG Solver has a constraint, `IlcInverse`, and the Sicstus finite domain constraint library has an `assignment` predicate which can be used to channel efficiently between the primal and dual viewpoints of a permutation.

To ensure that we have a permutation, we can post a global all-different constraint on the primal variables. Alternatively, we can post binary not-equals constraints between any two primal variables. However, if we have both primal and dual variables, the channelling constraints are on their own sufficient to ensure we have a permutation. Indeed, the channelling constraints provide an intermediate level of pruning between what GAC achieves on a primal all-different constraint and AC on binary not-equals constraints on the primal [3, 4]. AC on the binary not-equals constraints identifies singleton variables (those variables with a single value left in their domain). AC on the channelling constraints identifies both singleton variables and singleton values (those values which are left in the domain of a single variable). GAC on an all-different constraint identifies both singleton variables and singleton values plus even more complex situation (e.g. three variables with just two values left between them).

There are thus a large number of different ways to model and solve a permutation problem. We can, for example, post an all-different constraint posted on the primal and use Regin's efficient algorithm [5] to maintain GAC on this constraint (in the tables of results, we will write "$\forall$" for this model and solution method). Alternatively, we can maintain AC on channelling constraints between primal and dual (we write "c" for this model and solution method). A third viewpoint is to maintain AC on binary not-equals constraints between any two primal variables (we write "$\neq$" for this model and solution method). Finally, we can take any combination of these viewpoints. For example, we can maintain GAC on an all-different constraint on the primal and AC on channelling constraints between primal and dual (we write "$\forall c$" for this model and solution method).

3 Variable and value ordering

The aim of this paper is to study how multiple viewpoints may benefit variable and value ordering heuristics. A variable ordering heuristic like smallest domain is usually justified in terms of a “fail-first” principle. We have to pick eventually all the variables, so it is wise to choose one that is hard to assign, giving us hopefully much constraint propagation and a small search tree. On the other hand, a value ordering heuristics like most promise [1] is usually justified in terms of a “succeed-first” principle [6]. We pick a value likely to lead to a solution, so reducing the risk of backtracking and trying one of the alternative values. In a permutation problem, we can branch on the primal or the dual variables or on both. We therefore consider the following heuristics.

Smallest domain, SD(p+d) : choose the primal or the dual variable with the smallest domain, and choose the values in numeric order.

Primal smallest domain, SD(p) : choose the primal variable with the smallest domain, and choose the values in numeric order.

Dual smallest domain, SD(d) : choose the dual variable with the smallest domain, and choose the values in numeric order.

Double smallest domain, SD^2(p+d) : choose the primal/dual variable with the smallest domain, and choose the value whose dual/primal variable has the smallest domain.

Primal double smallest domain, SD^2(p) : choose the primal variable with the smallest domain, and choose the value whose dual variable has the smallest domain.

Dual double smallest domain, SD^2(d) : choose the dual variable with the smallest domain, and choose the value whose primal variable has the smallest domain.

The idea of using the smallest domain heuristic on the dual as a value ordering heuristic can be traced at least as far back as [7]. It was also used in [2, 3]. We shall now argue that the variable and value ordering provided by the double smallest domain heuristics is consistent with the fail first principle for variable ordering and the succeed first for value ordering. Barbara Smith in a personal communication to the authors made a similar argument. Suppose we assign the primal value k to the primal variable X (an analogous argument can be given if we branch on a dual variable). Constraint propagation will prune the primal value k from the other primal variables, and the dual value X from the other dual variables. Of course, constraint propagation may do more than this if we have an all-different constraint or channelling constraints. However, to a first approximation, this is a reasonable starting point. Geelen’s succeed first value ordering heuristic computes the “promise” of the different values by multiplying together the domain sizes of the uninstantiated variables [1]. Each term in this product is constant if k and X do not occur in the domain and is reduced by 1 if k or X occurs in the domain. This is likely to be maximized by ensuring we reduce as few terms as possible. That is, by ensuring k and X occur in as few domains as possible. That is X and D_k have the smallest domains possible. Hence double smallest domain will tend to branch on the variable with smallest domain and assign it the value with most promise.

4 Problem domains

We will compare these different models and heuristics on the following collection of permutation problems. All the models are implemented in Solver 5.300, and are available at CSPLib.

Langford's problem: Given two integers n and m, Langford's problem is to permute n sets of numbers 1 to m, so that each appearance of the number i is i on from the last. This is **prob024** in CSPLib.

Quasigroup existence problem: An order n quasigroup is a Latin square of size n. That is, an $n \times n$ multiplication table in which each row and column is a permutation of the numbers 1 to n. Quasigroups existence problem determines the existence or non-existence of quasigroups of a given size with additional properties. QG3(n) denotes quasigroups of order n for which $(a * b) * (b * a) = a$. QG4(n) denotes quasigroups of order n for which $(b * a) * (a * b) = a$. Furthermore, we may additionally demand that the quasigroup is idempotent, i.e., $a * a = a$ for every element a. This is **prob003** in CSPLib.

Golomb rulers problem: A Golomb ruler has n marks arranged on the ticks of a ruler of length m such that the distances between any pair of marks are all distinct. This is **prob006** in CSPLib.

Sport scheduling problem: We want to schedule games between n teams over $n - 1$ weeks when n is even (n weeks when n is odd). Each week is divided into $n/2$ periods when n is even ($(n - 1)/2$ when n is odd). Each game is composed of two slots, "home" and "away", where one team plays home and the other team plays away. The objective is to schedule a game for each period of every week such that: every team plays against every other team; a team plays exactly once a week when we have an even number of teams, and at most once a week when we have an odd number of weeks; and a team plays at most twice in the same period over the course of the season. This is **prob026** in CSPLib.

Magic squares problem: An order n magic square is an n by n matrix containing the number 1 to n^2, with each row, column, and diagonal equal the same sum. This is **prob019** in CSPLib.

5 Experimental results

We now compare the different models and branching heuristics in an extensive set of experiments. The hypothesis we wish to test is that branching heuristics can profit from multiple viewpoints.

5.1 Langford's problem

The results are given in Table 1. We make a number of observations. The primal not-equals viewpoint ("$\neq$") gives the worst results (as it does in almost all the subsequent problem domains). We will not therefore discuss it further. The best

		L(3,12)		L(3,13)		L(3,14)		L(3,15)	
model	heuristic	fails	sec.	fails	sec.	fails	sec.	fails	sec.
$\neq$	SD(p)	62016	10.27	300800	53.72	1368322	272.03	7515260	1601.00
$\forall$	SD(p)	20795	3.59	93076	16.95	405519	78.18	2072534	414.71
c	SD(p+d)	11683	**2.16**	45271	**8.66**	184745	**36.46**	846851	**171.97**
c	SD(p)	21148	3.68	94795	16.84	412882	74.99	2112477	389.69
c	SD(d)	15214	2.64	59954	10.73	249852	46.39	1144168	221.01
c	SD^2(p+d)	11683	2.2	45271	9.04	184745	38.32	846851	180.00
c	SD^2(p)	20855	3.89	93237	17.07	406546	75.38	2077692	393.21
c	SD^2(d)	14314	2.62	56413	10.61	234770	45.68	1076352	213.51
$\forall c$	SD(p+d)	**11449**	2.84	**44253**	11.47	**180611**	48.71	**827564**	231.80
$\forall c$	SD(p)	20795	4.93	93076	22.61	405519	102.45	2072534	537.14
$\forall c$	SD(d)	14459	3.44	56701	13.94	234790	60.13	1069249	282.42
$\forall c$	SD^2(p+d)	11451	2.91	44254	11.72	180631	49.71	827605	235.56
$\forall c$	SD^2(p)	20488	4.98	91513	22.86	399092	103.09	2037159	540.04
$\forall c$	SD^2(d)	13639	3.38	53483	13.78	221307	59.33	1009250	278.32

Table 1. No. of backtracks (fails) and running time to find all solutions, or prove that there are no solutions, to four instances of Langford problem. Runtimes are for ILOG Solver 5.300 on 1200MHz, Pentium III processor, and 512 MB of RAM.

		QG3(6)		QG(7)		QG3(8)		QG3(9)	
model	heuristic	fails	sec.	fails	sec.	fails	sec.	fails	sec.
$\neq$	SD(p)	8	**0.01**	100	0.22	1895	8.46	83630	600.61
$\forall$	SD(p)	7	**0.01**	59	0.17	955	5.76	**35198**	385.57
c	SD(p+d)	7	0.02	63	**0.16**	1117	5.81	53766	463.40
c	SD(p)	7	0.02	59	0.17	1039	5.70	38196	373.38
c	SD(d)	6	**0.01**	54	0.19	888	**5.40**	46539	418.96
c	SD^2(p+d)	7	0.02	63	0.17	1117	5.83	53785	461.05
c	SD^2(p)	7	**0.01**	58	0.17	1043	5.68	38198	372.41
c	SD^2(d)	6	**0.01**	54	0.18	887	5.42	46741	419.94
$\forall c$	SD(p+d)	7	0.02	54	**0.16**	999	6.00	49678	474.82
$\forall c$	SD(p)	7	0.02	59	0.18	955	5.85	**35198**	376.06
$\forall c$	SD(d)	**5**	0.02	**52**	0.2	824	5.73	43278	438.81
$\forall c$	SD^2(p+d)	7	0.03	54	0.17	999	6.05	49702	477.04
$\forall c$	SD^2(p)	7	0.02	58	0.18	959	5.84	35201	**368.87**
$\forall c$	SD^2(d)	**5**	0.02	**52**	0.19	**823**	5.80	43452	432.89

Table 2. No. of backtracks (fails) and running time to find all solutions, or prove that there are no solutions, to four instances of QG3 problem. Runtimes are for ILOG Solver 5.300 on 1200MHz, Pentium III processor, and 512 MB of RAM.

runtimes are obtained with the channelling constraints and branching on the primal or dual variable with smallest domain. Being forced to branch on just the primal or dual tends to increase runtimes. The branching heuristic therefore profits from the multiple viewpoints. Note that maintaining GAC on the all-different constraint is neither the best startegy in terms of failures or runtimes. This is despite the fact that it has the strongest propagator. This model has only one viewpoint and this hinders the branching heuristic. Note also that the smallest search trees (but not runtimes) are obtained with models that combines the all-different constraint on the primal with the channelling constraints between the primal and dual. In such models, we have the benefits of the strongest propagator and a dual viewpoint for the branching heuristic. Finnally, note that the model with just an all-different constraint only has primal variables, and gives the same search tree as the model with the all-different constraint and channelling when it is forced to branch on just the primal variables. The pruning performed by the all-different constraint subsumes that performed by the channelling [3, 4].

5.2 Quasigroups

The quasigroups existence problem can be modelled as a multiple permutation problem with $2n$ intersecting permutation constraints. We introduce a variable for each entry in the multiplication table of the quasigroup. We then post permutation constraints on each row and column of variables. In Table 2, we give results for the QG3 family of problems. All the models and branching heuristics except the primal not-equals models are competitive. A dual viewpoint doesn't appear to offer much advantage, but it also does not hurt. In Table 3, we give results for the QG4 family of problems. All the models and branching heuristics are again competitive except the primal not-equals models and those models which force the branching heuristic to branch on a dual variable. Note also that the best runtime on the largest problem is with the double smallest domain heuristic.

5.3 Golomb rulers

To model the Golomb rulers problem as a permutation problem, we introduce a variable for each pairwise distance between marks. Since we may have more values than variables, we introduce additional variables to ensure that there are as many variables as values. Geelen advocates such a construction in [1] as we can then post a permutation constraint on the enlarged set of variables. In Table 4, we give results for finding four optimal length rulers. Despite the fact that it has the strongest propagator, the primal all-different model is not competitive on the larger problems. The best runtimes are obtained with the channelling constraints and branching on the primal or dual variable with smallest domain. Being forced to branch on just the primal variables hurts the branching heuristic.

		QG4(6)		QG4(7)		QG4(8)		QG4(9)	
model	heuristic	fails	sec.	fails	sec.	fails	sec.	fails	sec.
$\neq$	SD(p)	6	**0.01**	82	0.23	1779	8.29	116298	843.26
$\forall$	SD(p)	**4**	**0.01**	**57**	**0.19**	**892**	5.12	52419	496.24
c	SD(p+d)	6	0.02	59	0.20	935	4.99	55232	489.89
c	SD(p)	6	**0.01**	59	0.20	931	4.92	55397	485.72
c	SD(d)	6	0.02	74	0.21	1266	7.59	83316	772.17
c	SD^2(p+d)	6	0.02	59	**0.19**	940	**4.81**	55264	**476.66**
c	SD^2(p)	6	**0.01**	59	**0.19**	936	4.87	55442	478.48
c	SD^2(d)	6	**0.01**	73	0.22	1267	7.37	82916	766.33
$\forall c$	SD(p+d)	**4**	0.02	57	**0.19**	900	5.19	**52045**	486.72
$\forall c$	SD(p)	**4**	0.02	57	0.20	**892**	5.29	52419	491.54
$\forall c$	SD(d)	**4**	0.02	67	0.21	1102	7.04	73997	745.09
$\forall c$	SD^2(p+d)	**4**	**0.01**	57	**0.19**	905	5.24	52077	491.45
$\forall c$	SD^2(p)	**4**	**0.01**	57	0.20	897	5.23	52463	493.70
$\forall c$	SD^2(d)	**4**	**0.01**	66	0.23	1104	7.02	73714	745.86

Table 3. No. of backtracks (fails) and running time to find all solutions, or prove that there are no solutions, to four instances of QG4 problem. Runtimes are for ILOG Solver 5.300 on 1200MHz, Pentium III processor, and 512 MB of RAM.

		Golomb(7,25)		Golomb(8,34)		Golomb(9,44)		Golomb(10,55)	
model	heuristic	fails	sec.	fails	sec.	fails	sec.	fails	sec.
$\neq$	SD(p)	912	0.15	5543	1.12	–	–	–	–
$\forall$	SD(p)	500	**0.11**	2949	**0.81**	–	–	–	–
c	SD(p+d)	606	0.12	3330	1.01	17002	**7.54**	72751	**49.14**
c	SD(p)	890	0.15	5343	1.25	–	–	–	–
c	SD(d)	626	0.12	3390	1.02	17151	7.55	73539	49.25
c	SD^2(p+d)	608	0.12	3333	1.03	17022	7.63	72853	49.37
c	SD^2(p)	928	0.17	5648	1.27	–	–	–	–
c	SD^2(d)	626	0.12	3390	1.03	17179	7.59	73628	49.59
$\forall c$	SD(p+d)	**493**	0.12	**2771**	1.10	**14313**	8.29	**61572**	54.63
$\forall c$	SD(p)	500	0.13	2949	1.08	–	–	–	–
$\forall c$	SD(d)	495	0.13	2782	1.10	14325	8.28	61616	54.46
$\forall c$	SD^2(p+d)	504	0.14	2787	1.1	14392	8.38	61898	54.94
$\forall c$	SD^2(p)	542	0.14	3258	1.12	–	–	–	–
$\forall c$	SD^2(d)	495	0.13	2794	1.11	14400	8.39	61893	54.97

Table 4. No. of backtracks (fails) and running time to find all solutions, or prove that there are no solutions, to four instances of Golomb rulers problem. Runtimes are for ILOG Solver 5.300 on 1200MHz, Pentium III processor, and 512 MB of RAM. A dash means that no results were returned after 1 hour.

		Sport(6)		Sport(8)		Sport(10)		Sport(12)	
model	heuristic	fails	sec.	fails	sec.	fails	sec.	fails	sec.
$\neq$	SD(p)	**0**	**0.00**	1248	0.22	1863275	397.70	5777382	1971.92
$\forall$	SD(p)	**0**	0.01	566	0.15	1361686	350.92	3522705	1444.44
c	SD(p+d)	624	0.09	4	**0.01**	**7**	**0.03**	5232	**1.78**
c	SD(p)	**0**	**0.00**	566	0.14	1376143	355.99	3537447	1368.84
c	SD(d)	589	0.07	**3**	**0.01**	336	0.07	6368	1.9
c	SD^2(p+d)	7	**0.00**	9	**0.01**	1112	0.30	46122	18.4
c	SD^2(p)	113	0.02	6601	0.94	820693	168.91	–	–
c	SD^2(d)	514	0.06	43	**0.01**	7028	1.58	6252	2.29
$\forall c$	SD(p+d)	624	0.10	4	**0.01**	**7**	**0.03**	**5190**	1.98
$\forall c$	SD(p)	**0**	0.01	566	0.16	1361686	372.10	3522705	1495.41
$\forall c$	SD(d)	589	0.09	**3**	**0.01**	329	0.08	6262	2.18
$\forall c$	SD^2(p+d)	7	**0.00**	9	**0.01**	1102	0.35	45125	20.98
$\forall c$	SD^2(p)	113	0.02	6563	1.09	812696	186.23	–	–
$\forall c$	SD^2(d)	514	0.07	43	0.02	6920	1.76	6129	2.55

Table 5. No. of backtracks (fails) and running time to find first solution to four instances of sport scheduling problem. Runtimes are for ILOG Solver 5.300 on 1200MHz, Pentium III processor, and 512 MB of RAM.

		Magic(3)		Magic(4)		Magic(5)		Magic(6)	
model	heuristic	fails	sec.	fails	sec.	fails	sec.	fails	sec.
$\neq$	SD(p)	6	**0.00**	20	**0.00**	1576	0.11	–	–
$\forall$	SD(p)	**4**	**0.00**	19	**0.00**	1355	0.11	2748609	196.45
c	SD(p+d)	5	**0.00**	18	**0.00**	4637	0.37	–	–
c	SD(p)	**4**	**0.00**	20	**0.00**	1457	0.14	3448162	249.84
c	SD(d)	5	**0.00**	37	0.01	49312	4.61	–	–
c	SD^2(p+d)	5	**0.00**	10	**0.00**	555	0.06	**463865**	**37.41**
c	SD^2(p)	**4**	**0.00**	11	**0.00**	495	**0.05**	1648408	132.35
c	SD^2(d)	5	**0.00**	18	**0.00**	928217	86.07	–	–
$\forall c$	SD(p+d)	5	0.01	18	**0.00**	4436	0.48	–	–
$\forall c$	SD(p)	**4**	**0.00**	19	**0.00**	1355	0.17	–	–
$\forall c$	SD(d)	5	**0.00**	**5**	**0.00**	42426	5.33	–	–
$\forall c$	SD^2(p+d)	5	0.02	10	0.01	435	0.07	290103	39.01
$\forall c$	SD^2(p)	**4**	**0.00**	11	**0.00**	**355**	**0.05**	1083993	148.73
$\forall c$	SD^2(d)	5	**0.00**	16	**0.00**	919057	106.55	–	–

Table 6. No. of backtracks (fails) and running time to find the first solution to four instances of magic square problem. Runtimes are for ILOG Solver 5.300 on 1200MHz, Pentium III processor, and 512 MB of RAM. A dash means that no results were returned after 1 hour.

5.4 Sport scheduling

The sport scheduling problem is modelled as follows. The set of teams is $T = \{1, \ldots, n\}$ (we assume n is even), the set of weeks is $W = \{1, \ldots, n-1\}$, the set of periods is $P = \{1, \ldots, n/2\}$, and the set of slots ("home" and "away") are $S = \{1, 2\}$. A schedule is then a *bijection* from $P \times S$ into T for each week such that all the other constraints of the problem are satisfied. We report results in Table 5. Unlike the previous tables which report results to find all solutions, here we report results to find just the first solution. Despite this significant change in the experimental setup, we observe similar trends in our results. Even though it has the strongest propagator, the primal all-different model is again not competitive on the larger problems. The best runtimes are obtained with the channelling constraints and branching on the primal or dual variable with smallest domain. As with the Golomb ruler problem, being forced to branch on just the primal variables hurts the branching heuristic. Multiple viewpoints appear to offer the branching heuristic very significant advantages on this problem.

5.5 Magic squares

We model the order n magic square problem as a matrix model in which there is an n by n matrix of variables which take values from 1 to n^2. We then post permutation constraint on all the variables in such a matrix, and sum constraints on the rows, columns and diagonals. Results are given in Table 6, again to find the first solution. The best strategy is the double smallest domain heuristic on either the model with just channelling constraints, or on the model with channelling constraints and a primal all-different constraint. The former explores a larger search tree, but does so very slightly quicker than the later. We conjecture that the large domain sizes in this problem favour a branching heuristic like double smallest domain which chooses its values with care.

6 Conclusion

On permuation problems, branching heuristics can be significantly more effective when they look at both the primal and dual viewpoint. Indeed, branching on primal or dual variables was often more important to our results than using a stronger propagator. For example, the model that enforced GAC on an all-different constraint often gave worse performance both in runtime and search tree size compared to the model that enforced AC on the channelling constraints. With the later model, the branching heuristic was able to use the multiple viewpoints to make better branching decisions. We also studied the double smallest domain heuristic [7]. This branches on the primal/dual variable with smallest domain and assigns it the value whose dual/primal variable has the smallest domain. This makes decisions which are consistent with the fail-first principle for variable ordering and the succeed-first principle for value ordering. On some of our problem domains, it offered the best performance of all the heuristics studied.

What general lessons can be learnt from these experiments? First, we have strong support for the hypothesis that branching heuristics can profit from multiple viewpoints. Second, our experimental results suggest that you should not necessarily aim for more propagation. For instance, we usually saw better performance when we threw out the all-different constraint. Third, when we model, we need to think about both the heuristics and the propagation. It would be interesting in the future to study the benefits of multiple viewpoints for branching heuristics on problems other than permutations where there might not be such a natural dual viewpoint.

Acknowledgments

The authors are supported by the Science Foundation Ireland. They wish to thank the members of the APES research group.

References

1. Geelen, P.: Dual viewpoint heuristics for binary constraint satisfaction problems. In: Proceedings of the 10th ECAI, European Conference on Artificial Intelligence (1992) 31–35
2. Cheng, B., Choi, K., Lee, J., Wu, J.: Increasing constraint propagation by redundant modeling: an experience report. Constraints **4** (1999) 167–192
3. Smith, B.: Modelling a Permutation Problem. In: Proceedings of ECAI'2000 Workshop on Modelling and Solving Problems with Constraints. (2000) Also available as Research Report from http://scom.hud.ac.uk/staff/scombms/papers.html.
4. Walsh, T.: Permtuation problems and channelling constraints. In: Proceedings of 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2001). (2001)
5. Régin, J.C.: A filtering algorithm for constraints of difference in CSPs. In: Proceedings of the 12th National Conference on AI, American Association for Artificial Intelligence (1994) 362–367
6. Smith, B.: Succeed-first or Fail-first: a case study in variable and value ordering heuristics. In: Proceedings of the Third International Conference on Practical Applications of Constraint Programming (PACT-97). (1997) Available as Research Report 96.26, School of Computer Studies, University of Leeds.
7. Jourdan, J.: Concurrent constraint multiple models in CLP and CC languages: toward a programming methodology by modelling. In: Proceedings of the INFORMS Conference. (1995)

Preference Constraints: New Global Soft Constraints Dedicated to Preference Binary Relations

Rémy-Robert Joseph[1], Peter Chan[2], Michael Hiroux[1,2], Georges Weil[1,2]

[1] Université Joseph Fourier-Grenoble 1, Laboratoire IMAG/TIMC, Equipe SIC, Institut Albert Bonniot, 38706 La Tronche, France
[2] Equitime S.A., Grand sablon, 4 Avenue de l'Obiou 38700 La Tronche, France
Remy.Joseph@imag.fr

Abstract. In this article, we propose a new soft constraint called *preference constraint*, squaring well with the decision theory concept of *preference binary relation*. We show how to use it for designing complex hierarchical preference information based on preference binary relations for combinatorial problems. Finally, *preference-based constraint systems* are defined and associated best quality choice problems are introduced. This new model offers greater flexibility to represent and make complex decisions with computers.

1 Introduction

Until now, the privileged preference representation used for combinatorial problems has been the objective function. It is exclusively used at every aggregation level [7] of a hierarchical preference model, and has remarkable structural properties as transitivity and completeness. These properties are often judged too restrictive, because some important aggregation concepts are incomplete by definition, as efficiency, unanimity, equity [10], to quote only few of them. Attributes cannot be necessarily transitive because of uncertainty [12]. Obviously these properties are desirable for a collective choice, but we shall not make a fetish of them [10]. For all these reasons which are the rule in practical works, it is necessary to enlarge objective function-based preference models toward weakly structured aggregation rules. In this work, we extend preference models to aggregation rules based on preference binary relations. Often used in multi-criteria decision aiding (MCDA) and social choice theory (SCT)[1], for their abilities of preference modelling and decision aiding, preference binary relations stay yet almost non-existent in combinatorial (and continuous) optimization[2]. Both interactivity and weaker preference models ("bounded" rationality) are necessary to improve the decision-making.

[1] MCDA [12] attends to evaluation of practical problems having solutions given in extension, and SCT [10] points out theoretical works on the characterization of adequate aggregation rules for collective and public problems.

[2] Note nevertheless that multi-objective mathematical programming (MOMP) has allowed going beyond classical optimization models by admitting the transitive incomplete Pareto dominance as final aggregation rule. Others as the lexicographic, the maximin and the

This article is outlined as follows: After a review of basic notions in constraint programming in section 2, we introduce preference constraints as a way to represent preference binary relations (section 3). The 4th section is devoted to preference binary relations obtained from aggregation rules for which preference constraints are adapted. Finally, preference-based constraint systems are introduced in 5.

2 Background in Constraints

A *finite constraint system CS* is defined as a set of *n variables* $V = \{v_1, \ldots, v_n\}$, a *finite domain* $D(v)$ of possible values for each variable $v \in V$, and a set of *m constraints* $C = \{c_1, \ldots, c_m\}$ among variables. A constraint *c* is characterized by a set of variables $V(c) \subseteq V$ and a feasible solution set $S(c)$ included in the Cartesian product of domains associated to $V(c)$: $S(c) \subseteq \mathcal{D}_{V(c)} = \times_{v \in V(c)} D(v)$. An element of $\mathcal{D}_{V(c)}$ is called a *solution* of *c*, and an element of $\mathcal{D}_V = D(v_1) \times \ldots \times D(v_n)$ is called a *solution* of the constraint system *CS*. A value *d* of a variable *v* ($\Leftrightarrow d \in D(v)$) is *consistent with* a constraint *c* iff $v \notin V(c)$, or there exists a feasible solution *x* of *c* such that $x(v) = d$. Otherwise, *d* of *v* is said *inconsistent with c*. As from this definition, it is possible to define different kind of consistency properties on variables, constraints and constraint systems[3]. A variable *v* is *consistent with* a constraint iff $D(v)$ is not empty and all its values are consistent with *c*. A constraint *c* is *globally inverse consistent* iff for all *v* in $V(c)$, *v* is consistent with *c*. Given a constraint *c* and a set of domains *D* associated to variables $V(c)$, a *filtering algorithm* for *c* is an algorithm establishing a consistency level for *c*. A constraint provided with an adjusted filtering algorithm is called a *global constraint* ([9], [8]).

A *combinatorial constraint satisfaction problem* (CSP) on a given finite constraint system $CS = (V, D, C)$ is concerned with the search for an element *x* of $\mathcal{D}_V$ such that for all constraints $c \in C$, the projection[4] of *x* on $V(c)$ is consistent with *c*. Such a solution *x* is called a *feasible solution* of the constraint system and the set of all feasible solutions of a constraint system *CS* is noted $S_{\text{CSP}}(CS)$.

3 Preference Binary Relations and Preference Constraints

After defining the preference binary relation, we present an adequate model to represent preference binary relations on constraint systems (i.e. Cartesian product sets).

leximin rules, have been used to synthesize objective functions. Although transitive and complete these aggregation rules are not representable by objective functions (see references given in [10]).

[3] For further information, see [11], [2], among others.

[4] Projection of $x \in \mathcal{D}_V$ on $V1 \subseteq V$ is the element x_1 of $\mathcal{D}_{V1}$ such that $x_1(v) = x(v)\ \forall\ v \in V1$.

3.1 Background in Decision Theory

There exists different ways of modelling preferences [12]. This article is devoted to preference binary relations, defined here:

A *binary relation* $\succcurlyeq$ on a set S is a subset of the Cartesian product $S \times S$. We will note here $x \succcurlyeq y$ instead of $(x, y) \in \succcurlyeq$, and not($x \succcurlyeq y$) to designate $(x, y) \notin \succcurlyeq$.

Given a set of solutions S, a *preference binary relation* $\succcurlyeq$ of an individual on S is a reflexive binary relation on S ($\Leftrightarrow x \succcurlyeq x$, for all $x \in S$) traducing the judgments of this individual concerning his preferences between the pairs of solutions. The assertion "$x \succcurlyeq y$" means "x is at least as good as y for the considered individual" for any solutions x and y of S. A preference binary relation $\succcurlyeq$ makes a partition of $S \times S$ into four fundamental binary relations called *fundamental attitudes*. Here is their definition:

(indifference)	$x \simeq y \Leftrightarrow (x \succcurlyeq y$ and $y \succcurlyeq x)$ for any $x, y \in S$
(strict preference)	$x \succ y \Leftrightarrow (x \succcurlyeq y$ and not($y \succcurlyeq x$)) for any $x, y \in S$
(strict aversion)	$x \prec y \Leftrightarrow y \succ x$ for any $x, y \in S$
(incomparability)	$x \parallel y \Leftrightarrow$ (not($x \succcurlyeq y$) and not($y \succcurlyeq x$)) for any $x, y \in S$

A preference binary relation can be also interpreted as a mapping from $S \times S$ to $AF = \{\simeq, \succ, \prec, \parallel\}$ with AF the set of fundamental attitudes. The set $\mathcal{PR}(AF)$, made up of elements of the power set of AF different from the empty set and AF, is called the *set of attitudes*.

3.2 Preference Modelling, Soft Global Constraints and Preference Constraints

In an explicit solution set environment, preferences are often explicitly represented [12]. But the implicit formulation of solutions $\mathcal{D}_V$ and feasible solutions $S_{\mathrm{CSP}}(\mathcal{CS})$ makes this way of modelling inconceivable. In constraint programming, preference representations have taken shape in *soft constraints*. Interesting soft constraints have been used in the frameworks of valued constraints systems and semiring-based constraint systems [1]. But they are limited to semiring structures on valuations. Otherwise, two kinds of preference models have handled global constraints: (*a*) *property constraints* dedicated to relevant basic properties which can be or not satisfied by a feasible solution and (*b*) *objective function constraints* devoted to objective functions by way of constraints. Such soft constraints are called *soft global constraints* ([8], [9]). To fill the gap about soft global constraints dedicated to preference binary relations, we present the preference constraints:

A preference binary relation $\succcurlyeq$ can be described by a set of constraints $c_{\succcurlyeq}[\alpha, x]$ parameterized by a solution x and an attitude α. By noting V the variable set and D the domain set on which scope the constraints $c_{\succcurlyeq}[\alpha, x]$, then the set $\{c_{\succcurlyeq}[\alpha, x], \forall (\alpha, x) \in \mathcal{PR}(AF) \times \mathcal{D}_V\}$ is called the *preference constraint* associated to the preference binary relation $\succcurlyeq$. For short, we will note $\{c_{\succcurlyeq}[\alpha, x]\}_{\alpha, x}$.

The feasible set of $c_{\succcurlyeq}[\alpha, x]$ is noted $S(c_{\succcurlyeq}[\alpha, x]) = \{y \in \mathcal{D}_V$ such that: $y\ \alpha_{\succcurlyeq}\ x\}$, with $\alpha_{\succcurlyeq}$ indicating the attitude α of the preference binary relation $\succcurlyeq$. In a digraph context, $S(c_{\succcurlyeq}[\alpha, x])$ describes the neighborhood of x in the set $\mathcal{D}_V$ according to the binary relation α. This modelling of the preference binary relation offers large perspectives in solving problems, as we will see in the following.

4 Aggregation and Preference Constraints

We show here possibilities offered by preference constraints in the building of complex hierarchical preference models.

4.1 Aggregation Rules and Preference Models

In complex real world problems, the evaluation of solutions can be done from several persons or/and from several viewpoints for each person. This preference information is methodically synthesized with several aggregations rules ([7], [3]) in order to obtain a collective preference binary relation representing the *preference model* of the problem.

Here, the term *individual* designates a human, a group, a society or someone's viewpoint; and $I = \{1, \ldots, n\}$ points out a set of individuals. From now $pref(i, S)$ refers to either the objective function f_i or the preference binary relation $\succcurlyeq_i$ of the individual i on the set S. The component of a preference model, allowing to synthesize preferential information, is the aggregation rule. An *aggregation rule* is a functional relation AR such that for any set of n individual preferences $pref(1, S), \ldots, pref(n, S)$ (one for each individual), one and only one collective preference $pref(I, S)$ is determined, $pref(I, S) = AR(pref(1, S), \ldots, pref(n, S))$.
As examples we mention: the weighted sum function ([4], [7], [12]), the majority method ([10], [12]) and the lexicographic rule ([4], [10], [12]).

4.2 Preference Constraints for Aggregation Rules.

The semantics of a preference constraint can be defined as an aggregation rule allowing preference binary relations and objective functions as individual preferences and a binary relation as collective preference. To each aggregation rule AR is associated one preference constraint noted $\{c_{AR}[\alpha, z]\}_{\alpha, z}$ or $\{c_{\succcurlyeq}[\alpha, z]\}_{\alpha, z}$, if $\succcurlyeq$ is the collective preference binary relation of AR. Individual preferences are noted $\{c_i[\alpha, z]\}_{\alpha, z}$ for any individual $i \in I = \{1, \ldots, n\}$. The variable set of $\{c_{AR}[\alpha, z]\}_{\alpha, z}$ is equal to the union of individual preferences variable sets. Whereas the variable set, the feasible solution set of $\{c_{AR}[\alpha, z]\}_{\alpha, z}$ is parameterized by an attribute and a solution. Here is their definition:

$$V(\{c_{AR}[\alpha, z]\}_{\alpha, z}) = V(\{c_1[\alpha, z]\}_{\alpha, z}) \cup \ldots \cup V(\{c_n[\alpha, z]\}_{\alpha, z})$$

$$S(c_{AR}[\alpha, x]) = \{y \in \mathcal{D}_{V(AR)} \text{ such that: } y\ \alpha_{AR}\ x\} \quad \forall\ (\alpha, x) \in PR(AF) \times \mathcal{D}_{V(AR)}\ .$$

with $\mathcal{D}_{V(AR)}$ the Cartesian product of domains $D(v)$ for all $v \in V(\{c_{AR}[\alpha, z]\}_{\alpha, z})$ and α_{AR} the attribute α associated to the collective preference of AR. Any filtering algorithm for $c_{AR}[\alpha, x]$ has to use only the elements $c_i[\alpha_1, z]$, $\forall\ (\alpha_1, z) \in PR(AF) \times \mathcal{D}_{V(AR)}$ in order to keep their generality. But for algorithmic efficiency they can be specialized.

Preference constraints devoted to aggregation rules give a recursive definition of preference constraints. They are components of a preference model. Like the cardinality operator [13], they are abstractions; which argue in favor of their modelling power.

5 Preference-Based Combinatorial Choice Problems

The instance of a combinatorial problem is made up of two parts. The first one, the feasibility model, describes feasible solutions by way of constraints and variables. The second one, the preference model, describes all the information necessary to compare solutions (different actors' viewpoints, etc). Preference constraints can be used to design the hierarchical preference model of some constraint-based choice problem instances. Thus, a *preference-based constraint system* is a couple (CS, $\{c_{\succcurlyeq}[\alpha, z]\}_{\alpha, z}$), where CS is a constraint system describing the set of solutions and feasible solutions, and $\{c_{\succcurlyeq}[\alpha, z]\}_{\alpha, z}$ is a preference constraint possibly defined recursively.

Several choice problems can be defined from a preference-based constraint system. For example a problem searching one best quality solution and giving some indications on the quality of the returned solution (optimality, maximality or only feasibility):

Preference-based combinatorial constraint search problem (P-CCSP): Given a preference-based constraint system (CS, $\{c_{\succcurlyeq}[\alpha, z]\}_{\alpha, z}$), returns one optimal solution x ($\Leftrightarrow x \in S_{\text{CSP}}(CS)$ and $\forall\, y \in S_{\text{CSP}}(CS)$, $x \succcurlyeq y$) with the label "optimal", if such a solution exists, else returns one maximal solution x ($\Leftrightarrow x \in S_{\text{CSP}}(CS)$ and $\forall$ $y \in S_{\text{CSP}}(CS)$, not($y \succ x$)) with the label "maximal", if such a solution exists, otherwise returns a feasible solution with the label "feasible", if such a solution exists, else returns "no".

Partial problems can be defined from P-CCSP, by only returning for example one maximal solution or else "no", etc. Next, these problems can be specialized according to properties of $\succcurlyeq$. In this way, the partial preorder-based combinatorial constraint search problem (PPO-CCSP) is defined. This latter problem returns either an optimal solution, or a maximal solution, or "no", because the existence of a feasible solution certifies at least the existence of a maximal solution (see [12]).
Solving a preference-based combinatorial problem is not limited to finding one maximal solution [12], if such solutions exist. It's necessary, in the general case, to propose algorithmic tools exploring the whole maximal (or optimal) set. Interactive tools are very well adapted for these kinds of tasks. Sometimes, when we have any guaranty on the size of such a set, and that the problem ventures to do it, the problem of generating all solutions of a whole maximal set can be envisaged. We call this enumerating version of P-CCSP, the preference-based combinatorial constraint choice problem (P-CCP). In the same way, the specialized version PPO-CCP can be defined.

A great amount of work have been carried on the search for a maximal solution of a transitive preference binary relation $\succcurlyeq$ by way of an objective function (see [4], [12], for review). For this goal, it is necessary to identify some objective functions having their optimal set included in this of ($S_{\text{CSP}}(CS)$, $\succcurlyeq$). On the other hand, global constraints give us the possibility to build a complex instance and then to solve it, without going through this theoretical identification. This possibility of customization of the preference models opens great perspectives. Recently, Gavanelli [5] presented two Branch-and-Bound-based algorithms solving the PPO-CCP, by using the particular case of $\alpha = \{\succ, \|\}$ of preference constraints. Afterwards, he designed a filtering algorithm for this partial preference constraint associated to the Pareto dominance aggregation rule [6].

6 Conclusions and Perspectives

This article gives a general framework to design and solve by way of constraint programming, combinatorial problems allowing complex preference models based on preference binary relations. It allows designing preference binary relations at an individual, intermediary and global level in preference models, conceding thus more importance to preference elicitation. Thus new soft global constraints, called preference constraints, and new combinatorial choice problems, called preference-based constraint problems, have been introduced. One future work leading from this approach is the building of filtering algorithms for different aggregation rules.

References

1. Bistarelli S., Montanari U., Rossi F., Schiex T., Verfaillie G. et Fargier H.: Semiring-Based CSPs and Valued CSPs: Frameworks, Properties, and Comparison. Constraints : An International Journal 4 (3), (1999) 199–240
2. Cooper M. C.: An Optimal *k*-Consistency Algorithm. Artif. Intelligence 41 (1989) 89–95
3. Corner J., Buchanan J., Henig M.: Dynamic Decision Problem Structuring. Journal of Multi-Criteria Decision Analysis 10 (2001) 129–141
4. Ehrgott M., Gandibleux X.: An Annotated Bibliography of Multiobjective Combinatorial Optimization. OR Spektrum 22 (2000) 425–460
5. Gavanelli M.: Interactive Constraint Satisfaction Problems for Artificial Vision. Ph.D. thesis of the University of Ferrara, Italy (2002)
6. Gavanelli M.: An Implementation of Pareto Optimality in CLP(FD). Proceedings of the CP-AI-OR’02 (2002) 49–64
7. Keeney R. L., Raïffa H.: Decisions with Multiple Objectives: Preferences and Value Tradeoffs. John Wiley & Sons, New York (1976)
8. Régin J. C., Petit T., Bessière C., Puget J. F.: New Lower Bounds of Constraints Violations for Over-Constrained Problems. In: Walsh T. (ed.): Proceedings CP’2001. Lecture Notes in Computer Science, Vol. 2239. Springer-Verlag, Berlin Heidelberg New York (2001) 332–345
9. Régin J.-C., Rueher M.: A Global Constraint Combining a Sum Constraint and Difference Constraints. In Dechter R. (ed.): Proceedings CP’2000. Lecture Notes in Computer Science, Vol. 1894. Springer-Verlag, Berlin Heidelberg New York (2000) 384–395
10. Sen A. K.: Collective Choice and Social Welfare. Series in Advanced textbooks in economics, vol. 11, Elsevier science publishers, Netherlands (1970)
11. Verfaillie G., Martinez D., Bessière C.: A Generic Customizable Framework for Inverse Local Consistency. In Proceedings of AAAI\IAAI’99, 18-22 july 1999, Orlando, Florida, USA, AAAI Press / The MIT Press (1999) 169–174
12. Vincke, P.: Multicriteria Decision-Aid. John Wiley & Sons (1992)
13. van Hentenryck P., Deville Y.: The Cardinality Operator : A New Logical Connective for Constraint Logic Programming. Logic Programming : Proceedings of the 8th International Conference (ICLP’91), MIT Press, Cambridge, Massachusetts (1991) 745–759

A Regular Language Membership Constraint for Sequences of Variables

Gilles Pesant[1,2]

[1] Centre for Research on Transportation,
Université de Montréal, C.P. 6128, succ. Centre-ville, Montreal, H3C 3J7, Canada
[2] École Polytechnique de Montréal, Montreal, Canada
pesant@crt.umontreal.ca

Abstract. For constraint programming to become widely used as a problem solving tool, it must be expressive, powerful, and accessible. Global constraints are an important step in that direction. This paper describes a global constraint on a finite sequence of variables requiring that the corresponding sequence of values taken by these variables belong to a given regular language, thereby generalizing some other known global constraints. We describe and analyze a filtering algorithm achieving generalized arc consistency. To be truly accessible, the constraint would probably benefit from a few, more natural interfaces as domain specific front-ends to regular expressions.

1 Introduction

For constraint programming (CP) to become widely used as a problem solving tool, it must be expressive, powerful, and accessible. Global constraints are an important step in that direction: they represent substructures commonly found in certain problems; they encapsulate efficient algorithms to reason about these substructures and about the rest of the problem through shared variables; their semantics is often formulated in the language of an application area.

This paper describes a global constraint on a finite sequence of variables requiring that the corresponding sequence of values taken by these variables belong to a given regular language. One finds such a substructure, for example, in rostering problems.

We briefly recall regular expressions, regular languages, and their connection with automata theory (the interested reader may consult, for example, [5]). An alphabet Σ is a finite set of symbols. A string over an alphabet is a finite sequence of symbols from that alphabet. The (infinite) set of all strings over Σ is denoted by $\Sigma^\star$. Any subset of $\Sigma^\star$ is called a language. Deciding whether a particular string belongs to a given language is easier for some languages than for others.

A regular expression over Σ is built from Σ and the symbols "(", ")", "ϵ", "+", and "$\star$", according to the following recursive definition:

- ϵ (the empty string) and each member of Σ is a regular expression;
- if α and β are regular expressions then so is $(\alpha\beta)$;

- if α and β are regular expressions then so is $(\alpha + \beta)$;
- if α is a regular expression then so is $\alpha^\star$.

Every regular expression represents a regular language in $\Sigma^\star$, according to the interpretation of "+" and "$\star$" as set union and Kleene star, respectively.

A deterministic finite automaton (DFA) may be described by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ is an alphabet, $\delta : Q \times \Sigma \to Q$ is a partial transition function, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final (or accepting) states. Given an input string, the automaton starts in the initial state q_0 and processes the string one symbol at a time, applying the transition function δ at each step to update the current state. The string is accepted if and only if the last state reached belongs to the set of final states F. The languages recognized by DFA's are precisely regular languages. Building a DFA from a regular expression can be done in linear time [2].

Definition 1 (regular language membership constraint). *Let Σ denote some alphabet, ρ a regular expression over Σ, and X a sequence of finite-domain variables $\langle x_1, x_2, \ldots, x_n\rangle$ with respective domains $D_1, D_2, \ldots, D_n \subseteq \Sigma$. Under a* regular language membership constraint `regular(`X, ρ`)`*, any sequence of values taken by the variables of X must belong to the regular language described by ρ.*

Example 1. Consider a sequence X of five variables with $D_i = \Sigma = \{a, b, c\}$, $1 \leq i \leq 5$ and $\rho = aa^\star bb^\star aa^\star + c^\star$. Under constraint `regular(`$X, \rho$`)`, assigned sequences $\langle a, a, b, b, a\rangle$ and $\langle c, c, c, c, c\rangle$ are valid but $\langle a, b, b, b, c\rangle$ and $\langle a, a, b, b, b\rangle$ are not.

Such a definition may appear restrictive since it only allows the strings in the regular language that have a given length n. But because that constraint, like most other global constraints, is destined to be integrated into a larger model where other constraints are present, it is a much better choice to use a fixed-length sequence of variables each ranging over a finite alphabet than a single variable ranging over an infinite set of strings of arbitrary length.

In CP, combinatorial problems are typically modeled with several finite-domain variables. Constraints are then expressed on different subsets of these variables and cooperate through shared variables. Consider for example a rostering problem modeled with a two-dimensional array of variables, each taking their value from the set of possible activities (different work shifts, day off, training, and so forth). Each row, made up of seven variables, represents a week and each column, a day of the week. There could be a `regular` constraint on each individual row to indicate valid patterns of activities during a week and another `regular` constraint on the column for Monday to limit the number of Monday evening work shifts in a row. There could also be a cardinality constraint on weekend variables to ensure enough weekends off, and so on. Breaking up the roster into single-day chunks allows us to express the previous constraints directly from the original variables.

The next section presents related work on the integration of regular languages and patterns in CP. Section 3 describes the data structures and the filtering algorithm encapsulated in the regular language membership constraint. Section 4 analyses the time and space complexity of the algorithm and argues that it achieves generalized arc consistency. Section 5 shows how some important global constraints are special cases of this one.

2 Related Work

In the literature, the study of regular language membership constraints has usually involved variables ranging over an infinite set of strings. We do not insist on such works and only mention them if they are embedded in CP systems. We rather concentrate on membership constraints for sequences of finite-domain variables.

The constraint logic programming language CLP($\Sigma^\star$) [10] introduced membership constraints on regular languages described by regular expressions, of the form "`X in` ρ", where `X` is a single variable. One important difference with the work described in this paper is that the domain of the variables is an infinite set, the strings in $\Sigma^\star$, as opposed to a finite set of symbols from Σ in our case. Its constraint solver is based on a flexible constraint scheduling strategy to ensure termination (which is cause for concern with infinite domains) and on a collection of deduction rules to determine the satisfaction of individual constraints.

Other constraint logic programming languages have offered some support to reason about sequences of symbols. PROLOG III [4] features equations on lists. Lists are built from constants, list variables and a concatenation operator. A length operator is also defined. CLP($\mathcal{S}$) [9] manipulates equations on strings, built very much like PROLOG III's lists except that each string variable has a length parameter associated to it. The constraint solver exploits equalities and inequalities on these lengths to speed up the rule-based unification algorithm used for string equations.

Some constraints enforcing patterns of values have also been defined for finite-domain variables. ILOG Solver's `IlcTableConstraint` [6] takes as main arguments a sequence of n finite-domain variables and a set of n-tuples representing the valid assignments of values to these variables. For sets described by a regular expression, as in our case, the number of n-tuples would usually need to be very large and this possibly reflects on the computational complexity of the constraint. CHIP's `sequence` constraint [3] is a very expressive global constraint on the number of times a certain pattern of length ℓ appears in a sequence of variables. Patterns are expressed as sums and cardinalities of values taken by some of the ℓ variables in the subsequence. The constraint is not designed for (regular) patterns over the whole sequence.

Interestingly, recent work on a lexicographic ordering constraint uses a specific DFA to build its filtering algorithm [1].

3 The Consistency Algorithm

The idea behind the consistency algorithm for the `regular` constraint is to build an automaton equivalent to the regular expression given as argument. The sequence X with the current domains of its variables is then processed by the automaton in a two-step forward-backward manner, collecting for each variable-value pair a set of states that constitute its support. The sets are updated as the domains change and whenever one of these sets becomes empty, a value becomes unsupported and can be removed from a domain.

Let $\Sigma = \{v_1, v_2, \ldots, v_m\}$. The two-step process is best seen as constructing a layered directed multigraph $(N^1, N^2, \ldots, N^{n+1}, A)$ where each layer $N^i = \{q_0^i, q_1^i, \ldots, q_{|Q|-1}^i\}$ contains a different node for each state of the DFA and arcs only appear between consecutive layers. Arcs are related to variable-value pairs: an arc from q_k^i to q_ℓ^{i+1} is admissible for inclusion in A only if there exists some $v_j \in D_i$ such that $\delta(q_k, v_j) = q_\ell$. The arc is labeled with the value v_j allowing the transition between the two states. Note that it is enough, given a variable-value pair, to store an arc as the origin state: the destination state can be obtained by one application of the transition function. Accordingly, we maintain sets of states Q_{ij} to stand for the arcs related to variable-value pair (x_i, v_j).

Set Q_{ij} also acts as the support for variable x_i taking value v_j. Multiset A will contain exactly the admissible arcs that belong to a path from q_0 in the first layer to a member of F in the last layer. When this condition is met for some arc related to variable-value pair (x_i, v_j), we are sure that there is at least one sequence of values built from the current domains of X, and with value v_j for variable x_i, that belongs to the regular language. As long as Q_{ij} is not empty, we know that one such arc exists.

Given the regular expression ρ and the sequence X of variables (with their respective domains) in the statement of the constraint, Algo. 1 first builds the DFA and then the layered digraph by initializing our main data structure, the Q_{ij} sets. In addition, the in- and out-degrees of the nodes are maintained in $\mathit{indeg}[i+1][k]$ and $\mathit{outdeg}[i][k]$, $1 \le i \le n,\ 0 \le k \le |Q|-1$. All of these data structures are restorable upon backtracking. In the first phase, candidate arcs are collected by reaching forward from the initial state q_0 in the first layer using the domains of the x_i's. In the second phase, the arcs collected during the first phase are only kept if they can be reached backward from a state of F in the last layer. Finally, domains are possibly filtered if some Q_{ij}'s are empty.

Algorithm 2 is executed whenever some value v_j is removed from the domain of variable x_i in the course of the computation. Removing a value corresponds to removing one or several arcs in the digraph (as many as there are supporting states in Q_{ij}). The in- and out-degrees of the corresponding nodes must then be updated accordingly (Algo. 3 and 4). If some degree reaches zero, the node no longer belongs to a path from q_0 in the first layer to a member of F in the last layer: that information is propagated along such former paths going through that node and other domains are possibly filtered.

```
procedure initialize():
$(Q, \Sigma, \delta, q_0, F) \leftarrow \rho$; {*build the DFA from regular expression $\rho$*}
{*clear the data structures*}
for all $i \in \{1, 2, \ldots, n\}$ do
  for all $j \in \{1, 2, \ldots, m\}$ do
    $Q_{ij} := \emptyset$;
  for all $k \in \{0, 1, \ldots, |Q| - 1\}$ do
    $outdeg[i][k] := 0$;
    $indeg[i+1][k] := 0$;
  $N_{i+1} := \emptyset$;
{*forward phase: accumulate*}
$N_1 := \{q_0\}$;
for $i := 1$ to $n$ do
  for all $v_j \in D_i,\ q_k \in N_i$ do
    if $\delta(q_k, v_j)$ is defined then
      $Q_{ij} := Q_{ij} \cup \{q_k\}$; {*state $q_k$ is a candidate for support*}
      $N_{i+1} := N_{i+1} \cup \{\delta(q_k, v_j)\}$;
{*backward phase: validate*}
$N_{n+1} := N_{n+1} \cap F$;
for $i := n$ downto 1 do
  for all $q_k \in N_i$ do
    $mark[k] :=$ false;
  for all $v_j \in D_i,\ q_k \in Q_{ij}$ do
    if $\delta(q_k, v_j) \in N_{i+1}$ {*state $q_k$ confirmed as support*} then
      $outdeg[i][k] := outdeg[i][k] + 1$;
      $indeg[i+1][\delta(q_k, v_j)] := indeg[i+1][\delta(q_k, v_j)] + 1$;
      $mark[k] :=$ true;
    else
      $Q_{ij} := Q_{ij} \setminus \{q_k\}$;
  for all $q_k \in N_i$ do
    if $mark[k] =$ false then
      $N_i := N_i \setminus \{q_k\}$;
{*clean up the domains*}
for all $i \in \{1, 2, \ldots, n\},\ j \in \{1, 2, \ldots, m\}$ do
  if $Q_{ij} = \emptyset$ then
    remove value $v_j$ from domain $D_i$;
```

Algorithm 1: Upon the constraint being posted, build the digraph to initialize the data structures.

```
procedure propagate($i, j$):
for all $q_k \in Q_{ij}$ do
  decrement_outdeg($i, k$);
  decrement_indeg($i + 1, \delta(q_k, v_j)$);
$Q_{ij} := \emptyset$;
```

Algorithm 2: Upon v_j being removed from D_i in the course of the computation, update the data structures and filter out inconsistent values.

```
procedure decrement_outdeg($i, k$):
$outdeg[i][k] := outdeg[i][k] - 1$;
if $outdeg[i][k] = 0$ and $i > 1$ then
  for all $v_j \in D_{i-1}$ do
    for all $q_\ell \in Q_{i-1,j}$ do
      if $\delta(q_\ell, v_j) = q_k$ then
        $Q_{i-1,j} := Q_{i-1,j} \setminus \{q_\ell\}$;
        decrement_outdeg($i - 1, \ell$);
    if $Q_{i-1,j} = \emptyset$ then
      remove value $v_j$ from domain $D_{i-1}$;
```

Algorithm 3: Decrementing the out-degree of node q_k^i.

```
procedure decrement_indeg($i, k$):
$indeg[i][k] := indeg[i][k] - 1$;
if $indeg[i][k] = 0$ and $i < n$ then
  for all $v_j \in D_{i+1}$ do
    if $q_k \in Q_{i+1,j}$ then
      $Q_{i+1,j} := Q_{i+1,j} \setminus \{q_k\}$;
      decrement_indeg($i + 1, \delta(q_k, v_j)$);
    if $Q_{i+1,j} = \emptyset$ then
      remove value $v_j$ from domain $D_{i+1}$;
```

Algorithm 4: Decrementing the in-degree of node q_k^i.

Example 2. Consider a sequence X of five variables with $D_1 = \{a, b, c, o\}$, $D_2 = \{b, o\}$, $D_3 = \{a, c, o\}$, $D_4 = \{a, b, o\}$, and $D_5 = \{a\}$. Between a's and b's, a's and c's, or b's and c's, there should be at least one o. Furthermore, a's followed by o's followed by c's is not allowed, and neither are b's followed by o's followed by a's nor c's followed by o's followed by b's. Figure 1 gives the corresponding DFA. (In the figures, we identify the states as integers, for readability.) Algorithm 1 would build the digraph of Fig. 2, where the arcs are shown in bold (the ones not in bold were discarded during the backward phase). As a result, value b is removed from D_2 and D_4.

4 Analysis

The worst-case running time of Algo. 1, called once, is dominated by the construction of the digraph and is in $\mathcal{O}(nm|Q|)$. Its space complexity is in $\mathcal{O}(nm|Q|)$ as well. An analysis of Algo. 2, called anytime a domain is modified, puts its worst-case running time in $\mathcal{O}(m|Q|)$ per arc removed from the graph, which is a better way to evaluate the amount of work performed at each call. Better data structures could probably bring down this time complexity. Also, because the number of states influences both the time and space complexity, it is desirable

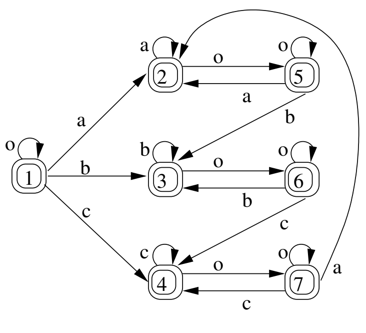


Fig. 1. A deterministic finite automaton.

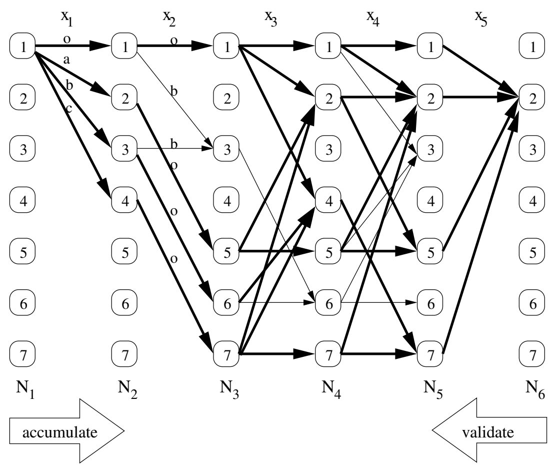


Fig. 2. The layered directed graph (v_j labels are only shown on the arcs between the first three layers, for clarity).

to compute the minimum state DFA. This can be achieved in $\mathcal{O}(|Q|\log|Q|)$ time [11].

We now argue that the filtering achieved is complete. By construction of the layered digraph, each arc exactly corresponds to a support for some variable x_i taking value v_j, in the form of a transition from one state of the DFA to another. Such an arc necessarily lies on a path from q_0 in the first layer (that is, before x_1 is processed by the DFA) to some final state in the last layer (that is, after x_n is processed). That path spells out the sequence of transitions in the DFA while processing an assigned sequence X. The arcs on that path also provide the supporting values for variables $\langle x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n\rangle$, from the identity of the Q_{ij} set to which each belongs. Since the digraph is updated throughout the computation in order to mirror changes in the domains, generalized arc consistency is achieved.

5 Instances of the constraint

Regular languages allow us to express many non trivial relationships between the variables of a sequence. In particular, a regular language membership constraint generalizes some known global constraints.

Definition 2 (stretch). *A* stretch *is a maximal subsequence of identical values in an assigned sequence X. The* type *of a stretch is the value v_j taken by its variables and we denote it as v_j-stretch.*

The `stretch` constraint [8] puts restrictions on the length of stretches in a sequence. This can also be expressed with a regular expression or equivalently with a DFA.

Example 3. The DFA of Fig. 3 enforces a-stretches of length 2 and b-stretches of length 3.

In general, minimum and maximum lengths for each type of stretch are expressed by choosing the final states appropriately.

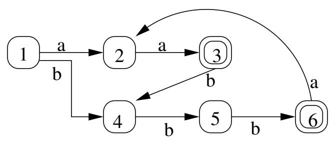


Fig. 3. A deterministic finite automaton for a stretch constraint.

Definition 3 (pattern). *We call* pattern *two or more consecutive stretches. A pattern made up of p stretches of type $v_1, v_2, \dots, v_p$, in that order, we call a* p-pattern *and denote it by $[\![v_1 v_2 \cdots v_p]\!]$. It follows from the definition of a stretch that $v_j \neq v_{j+1}$, $1 \leq j \leq p-1$.*

Patterns are a proper subclass of regular expressions: an a-stretch can be described as $aa^\star$; the 3-pattern $[\![aob]\!]$, as $aa^\star oo^\star bb^\star$ (recall Ex. 2).

Early experimental results suggest that using the `regular` constraint to enforce patterns performs better in terms of running time than a special-purpose pattern constraint previously designed by the author [7], whereas using it to replace the `stretch` constraint yields a slightly slower code. This may be due to the larger number of automaton states in the latter case (twenty-nine in the instances considered). Note however that the algorithm described in [8] exhibits weaker filtering: the inconsistency of Ex. 3 on a sequence of 6 variables, a very special case, would not be immediately detected whereas it necessarily would with the `regular` constraint.

6 Conclusion

This paper introduced an expressive global constraint that can help model complex sequencing rules present in many problems. A filtering algorithm that achieves generalized arc consistency was described for the constraint. From a user interface point of view, it would probably benefit from front-ends designed for particular contexts (such as stretch or pattern constraints) that would feel more natural than a regular expression to the ordinary user and that would automatically translate it into the appropriate DFA.

Acknowledgements

The author wishes to thank Geneviève Hernu for implementing the constraint and conducting the experiments, and the two anonymous referees for their constructive comments. This work was partially supported by the Canadian Natural Sciences and Engineering Research Council under grant OGP0218028.

References

1. M. Carlsson and N. Beldiceanu. Revisiting the Lexicographic Ordering Constraint. Technical Report T2002:17, SICS, 2002. 13 p.
2. C. Chang and R. Paige. From Regular Expressions to DFA's Using Compressed NFA's. *Theoretical Computer Science*, 178:1–36, 1997.
3. The Sequence Global Constraint of CHIP. Technical Report COSY/SEQ/032, COSYTEC, Orsay, France, 1999.
4. A. Colmerauer. An Introduction to PROLOG III. *Communications of the ACM*, 33(7):69–90, 1990.
5. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 1979.
6. ILOG S.A., Gentilly, France. *ILOG Solver Reference Manual, version 4.4*, 1999.
7. G. Pesant. The Pattern Constraint. In preparation.
8. G. Pesant. A Filtering Algorithm for the Stretch Constraint. In *Principles and Practice of Constraint Programming – CP 2001: Proceedings of the Seventh International Conference*, pages 183–195. Springer-Verlag LNCS 2239, 2001.

9. A. Rajasekar. Applications in Constraint Logic Programming with Strings. In *Principles and Practice of Constraint Programming: Proc. Second International Workshop*, pages 109–122, Rosario, Orcas Island, Washington, USA, 1994. Springer-Verlag LNCS 874.
10. C. Walinsky. CLP($\Sigma^\star$): Constraint Logic Programming with Regular Sets. In *Proceedings of the Sixth International Conference on Logic Programming*, pages 181–196, Lisbon, Portugal, 1989. MIT Press.
11. B.W. Watson. A taxonomy of finite automata minimization algorithms. Technical Report Computing Science Note 93/44, Eindhoven University of Technology, The Netherlands, 1993.

Boolean and Pseudo-Boolean Models for Scheduling

Steven Prestwich and Colin Quirke

Cork Constraint Computation Centre
Computer Science Department
University College, Cork, Ireland
www.4c.ucc.ie
{s.prestwich@cs,c.quirke@4c}.ucc.ie

Abstract. Several Boolean satisfiability (SAT) algorithms have been generalized to the more expressive linear pseudo-Boolean (PB) language. PB models are sometimes much more compact than SAT models, allowing the practical application of SAT-based methods to new problems. We show that reformulating job-shop and round-robin scheduling from SAT to PB can greatly reduce their model sizes. Using generalized SAT algorithms we show that, computationally speaking, little or nothing is lost by the reformulation.

1 Introduction

The propositional satisfiability (SAT) problem is to determine whether a Boolean expression has a satisfying labeling (set of truth assignments). Problems are usually expressed in *conjunctive normal form*: a conjunction of clauses $C_1 \wedge \ldots \wedge C_m$ where each clause C is a disjunction of literals $l_1 \vee \ldots \vee l_n$ and each literal l is either a Boolean variable x or its negation $\bar{x}$. A Boolean variable can be labeled either *true* or *false*. A satisfying assignment has at least one true literal in each clause.

Many combinatorial problems have been successfully modeled and solved as SAT, but not all problems succumb easily to SAT methods. One reason is that SAT models may turn out to be very large. Some researchers have proposed more expressive languages, partly in order to reduce model sizes. Examples include nested equivalences [19], exclusive-or [3], disjunctions of conjunctions of literals [31], cardinality constraints [11], finite-domain literals [5], Quantified Boolean Formulae (several papers), lifted models [12] and linear pseudo-Boolean (PB) models [1, 2, 22, 26, 29]. In particular, some SAT models can be exponentially reduced in size by reformulating them as PB models [10] (or of course by formulating them as PB instead of SAT in the first place).

PB models are a special form of integer linear program (ILP). ILP problems have been solved for decades using Operations Research algorithms, but these algorithms turn out to be unsuccessful on many SAT problems, typically those containing a great deal of structure. However, SAT-based methods have recently been shown to be remarkably successful on such problems after transformation to PB form. Algorithms based on SAT backtrackers [1, 2] performed well on several integer programming benchmarks and randomly-generated problems. Algorithms based on SAT local search algorithms [22, 26, 29] performed well on radar surveillance benchmarks, the Progressive Party Problem, SAT problems, combinatorial auctions, block designs and sports scheduling.

Faced with very large SAT problems it therefore seems reasonable to try reformulating them as PB problems to obtain smaller models. However, reducing the size of a model does not necessarily make it easier to solve. In general it is hard to predict the effect of changing models on search performance, and the best model for one algorithm may be the worst model for another. This paper studies the effect of reformulating job-shop (in Section 2) and round-robin (in Section 3) scheduling problems from SAT to PB. Both problems generate large SAT models but have much smaller PB models. We show that as PB problems these problems can be solved in similar times by suitably generalized SAT algorithms. Conclusions are drawn in Section 4. All experiments are performed on a 733 MHz Pentium III.

2 Job-shop scheduling

An $n \times m$ job-shop scheduling problem (JSP) is defined as follows. We are given n jobs and m tasks and resources, with a specified duration for each task and a resource on which it must execute. Each resource can only be used by one task at any given time, each job's tasks must occur in the given order without overlapping in time, and in each job each task is assigned to a unique resource. Tasks cannot be interrupted. Given a due date for the desired *makespan* we must schedule all tasks (or show that no schedule is possible).

Almost a decade ago Crawford & Baker [8] studied SAT encodings of JSPs. It has been revisited since [4, 7] but is worth revisiting for several reasons. Firstly, the SAT encodings were so large that the time taken to read them was greater than the time taken to solve them by a good JSP algorithm. It was suggested that generalizations of SAT could be used to reduce model size, and PB is such a generalization. Cadoli & Schaerf [7] also generated very large SAT models for the JSP. Secondly, advances have been made in problem modeling in recent years. The importance of implied constraints is now better appreciated, and alternative SAT encodings techniques have been studied. Thirdly, these problems appear to have unusual characteristics. The simple Iterative Sampling [18] algorithm gave better performance than both local search and backtracking. Preprocessing the problems by applying unit propagation on the unit clauses greatly improved local search performance, but it was still beaten by Iterative Sampling. It was conjectured that a hybrid of forward checking and local search might solve the problems even more quickly. More than one such hybrid now exists [16, 22] and other improvements have been made in both backtracking and local search algorithms. It would be interesting to test whether the problems still appear unusual with newer search algorithms, under alternative SAT and PB models, and using other benchmarks. Bayardo & Schrag [4] also found the JSP atypical as a SAT problem, more easily solved by Iterative Sampling than by backtracking with look-back techniques.

2.1 A minimal ILP model

We model the possible start times as positive integers $\{0, \ldots, T-1\}$ where T is the maximum allowed makespan. We define integer variables s_{ij} to model the start time of each task, where $1 \leq i \leq n$ and $1 \leq j \leq m$. Each start time has domain $\mathrm{dom}(s_{ij}) = \{0, \ldots, T-1\}$. The duration of each task is given by a constant d_{ij}. We impose *sequencing constraints* to ensure that tasks occur in the correct order within each job:

$$s_{ij} - s_{i\,j-1} \geq d_{i\,j-1}$$

where $1 \leq i \leq n$ and $2 \leq j \leq m$. It is much easier to express these conditions in ILP than in SAT: each requires a single linear constraint, whereas a SAT model must enumerate all violating pairs. To ensure that each resource is used by at most one task at any given time, for each pair of jobs $1 \leq i < i' \leq n$ and each resource $1 \leq r \leq m$ let the two tasks sharing r be j and j' respectively. The *resource capacity* constraints

$$(s_{ij} + d_{ij} \leq s_{i'j'}) \ \mathsf{XOR} \ (s_{i'j'} + d_{i'j'} \leq s_{ij})$$

express the restriction but are not in linear form. To achieve this we define an auxiliary 0/1 variable $v_{ii'r}$ for each job pair $i < i'$ and resource r, and impose two linear *channeling constraints*

$$s_{ij} + d_{ij} - T v_{ii'r} \leq s_{i'j'} \qquad s_{i'j'} + d_{i'j'} - T(1 - v_{ii'r}) \leq s_{ij}$$

If $v_{ii'r} = 0\,[1]$ then the first [second] constraint is imposed and the second [first] is trivially satisfied. Intuitively, the meaning of $v_{ii'r} = 0\,[1]$ is that task j precedes [succeeds] task j', where j and j' share resource r and are in jobs i and i' respectively. Crawford & Baker define similar variables but for all pairs i, i', not restricted to $i < i'$, in other words using inclusive-or instead of exclusive-or. They note that the auxiliary variables may be the most important search variables for these problems. The main decisions to be made concern the relative ordering of tasks, not their precise start times.

2.2 Implied constraints

The model can be improved by adding the following implied constraints, which are used in the experiments below (other implied constraints are possible, but those we tried so far did not improve performance). Firstly, start times cannot range over the full set of times, because the durations of previous and subsequent tasks restrict them to a subset of times. Their domains can be made explicit by *domain constraints*

$$\sum_{j'=1}^{j-1} d_{ij'} \leq s_{ij} \leq T - \sum_{j'=j}^{m} d_{ij'}$$

where $1 \leq i \leq n$ and $1 \leq j \leq m$. Secondly, the sequencing constraints can be extended to all pairs of tasks in a given job, giving *strong sequencing constraints*

$$s_{ij'} - s_{ij} \geq \sum_{k=j}^{j'-1} d_{ik}$$

where $1 \leq i \leq n$ and $1 \leq j < j' \leq m$ (Crawford & Baker appear to use constraints corresponding to our strong sequencing constraints). These constraints clearly follow from the original sequencing constraints and the transitivity of $<$.

2.3 Log encoding

Next we transform this model to 0/1 form. There are several ways to do this, just as there are several ways to SAT-encode a constraint satisfaction problem (CSP). One SAT encoding is sometimes called the *log encoding*. Instead of defining a SAT variable for each possible CSP variable-value assignment, we define a variable for each CSP variable/bit where the bits occur in a binary representation of the CSP domain values. The advantage of this encoding is that the number of variables is reduced from the size D of the CSP domains to $\lceil \log D \rceil$.

Unfortunately the SAT log encoding does not reduce the number of clauses. In fact it increases the space complexity of the model because each conflict between V CSP variables now requires a clause of size $V \lceil \log D \rceil$ instead of size V. However, a *linear* constraint over V variables is a special case: using a log encoding it can be modeled by a *single* 0/1 constraint of size $V \lceil \log D \rceil$, instead of enumerating all the conflicts. We replace a CSP constraint such as

$$\sum_i w_i v_i \geq d$$

by

$$\sum_i w_i \left(\sum_{k=1}^{b} 2^{k-1} v_{ik} \right) \geq d$$

where the integer v_i has binary representation $v_{i1} \ldots v_{ib}$.

Because JSP instances typically have a large number of possible times we use a log encoding to represent time: instead of defining a binary variable for every possible start time and task, we use a binary representation for the start time of each task. We define a 0/1 variable s_{ijk} for each integer variable s_{ij} and $1 \leq k \leq b$ where $b = \lceil \log_2 T \rceil$ is the number of bits needed to represent all possible start times. Then

$$s_{ij} = \sum_{k=1}^{b} 2^{k-1} s_{ijk}$$

and all the above constraints can be transformed to 0/1 form by substituting this expression.

Note that if we omit the implied domain constraints then the 0/1 model only enforces a makespan of $2^b \geq T$, and to enforce a makespan of T we must add constraints $s_{im} + d_{im} \leq T$. However, we use the implied constraints in our experiments.

2.4 Pseudo-Boolean form

Finally we transform the model to PB form so that generalized SAT algorithms can be applied. A 0/1 ILP model has constraints of the form

$$\sum_i w_i v_i \sim d$$

where $\sim$ is one of the operators $\{<, >, \leq, \geq, =\}$, the weights w_i and the constant d are (possibly negative) integers, and the variables v_i have domain $\{0, 1\}$. Any such constraint can be transformed into a normal form involving only $\geq$ and positive d and w_i, but with variables replaced by *literals*. A literal here l_i is either a variable v_i or its negation $\bar{v}_i = 1 - v_i$. SAT unit propagation generalizes easily to such constraints and is not much more expensive in runtime; see [2, 9, 22] for details. As an example consider the constraint $3v_1 + 2v_2 - v_3 = 2$. This expands to $3v_1 + 2v_2 - v_3 \geq 2$ and $3v_1 + 2v_2 - v_3 \leq 2$. In the first inequality the negative weight can be removed by using the fact that $-wl_i$ is equivalent to $w\bar{l_i} - c$, giving $3v_1 + 2v_2 + \bar{v}_3 \geq 3$. The second inequality can be expressed as $-3v_1 - 2v_2 + v_3 \geq -2$. Again the negative weights can be removed giving $3\bar{v}_1 + 2\bar{v}_2 + v_3 \geq 3$.

To improve the PB model we apply the following rules until no change occurs: (i) any constraint $\sum_i a_i x_i \leq b$ such that $\sum_i a_i \leq b$ is deleted; (ii) any term $a_i x_i$ such that $a_i > b$ is removed and a unary constraint $x_i \leq 0$ added.

2.5 Comparison of the SAT and PB models

In terms of the number of variables we obtain a reduction by using a log encoding. Crawford & Baker's model has $O(mn(n+T))$ variables while ours has $O(mn(n+\log T))$. In addition we have roughly half as many auxiliary variables because (as noted above) we encode exclusive-or conditions instead of inclusive-or. The SAT model also has variables denoting limits on both the start and end times of tasks, whereas ours only has variables denoting start times.

We also obtain a reduction in model size, which we define as the total number of literals in the model; this is fairer than comparing the number of constraints because our model has larger constraints. Crawford & Baker's SAT model is dominated by the sequencing constraints and a set of *coherence constraints*, giving $O(mnT(m + n))$ constraints. The constraint sizes are independent of the problem parameters so in terms of literals their model has size $O(mnT(m+n))$. Our model is dominated by the equivalent constraints, with $O(mn(m + n))$

constraints of size $\log T$, giving a model size of $O(mn \log T(m+n))$ literals. (If the domain constraints are not used then the two models are of size $O(mn^2T)$ and $O(mn^2 \log T)$ respectively.) T is typically a large number so this is a significant reduction.

2.6 Experiments

We recreated some of the experiments in [8] using the same problems: Sadeh's well-known JSP instances [25]. These are as described above but with additional constraints restricting release dates and due dates for each job, which are easily added to our model. Figure 1 compares the Walksat(OIP) and Saturn local search algorithms, showing local moves and CPU times computed as means over 1000 runs. It also reproduces the results for GSAT and Iterative Sampling (denoted "Isamp") from [8] with times normalized to our machine, taking means over 10 runs. The results were obtained after simplification consisting of unit propagation, some clause subsumption, and removal of unused variables; without simplification GSAT was up to 26 times slower and Isamp slightly slower. Normalization was done by comparing the runtimes of the DIMACS benchmark program *dfmax r500.5* [17] on both machines. "Moves" denotes flips in the case of GSAT and Walksat(OIP), iterations in the case of Isamp, and non-systematic backtracks in the case of Saturn. For each local search algorithm its noise parameter was set to a value that was roughly optimal for the entire problem set.

GSAT [28] was one of the first local search algorithms, Walksat [27] is a related but improved algorithm, Saturn [22] combines local search with unit propagation, and Iterative Sampling [18] is a modified backtracker that restarts the search from a random configuration at each dead-end. We expected Saturn to beat both GSAT and Walksat(OIP) on these problems, because GSAT performed poorly on the SAT model and it was conjectured that local search with constraint propagation would do well. Saturn is such an algorithm yet there is no clear winner among the three local search algorithms. We hope to test Saturn and Walksat(OIP) on the SAT model to test the conjecture on hybrid local search.

It was not clear whether reformulation to PB would improve Walksat-style local search: Walksat(OIP) has more advanced heuristics than GSAT and the PB model is smaller than the SAT model, but the use of a log encoding is likely to be a disadvantage. These factors appear to roughly cancel each other out, and Walksat(OIP) on the PB model is comparable to GSAT on the SAT model.

Perhaps surprisingly, Isamp is still the best algorithm. To investigate further we implemented a PB version of Isamp with unit propagation (like Crawford & Baker's SAT version) and found that it performed very badly on these problems. The explanation may be the use of a log encoding. The SAT log encoding is known to generate less powerful constraint propagation than the more usual direct encoding [30]. So our model may be less well-suited to Isamp than the SAT model, because the algorithm is less likely to be guided directly to a solution by unit propagation. We leave this question for future work.

Because our models are relatively small we can evaluate the PB algorithms on larger benchmarks. So far we have results on only two benchmarks, the well-

	SAT				PB			
	GSAT		Isamp		Saturn		Walksat(OIP)	
problems	moves	secs†	moves	secs†	moves	secs	moves	secs
w/1	390000	4.83	7	1.25	2056863	26.8	42037	1.42
w/2	290000	4.11	18	1.79	134742	1.66	188782	6.17
n/1	310000	4.11	13	1.43	279787	3.56	79487	2.61
n/2	550000	7.69	42	2.68	1683873	21.8	472989	17.3
t/1	1100000	13.8	62	2.86	369658	4.75	122155	4.03
t/2	2900000	37.7∗	180	7.69	4383837	57.2	1150940	58.9

∗ With 97% success rate † Normalized to our platform

Fig. 1. Results on Sadeh's benchmarks

known FT06 and FT10 problems of Muth & Thompson [21]. On FT06 Walksat(OIP) and Saturn both took approximately 10 seconds to find an optimal solution, which is comparable to the times found by Cadoli & Schaerf [7]. On FT10 neither was able to find an optimal solution (with makespan 930). Given roughly 1 hour Saturn reached makespan 940 once and 950 on several runs, while Walksat(OIP) managed to reach 970. Saturn is helped here by its ability to reuse previous solutions for each makespan. With both algorithms, on finding a schedule with a given makespan T we generate a new model with upper bound $M' < M$ on the makespan and restart the search (we chose $M' = M - 10$). Saturn reuses the previous schedule by reading in the values for the variables common to both models and initializing as many variables as possible to those values.

These JSP results are not competitive with state-of-the-art scheduling algorithms, and these instances are no longer regarded as difficult. But by focusing on problems that are difficult for SAT algorithms we may find better modeling and search techniques.

3 Round-robin scheduling

Next we consider the construction of *round-robin tournaments*, an example of sports scheduling. Though simple to state, this problem turns out to be a hard task for many algorithms. The problem is to schedule a tournament of n teams over $n - 1$ weeks with each week divided into $n/2$ periods. A tournament must satisfy the following three constraints: every team plays once a week; every team plays at most twice in the same period over the tournament; every team plays every other team.

This problem has been SAT-encoded and solved by Béjar & Manyà [6]. They compared several SAT models and algorithms, and obtained best results using the R-novelty variant of Walksat [27]. The model giving best results has $O(n^3)$ variables and $O(n^6)$ clauses. Clause sizes are independent of n so in terms of literals the model is of size $O(n^6)$. They point out that this can be reduced to

$O(n^4)$ clauses (or literals) at the expense of introducing new variables, but found that this led to inferior results.

This problem has no need for a log encoding but it can still be expressed more compactly as a PB model. An elegant ILP model was described by McAloon et al. [20] containing $O(n^2)$ constraints of size $O(n^2)$, giving $O(n^4)$ literals. However, it has $O(n^4)$ variables instead of the SAT model's $O(n^3)$. McAloon et al. state that it is unsuitable for extension to large sports scheduling problems because it has too many variables; but this is not necessarily the case with local search, which can often handle problems with more variables than backtracking algorithms can. In the model we define variables v_{wpxy} where $1 \leq w \leq n-1$, $1 \leq p \leq n/2$ and $1 \leq x < y \leq n$, meaning that game (x, y) is played in week/period (w, p) with no distinction between home and away. To enforce exactly one game per week/period:

$$\sum_{x=1}^{n} \left(\sum_{y=x+1}^{n} v_{wpxy} \right) = 1$$

where $1 \leq w \leq n-1$ and $1 \leq p \leq n/2$. To enforce exactly one week/period per game:

$$\sum_{w=1}^{n-1} \left(\sum_{p=1}^{n/2} v_{wpxy} \right) = 1$$

where $1 \leq x < y \leq n$. To ensure that each team plays exactly once per week:

$$\sum_{p=1}^{n/2} \left(\sum_{y=1}^{x-1} v_{wpyx} + \sum_{y=x+1}^{n} v_{wpxy} \right) = 1$$

where $1 \leq w \leq n-1$ and $1 \leq x \leq n$. To ensure that each team plays at most twice per period:

$$\sum_{w=1}^{n-1} \left(\sum_{y=1}^{x-1} v_{wpyx} + \sum_{y=x+1}^{n} v_{wpxy} \right) \leq 2$$

where $1 \leq p \leq n/2$ and $1 \leq x \leq n$. This model is already in 0/1 form and can be transformed to PB form as described in Section 2.4.

3.1 Implied constraints

An implied constraint is that each team plays at least once per period:

$$\sum_{w=1}^{n-1} \left(\sum_{y=1}^{x-1} v_{wpyx} + \sum_{y=x+1}^{n} v_{wpxy} \right) \geq 1$$

where $1 \leq p \leq n/2$ and $1 \leq x \leq n$. It is easy to show that these are implied. Assume that a team does not play in a period. Then even if it plays twice in every other period, it plays only $2 \times (\frac{1}{2}n - 1) = n - 2$ times. But each team must play every other team and must therefore play $n-1$ times, which contradicts our assumption. These constraints are not used by McAloon et al. but in tests we found that they improve local search performance.

Further constraints can be introduced by using a technique described by van Hentenryck et al. [15]: add a dummy week number n and insist that each team plays exactly twice, then extend the other constraints to ensure one match per week/period in the dummy week. Surprisingly this technique gave worse results with our model and search algorithm, possibly because it introduces extra variables for the dummy week.

3.2 Experiments

McAloon et al. found that the state-of-the-art MIP solver CPLEX was unable to solve the N=14 problem using the ILP model, though with symmetry breaking it found solutions up to N=12. (They solved N=14 in about 45 minutes on an UltraSparc using ILOG Solver with alldifferent constraints.) On a 250 MHz Sun UltraSparc, Béjar & Manyà used their SAT model and Walksat with the RNovelty heuristic to solve N=14 in 1.74 minutes, N=16 in 0.28 hours, N=18 in 1.98 hours and N=20 in 12.73 hours. Saturn was previously evaluated on round-robin problems in [22] but took roughly 1 hour to solve N=14 and never solved larger instances. The current version gives much better results by using a form of conflict analysis (to be described in future work). On the PB model with implied constraints it solved N=14 in just over 1 minute, N=16 in 2170 seconds, and N=18 in a few hours. Applying Walksat(OIP) with noise value $p = 0.05$ we found slightly better results for N=14: a median of 34 seconds. For N=16 with the same noise value we also obtained a better result: a median of 1428 sec. (We do not yet have results for larger N.) So far we have not quite matched the results of Béjar & Manyà but achieve results that are almost as good.

These results improve on those of McAloon et al. [20], are similar to those of Gomes et al [13], and are not quite as good as those of Béjar & Manyà [6]. But they fall short of the best known algorithms: Régin [24] solved N=40 in 6 hours using constraint-based techniques, and Hamiez & Hao [14] solve N=40 in less than 1 minute with Tabu search (but took longer for some smaller problems). Nevertheless, our results are among the best for SAT-based approaches but use smaller PB models.

4 Conclusion

SAT model compactness is important for two reasons: memory consumption and runtime overheads associated with checking a large number of clauses. Linear pseudo-Boolean models are sometimes much more compact than SAT models, and SAT algorithms have been extended to handle pseudo-Boolean models. We showed that SAT models for two scheduling problems can be replaced by much smaller pseudo-Boolean models, and solved in comparable times by generalized SAT algorithms. This supports the view that such reformulation is a practical way of solving problems with very large SAT models. Though SAT-based approaches are not competitive with the best known algorithms on these scheduling problems, these results help to advance the use of SAT techniques to such problems, and to other problems with similar characteristics.

We also partially tested a conjecture of [8]: that a hybrid of local search and propagation may beat pure local search on SAT-encoded job-shop scheduling problems. Though this has since been shown on other problems (for example query optimization [23]) we found in this paper that a hybrid (Saturn) had no clear advantage over pure local search (Walksat(OIP)). This may be because we used a pseudo-Boolean rather than a SAT model, or because our hybrid algorithm requires improvement. Alternatively, these problems may not require hybrid search, and the poor local search performance found in [8] may have been caused by the use of a relatively primitive algorithm. In future work we hope to test these ideas by comparing algorithm performance on the original SAT models.

Finally, we found an interesting result for Iterative Sampling. Though it gave better results than several local search algorithms on a SAT encoding, it performed very badly on a pseudo-Boolean encoding. We believe that this was caused by our use of a log encoding in particular, which has been shown to weaken constraint propagation in SAT-encoded CSPs [30] (we assume that this result extends to pseudo-Boolean encodings). We know of no previous combination of Iterative Sampling and log encoding in the literature.

Acknowledgments This work was supported in part by the Boole Centre for Research in Informatics, University College, Cork, Ireland, and by Science Foundation Ireland under Grant 00/PI.1/C075.

References

1. F. A. Aloul, A. Ramani, I. Markov, K. Sakallah. PBS: a Backtrack-Search Pseudo-Boolean Solver and Optimizer. *Fifth International Symposium on Theory and Applications of Satisfiability Testing*, Cincinnati, Ohio, USA, 2002, pp. 346–353.
2. P. Barth. A Davis-Putnam Based Enumeration Algorithm for Linear Pseudo-Boolean Optimization. Research Report mpi-i-95-2-003, Max-Plank Institut fur Informatik, Saarbrucken, 1995.
3. P. Baumgartner, F. Massacci. The Taming of the (X)OR. *First International Conference on Computational Logic, Stream on Automated Deduction: Putting Theory into Practice, Lecture Notes in Artificial Intelligence* vol. 1861, Springer-Verlag, 2000, pp. 508–522.
4. R. J. Bayardo, R. C. Schrag. Using CSP Look-Back Techniques to Solve Real-World SAT Instances. *Fourteenth National Conference on Artificial Intelligence*, 1997, pp. 203–208.
5. R. Béjar, A. Cabiscol, C. Fernandez, F. Manyà, C. P. Gomes. Capturing Structure with Satisfiability. *Seventh International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 2239, Springer-Verlag, 2001, pp. 137–152.

6. R. Béjar, F. Manyà. Solving the Round Robin Problem Using Propositional Logic. *Seventeenth National Conference on Artificial Intelligence*, 2000, pp. 262–266.
7. M. Cadoli, A. Schaerf. Compiling Problem Specifications Into SAT. *European Symposium On Programming, Lecture Notes in Computer Science* vol. 2028, Springer Verlag, 2001.
8. J. M. Crawford, A. B. Baker. Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems. *Twelfth National Conference on Artificial Intelligence* vol. 2, AAAI Press, 1994, pp. 1092–1097.
9. H. E. Dixon, M. L. Ginsberg. Inference Methods for a Pseudo-Boolean Satisfiability Solver. *Eighteenth National Conference on Artificial Intelligence*, Edmonton, Alberta, Canada, 2002.
10. H. E. Dixon, M. L. Ginsberg, A. J. Parkes. Likely Near-term Advances in SAT Solvers. *Workshop on Microprocessor Test and Verification*, Austin, Texas, USA, 2002.
11. M. R. Dransfield, V. W. Marek, M. Truszczynski. Satisfiability and van der Waerden Numbers. *Sixth International Conference on Theory and Applications of Satisfiability Testing*, Portofino, Italy, 2003, pp. 325-336.
12. M. L. Ginsberg, A. J. Parkes. Satisfiability Algorithms and Finite Quantification. *Seventh International Conference on Principles of Knowledge Representation and Reasoning*, Breckenridge, Colorado, USA, 2000.
13. C. P. Gomes, B. Selman, H. A. Kautz. Boosting Combinatorial Search Through Randomization. *Fifteenth National Conference on Artificial Intelligence*, AAAI Press / The MIT Press, 1998, pp. 431–437.
14. J. P. Hamiez, J. K. Hao. Solving the Sports League Scheduling Problem With Tabu Search. *Lecture Notes in Artificial Intelligence* vol. 2148, Springer, 2001, pp. 24–36.
15. P. van Hentenryck, L. Michel, L. Perron, J. C. Régin. Constraint Programming in OPL. *International Conference on Principles and Practice of Declarative Programming*, Springer-Verlag, 1999.
16. E. A. Hirsch, A. Kojevnikov. UnitWalk: A New SAT Solver That Uses Local Search Guided by Unit Clause Elimination. *Fifth International Symposium on the Theory and Applications of Satisfiability Testing*, University of Cincinnati, 2002, pp. 35–42.
17. D. S. Johnson, M. A. Trick (Eds). *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science* vol. 26, American Mathematical Society, 1996.
18. P. Langley. Systematic and Nonsystematic Search Strategies. *First International Conference on Artificial Intelligence Planning Systems*, 1992.
19. C.-M. Li. Integrating Equivalency Reasoning into Davis-Putnam Procedure. *Seventeenth National Conference on Artificial Intelligence*, Austin, Texas, USA, 2000, pp. 291–296.
20. K. McAloon, C. Tretkoff, G. Wetzel. Sports League Scheduling. *ILOG Optimization Suite International Users' Conference*, Paris, July, 1997.
21. J. Muth, G. Thompson. *Industrial Scheduling.* Prentice Hall, 1963.
22. S. D. Prestwich. Randomised Backtracking for Linear Pseudo-Boolean Constraint Problems. *Fourth International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems* le Croisic, France, 2002, pp. 7–20.
23. S. D. Prestwich, S. Bressan. A SAT Approach to Query Optimization in Mediator Systems. *Fifth International Symposium on the Theory and Applications of Satisfiability Testing*, University of Cincinatti, 2002, pp. 252–259. Submitted to *Annals of Mathematics and Artificial Intelligence*.

24. J.-C. Régin. Sports Scheduling and Constraint Programming. *INFORMS*, Cincinnatti, Ohio, 1999.
25. N. Sadeh. Look-Ahead Techniques for Micro-Opportunistic Job Shop Scheduling. Technical report CMU-CS-91-102, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, 1991.
26. D. Schuurmans, F. Southey, R. C. Holte. The Exponentiated Subgradient Algorithm for Heuristic Boolean Programming. *Seventeenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 2001, pp. 334–341.
27. B. Selman, H. Kautz, B. Cohen. Noise Strategies for Improving Local Search. *Twelfth National Conference on Artificial Intelligence*, AAAI Press, 1994, pp. 337–343.
28. B. Selman, H. Levesque, D. Mitchell. A New Method for Solving Hard Satisfiability Problems. *Tenth National Conference on Artificial Intelligence*, MIT Press, 1992, pp. 440–446.
29. J. P. Walser. Solving Linear Pseudo-Boolean Constraints With Local Search. *Eleventh Conference on Artificial Intelligence*, AAAI, 1997, pp. 269–274.
30. T. Walsh. SAT v CSP. *Sixth International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science* vol. 1894, Springer-Verlag, 2000, pp. 441–456.
31. J. Whittemore, J. Kim, K. Sakallah. SATIRE: A New Incremental Satisfiability Engine. *Thirty-Eighth Design Automation Conference*, 2001, pp. 542–545.

A Reformulation of the Bridge Building Problem as Vehicle Routing[⋆]

Evgeny Selensky

Department of Computing Science, University of Glasgow, Scotland.
evgeny@dcs.gla.ac.uk

Abstract. We empirically study the effects of problem formulations and, more specifically, the influence of symmetries in the problem on algorithmic performance. We present a reformulation of the Bridge Building Problem as an instance of vehicle routing with time windows. We apply systematic and neighbourhood search to the original and reformulated problem. We also relax the original and reformulated problem by introducing additional symmetries and show how this impacts the performance of the solving techniques.

1 Introduction

Choosing a suitable problem representation has long been in the focus of AI research [13], and, more specifically, in constraint satisfaction [15], scheduling and routing [4, 5, 12]. Considerable attempts have been made to automate the process of finding "good" formulations [6]. Nonetheless, the issue of adequately formulating the given problem still seems too broad to embrace it in all entirety. E.g., it is known that symmetries in a problem formulation are undesirable for constructive search because a solver spends redundant effort in exploring symmetric states. However, recent work [10] has revealed that different neighbourhood search techniques may in contrast benefit from having symmetries in the problem representation. The intuition is that adding symmetry-breaking constraints transforms solution states into local minima. It appears that more in-depth research is needed to study possible effects of symmetries in the context of local search in particular in the scheduling and routing domains.

This paper presents an empirical study of the effects of changes in the formulation of a scheduling problem and of the influence of symmetries in the different problem formulations on algorithmic performance. In the study we use an example problem of constructing a five-segment bridge (the Bridge Building Problem, BBP). It was introduced in [3] and has since been widely accepted as a benchmark [9]. In the study, we first solve the problem by a systematic branch and bound based technique. As a second step, we reformulate the original BBP as an instance of vehicle routing and apply to the routing-reformulated instance a neighbourhood search technique specifically designed for routing. Finally, we

⋆ This work was supported by EPSRC research grant GR/M90641 and ILOG SA.

inject various symmetries into the BBP, both direct and reformulated, and compare the performance of systematic and neighbourhood search techniques with and without additional symmetries.

The paper is organised as follows. Section 2 is a recap on the Bridge Building Problem. In section 3 we present two encodings of the BBP, the direct and reformulated as the vehicle routing problem. Section 4 explains how we relax the BBP by introducing symmetries. Section 5 presents the design and results of our experiments. Section 6 concludes the paper.

2 Bridge Building Problem

In the Bridge Building Problem and 2) we have to construct a five-segment bridge.

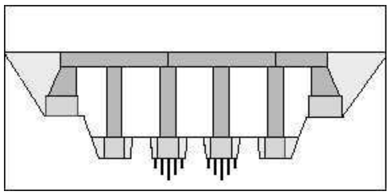


Fig. 1. The five-segment bridge (taken from the Oz online manual [9]).

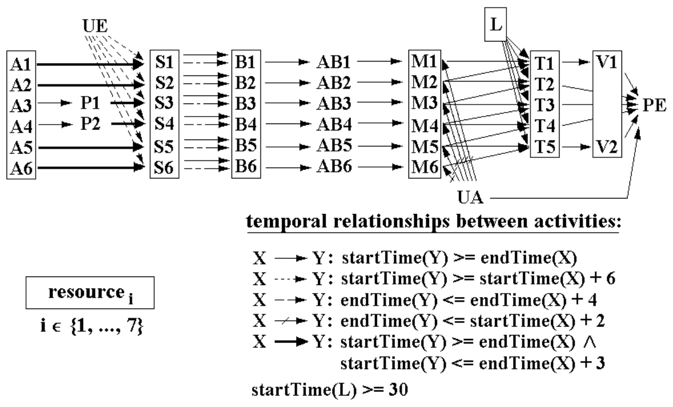

UE
L
A1
A2
A3
A4
A5
A6
P1
P2
S1
S2
S3
S4
S5
S6
B1
B2
B3
B4
B5
B6
AB1
AB2
AB3
AB4
AB5
AB6
M1
M2
M3
M4
M5
M6
T1
T2
T3
T4
T5
V1
V2
PE
UA
temporal relationships between activities:
X → Y: startTime(Y) >= endTime(X)
X → Y: startTime(Y) >= startTime(X) + 6
X → Y: endTime(Y) <= endTime(X) + 4
X → Y: endTime(Y) <= startTime(X) + 2
X → Y: startTime(Y) >= endTime(X) ∧ startTime(Y) <= endTime(X) + 3
startTime(L) >= 30
resource i
i ∈ {1, ..., 7}

Fig. 2. The Bridge Building Problem: resources are shown as rectangles, activities as identifiers. Arrows represent temporal relationships between activities.

Table 1. The Bridge Building Problem: Specification

No.	ID	Description	Duration (days)	Predecessors	Resource
1	A1	excavation (abutment 1)	4	-	excavator
2	A2	excavation (pillar 1)	2	-	excavator
3	A3	excavation (pillar 2)	2	-	excavator
4	A4	excavation (pillar 3)	2	-	excavator
5	A5	excavation (pillar 4)	2	-	excavator
6	A6	excavation (abutment 2)	5	-	excavator
7	P1	foundation piles 2	20	A3	pile driver
8	P2	foundation piles 3	13	A4	pile driver
9	UE	erection of temporary housing	10	-	-
10	S1	formwork (abutment 1)	8	A1	carpentry
11	S2	formwork (pillar 1)	4	A2	carpentry
12	S3	formwork (pillar 2)	4	P1	carpentry
13	S4	formwork (pillar 3)	4	P2	carpentry
14	S5	formwork (pillar 4)	4	A5	carpentry
15	S6	formwork (abutment 2)	10	A6	carpentry
16	B1	concrete foundation (abutment 1)	1	S1	concrete mixer
17	B2	concrete foundation (pillar 1)	1	S2	concrete mixer
18	B3	concrete foundation (pillar 2)	1	S3	concrete mixer
19	B4	concrete foundation (pillar 3)	1	S4	concrete mixer
20	B5	concrete foundation (pillar 4)	1	S5	concrete mixer
21	B6	concrete foundation (abutment 2)	1	S6	concrete mixer
22	AB1	setting time (abutment 1)	1	B1	-
23	AB2	setting time (pillar 1)	1	B2	-
24	AB3	setting time (pillar 2)	1	B3	-
25	AB4	setting time (pillar 3)	1	B4	-
26	AB5	setting time (pillar 4)	1	B5	-
27	AB6	setting time (abutment 2)	1	b6	-
28	M1	masonry (abutment 1)	16	AB1	bricklaying
29	M2	masonry (pillar 1)	8	AB2	bricklaying
30	M3	masonry (pillar 2)	8	AB3	bricklaying
31	M4	masonry (pillar 3)	8	AB4	bricklaying
32	M5	masonry (pillar 4)	8	AB5	bricklaying
33	M6	masonry (abutment 2)	20	AB6	bricklaying
34	L	delivery of preformed bearers	2	-	crane
35	T1	positioning (bearer 1)	12	M1,M2,L	crane
36	T2	positioning (bearer 2)	12	M2,M3,L	crane
37	T3	positioning (bearer 3)	12	M3,M4,L	crane
38	T4	positioning (bearer 4)	12	M4,M5,L	crane
39	T5	positioning (bearer 5)	12	M5,M6,L	crane
40	UA	removal of temporary housing	10	-	-
41	V1	filling 1	15	T1	caterpillar
42	V2	filling 2	10	T5	caterpillar
43	PE	end of project	0	T2,T3,T4,V1,V2,UA	-

Table 2. The Bridge Building Problem: Additional Constraints

No.	Additional Constraint
1	$StartTime(S_i) \geq StartTime(UE) + 6,\ i \in \{1, ..., 6\}$
2	$EndTime(B_i) \leq EndTime(S_i) + 4,\ i \in \{1, ..., 6\}$
3	$StartTime(S_i) \leq EndTime(A_i) + 3,\ i \in \{1, ..., 6\}$
4	$EndTime(M_i) \leq StartTime(UA) + 2,\ i \in \{1, ..., 6\}$
5	$StartTime(L) \geq 30$

The complete project includes excavations, making foundation piles, formwork, masonry work, positioning of the bearers, etc. as displayed in Table 1 and explained in detail in [3, 9]. Each row in Table 1 shows activity number (column 1), identifier (column 2), description (column 3), duration (column 4), a list of other activities that should precede it (column 5) and finally the required resource (column 6).

We also have the following additional constraints that cannot be represented by showing the predecessor of an activity:

– The erection of the temporary housing must begin at least 6 days before each formwork (constraint 1 in Table 2).

- The time between the completion of the formwork and the completion of the corresponding concrete foundation is at most 4 days (constraint 2).
- Between the end of a particular foundation and the beginning of the corresponding formwork can at most 3 days elapse (constraint 3).
- The removal of the temporary housing can start at most 2 days before the end of the last masonry (constraint 4).
- The delivery of the preformed bearers occurs exactly 30 days after the beginning of the project (constraint 5 in Table 2).

3 Bridge Building Problem Formulations

3.1 Direct Encoding

The BBP is clearly a scheduling problem: we have to schedule 43 activities, 34 of which are allocated to the 7 available resources while the rest require no resource. The schedule must respect the temporal constraints displayed in Tables 1 and 2.

In the direct encoding there are 43 activities and 7 unary resources. Each unary resource can be occupied by only one activity at any instant. We post all the resource allocation and temporal constraints shown in Tables 1 and 2. For example, we state that activity $P1$ requires unary resource *pile driver* for its entire duration (20 days) and that *pile driver* can only start execution of $P1$ after *excavator* has finished execution of $A3$. While building a schedule we minimise the finish time of the latest activity PE, known as the makespan in the scheduling literature [1].

3.2 BBP as Vehicle Routing

Now we represent the Bridge Building Problem as an instance of the Vehicle Routing Problem.

In the *delivery* variant of the *vehicle routing problem* (VRP), m identical vehicles initially located at the base are to deliver discrete quantities of goods to n customers[1]. The locations of the base and customers are known. Each customer has a certain discrete demand for goods and each vehicle has a discrete capacity, i.e., it can carry quantities less than or equal to its capacity. A vehicle can make only one tour starting at the base (or the depot), visiting a subset of customers and getting back to the base. It is also assumed that travel time equals travel distance computed using the Euclidean metric. There are also temporal constraints known as *time windows* for servicing customers. Required is to find tours for a subset of vehicles such that (i) all customers are served, (ii) each customer is served only once and (iii) the total distance travelled by the fleet is minimum.

In the BBP reformulated as a routing problem, an activity is a customer visit, a resource is a vehicle. E.g., there is a vehicle corresponding to the *caterpillar*.

[1] In the *pickup and delivery* version, customers may require to pick up goods as well as to deliver.

This vehicle can only visit customers $V1$ and $V2$ and these visits should satisfy all precedence constraints involving $V1$, $V2$ and other visits. Clearly, the time window and duration of an activity is the time window and duration of the respective customer visit. Apart from that, we have to introduce two additional dummy visits for each vehicle to mark the start and end of its tour. Consequently, if in a solution a vehicle is not used, its empty tour will contain only two dummy visits. If a vehicle is used in a solution, any customer visit occurs after the start dummy visit and before the end dummy visit.

Travel between the depot and customers in the VRP corresponds to transition (setup, teardown) times between activities. Because we have zero transition times between activities in the BBP, all travel is also zero. So we can think of the BBP as a strange routing problem where everything occurs in one place! This representation of a scheduling problem is analogous to [5].

The end of project PE is a unique activity in this encoding. We do not represent it as a customer visit, but model it using a deadline variable D. To minimise the deadline, we iteratively reduce D during search. The cost function is the sum of tardiness on each vehicle, not the total distance as in the original VRP formulation.

4 BBP with Symmetries

As shown in [5], two important problem characteristics responsible for the performance of typical scheduling and routing techniques are the number of alternative resources in a problem and the number of temporal relationships between operations. Our long-term goal is to develop a deeper understanding of the influence of these parameters on solving techniques.

In this study, we can think of introducing two kinds of symmetries to the Bridge Building Problem, i.e., symmetries with respect to ordering activities and with respect to allocating resources. The first type of symmetry in the extreme turns a conventional scheduling problem into a variation of the open shop problem [8], where any order of jobs is satisfactory. The resource allocation symmetry amounts to having alternative resources. Examples of symmetries are given below in sect. 5.

In the next section, we first solve the original BBP as a scheduling problem and as a routing problem. Then we systematically vary the original Bridge Building Problem by adding symmetries in the problem formulation and observe how these variations in the problem influence algorithmic behaviour.

5 Experiments

5.1 No additional symmetries

The first set of experiments is to run the direct and routing-reformulated encodings of the Bridge Building Problem and compare the performance of solving

Table 3. Experiments with Scheduling Techniques. CPU time limit: 1 second

Techn.	**Description**	**#Sols.**	**First Solution**			**Best Solution**		
			Cost	Nodes	Backtr.	Cost	Nodes	Backtr.
ST1	Branch & bound + restart resource selection: min global slack activity selection: min earliest start time	4	120	27	1	104	90	2
ST2	Branch & bound without restart, the rest is as ST1	1	104	102	87	104	102	87

Table 4. Comparison of Routing Techniques. CPU time limit: 60 seconds

Technique	**Description**	**Cost**	
		First Sol.	**Best Sol.**
RT1	First sols. by savings, first-accept local search guided local search metaheuristic	123	120
RT2	First sols. by ST1, first-accept local search guided local search metaheuristic	120	116

techniques. The software we use throughout the paper is the ILOG optimisation suite (in particular, ILOG Scheduler 5.2 and Dispatcher 3.2).

Because every resource in the problem is unary and, consequently, at any instant it is occupied by no more than one activity, for any pair of activities A and B on the same resource, we have to decide if A goes before B or vice versa. Therefore as a scheduling technique, ST, for the direct encoding we use the approach of disjunctive scheduling [11], which is known to be efficient for this type of problem. At each step while we have activities to sequence, we dynamically choose the most critical resource to schedule activities on. Resource criticality is calculated in terms of global slack [2] as the difference between the demand and supply over a given time interval. Among the unscheduled activities on the selected resource, scheduled first is the activity that has the minimum earliest start time, breaking ties with the minimum latest start time. $ST1$ in Table 3 searches for solutions iteratively, posting constraints on the quality of each next solution and restarting search after each solution found. On the contrary, $ST2$ adds a new constraint on the quality of solutions at a leaf node and backtrack from there instead of starting search from scratch.

We see from Table 3 that $ST1$ improves on the overall number of search states explored (90 nodes and 2 backtracks) compared to $ST2$ (102 nodes and 87 backtracks) even though the very first solution obtained by $ST2$ is optimum (with a cost of 104 days). It means that propagation on the optimisation criterion helps the scheduling heuristics make better decisions than by $ST2$. Using either technique, the optimum solution was found within 0.1s CPU time (Windows NT workstation, 1 Gb RAM, 933 MHz Pentium III processor).

As a routing technique, RT, we apply a neighbourhood search based procedure to the BBP reformulated as vehicle routing. In $RT1$ (Table 4), the first solution is obtained by the savings heuristic [11]. In $RT2$, the first solution is imported from $ST1$. To improve solutions locally we use first-accept local search enhanced by the guided local search metaheuristic [14] to drive search out of local minima.

It has been found in [5] that impurities of the vehicle routing problem such as precedence constraints between visits or resource allocation constraints between visits and vehicles may deteriorate the performance of first solution construction heuristics (such as savings). It appears that it would have been the case for this problem as well if we had had more complex temporal relationships between activities, e.g., between activities on the same resource. When a routing technique cannot come up with a first solution, an alternative approach is to start search from solutions found by scheduling technology as is done by $RT2$.

It is clear from Tables 3 and 4 that the scheduling technique outperforms the routing technique for the Bridge Building Problem encoded as a VRP instance. This agrees with previous research in routing-scheduling reformulation. In [4, 5, 12] several problem parameters were identified that were responsible for the observed difference in algorithmic performance. Problems with a “scheduling” optimisation criterion (for example, makespan), many precedence constraints between operations and high specialisation of resources are amenable to typical scheduling techniques (for example, systematic branch and bound search, powerful global constraint propagation mechanisms and sophisticated heuristics). On the contrary, instances with a “routing” optimisation criterion (e.g., travel time), very few or no precedences between operations and many alternative resources for an operation benefit from using conventional routing techniques (fast and efficient first solution construction methods and local search in a metaheuristic framework).

5.2 Introducing symmetries to the BBP

Now we are transforming the BBP by relaxing temporal constraints between activities or adding alternative resources. Solutions to the transformed problems are therefore no longer solutions to the original BBP.

Precedence Symmetries In this section we systematically transform precedence constraints (Fig. 2) such that if in the original problem activity B could start only after activity A has finished execution ($A \rightarrow B$), now either A can precede B or vice versa, $A \leftrightarrow B$. In Table 5 we present the results of several experiments:

- PS_0: no additional symmetries;
- PS_1: precedence symmetries between operations B_i, AB_i, M_i, $i \in \{1, ..., 6\}$ to say that the order of these operations is no longer significant. Assuming

Table 5. Influence of Precedence Symmetries on Performance of Scheduling and Routing Techniques. CPU time limit: 60 seconds

Exp.	**Technique**	**# added symms.**	**Cost**		**Cost Difference,%**	
			First Sol.	**Best Sol.**	**First Sol.**	**Best Sol.**
PS_0	ST1	0	120	104	0	0
	RT1	0	123	120	2.5	15.4
PS_1	ST1	36	104	92	0	0
	RT1	36	107	95	2.9	3.3
PS_2	ST1	66	104	90	0	0
	RT1	66	95	95	-8.7	5.6
PS_3	ST1	126	-	-	-	-
	RT1	126	91	76	-	-

the operations do not overlap, it is equivalent to having $6 \cdot 3! = 36$ additional symmetries.

– PS_2: as in PS_1 plus symmetries between L and T_i, $i \in \{1, ..., 5\}$. The number of symmetries is $3! \cdot 2! \cdot 5 + 3! = 66$ because for each $i \in \{1, .., 5\}$ we have $3!$ permutations of B_i, AB_i, M_i, $2!$ permutations of L, T_i, and for $i = 6$ we have $3!$ permutations of B_6, AB_6 and M_6;
– PS_3: in addition to PS_2, symmetries between M_i and T_i, $i \in \{1, ..., 5\}$. The number of symmetries is $3! \cdot 2! \cdot 2! \cdot 5 + 3! = 126$.

From Table 5 we see that the performance of routing technology is improving relative to scheduling technology as we inject more symmetries. In the last experiment, the routing technology successfully found solutions while $ST1$ did not find any over the allocated CPU time.

Resource Allocation Symmetries Now we inject resource allocation symmetries into the original BBP to allow alternative resources (Table 6):

– RAS_0: no resource allocation symmetries (the same as PS_0);
– RAS_1: we introduce six sets of alternative resources such that B_i, AB_i and M_i, $i \in \{1, ..., 6\}$ are processed by any one resource in the i-th set[2]. There are $3^3 = 27$ different variants for allocating 3 activities on 3 alternative resources (not taking into account the fact that there may be other constraints in the problem that disallow a subset of these allocations). Because we have six sets of alternative resources, this gives 162 symmetries.
– RAS_2: in addition to RAS_1 we have resource allocation symmetries with respect to L and each of T_i, $i \in \{1, ..., 5\}$. This gives $6^6 = 46656$ symmetries in addition to RAS_1 (46818 in total).

As displayed in Table 6, the performance of the routing technique gets worse as we add in more resource allocation symmetries. More specifically, $RT1$ failed

[2] To do this we have to introduce extra resources.

Table 6. Influence of Resource Allocation Symmetries on Performance of Scheduling and Routing Techniques. CPU time limit: 60 seconds

Exp.	**Technique**	**# added symms.**	**Cost**		**Cost Difference,%**	
			First Sol.	**Best Sol.**	**First Sol.**	**Best Sol.**
RAS_0	ST1	0	120	104	0	0
	RT1	0	123	120	2.5	15.4
	RT2	0	120	116	0	3.3
RAS_1	ST1	162	98	92	0	0
	RT1	162	-	-	-	-
	RT2	162	98	98	0	6.5
RAS_2	ST1	46818	75	72	0	0
	RT1	46818	-	-	-	-
	RT2	46818	75	75	0	4.2

to produce first solutions in the experiments RAS_1 and RAS_2. That is why we had to resort to $RT2$ that imported first scheduler solutions. This is because the introduction of some resource allocation symmetries increases the possibility of the wrong resource choice by the first solution heuristic and, consequently, the possibility of subsequent failure. Finally, when we have substantially many alternatives the performance of the heuristic gets improved since increasingly more resource allocations will lead to solutions. Similar effects were noticed in [5].

Second, even $RT2$ could not find any improvements to the injected first solutions. One possible explanation is that in the presence of many temporal constraints a lot of moves in the neighbourhood are discarded. On the contrary, scheduling technology actually benefited from having many temporal constraints even with additions of resource allocation symmetries. Despite extra variants of operation placement introduced, temporal and resource constraint propagation was able to detect inconsistencies early on.

Precedence and Resource Allocation Symmetries We have also performed tests to assess the response of solution quality to the introduction of both types of symmetry at the same time. In total, we generated six different problems based on problems PS_1,PS_2,PS_3 from Table 5, and RAS_1 and RAS_2 from Table 6. As before, we have run branch and bound based systematic search algorithm ST_1 and, separately, local search algorithms RT_1 and RT_2 on each of the six problems. The experiments showed that RT_1 was not able to construct a first solution for any instance. Moreover, for all but one problems the first solution ST_1 found was optimal. For the remaining instance with precedence symmetries taken from PS_3 and resource allocation symmetries taken from RAS_2, even ST_1 could not find a solution over the allotted time. This remaining instance had the biggest number of symmetries of all the six problems.

5.3 Summary of Experimental Results

We observe that the introduction of precedence symmetries to the original problem instance leads to increasingly better performance of the local search technique as compared to the systematic search technique. Resource allocation symmetries, in contrast, have a negative influence on the performance of the local search technique because they make it harder to construct first solutions. These observations also hold when we introduce both types of symmetry at the same time.

6 Conclusion

This paper provides empirical evidence to emphasise the importance of problem formulation in the process of problem solving. More specifically, the study is focused on two aspects of problem reformulation: mapping an instance from one problem class to another and having symmetries in a problem representation.

Using a simple scheduling problem, we have seen that mapping to a different problem class can have a profound effect (positive or negative) on algorithmic performance. In particular, we have observed that a typical local search based routing technique performs worse on the Bridge Building Problem than a conventional systematic search based scheduling technique does. As [5] points out, it is important to realise what features a problem has as this can give an indication of what solving technique is best to use.

We could also view the results in this paper the other way around. Suppose that we had a “routing” problem with many predefined vehicle specialisations, precedences between customer visits and that we were interested in minimising the finish time of the latest tour. After all, we can have it in practice, for example, when certain trucks can only carry certain types of goods, trucks are loaded in a way that dictates some order of visits to minimise unloading delays, visits are time-critical and the only concern is to do all of them as soon as possible regardless of travel. This study provides evidence that the scheduling technique would be a much better alternative to the routing technology in this scenario.

We have also demonstrated that symmetries may have a dramatic impact on the performance of solving techniques. It is no surprise that, in general, symmetries are not desirable for systematic search [7]. However, local search techniques can benefit from having symmetries, as noted in [10]. In our study, when we added operation precedence symmetries, we observed a notable increase in the performance of the routing technique even for a problem of modest size such as the Bridge Building Problem. The introduction of resource allocation symmetries in isolation had a negative effect on the performance of the routing technique because it was not amenable to first solution construction and also because it proved hard for guided local search to even improve scheduling solutions in the presence of many temporal and resource constraints.

Based on a small set of test problems generated using a known scheduling benchmark, this study gives preliminary indications of the importance of issues of problem representation and symmetries for efficient problem solving. To develop a deeper understanding of their influence on search performance, in particular for scheduling problems, experiments need to be carried out systematically on larger sets of instances.

Acknowledgement

The author wishes to thank Patrick Prosser (Glasgow University, Scotland) and Paul Shaw (ILOG, France) for their ideas and comments.

References

1. Kenneth R. Baker. *Introduction to Sequencing and Scheduling.* John Wiley and Sons, 1974.
2. Ph. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems.* Kluwer Academic Publishers, 2001.
3. M. Bartusch. *Optimierung von Netsplanen mit Anordnungsbeiziehungen bei knappen Betriebsmitteln.* PhD thesis, Universitat Passau, Fakultat fur Matematik und Informatik, 1983.
4. J.C. Beck, P. Prosser, and E. Selensky. On the reformulation of vehicle routing problems and scheduling problems. In *LNAI 2371, Proceedings of the Symposium on Abstraction, Reformulation and Approximation (SARA 2002)*, pages 282–289, 2002.
5. J.C. Beck, P. Prosser, and E. Selensky. Vehicle routing and job shop scheduling: What's the difference? In *Proceedings of the 13th International Conference on Artificial Intelligence Planning and Scheduling (ICAPS 2003), to appear*, 2003.
6. E. Fink. *Changes of Problem Representation: Theory and Experiments.* Physica-Verlag, 2003.
7. I.P. Gent, W. Harvey, and T. Kelsey. Groups and constraints: Symmetry breaking during search. In *Principles and Practice of Constraint Programming (CP'02)*, 2002.
8. T. Gonsalez and S. Sahni. Open Shop Scheduling To Minimize Finish Time. *Journal of the Association for Computing Machinery*, 23(4):665–679, 1976.
9. The Oz Constraint Programming Language Online Manual. http://www.mozart-oz.org/documentation/fdt/node48.html.
10. S. Prestwich. Negative Effects of Modeling Techniques on Search Performance. *Annals of Operations Research*, 18:137–150, 2003.
11. ILOG SA. ILOG Scheduler 5.1 User Manual, France, 2001.
12. E. Selensky. On mutual reformulation of shop scheduling and vehicle routing. In *Proceedings of the 20th UK PLANSIG*, 2001.
13. Yu. Smirnov and M. Veloso. Efficiency competition through representation changes: Pigeonhole principle versus linear programming relaxation. In *5th Int. Conference on Principles of Knowledge Representation and Reasoning (KR'96)*, 1996.
14. C. Voudouris and E.P.K. Tsang. Guided Local Search. *European Journal of Operational Research*, 113(2):80–110, 1998.
15. T. Walsh. Reformulating propositional satisfiability as constraint satisfaction. In *Symposium on Abstraction, Reformulation and Approximation (SARA)*, 2000.

Search Strategies for Optimization: Modelling the SONET Problem

Barbara M. Smith

School of Computing & Engineering, University of Huddersfield, U.K.
b.m.smith@hud.ac.uk

Abstract. An optimization problem arising in the design of optical fibre networks is considered. Initially, it proved very difficult to solve using constraint programming: a straightforward CP model could only solve very small instances. It was remodelled using a variety of CP modelling techniques that are by now standard: symmetry breaking, implied constraints, new variables linked to the original search variables by channelling constraints. A novel two-phase search has been introduced; first assigning values to a set of intermediate variables in order to guide the subsequent assignments to the original search variables. Variable ordering heuristics have a huge impact on search effort; the two-phase search complicates the choice, but good heuristics have been found which allow the test instances to be solved. Finally, the optimization process has been changed to guide the search better.

1 Introduction

In this paper, the development of constraint programming models for an optimization problem arising in the design of optical fibre networks is discussed. The problem was studied by Sherali and Smith [6] who used a mixed integer linear programming model. A model using similar variables and constraints, translated into a CP solver, could solve only very small instances of a simplified version of the problem.

The problem has been remodelled, using the armoury of CP modelling, including symmetry breaking; introduction of implied constraints (including both those logically implied by the other constraints, and those implied by the fact that this is an optimization problem); new variables representing different aspects of the problem linked to the original variables by channelling constraints. The most important feature of the remodelling has been the design of the search strategy: firstly, the search for solutions is conducted in two phases, first assigning values to a set of intermediate variables, insufficient by themselves to solve the problem, and then to the original search variables. Secondly, the variable ordering heuristics used are of crucial importance; several choices are discussed. Finally, the usual approach of using successive values of the objective to provide a bound on future solutions has been changed to avoid wasted effort considering non-optimal solutions.

2 Problem Description

The problem is described by Sherali and Smith [6] in a paper on dealing with symmetry in linear programming problems (integer and mixed integer) by adding symmetry-breaking constraints to the model. They treat three different problems including the SONET problem.

To quote from Sherali & Smith: "The SONET is a standard of transmission technology over optical fiber networks." The network contains a number of client nodes and there are known demands (in terms of numbers of channels) between pairs of nodes. A SONET ring joins a number of nodes; a node is installed on a SONET ring using an 'add-drop multiplexer' (ADM) that is capable of adding and dropping the traffic. Each node can be installed on more than one ring, and traffic can be routed between a pair of client nodes only if they are both installed on the same ring: there is no traffic allowed between rings. There are capacity limits on the rings (in terms of both nodes and channels). The objective is to minimise the total number of ADMs required, while satisfying all the demands.

The 15 randomly-generated test instances used by Sherali & Smith have 13 nodes, with 24 demand pairs. The SONET rings can accommodate 5 nodes and 40 traffic channels, and there are 7 rings available. (From other descriptions, it appears that the cost of SONET rings is negligible, so that there is no practical limit on the number of rings used. The limit is specified in order to model the problem.) 80% of the demand pairs were uniformly generated between 1 and 5, and 20% between 1 and 25. There are also two smaller sets of 15 problems each[1].

A sample set of demand pairs, taken from one of the test problems, is given below. The first line gives the origin nodes, the second the destination nodes, and the third the demands in terms of number of channels.

```
1  1  2 2 2  2  2  3 4 4  4  4 5 5 7  7  7  8  8  8  9 10 11 12
9 11  3 5 9 10 13 10 5 8 11 12 6 7 9 10 12 10 12 13 12 13 13 13
8  2 25 5 2  3  4  2 4 1  5  2 5 4 5  2  6  1  4  1  5  9  3  2
```

The data suggest that the limit on the number of nodes on each ring will have more influence on the solution than the limit on demand. Hence, initially the level of demand between pairs of nodes will be ignored. At worst, the solutions found will give a lower bound on the minimum number of ADMs possible when the demand levels are taken into account.

For the instance just given, an optimal solution ignoring the traffic levels uses 22 ADMs on 4 rings. The sets of nodes installed on the rings are: {4, 8, 10, 12, 13}, {4, 5, 6, 11}, {1, 2, 9, 11, 13}, {7, 9, 12}. However, this is not a feasible solution if the level of demand is taken into account, because the third ring requires 41 channels, and the demand pairs on this ring are not on any other ring, so that the demand cannot be split between two rings. An optimal solution for this instance respecting the traffic limit uses 23 ADMs, as will be seen below.

[1] These are not used in their paper, but were sent to me by Cole Smith, along with the larger instances

3 A CSP Model with Integer and Set Variables

Sherali and Smith use 0/1 variables in modelling the SONET problem:

$$x_{ik} = 1 \text{ if node } i \text{ is assigned to ring } k, 1 \leq i \leq n, 1 \leq k \leq m$$
$$0 \text{ otherwise}$$

where n is the number of nodes and m is the number of available rings.

It is of course possible to model the problem as a CSP using only these variables. However, the resulting CSP can only be solved in any reasonable time for the 15 smallest instances (with 7 nodes, 8 demand pairs, 4 available rings, and a maximum of 4 nodes allowed on each ring). It is in any case difficult to model the constraint that if there is a demand between a pair of nodes, they must both appear on the same ring, and partly to make it easier to express this constraint, two dual sets of set variables were introduced: variable R_k represents the set of nodes assigned to ring k, and N_i is the set of rings that node i is assigned to. The x_{ik} variables are still used as the search variables.

The constraints are:

– $|R_k| \leq r$ where r is the maximum number of nodes on each ring, i.e. 5. The constraint could alternatively be expressed as: $\sum_i x_{ik} \leq r$.
– if there is a demand between nodes i and j, then $|N_i \cap N_j| \geq 1$ (i.e. there must be at least one ring that they are both assigned to).
– channelling constraints: $x_{ik} = 1$ iff $i \in R_k$ and iff $k \in N_i$.

The objective, the total number of ADMs in the network, is represented by an integer variable t, where $t = \sum_k |R_k|$. There is an implied constraint that $t = \sum_i |N_i|$. The optimal solution can be found using the optimization facilities in ILOG Solver: whenever a solution is found, Solver adds a constraint that in future solutions, the value of t must be smaller. Hence, when there is no solution satisfying the current constraint on t, the last solution found is known to be optimal.

Trying to construct good solutions by hand led to variable and value ordering heuristics which often find optimal or near-optimal solutions quickly. The next variable chosen is x_{ik}, where ring k is the smallest numbered ring not yet fully occupied and i (the node to place on this ring) is chosen according to the following criteria:

– if there is a node whose future (i.e. not yet accommodated) connections are all to nodes already on ring k, and those nodes in turn have no future connections other than to this node, choose it. (If the node is not allocated to ring k, it will have to be placed on another ring, along with its neighbouring nodes that are already on ring k, which is likely to be much more expensive.)
– if there is no such node, choose the node that is connected to the largest number of nodes already on ring k.

– break ties by choosing the node with largest future degree, defined to be the number of demand pairs involving the node that have not yet been accommodated on a ring. The tie-breaker is always used when starting a new ring, for instance.

The value ordering is to choose 1 before 0, i.e. when considering variable x_{ik}, choose to place node i on ring k before choosing not to place it.

A number of implied constraints can be added to the model, as well as the one already mentioned, that $\sum_k |R_k| = \sum_i |N_i|$.

– if the degree δ_i of node $i \geq r$, it must be placed on more than one ring, i.e. $|N_i| > 1$;
– if two nodes i and j are connected, and each of them is connected to fewer than r other nodes (so that the first implied constraint does not apply), but together they are connected to at least $r - 1$ other nodes, then at least one of them must be on at least two rings i.e. $|N_i| + |N_j| \geq 3$.

As an example of the second type of implied constraint, suppose two nodes with a demand between them each have 2 other neighbours, and they have no neighbours in common, e.g. nodes 1 and 7 in Figure 1 below. If we consider each node in isolation, it seems that we could put it on just one ring. However, there must be a ring that both nodes are installed on, and there is not room for both sets of neighbouring nodes on the same ring. Hence, at least one of the nodes must also be on another ring.

The following constraints can also be added. They are not true of every consistent solution, and so are not logical consequences of the existing constraints, as implied constraints are, but they will be satisfied by at least one optimal solution:

– a ring cannot have just one node on it, i.e. $|R_k| \neq 1$, for $1 \leq k \leq m$;
– the total number of nodes allocated to two non-empty rings must be more than the number that can be accommodated on one ring, i.e. if $|R_k| > 0$ and $|R_l| > 0$ then $|R_k| + |R_l| > r$, for $1 \leq k < l \leq m$. Otherwise there is an equally good solution in which the two rings are combined into one.

All these additional constraints are useful: they reduce both search and runtime.

4 Symmetry Breaking

The available rings are indistinguishable: in any solution, the rings, with their associated nodes, can be permuted without changing the solution. If the variables x_{ik} are thought of as corresponding to the elements of a 2-dimensional matrix, the columns of the matrix (corresponding to the 2nd subscript) can be permuted. This symmetry can be eliminated using SBDS (Symmetry Breaking During Search) [3] by supplying symmetry functions describing the transpositions of pairs of rings/columns.

It would be possible to remove the symmetry between the rings in some other way, for instance by imposing ordering constraints between the sets of nodes on each ring. But it is important to ensure that symmetry-breaking constraints do not conflict with the variable ordering; it would be difficult to do this here, since the ordering is dynamic.

With SBDS and with the implied constraints and heuristics described earlier, the 15 medium-sized instances (10 nodes, 15 demand pairs, 6 rings and a maximum of 5 nodes allowed on each ring) are solvable in a reasonable time. They can be solved, even without breaking the symmetry, if the number of available rings is restricted to 4, instead of 6. Four rings are in fact sufficient, since none of the optimal solutions require more. These results are shown in Table 1: for the instances which are easiest to solve, eliminating the symmetry makes little difference, if any, especially in finding the optimal solution. For the most difficult instances, it reduces search overall by an order of magnitude.

Table 1. Solving medium-sized SONET instances, allowing only 4 rings, with and without symmetry-breaking, using ILOG Solver. 'Value' is the minimum number of ADMs required. F is the number of backtracks (fails) to find the optimal solution, P is the total number of backtracks to prove optimality. Time is the cpu time in seconds on a 600MHz Celeron PC.

		With SBDS			No symmetry breaking		
Instance	Value	F	P	Time	F	P	Time
1	14	1	15	0.11	1	15	0.07
2	14	1	25,044	11.6	1	188,664	52.4
3	14	24	103	0.14	24	324	0.21
4	13	1,618	27,264	12.8	3,922	172,315	58.2
5	15	1	82,531	40.6	1	745,525	287
6	14	1	36,901	16.8	1	279,085	91.6
7	13	922	933	0.81	2,126	2,137	1.33
8	14	1,657	28,206	12.4	4,180	187,902	52.9
9	15	36,395	71,960	41.3	133,684	507,068	196
10	14	374	435	0.36	983	1,256	1.03
11	12	320	331	0.35	751	762	0.91
12	15	1,021	117,855	62.3	4,976	1,292,875	553
13	15	1,870	89,117	47.0	10,770	1,018,949	433
14	15	9	71,127	35.5	9	908,497	387
15	15	4	53,851	26.0	5	468,969	180

5 Faster Proofs of Optimality

Figure 1 shows the demand graph of one of the medium sized instances (instance 15). An optimal solution for this instance has 15 ADMs on 3 rings, represented by the sets {2,3,6,8,9}, {1,3,4,7,9}, {2,5,6,7,10}, i.e. the value of the objective variable t is 15. This solution is found in 4 backtracks, using the model described

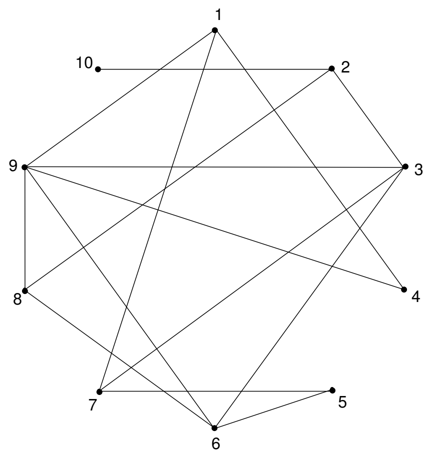


Fig. 1. Demand pairs in a sample instance of the SONET problem

in the last section, but proving that it is optimal takes 53,581 backtracks. (This demonstrates that in optimization problems, it is worthwhile to record separately the effort required to *find* the optimal solution for the first time as well as the overall search effort, including *proving* that this solution is optimal. When trying to improve performance, it is useful to know whether most effort is going into finding good enough solutions, or into the proof of optimality, as here.)

How could we prove optimality more quickly? To prove that there is no solution with value 14, i.e. with $t = 14$, we attempt to find such a solution. To do that we need to try to reduce the number of nodes that appear on more than one ring. Node 9 must appear at least twice (because its degree in the demand graph is 5, and there are at most 5 nodes on each ring). There are also implied constraints on several pairs of nodes, specifying that at least one of the pair must appear at least twice, e.g. 1 and 7, 2 and 3. In the solution we are considering, most of these pairs already appear the minimum number of times; for instance, node 1 appears once and node 7 twice. The only exception is nodes 2 and 3, which both appear twice. To reduce the number of ADMs, we must find a solution where either node 2 or node 3 only appear once.

Suppose we introduce new integer variables n_i, where $n_i = |N_i|$. (Recall that N_i is a set variable representing the rings that node i is assigned to, and that $t = \sum_i |N_i|$.) n_i is the number of ADMs used for node i, i.e. the number of times that node i appears in the solution. These variables can be used to prove that the solution given earlier is optimal. First, the constraint that $t < 15$ is imposed, so that only solutions better than the one already found will be accepted. Adding the constraint $n_2 < 2$ leads to an immediate fail, so $n_2 \geq 2$. If the constraints $n_2 \geq 2$ and $n_3 < 2$ are imposed, there is no solution: the proof of this takes 122 fails. Hence we cannot have either node 2 or node 3 appearing just once in a solution with value < 15, and so cannot reduce the number of nodes in the solution: hence, the solution given earlier has been proved optimal, with little search.

The proof just outlined resulted from adding constraints to the model by hand. To automate the process, the n_i variables are used as search variables, and values are assigned first to them and then to the x_{ik} variables, i.e. the search first decides how many rings each node is on, and then decides which nodes are on each ring. This is still a complete algorithm; if no consistent assignment is found to the x_{ik} variables, the search will reassign the n_i variables.

To avoid assignments to the n_i variables which cannot be extended to a consistent solution, additional upper bounds on the n_i variables are added to the model; if the degree of node i is δ_i, it must appear on no more than δ_i rings, i.e. $n_i \leq \delta_i$. Note that this would not be a legitimate constraint if the demand levels were being taken into account.

Furthermore, since values are being directly assigned to the n_i variables, the following constraints can be added:

- if $n_i = 1$, and node i is installed on ring k, then all the neighbours of node i must also be on ring k, i.e. if A_i is the set of neighbours of node i:

$$\text{if } n_i = 1 \text{ and } x_{ik} = 1 \text{ then } \{i\} \cup A_i \subseteq R_k \quad \text{for } 1 \leq i \leq n, 1 \leq k \leq m \tag{1}$$

- similarly, if $n_i = 2$, once node i has been allocated to two rings, all its neighbours must be on these two rings as well, i.e.

$$\begin{aligned} &\text{if } n_i = 2 \text{ and } x_{ik} = 1 \text{ and } x_{il} = 1 \text{ then } \{i\} \cup A_i \subseteq R_k \cup R_l \\ &\qquad\qquad\qquad\qquad\qquad\qquad \text{for } 1 \leq i \leq n, 1 \leq k < l \leq m \end{aligned} \tag{2}$$

These constraints help to complete the solution quickly or decide that it cannot be completed, once an assignment to the n_i variables has been made.

The symmetry between the rings can be broken in the same way as before, using the same SBDS functions. Symmetry breaking is only needed when assigning values to the x_{ik} variables; the symmetry does not affect the n_i variables. With the two-phase model, it would be even more difficult than before to use symmetry-breaking constraints and ensure compatibility with the variable ordering, since the order in which the x_{ik} variables are assigned depends on the previous assignments to the n_i variables.

The variable ordering heuristics used for the two sets of search variables are discussed separately below in section 8.

With the two-phase search just described, the medium-sized instances can be solved easily, as shown in Table 2. These are the same instances as in Table 1, except that the number of rings available is 6, as in the original data, instead of 4, which makes them harder to solve. Even so, with SBDS, these instances are solved much more quickly than with the simpler model used in Table 1. As before, symmetry breaking is clearly essential to being able to solve these problems.

Each of the largest instances (i.e. those used in Sherali & Smith's paper) can now easily be solved in under 25 seconds, provided that the number of available

Table 2. Solving medium-sized SONET instances with two-phase search, with and without symmetry-breaking.

		With SBDS			No symmetry breaking		
Instance	Value	F	P	Time	F	P	Time
1	14	13	21	0.62	553	561	2.27
2	14	1	13	0.10	1	13	0.06
3	14	16	29	0.64	241	251	1.34
4	13	1,320	1,338	6.98	>50K	-	-
5	15	156	174	1.24	>50K	-	-
6	14	0	14	0.57	0	14	0.10
7	13	4	16	0.55	13	25	0.11
8	14	1	13	0.56	1	13	0.09
9	15	7	26	0.63	19	715	2.43
10	14	707	715	3.74	40,989	40,997	118
11	12	1	12	0.54	1	12	0.08
12	15	0	25	0.61	0	590	2.27
13	15	32	63	0.90	5,366	5,965	19.2
14	15	398	498	2.85	30,169	38,083	128
15	15	2	17	0.57	2	588	2.36

rings is restricted to 5 and the demand level between pairs of nodes is still ignored. It is now time to reconsider the original problem, and in particular the demand capacity of the rings.

6 Modelling demand capacity

Sherali & Smith modelled the demand capacity of the rings by introducing a variable f_{kij} for every pair of nodes i, j with a traffic demand between them, and every ring k: f_{kij} is the fraction of the demand between nodes i and j that is assigned to ring k ($0 \leq f_{kij} \leq 1$). This gives a mixed integer programming model. These variables are used to model the fact that if a pair of nodes occurs on more than one ring, the demand between the nodes can be split over these rings.

Using fractional variables seems an unpromising approach for constraint programming, and moreover introduces alternative solutions. For instance, the optimal solution for the example in section 2, taking the demand levels into account, has a ring with the set of nodes {2,3,7,10,13}. The total traffic demand between these nodes is 45 channels, whereas the capacity of a ring is 40. However, this solution is feasible, because nodes 10 and 13, with a demand of 9 units between them, are also both installed on another ring with enough spare capacity to take all the demand between these two nodes. Hence, a feasible solution can allocate either 0, 1, 2, 3 or 4 channels on the first ring to the demand between nodes 10 and 13 and the remaining demand from this pair to the second ring.

To avoid exploring equivalent alternative solutions during search, we should avoid making the decision about how the demand is split between the rings, and simply ensure that it can be done feasibly by adding further constraints to the existing model.

First, more variables are needed, relating to the edges in the demand graph, to allow the constraints to be expressed. Let

$$\begin{aligned} w_{pk} = {} & 1 \text{ if edge } p \text{ is on ring } k, 1 \leq l \leq e, 1 \leq k \leq m \\ & 0 \text{ otherwise} \end{aligned}$$

If edge p links nodes i and j, then we have the following channelling constraints: $w_{pk} = 1$ iff $x_{ik} = 1$ & $x_{jk} = 1$, for every edge p. Also let E_k be the set of edges on ring k, giving the further channelling constraints: $w_{pk} = 1$ iff $p \in E_k$.

We could insist that the total demand for channels from the edges on any ring must be no more than the capacity of the ring, i.e.

$$\sum_{p:p \in E_k} d_p \leq b \quad 1 \leq k \leq m \tag{3}$$

where d_p is the traffic demand on edge p, and b is the traffic capacity of a ring.

However, this constraint is too tight and would not allow the demand between a pair of nodes to be split between two or more rings. Hence, although any solution found that satisfies this constraint is feasible, it may not be optimal.

It is clearly necessary that if a demand pair occurs on only one ring, the demand between the nodes must fit onto that ring, and a ring must accommodate all the demands that cannot be assigned to any other ring. Hence, for any ring, the sum of the demands between demand pairs that occur only on that ring must be no more than the capacity of the ring:

$$\sum_{p, p \in E_k; p \notin E_l, l \neq k} d_p \leq b \quad 1 \leq k \leq m \tag{4}$$

Although these constraints allow a demand pair to be on more than one ring, they do nothing to check that the total demand between the pair can be accommodated along with the other demand pairs on these rings. Hence the constraints are too loose and may allow infeasible solutions.

We keep the constraints (4) and add constraints on pairs of rings. If a demand pair occurs on two rings, the sum of the demands that occur only on those two rings must be no more than twice the ring capacity:

$$\sum_{p, p \in E_k \cup E_l; p \notin E_h, h \neq k,l} d_p \leq 2b \quad 1 \leq k < l \leq m, E_k \cap E_l \neq \emptyset \tag{5}$$

The model is extended to include constraints (4) and (5) for every ring and for every pair of rings. Although tighter than constraints (4) alone, they might still conceivably allow infeasible solutions, since there may be demand pairs on either or both of the rings k, l which are also on other rings and so not included in the constraints. To prevent this the constraints could be extended to sets of three rings, and so on. However, in practice, for the instances considered, the optimal solutions found are feasible, and the demands will fit onto the rings as allocated.

Some of the implied constraints added earlier must also be modified when the demand capacity of the rings is taken into account. For instance, the constraint that any two rings must have a total of more than r nodes installed on them is no longer correct, but can be modified so that two rings *either* have more than r nodes *or* more than b channels between them. Further, it is no longer necessarily true that a node with degree δ_i should appear on at most δ_i rings, and this constraint is dropped.

Having modified the model to handle the level of demand between nodes, the large instances can be solved. First, an upper bound on the new optimal value is found. This can be done easily; the existing solutions only violate the demand capacity on one or two rings, and the infeasibility can be fixed by duplicating one or two demand pairs on a new ring, or an existing ring with spare capacity. (Currently, this is done by hand but the process could be automated.) This gives a feasible solution using only a few more nodes than the solution already found, and hence an upper bound on the value of the new optimum.

Table 3 shows, in the left-hand columns, the performance of the two-phase model in solving the original Sherali & Smith instances (13 nodes, 24 demand pairs, ring capacity 5 nodes and 40 traffic channels, and 7 rings available) if the traffic capacity of the rings is ignored. For six of these 15 instances, the solutions found are still feasible (and therefore optimal) if the traffic between each pair of nodes and the traffic capacity of each ring is taken into account. In the other cases, the optimal solution already found gives lower and upper bounds on the new optimum, and the right-hand columns show the performance of the two-phase search in re-optimizing, using these bounds to limit the search. The average solution time is 167 sec. (including both finding the optimal solution ignoring the demand levels and then if necessary re-optimizing using upper and lower bounds derived from this solution).

7 The Objective as a Search Variable

The final improvement in performance arose from examining the assignments to the n_i variables considered during search. When a variable n_i is chosen as the next variable to assign, each value in its domain is tried in turn. Before a solution has been found, values up to $\min(\delta_i, m)$ will be tried. However, if the degree δ_i of node i is large, for the largest values in a variable's domain, $\sum_i n_i$ will be larger than in the optimal solution. Considering such assignments is ultimately wasted effort. This leads to the idea of considering assignments for which the objective variable t has a specified value, t_0, and only increasing t_0 once every assignment to the n_i variables for which $\sum_i n_i = t_0$ has been considered without success. This is easily implemented by making t a search variable and choosing it first.

In this way, the search will never consider assignments in which $\sum_i n_i$ is more than the optimal value; indeed, the first solution found is guaranteed to be optimal. Every assignment in which $\sum_i n_i$ is less than the optimum must still

Table 3. Solving large SONET instances, with a two-phase search process. Dashes in the right-hand columns indicate that the solution already found ignoring the demand capacity of the rings is still optimal when it is taken into account.

	Without demands				With demands				
Instance	Optimal	F	P	Time	Initial bound	Optimal	F	P	Time
1	22	32	1,424	12.8	24	22	3,864	3,883	69.2
2	20	11,988	12,004	122.4	-	20	-	-	-
3	22	427	1,370	14.8	-	22	-	-	-
4	23	19,824	24,186	196	25	23	8,668	8,686	144
5	20	221	234	2.85	22	22	17	1,992	39.0
6	22	148	1,200	10.5	-	22	-	-	-
7	20	61,492	61,514	451	22	20	1,685	6,534	127
8	20	2,222	2,256	26.4	-	20	-	-	-
9	22	6,144	7,480	57.5	26	23	18,218	25,932	421
10	23	3,252	5,944	52.4	25	24	1,056	2,708	57.9
11	22	5,220	5,322	44.7	-	22	-	-	-
12	20	914	944	8.98	24	22	249	1,209	43.3
13	21	1,506	2,998	26.2	-	21	-	-	-
14	23	405	1,154	11.8	25	23	3,130	3,145	59.6
15	22	41,277	41,349	416	23	23	3,312	4,284	83.3

be considered, as before, in order to prove optimality. In the previous model, all such assignments which have not previously been encountered during the search are considered once the optimal solution is found, in order to complete the proof of optimality. Now, all these assignments will already have been met by the time the optimal solution is found.

This would not be a useful solution strategy if finding the optimal solution were likely to be very difficult and we were prepared to settle for a good solution and forgo the proof of optimality. For these instances of the SONET problem, the minimum value of t which need be considered is between 18 and 20 (given the implied constraints on the n_i variables) and the optimal value is between 20 and 23. If the gap were much larger, proving optimality might be unrealistic.

Table 4 shows the results with this strategy. It is no longer necessary to derive an upper bound on the optimal solution with demand levels from the optimal solution ignoring them. The optimal solution with demand levels was found in all cases, even when the solution already found was still feasible. It is usually very quick to find the same solution again in these cases, and avoids checking the solution for feasibility by hand (although an automatic check would be quicker).

The average solution time for the instances used in the Sherali & Smith paper is now 63 sec. (including an average of 15 sec. to find the optimal solution ignoring the demand levels). In [6], the average solution time (with the level of symmetry breaking giving best performance) was 21,000 sec., although on Sun Ultra 10 (a much slower machine), and the corresponding average number of nodes in the branch and bound tree using CPLEX was about 600. It is hard to compare the performance of the CP model and the mixed integer LP model, but it is clear that these instances are now relatively easy to solve using the CP model and search strategy described.

On the other hand, the success of the CP model does to some extent depend on the characteristics of the set of instances. In particular, the solution ignoring the level of demand is not too far from the true optimal solution and constraints (4) and (5) in section 6 are sufficient to give a feasible solution.

Table 4. Solving large SONET instances, with the objective as the primary search variable. * indicates that the optimal solution without taking the demand levels into account is also feasible for the full problem.

	Without demands			With demands		
Instance	Optimal	F	Time	Optimal	F	Time
1	22	1,425	15.8	22	442	9.14
2	20	61	1.27	20*	52	1.90
3	22	1,065	13.3	22*	72	2.44
4	23	5,722	58.8	23	6,147	101
5	20	192	2.32	22	2,036	42.6
6	22	1,148	12.9	22*	25	0.69
7	20	151	2.12	20	6,525	128
8	20	118	2.14	20*	49	1.78
9	22	1,626	18.1	23	12,154	219
10	23	3,252	41.8	24	2,516	57.0
11	22	263	3.85	22*	90	2.80
12	20	183	2.67	22	1,135	22.7
13	21	2,637	29.0	21*	673	14.6
14	23	938	12.1	23	1,257	24.8
15	22	668	9.06	23	4,271	88.5

8 Variable Ordering for Two-Stage Search

For the two-stage search, with two separate sets of search variables being assigned consecutively, two variable ordering heuristics are needed. The results already given for these models (Tables 2, 3 and 4) depend on the variable ordering heuristics used. The same heuristics have been used for the initial two-stage search, described in section 5, and also for the variants that deal with the demand capacities of the rings and/or choose the value of the objective variables first.

A number of variable ordering heuristics for the first phase, i.e. for assigning the n_i variables, have been tried. However, the comparison is clearer in the final model, ignoring the demand capacity. In that model, three heuristics have been compared: smallest domain, minimum degree and maximum degree.

Any variable ordering will generate the same set of sub-optimal complete assignments to the n_i variables to pass to the second stage of search (i.e. assignments for which the value of t is less than its optimal value), since all assignments satisfying the implied constraints on the n_i variables must be found, whatever heuristic is used, and will fail only when the second stage tries to assign the x_{ik} variables. The different heuristics tested sometimes require slightly different numbers of backtracks to explore the suboptimal assignments, but they principally differ in the number of complete assignments to the n_i variables that they consider at the optimal value of t, before finding one that leads to a solution. On average, minimum degree is much worse, and smallest domain slightly worse, than maximum degree. Evidently the total number of complete assignments to the n_i variables at the optimal value of t is again the same for all three heuristics, but they each consider them in a different order, and hence find a solution earlier or later. Its poor performance indicates that the assignments considered first by minimum degree are less likely to be successfully extended to the x_{ik} variables than those considered first by maximum degree, but it is difficult to identify the reason.

Maximum degree also compares well with the other two variable ordering heuristics for the other variants of the two-stage search, i.e. for the previous model with the standard form of optimization and for both models with the constraints to take into account the demand capacities of the rings.

A number of variable ordering heuristics have been investigated for the x_{ik} variables, which are assigned once the a complete assignment to the n_i variables have been found. In all cases, the smallest numbered ring not yet fully occupied is chosen, and then a node chosen to place on this ring. The best performance was found by choosing the node i assigned to fewest rings (i.e. for which n_i is smallest), breaking ties by choosing the node with largest degree. This means that any node that is only on one ring (i.e. $n_i = 1$) is placed first. Given the set of constraints (1) in section 5, all its neighbours will then be placed on the same ring; by choosing the node with maximum degree, the largest number of neighbouring nodes will be placed. Since these nodes must be on the same ring, postponing placing them is likely to lead to future failure and wasted search.

The heuristics investigated were selected largely by trial and error, and so it is very likely that better heuristics exist. The choice, especially in the first stage, is more complicated than when there is only one set of search variables. The aim is not just to find an assignment to satisfy the direct constraints on the first-stage search variables (which is easily done), but to find one that can be extended to the second-stage variables. Hence, although the second stage variables do not directly influence the first stage of search, they have an indirect influence on the performance of variable ordering heuristics that is hard to take into account.

9 Discussion

The first simple CP model for the SONET problem could only solve very small instances; after a lot of effort redesigning the model and devising better search strategies, much larger instances can now be solved comfortably.

The two-phase search, in the second model presented, is novel and could be useful in other contexts. Here, the search first decides how many rings each node should be installed on and then decides which rings they should be. A similar idea was used for a difficult instance of the template design problem [4]. The common features of the two-phase search in the two problems are:

- the first phase assigns values to a set of 'intermediate' variables. These are closer to the objective than the original variables, in the sense that an assignment to an intermediate variable can propagate to the objective. Conversely, tightening the bounds on the objective can prune the domains of the intermediate variables. In contrast, the original variables and the objective have very little direct effect on each other;
- in both the template design problem and the SONET problem, the objective is the sum of the intermediate variables. However, other relationships might also be possible;
- an assignment to the intermediate variables does not give a complete solution to the problem, but it does make finding a complete solution (or proving that there is no solution) much easier.

The intermediate variables can be thought of as a bridge between the objective variable and the original decision variables, allowing a new bound on the objective to propagate through to the decision variables. This could be a useful strategy in other cases where the objective can, in effect, be decomposed into a set of intermediate variables.

An extension to the two-stage search minimizes the objective by considering progressively increasing values, proving that each value cannot be attained before going on to the next. This means that the first solution found is guaranteed to be optimal. This is in contrast to the usual branch-and-bound approach where an incumbent solution is improved until no further improvement can be made, and proving optimality often requires extensive search after the optimal solution is found. The approach adopted here could be useful for other cases where the objective has only a small range of possible values.

Another feature of modelling the SONET problem that is found in modelling other problems is that the variable ordering heuristics have a profound effect on the performance. As noted earlier, the variable ordering heuristics used were derived by trial and error. Currently, we do not know enough about how to choose variable ordering heuristics; much of what we do know is based on studies on random binary constraint satisfaction problems, and may not transfer to CSPs derived from real problems if the constraints are neither binary nor uniform. Standard heuristics may be still less applicable to optimization problems and the two-stage search adopted for this problem complicates matters further. Previous experience has shown that heuristics which are a good choice when trying to find a good solution may be much less effective when it comes to proving optimality [7]; in general, heuristics for optimization problems have to do both. In order to be able to choose good heuristics for optimization problems, we will either need a better understanding of these issues, or a systematic way of choosing heuristics tailored to specific problems (as in [1], for instance); preferably, both. It is quite possible that much better heuristics for the SONET problem could then be found and that larger problems could be solved more quickly.

Considerable effort has been put into remodelling the problem; several iterations were needed before the larger problems could be solved satisfactorily. Remodelling currently requires expertise in constraint programming, and it would be preferable if the process could be automated or at least supported. A proposal for a system to do this is presented by Frisch *et al.* [2], using models of the SONET problem as an illustration.

Remodelling an initially intractable problem has eventually resulted in a CP model that appears competitive with Sherali & Smith's mixed integer LP model. Optimization is generally considered to be difficult for CP, but more effort put into CP modelling could pay off in other optimization problems too. Régin [5], for instance, has developed a CP approach to solving the maximum clique problem and has shown that it is competitive with existing methods, achieving new solutions to benchmark problems. Improved CP models might improve overall performance even for problems that are eventually solved using a CP/IP hybrid.

Acknowledgments. The author is a member of the APES group (see http://www.dcs.st-and.ac.uk/~apes) and would like to thank the other members for their continuing encouragement and interest through several iterations of work on this problem. I am also grateful to Hanif Sherali and Cole Smith for sending me their sample instances, and to Sarah Fores and Les Proll for bringing the problem to my attention. This work was supported by EPSRC grant GR/M90641.

References

1. S. L. Epstein, E. C. Freuder, R. Wallace, A. Morozov, and B. Samuels. The Adaptive Constraint Engine. In P. van Hentenryck, editor, *Principles and Practice of Constraint Programming - CP 2002*, LNCS 2470, pages 525–540. Springer, 2002.
2. A. M. Frisch, B. Hnich, I. Miguel, B. M. Smith, and T. Walsh. Towards Model Reformulation at Multiple Levels of Abstraction. In A. M. Frisch, editor, *Proceedings of the International Workshop on Reformulating Constraint Satisfaction Problems: Towards Systematisation and Automation*, 2002. Available from http://www-users.cs.york.ac.uk/~frisch/Reformulation/02/Proceedings/.
3. I. P. Gent and B. M. Smith. Symmetry Breaking During Search in Constraint Programming. In W. Horn, editor, *Proceedings ECAI'2000*, pages 599–603, 2000.
4. L. G. Proll and B. M. Smith. ILP and Constraint Programming Approaches to a Template Design Problem. *INFORMS Journal on Computing*, 10:265–275, 1998.
5. J.-C. Régin. Using Constraint Programming to Solve the Maximum Clique Problem. In F. Rossi, editor, *Principles and Practice of Constraint Programming - CP 2003*, LNCS. Springer, 2003.
6. H. D. Sherali and J. C. Smith. Improving Discrete Model Representations Via Symmetry Considerations. *Management Science*, 47:1396–1407, 2001.
7. B. M. Smith, K. E. Petrie, and I. P. Gent. Models and Symmetry breaking for 'Peaceable Armies of Queens'. Technical Report APES-50-2002, APES Research Group, May 2002. Available from http://www.dcs.st-and.ac.uk/~apes/apesreports.html.

A Generalisation of the Backtracking Algorithm

M.R.C. van Dongen[⋆]

Cork Constraint Computation Centre, Computer Science Department, University College Cork,
dongen@cs.ucc.ie

Abstract. This paper presents a generalisation of the well known chronological backtracking algorithm. The algorithm is a generalisation in the sense that it can use *any* kind of constraint (as opposed to just unary constraints—the domains of the variables) to decompose a problem into smaller problems. This choice keeps the height of the search tree less than or equal to the number of variables. Most importantly, however, this choice will never increase the local branching factor of the search tree but will frequently make it smaller.

1 Introduction

This paper presents tools to analyse and dissect constraints. The tools are the building blocks for a new *generalised backtracking algorithm* which is a generalisation of the well known chronological backtracking algorithm. Generalised backtracking is sound and complete.

The motivation for the generalised backtracking algorithm is as follows. It has been observed in the mathematical community that a solution strategy for systems of multivariate polynomial equations where Gröbner bases with respect to total degree orders are factorised[1] and the induced problems are solved is to be preferred to a strategy where lexicographical Gröbner bases (elimination ideals/strict "lexicographical" rules) dictate the order in which equations should be used to decompose the problem [1, 3, 10, 15, 16, 18]. The reasons are two-fold. Firstly, the Gröbner bases with respect to total degree orders are (normally) easier to compute. Secondly, the polynomials occurring in the total degree bases (normally) have a lower degree thereby leading to factors of lower degree. Gräbe furthermore observes that problems coming from real life often fulfill the condition of being factorisable [10].

For example, in the presence of a *binary* algebraic constraint $(x-1)\times(x-y) = 0$ it is probably a good decision to factorise it to get two constraints $x = 1$ and $y = x$, to branch on these two constraints, and to do a bit of extra work to remove

[⋆] This work has received support from Science Foundation Ireland under Grant 00/PI.1/C075.

[1] Here and in the remainder of this paper a factorisation of a problem is a set of problems the union of whose solutions are equal to the solutions of the original problem. This is analogous to the notion of factorisation of a polynomial: the zeros of the original polynomial are equal to the zeros of the members of the factorisation.

duplicate solutions even if there is a *unary* constraint $(x-1) \times \cdots \times (x-d) = 0$, for some $d > 2$, for which a factorisation is known. There are at least two reasons why this is probably better. The first reason is that branching on this binary constraint requires only 2 branches as opposed to $d > 2$ branches for the unary constraint. The second reason is that the resulting constraints $x = 1$ and $y = x$ are about as tight as you can get. Both allow for the elimination of a variable, just like each of the members $x = i$, $1 \leq i \leq d$, of the factorisation of the unary constraint allow for the elimination of a single variable.

The strategy used by the chronological backtracking algorithm is similar to the "lexicographical" approach mentioned before. The generalised backtracking algorithm on the other hand is motivated by similar observations as the "total degree" approach. To be more specific, the generalised backtracking algorithm is not restricted to the use of unary constraints (the domains of the variables) alone to decompose problems.

Both the chronological backtracking algorithm and the generalised backtracking algorithm traverse search trees. The chronological backtracker decomposes problems at each internal node of the search tree by considering a unary constraint (the domain of a variable). For each member in the unary constraint it creates a sub-problem. The number of sub-problems that has to be considered is equal to the cardinality of the unary constraint. In terms of tree traversals the unary constraint determines the (local) number of branches whose sub-trees have to be traversed. This number is called the *local branching factor*. As already indicated, the generalised backtracking algorithm can use *any* kind of constraint to obtain a problem decomposition. It will be demonstrated that this will never result in a higher local branching factor but may result in a lower local branching factor in return for a marginal increase in the space complexity.

The chronological backtracking algorithm has received much attention from many researchers. Variants of the algorithm range from a vanilla version [9], to forward checking [11] and MAC [20], and to backjumping [6], conflict directed backjumping [19], and dynamic backtracking [8]. For detailed treatments of and surveys of backtracking the reader may wish to consult [5, 13, 14, 17]. The reader may wish to consult [7, 21] for an introductory treatment to backtracking.

It is a well established fact that in order to keep (backtrack) search efficient it is imperative that the branching factors of the nodes near the root of the search tree be kept as small as possible. The contribution of generalised backtracking is that it is the first attempt to keep the branching factor of the search tree small by analysing the structure of *any* kind of constraint and by using an alternative (exhaustive) way to enumerate the members of the constraint. This alternative way to explore the search space generalises the notion of static and dynamic variable orders.

The remainder of this paper is organised as follows. Section 2 is a brief introduction to constraint satisfaction. Section 3 introduces the notions of *covers* and *partitions* of constraints. This is followed by Section 4 which introduces the notions of a the *degree* of a constraint of *linear* constraints and shows how linear constraints can be used to simplify CSPs. Arguments are presented that this simplification corresponds to a "localised" breadth-first search. Special kinds of partitions of constraints are discussed in Section 5. Section 5 also describes the generalised backtracking algorithm. Experimental results are discussed in Section 6. A summary is presented in Section 7.

2 Constraints and CSPs

Let $\cdot \prec \cdot$ be the usual lexicographical variable ordering. In this paper it is assumed that if x_i and x_j are two variables then $x_i \prec x_j \iff i < j$. Let $T = \{ x_{i_1}, \ldots, x_{i_m} \}$ be a non-empty set of variables. For the sake of this paper a *constraint* among the members of T is a subset of the cartesian product of the domains of the members of T. However, this is by no means a necessity. If C_T is a constraint then it will be assumed that the constraint is between the members of T. In addition it will be assumed that if $v \in C_T$ then $(x_{i_1}, \ldots, x_{i_m}) = v$ is a valid "assignment" which is allowed by C_T. A *Constraint Satisfaction Problem* (CSP) is a tuple (X, D, C), where X is a set of variables, $D(\cdot)$ is a function mapping each member of X to its domain, and C is a set of constraints.

3 Covers and Partitions of Constraints

This section presents methods to decompose any CSP into several CSPs whose solutions are pairwise disjoint and the union of whose solutions is equal to the solutions of the original CSP. It is shown that certain kinds of decompositions are essentially the same as the decompositions that are (implicitly) computed by the chronological backtracking algorithm.

Let S be a set and let 2^S denote the *power set* of S, i.e. the set of all subsets of S. A set $\kappa \subseteq 2^S$ is called a *cover* of S if $S = \cup_{c \in \kappa} c$. The set containing all covers of S is denoted $K(S)$, i.e. $K(S) = \left\{ \kappa \subseteq 2^S : S = \cup_{c \in \kappa} c \right\}$. Partitions are covers whose members are pairwise disjoint. The set of all partitions of S is denoted $\Pi(S)$. The *maximal partition* of a set S is the set $\{ \{ s \} : s \in S \}$.

The following proposition tells us that if we factorise a constraint C_T from a given CSP then the solutions of that CSP are equal to the union of the solutions of the CSPs that can be obtained by replacing C_T by the members of its factorisation.

Proposition 1 (Covers of Constraints). *Let* (X, D, C) *be a* CSP, *let* $T \subseteq X$, *let* $C_T \in C$, *let* $C'(c) = (C \setminus \{ C_T \}) \cup \{ c \}$ *and let* $\cdot \bowtie \cdot$ *denote* natural join. *The following holds for all covers* κ *of* C_T*:*

$$\bowtie_{c \in C} c = \bigcup_{c' \in \kappa} \bowtie_{c \in C'(c')} c \,.$$

To prove the proposition is not difficult. A formal proof may be found in [22]. Notice that partitions are covers. Therefore, Proposition 1 also applies to partitions and maximal partitions.

Proposition 1 allows for the decomposition of a CSP into a collection of CSPs. The collection represents the CSP in the sense that the union of the solutions of the members of that collection is equal to the solutions of that CSP. The following example demonstrates how Proposition 1 can be used to explain how the chronological backtracking algorithm works.

Example 2 (Chronological Backtracking). Let $\mathcal{C} = (\, X, D, C\,)$ be the CSP, where $X = \{\, x, y\,\}$, $C = \left\{\, C_{\{\,x\,\}}, C_{\{\,y\,\}}, C_{\{\,x,y\,\}}\,\right\}$, $D(x) = \{\,1, 2\,\}$, $D(y) = \{\,1, 2, 3\,\}$, $C_{\{\,x\,\}} = \{\,1, 2\,\}$, $C_{\{\,y\,\}} = \{\,1, 2, 3\,\}$ and $C_{\{\,x,y\,\}} = \{\,(\,1, 1\,), (\,1, 2\,), (\,2, 3\,)\,\}$. The solution set of $\mathcal{C}$ is $C_{\{\,x,y\,\}}$. To backtrack with x as the current variable corresponds to the application of Proposition 1 to the CSP for the maximal partition $\pi = \left\{\, C'_{\{\,x\,\}}, C''_{\{\,x\,\}}\,\right\}$, where $C'_{\{\,x\,\}} = \{\,1\,\}$ and $C''_{\{\,x\,\}} = \{\,2\,\}$. The application of Proposition 1 to π allows for the decomposition of the constraint $C_{\{\,x\,\}}$ into the two constraints $C'_{\{\,x\,\}}$ and $C''_{\{\,x\,\}}$ that can be used to dissect $\mathcal{C}$ into the two CSPs $\mathcal{C}'$ and $\mathcal{C}''$, where

$$\mathcal{C}' = \left(X, D, \left\{\, C'_{\{\,x\,\}}, C_{\{\,y\,\}}, C_{\{\,x,y\,\}}\,\right\}\right)\,,$$
$$\mathcal{C}'' = \left(X, D, \left\{\, C''_{\{\,x\,\}}, C_{\{\,y\,\}}, C_{\{\,x,y\,\}}\,\right\}\right)\,.$$

The solutions of $\mathcal{C}'$ are given by $C'_{\{\,x\,\}} \bowtie C_{\{\,y\,\}} \bowtie C_{\{\,x,y\,\}} = \{\,(\,1, 1\,), (\,1, 2\,)\,\}$ and the solutions of $\mathcal{C}''$ are given by $C''_{\{\,x\,\}} \bowtie C_{\{\,y\,\}} \bowtie C_{\{\,x,y\,\}} = \{\,(\,2, 3\,)\,\}$. The union of the solutions of $\mathcal{C}'$ and $\mathcal{C}''$ is equal to the solution set of $\mathcal{C}$. If the "standard" lexicographical heuristics are used then a chronological depth-first backtracking algorithm will first solve $\mathcal{C}'$ and then solve $\mathcal{C}''$.

4 Linear Constraints

This section discusses how to use certain properties of constraints to simplify binary CSPs. In particular it is shown that what are called *linear* constraints (many-to-one-relations)[2] can be used to simplify binary CSPs. The simplifications consist of a transformation of a binary CSP to a binary CSP where a variable has been eliminated (modulo renaming). The sizes of the domains in the resulting CSP are less than or equal to the sizes of the domains in the original CSP. The number of binary constraints in the resulting CSP is less than the number of binary constraints in the original CSP. It is argued that such transformation can be regarded as a "localised" breadth-first search.

[2] A suitable name for linear constraints could also have been *functional* constraints had it not been for the fact that there is a "name clash" for such constraints. For example, [23] and [4] both use functional constraints with a different meaning. Functional constraints in the context of [23] are what will be called *bi-linear* constraints here further on. They correspond to one-to-one relations. Functional constraints in the context of [4] correspond to the notion of what will be called linear binary constraints in this work.

Constraints that have finitely many members can be translated to polynomial ideals (systems of polynomial equations) and vice versa [22, Chapter 4]. This suggests that constraints also have “degrees” and “total degrees.” The following is an attempt to generalise these notions of degree and total degree to that of the degree of a set of variables in a constraint. It is a reformulation of the notion presented in [22, Chapter 5]. Notions similar to that of the degree of a constraint do not seem to have appeared before. The number of substitutions of values for a variable in a polynomial for which the polynomial vanishes (“becomes” zero) corresponds to a branching factor of a constraint in a search tree. As will be shown later, the degree of a set of variables in a constraint relates these variables to the branching factor. Constraints with low degrees correspond to low branching factors in search.

```
function deg($C_T, S$) : Integer
var $R$, $Partitions$, $Projections$;
begin
    if $C_T = \emptyset$ then
        return 0;
    else if $|S| = 1$ then
        $R := T \setminus S$;
        $Projections := \{$ projection of $t$ onto $R \,:\, t \in C_T \}$;
        return $\max_{p \in Projections} |\{ t \in C_T \,:\, p =$ projection of $t$ onto $R \}|$;
    else
        $Partitions := \{ \pi \in \Pi(C_T) \,:\, (\forall c \in \pi)(\exists x \in S)(\deg(c, \{ x \}) = 1) \}$;
        return $\min(\{ |\pi| \,:\, \pi \in Partitions \})$;
    fi;
end;
```

Fig. 1. Degree function.

Definition 3 (Degree). Let S and T be non-empty sets of variables such that $S \subseteq T$. Furthermore, let C_T be a constraint whose cardinality is finite, and let $\deg(\cdot, \cdot)$ be the function depicted in Figure 1. The number $\deg(C_T, S)$ is called the *degree* of S in C_T. The degree of $\{ x \}$ in C_T is also called the *degree* of x in C_T or the *x-degree* of C_T.

If $|S| = 1$ then the degree of S in C_T is the maximum number of solutions for the variable in S that are “allowed” by C_T given fixed assignments to the variables in $T \setminus S$. For binary constraints the degree of x in $C_{\{ x,y \}}$ is the maximum support size in $D(x)$ for y. A constraint C_T is called *linear* in $x \in T$ if the degree of x in C_T is one. A binary constraint $C_{\{ x,y \}}$ is called *bi-linear* if it is linear in both x and y. A constraint C_T is called *sub-linear* (in $S \subseteq T$) if $C_T = \emptyset$. Finally, a constraint C_T is called *linear* if it is linear in some variable in T.

Example 4 (Degree). Consider the binary constraint $C_{\{ x,y \}}$ given by:

$$C_{\{ x,y \}} = \{ (0,0), (0,1), (0,2), (0,3), (1,0), (2,0) \} .$$

The constraint $C_{\{x,y\}}$ is not linear in x. For example, the support for $y = 0$—the tuples whose projection onto y is equal to 0—is $\{0, 1, 2\}$. Therefore, the degree of x in $C_{\{x,y\}}$ is at least three. Similarly, $C_{\{x,y\}}$ is not linear in y either. It is left to the reader to verify that the degree of x in $C_{\{x,y\}}$ is three and that the degree of y in $C_{\{x,y\}}$ is four. Note that $d_{xy} = \deg(C_{\{x,y\}}, \{x, y\}) > 1$ because $C_{\{x,y\}}$ is non-empty and is neither linear in x nor linear in y. However, it can be shown that $d_{xy} = 2$. For example, consider the following two constraints:

$$\begin{aligned} C'_{\{x,y\}} &= \{(0,1), (0,2), (0,3)\} \ , \\ C''_{\{x,y\}} &= \{(0,0), (1,0), (2,0)\} \ . \end{aligned}$$

Both $C'_{\{x,y\}}$ and $C''_{\{x,y\}}$ are linear. The former is linear in x and the latter is linear in y. The set $\pi = \left\{ C'_{\{x,y\}}, C''_{\{x,y\}} \right\}$ is a partition of $C_{\{x,y\}}$. It follows from the definition of the degree of $\{x, y\}$ in $C_{\{x,y\}}$ that $d_{xy} \leq |\pi| = 2$. As observed before $C_{\{x,y\}}$ is not linear in x or in y. Therefore, $d_{xy} > 1$. Clearly, $d_{xy} = 2$.

Linear constraints can be used to simplify binary CSPs. If an arc-consistent constraint $C_{\{x,y\}}$ is linear then the variables x and y can be *amalgamated* into a "super-variable" which represents the values in the Cartesian product of the domains of x and y that are in $C_{\{x,y\}}$. The cardinality of the domain of the super-variable is equal to $\max(|D(x)|, |D(y)|)$ and the number of constraints in the resulting CSP will be less than the number of constraints of the original CSP. The transformation will leave all constraints of the form $C_{\{w\}}$ or $C_{\{w,z\}}$ intact, for w and $z \notin \{x, y\}$.

Thus, linear binary constraints allow for the elimination of a variable without causing an increase in the domain sizes of the variables or the number of constraints. The remainder of this section provides concrete examples about the flavour of linear constraints and how to exploit their properties.

Example 5 (Singleton Domains). During backtrack search it often occurs that the domain of a future variable reduces to a singleton set. Let x be such variable.

If the problem is binary and if the problem is arc-consistent then x can be removed. Should there be solutions then the projections of these solutions onto the domain of x will be the value in its domain.

If C_T is a constraint which involves x and if the domain of x is a singleton then C_T is linear or sub-linear in x. C_T is linear or sub-linear in x because any assignment to the variables in $T \setminus \{x\}$ which satisfies the projection of C_T onto $T \setminus \{x\}$ can be extended to at most one assignment to the variables in T such that this extended assignment satisfies C_T. If the projection of C_T onto the domain of x is non-empty then the constraint C_T can be contracted to a constraint on $T \setminus \{x\}$ without "losing" any solutions; the solutions for x can always be recovered.

The *reason* why the solutions can be recovered is that T is linear in x. Therefore, there is a function from the variables in $T \setminus \{x\}$ to x which can be used to "recover" x. If C_T is binary, then the contraction of C_T entails the creation of a unary constraint on the remaining variable, say y, in $T \setminus \{ x \}$. If the problem is arc-consistent then the contraction of C_T is the same as $D(y)$ and it can be ignored. In binary CSPs that are arc-consistent, variables whose domains are singletons can therefore be eliminated.

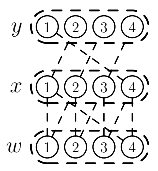


Fig. 2. Before Amalgamation.

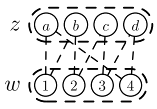


Fig. 3. After Amalgamation.

In the previous example it was argued that a variable x whose domain is a singleton can be removed from binary arc-consistent CSPs because there is a function from the variables in $T \setminus \{ x \}$ to x and that this mapping could be used to recover the value of x. This is *exactly* the same reason as the one upon which MAC (a backtracker which maintains arc-consistency [20]) relies, namely that after the assignment of a value to the current variable and after arc-consistency processing the current variable can be removed from a problem if it is arc-consistent because there is a function from the future variables to the value in the domain of the current variable. Some people may argue that x can be removed because its *only* value has been "saved" and can therefore be recovered. Other people may argue that after the assignment to x any arc-consistent constraint between x and another variable has "become" universal and can therefore be removed. However, the concept of x being "dependent on" a function is more general because, as the following example will demonstrate, it allows for the simultaneous recovery of *several different* values as opposed to only one.

Example 6 (Amalgamation of Nodes (1)). Consider the binary CSP whose micro-structure is depicted in Figure 2. The dashed lines represent the tuples that are allowed. The constraint $C_{\{ w,x \}}$ is not linear. The remaining constraint $C_{\{ x,y \}}$ is bi-linear.

Consider the sub-problem consisting of the two variables x and y, their domains, and the constraint $C_{\{ x,y \}}$. As it turns out the sub-problem has exactly four solutions. The nodes x and y can be transformed into a new node z whose domain contains four values a, b, c and d without increasing the maximum domain size. The transformation is such that these four values represent the four solutions of the sub-problem.

Figure 3 depicts the micro-structure of the same CSP where x and y have been "amalgamated" into one fresh variable z. The value a in the domain of z represents the tuple $(x, y) = (1, 2)$, b corresponds to $(x, y) = (2, 3)$, c corresponds to $(x, y) = (3, 4)$, and d corresponds to $(x, y) = (4, 1)$. The problem is satisfiable if and only if the original problem is satisfiable, and its solutions are in one-to-one correspondence with the solutions of its original problem. The structure of the new problem is simpler than that of the original problem. Transformations, like the one from the CSP whose micro-structure is depicted in Figure 2 to the CSP whose micro-structure is depicted in Figure 3 may also be regarded as the elimination of a variable which linearly depends on a linear constraint (modulo renaming).

For binary CSPs the worst-case time-complexity for the detection of *all* linear constraints is in $\mathcal{O}(ed^2)$. This is exactly the worst case time-complexity for making a CSP arc-consistent, an "overhead" which is considered to be well spent by researchers in the constraint satisfaction area. It is not difficult to see how to incorporate part of the work for the detection of linear constraints into existing arc-consistency algorithms. If the domain sizes are large then most binary constraints are not linear and this can be found out without much overhead. The reason why this does not require much overhead is that it is not difficult (on average) to detect that there are at least two tuples in a binary constraint whose first members are equal and to find two tuples in a binary constraint whose second members are equal. However, when domain sizes become small a relatively large proportion of all the possible binary constraints are linear. To detect that a constraint is linear is relatively easy if the sizes of the domains are small.

The following example demonstrates that, in the presence of constraint propagation, to amalgamate the variables of a linear binary constraint does not only allow for the elimination of variables but may sometimes allow for the elimination of values.

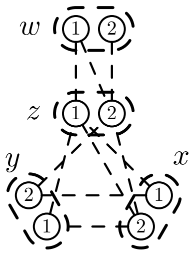


Fig. 4. Before Amalgamation of x and y.

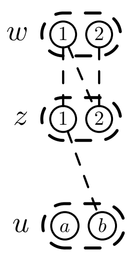


Fig. 5. After Amalgamation of x and y.

Example 7 (Amalgamation of Nodes (2)). Consider the CSP whose micro-structure is depicted in Figure 4. The CSP consists of four variables w, x, y and z, their domains, and four binary constraints $C_{\{w,z\}}$, $C_{\{x,z\}}$, $C_{\{x,y\}}$, and $C_{\{y,z\}}$. The CSP is arc-consistent. The binary constraints $C_{\{w,z\}}$, $C_{\{x,z\}}$, $C_{\{x,y\}}$, and $C_{\{y,z\}}$ which were mentioned before are explicit. Besides these explicit constraints there are also implicit constraints. These constraints are determined by projections, (natural) joins, and intersections of constraints. For example, there is an implicit constraint $C_{\{\,x,y,z\,\}}$ between x, y and z. $C_{\{\,x,y,z\,\}} = \{\,(\,2,1,1\,)\,\}$. The constraint between x, y, and z may also be considered as a constraint between x and y on the one hand and z on the other. This constraint is given by $C_{\{\,(\,x,y\,),z\,\}} = \{\,(\,(\,2,1\,)\,,1\,)\,\}$. The members of $C_{\{\,(\,x,y\,),z\,\}}$ are in one-to-one correspondence to the solutions of the sub-problem involving the variables x, y, and z, their domains, and the constraints $C_{\{\,x,y\,\}}$, $C_{\{\,x,z\,\}}$, and $C_{\{\,y,z\,\}}$. $C_{\{\,x,y\,\}}$ is linear. Therefore, the nodes x and y can be amalgamated into a node u whose domain contains one value for each of the tuples that are "allowed" by $C_{\{\,x,y\,\}}$ without increasing the size of the domains. Renaming $(\,x,y\,)$ to u, $(\,1,2\,)$ to a, and $(\,2,1\,)$ to b results in a CSP the solutions of which are in one-to-one correspondence with the solutions of the original CSP. In particular, $(\,w,z,u\,) = (\,1,1,b\,) \iff (\,w,x,y,z\,) = (\,1,2,1,1\,)$. The micro-structure of the resulting CSP is depicted in Figure 5. Note that the CSP is not arc-consistent. The values 2 in the domain of z and a in the domain of u have lost support as a "result" of the constraint $C_{\{\,(\,x,y\,),z\,\}}$ which was implicit between x, y, and z. It is straightforward to make the CSP arc-consistent again.

To conclude this section it should be observed that the amalgamation of two variables x and y may be regarded as a "localised" breadth-first search of depth two. To see why this is true observe that the domain of the amalgamation of two variables contains the representatives of the values in the constraint between the variables. This set is equal to the set containing the allowed assignments of the nodes of the search tree at depth two which uses an ordering where x and y are the first variables. The advantage of amalgamation is that no decision has to be made yet about which value to assign to which variable, that it simplifies the problem, and that it allows for cheap constraint propagation. The advantage of not making a decision about which variable is to become the next current variable is that to postpone this decision may avoid assignments leading to traversals of sub-trees that are infeasibly large.

5 Linear Partitions and Generalised Backtracking

The previous sections have demonstrated the usefulness of partitions of constraints and linear constraints. This section studies a special kind of partition called *linear* partitions and a function to compute such partitions. The function is non-trivial in the sense that the cardinality of its result is "low." We shall see that linear partitions and the transformation to amalgamate nodes can be used to enumerate the nodes in the search tree of constraints more efficiently than chronological backtracking.

In the following let $\mathrm{proj}_{x_{i_j}}(\cdot)$ be the *projection function* defined as follows:

$$\mathrm{proj}_{x_{i_j}}((\,v_{i_1},\ldots,v_{i_m}\,)) = \begin{cases} v_{i_j} & \text{if } 1 \le j \le m; \\ \bot & \text{otherwise.} \end{cases}$$

Definition 8 (Layer). Let S be a non-empty set of variables, let $x \in S$, and let C_S be a non-empty constraint. Furthermore, let $C_{\{\,x\,\}} = \{\,\mathrm{proj}_x(t) \,:\, t \in C_S\,\}$. A set $S_x \subseteq C_S$ is called an *x-layer* of C_S if $|S_x| = |C_{\{x\}}|$ and $\{\, \mathrm{proj}_x(t) \,:\, t \in S_x \,\} = C_{\{x\}}$.

Example 9 (Layer). Let $C_{\{x,y\}} = \{\,(0,0),(0,1),(1,2)\,\}$. There are two x-layers of $C_{\{x,y\}}$. They are given by $\{\,(0,0),(1,2)\,\}$ and by $\{\,(0,1),(1,2)\,\}$. The only y-layer of $C_{\{x,y\}}$ is given by $C_{\{x,y\}}$ itself.

Lemma 10 (Linearity of Layers of Binary Constraints). *Let $C_{\{x,y\}}$ be a non-empty binary constraint and let $(z,z') \in \{\,(x,y),(y,x)\,\}$. If S_z is a z-layer of $C_{\{x,y\}}$ then S_z is linear in z'.*

Proof. Let $S_z = \{\,(x_1,y_1),\ldots,(x_m,y_m)\,\}$. Without loss of generality assume that $z = x$. Then $x_i \neq x_j \Leftrightarrow i \neq j$, for $1 \leq i,j \leq m$, and $x_i \mapsto y_i$ defines a *function* from $z = x$ to $y = z'$.

Lemma 11 (Monotonicity). *Let $C_{\{x,y\}}$ be a binary constraint, let $z \in \{\,x,y\,\}$, and let S_z be a z-layer of $C_{\{x,y\}}$. Furthermore, let $d_w = \deg(C_{\{x,y\}},\{\,w\,\})$, and let $d'_w = \deg(C_{\{x,y\}} \setminus S_z, w)$, for $w \in \{\,x,y\,\}$. Then $\min(d'_x,d'_y) < \min(d_x,d_y)$.*

Proof. Trivial.

```
function $P_l(C_i)$ :
begin var $B_i$, $C_{i+1}$, $R_i$, $z$;
    if $C_i = \emptyset$ then
        return $\emptyset$;
    else
        if $\deg(C_i,\{x\}) > \deg(C_i,\{y\})$ then
            $z := x$;
        else if $\deg(C_i,\{x\}) < \deg(C_i,\{y\})$ then
            $z := y$;
        else
            $z :=$ any member from $\{x,y\}$;
        fi;
        $B_i :=$ any $z$-layer of $C_i$;
        $C_{i+1} := C_i \setminus B_i$;
        $R_i := \{B_i\} \cup P_l(C_{i+1})$;
        return $R_i$;
    fi;
end;
```

Fig. 6. Partition function

A *linear partition* is a partition whose members are linear. The following defines a function to transform a binary constraint to a linear partition of that constraint.

Proposition 12 (Linear Partition of Binary Constraint). *Let $C_{\{x,y\}}$ be a non-empty finite constraint. Furthermore, let $d_z = \deg(C_{\{x,y\}},\{\,z\,\})$, for $z \in \{\,x,y\,\}$. Finally, let $P_l(\cdot)$ be the function defined in Figure 6, then $P_l(C_{\{x,y\}})$ is a linear partition of $C_{\{x,y\}}$. Furthermore, $|P_l(C_{\{x,y\}})| \leq \min(d_x,d_y)$.*

Proof. Let $C_1 = C_{\{x,y\}}$. To prove that the proposition is correct it has to be demonstrated that $P_l(C_1)$ terminates, that $P_l(C_1)$ is a partition of C_1, that the members of $P_l(C_1)$ are linear, and that the cardinality of $P_l(C_1)$ does not exceed $\min(d_x, d_y)$.

termination Assume that $P_l(C_1)$ does not terminate. By assumption $|C_1|$ is finite. It follows from the non-termination of $P_l(C_1)$ and its termination criterion that $C_i \supset \emptyset$ for $i \in \mathbb{N} \setminus \{0\}$. This together with the definition of B_i allows us to infer that $\emptyset \subset B_i \subseteq C_i$ must hold. Therefore, $C_{i+1} = C_i \setminus B_i \subset C_i$ and it follows that the sequence

$$C_1 \supset C_2 \supset C_3 \supset \cdots$$

is infinite. This contradicts the premise that $|C_1|$ is finite.

partition property Let $C_1 = C_{\{x,y\}}$ and let $d = |P_l(C_1)|$. To prove that $P_l(C_1)$ is a partition of C_1 we must prove that $C_1 = \cup_{c \in P_l(C_1)} c$ and that $(\forall C_S, C_T \in P_l(C_1))(C_S \neq C_T \iff \emptyset = C_S \cap C_T)$.
First note that $P_l(C_i) = \cup_{i=1}^{d} \{B_i\}$. Next note that $C_{i+1} \cup B_i = C_i$, for $i = d, d-1, \ldots, 1$. Clearly, $P_l(C_1)$ is a cover of C_1. To see why the members of $P_l(C_1)$ are pairwise disjoint, observe that $B_j \subseteq C_{i+1} = C_i \setminus B_i$, for $1 \leq i < j \leq d$.

linearity property By Lemma 10, B_i is linear, for $1 \leq i \leq d$.

cardinality property Use Lemma 11 and induction on $\min(d_x, d_y)$.

Definition 13 (Generalised Branching Factor). The *generalised branching factor* of a linear partition of a constraint is given by the cardinality of that partition.

Note that maximal partitions of unary constraints are linear. Therefore, the notions of generalised branching factor and that of the branching factor coincide for maximal partitions of unary constraints.

The application of linear partitions will become apparent in the next example. Before we go on to that example, it should be pointed out that the minimum of the degrees of the variables that are involved in an arc-consistent binary constraint, cannot exceed the minimum of their domain sizes. This is formulated as the following proposition.

Proposition 14. *Let $C_{\{x,y\}}$ be a non-empty constraint, let $d_z = \deg(C_{\{x,y\}}, \{z\})$, and let $D(z) = \left\{ \operatorname{proj}_z(t) : t \in C_{\{x,y\}} \right\}$, for $z \in \{x, y\}$, then*

$$\min(d_x, d_y) \leq \min(|D(x)|, |D(y)|)\,.$$

Proof. $|D(y)| \geq d_x$ and $|D(x)| \geq d_y$.

In the previous section we saw that backtracking uses linear partitions of unary constraints to enumerate the members of the domain of the current variable. We have also seen that linear binary constraints can be used to amalgamate two variables. We have argued that this may be viewed as variable elimination (modulo renaming). By combining linear constraints and amalgamation, we can obtain lower (generalised) branching factors.

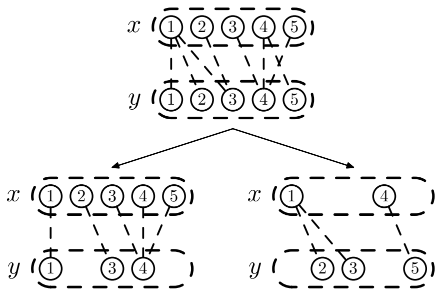


Fig. 7. Linear partition.

Example 15 (Generalised Backtracking). Consider the constraint $C_{\{x,y\}}$ whose micro-structure is depicted at the top of Figure 7. The constraint is cubic in x and in y. The two constraints whose micro-structures are depicted at the bottom of Figure 7 form the partition $\pi = \{\mathcal{C}_1, \mathcal{C}_2\}$ of $C_{\{x,y\}}$, where

$$\mathcal{C}_1 = \{(1,1),(2,3),(3,4),(4,4),(5,4)\} ,$$
$$\mathcal{C}_2 = \{(1,2),(1,3),(4,5)\} .$$

$\mathcal{C}_1$ is linear in y, whereas $\mathcal{C}_2$ is linear in x. The partition was computed using $P_l(\cdot)$ by always selecting the lexicographically smallest z-layer to compute the sets B_i.

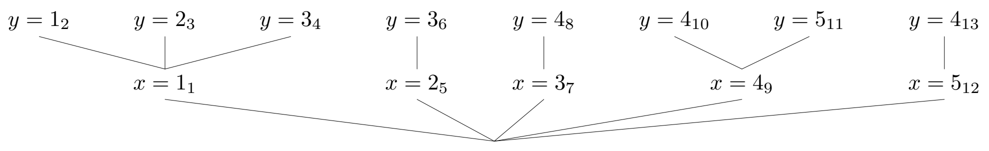


Fig. 8. In-order search tree for $x \prec y$. Branching factor is 5. #Visited nodes is 13.

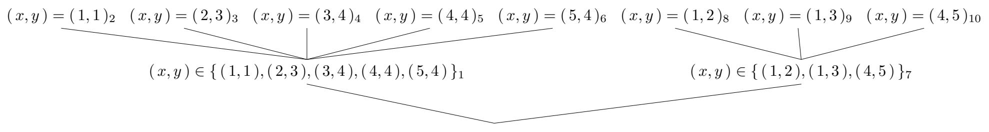


Fig. 9. Generalised search tree. Generalised branching factor is 2. #Visited nodes is 10.

Note that the generalised branching factor of π (the cardinality of π) is 2. This is strictly less than the x-degree and the y-degree of the original constraint. This demonstrates that the inequality in Proposition 12 may be strict.

The in-order search tree for the chronological backtracking algorithm for the variable order $x \prec y$ is depicted in Figure 8. The subscripts of the nodes and leaves of the trees represent the visiting order. The tree has 13 visited nodes and a branching factor of 5 at the root. The number of leaves of the tree is equal to the cardinality of the constraint $C_{\{x,y\}}$. The search tree for $y \prec x$ has similar properties at that for $x \prec y$.

The *generalised search tree* corresponding to the partition π is depicted in Figure 9. The nodes and leaves of this tree are visited by a *generalised backtracking algorithm.* At the root of the tree there are two branches—one for each member of π. Each of the members of π is linear. As argued before, linear binary partitions of arc-consistent constraints correspond to the domain of a variable. As also argued before, a linear arc-consistent binary constraint can be used to eliminate a variable (modulo renaming) by amalgamating the variables that are involved.

1. The number of leaves of the generalised search tree is equal to the number of leaves of the in-order search trees. There is one leaf for each member of the constraint.
2. The linearity of π ensures that the maximum domain size does not increase.
3. The generalised branching factor at the root does not exceed the minimum domain size (the branching factors of the in-order search trees) of x and y and is usually smaller.
4. The linearity of π ensures that the height of the generalised search tree does not exceed the number of variables.
5. The number of visited nodes of the generalised backtracking tree is less than the number of nodes of each of the chronological backtracking trees. This is a consequence of 1, 2, and 3, and the fact that (for this example) the generalised branching factor is strictly less than the minimum domain size.

The most important result of the generalised backtracking approach is that it has decreased the generalised branching factor. Generalised backtracking works because the degrees of constraints determine the generalised branching factor. If the degree of a variable x in a binary constraint is d_x then x can be eliminated (modulo renaming) at the cost of a branching factor of d_x or less. We already observed that d_x never exceeds the cardinality of the domain of x and may be less than it. If d_x is less than the domain size of x then a smaller branching factor can be achieved than with the traditional backtracking approach. Since the domain sizes do not increase and since the height of the search tree remains the same this results in strictly fewer visited nodes.

Figures 7–9 suggest that every leaf in the tree (read the representatives of the members of $C_{\{x,y\}}$) can be visited. This is not true in general. There are at least two reasons. The first reason is that in general there may be more variables in a problem and the variables which will be the current variable at depth two from the root of the tree may be different. The second reason is that branches may become dead-ends as a result of the use of constraint propagation techniques.

6 Experimental Results

This section briefly discusses results from the application of a toy implementation in `Haskell` [12] of the generalised backtracking algorithm and discusses possible improvements on the implementation. It is still an item on the to-do-list to implement an efficient version of the algorithm.

The algorithm repeatedly selects a binary constraint. The constraint is always such that the domains of the variables that are involved are as small as possible. If the constraint is linear, it is used to eliminate a variable. Otherwise, if the constraint is universal, the constraint is removed. Otherwise the algorithm computes a linear partition of the constraint and uses it for branching. In addition, the algorithm maintains arc-consistency during search.

For RLFAP 3 and RLFAP 4 some large (but relatively easy) problems [2], it was observed that for deciding the satisfiability of these problems the generalised branching factor was almost always significantly smaller than the smallest domain size of any of the variables in the problem. It is recalled that the smallest domain size is a lower bound on the branching factor for chronological backtracking. For RLFAP 3, for example, the smallest domain size would almost always be between 20 and 40 and the ratio between the generalised branching factor and the smallest domain size would be between 0.6 and 0.9. Both problems could be solved without backtracking.

Constraints were implemented as black boxes (function calls) because an extensional representation would have been impossible due to the size of the input CSPs. The black box representation of constraints resulted in a lot of overhead for the computation of partitions. As mentioned before, it is our intention to look at more efficient implementation techniques.

The fact that the problems could be solved is not world shocking news; Any backtracker that maintains arc-consistency can do this almost effortlessly. However, the fact that the generalised branching factor b_g significantly improved upon the ordinary branching factor b_o may be significant. For example, a good heuristic for selecting a good constraint for partitioning need not be expensive. To partition a constraint $C_{\{x,y\}}$ is in $\mathcal{O}(d_x d_y)$, where d_x and d_y are the sizes of the domains of x and y. Furthermore, it seems that the variable elimination step which is carried out as part of the algorithm can be incorporated in the maintenance of arc-consistency without too much overhead (however, this still remains to be proved). Therefore, it seems that the “overhead” of partitioning is equivalent to about 1 times the work to make the remaining problem arc-consistent. Most of the time in search is spent on deciding that there are no solutions, i.e. *all* branches have to be searched and this requires at least b times the work to make the remaining problem arc-consistent, where b is the number of branches. If this “analysis” is correct then the savings of the generalised backtracking algorithm at the current level in the search tree are given by $b_o + 1 - b_g$ times the work of making the remaining problem arc-consistent.

In summary, much work remains to be done both in terms of implementation and experimental evaluation. However, the results from a few tests seem promising.

7 Summary

In this paper, the new notion of the degree of a constraint has been presented. This notion allowed us to reason about branching factors in search and allowed us to generalise the branching strategy of chronological backtracking.

The notion of the degree of a set of variables in a constraint led to the notion of a linear constraint and it has been shown how linear constraints can be used for the simplification of CSPs by amalgamating the variables involved in a linear binary constraint. This amalgamation operation corresponds to a variable elimination (modulo renaming). Arguments have been presented that for binary CSPs the average costs for the detection of linear constraints is low if arc-consistency is maintained.

It has been shown that the essence of chronological backtracking is that it uses linear partitions of unary constraints (the domains of the variables) to decompose a CSP into a set of CSPs whose solutions are disjoint and the union of whose solutions is equal to the solutions of the original CSP. The cardinality of a linear partition is called the generalised branching factor of that partition. The minimal generalised branching factor for a chronological backtracking algorithm which maintains arc-consistency is equal to the minimum domain size.

A generalisation of the chronological backtracking algorithm has been presented. This algorithm, called generalised backtracking, is not restricted to the use of linear partitions of unary constraints to decompose CSPs but can use any kind of constraint for this purpose. A function $P_l(\cdot)$ has been presented to compute linear partitions of binary constraints. The cardinalities of these partitions are small. If $C_{\{x,y\}}$ is a constraint such that the size of the domain of x or y is equal to the minimum domain size then the generalised branching factor of $P_l(C_{\{x,y\}})$ is never larger then the minimum domain size but may be smaller.

A few results have been presented of applications of a toy implementation of the generalised backtracking algorithm. The results are promising in the sense that they demonstrated that significant reductions of the generalised branching factor can be obtained. With proper adjustments, it may be possible that the algorithm becomes an improvement on the standard backtracking algorithm in the sense that it will also save consistency-checks. Suggestions have been presented on how to properly implement the algorithm. However, future research has to demonstrate whether generalised backtracking can be implemented efficiently and proper experiments have to be set up to compare chronological and generalised backtracking. Of course, these experiments should be complemented by a theoretical investigation.

References

1. W. Boege, R. Gebauer, and H. Kredel. Some examples for solving systems of algebraic equations by calculating Groebner bases. *Journal of Symbolic Computation*, 2(1):83–98, 1986.
2. B. Cabon, S. De Givry, L. Lobjois, T. Schiex, and J.P. Warners. Radio link frequency assignment. *Journal of Constraints*, 4:79–89, 1999.

3. S.R. Czapor. Solving algebraic equations: Combining Buchberger's algorithm with multivariate factorization. *Journal of Symbolic Computation*, 7(1):49–54, 1989.
4. P. David. Using pivot consistency to decompose and solve functional CSPs. *Journal of Artificial Intelligence Research*, 2:447–474, May 1995. AI Access Foundation and Morgan Kaufmann Publishers.
5. R. Dechter and D. Frost. Backtracking algorithms for constraint satisfaction problems. Technical report, University of California, Irvine, 1999.
6. J. Gaschnig. Experimental case studies of backtrack vs. Waltz-type vs. new algorithms for satisficing assignment problems. In *Proceeding of the* 2^{nd} *Biennial Conference, Canadian Society for the Computational Studies of Intelligence*, pages 268–277, 1978.
7. M.L. Ginsberg. *Essentials of Artificial Intelligence.* Morgan Kaufmann Publishers, San Mateo, California, 1993.
8. M.L. Ginsberg and D.A. McAllester. GSAT and dynamic backtracking. In A. Borning, editor, *Principles and Practice of Constraint Programming*, number 874 in Lecture Notes in Computer Science, pages 243–265. Springer-Verlag, Berlin/Heidelberg, 1994.
9. S.W. Golomb and L.D. Baumert. Backtrack programming. *Journal of the ACM*, 12(4):516–524, 1965.
10. H.-G. Gräbe. On factorized Gröbner bases. Technical Report 6, Institut für Informatik, Universität Leipzig, Germany, 1994.
11. R.M. Haralick and G.L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313, 1980.
12. P. Hudak, J. Peterson, and J.H. Fasel. A gentle introduction to haskell, version 1.4, 1997.
13. G. Kondrak and P. van Beek. A theoretical evaluation of selected backtracking algorithms. In C.S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 541–547, 1995.
14. G. Kondrak and P. van Beek. A theoretical evaluation of selected backtracking methods. *Artificial Intelligence*, 89:365–387, 1997.
15. H. Melenk. Solving polynomial equation systems by Groebner methods. CWI Quarterly 3, 1990.
16. H. Melenk. Algebraic soltion of nonlinear equation systems in REDUCE. Technical report, Conrad-Zuse-Zentrum für Informationstechnik, Berlin, Germany, 1993.
17. B.E. Nadel. Constraint satisfaction algorithms. *Computational Intelligence*, 5(4):188–224, 1989.
18. M. Pesch. Factorizing Gröbner bases, 1996.
19. P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9:268–299, 1993.
20. D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In A.G. Cohn, editor, *Proceedings of the* 11^{th} *European Conference on Artificial Intelligence*, pages 125–129. John Wiley and Sons, 1994.
21. E. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.
22. M.R.C. van Dongen. *Constraints, Varieties, and Algorithms.* PhD thesis, Department of Computer Science, University College Cork, Ireland, 2002.
23. Pascal Van Hentenryck, Yves Deville, and Choh-Man Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57(2–3):291–321, 1992.