Propagation Redundancy in Redundant Modelling *

C.W. Choi[1], J.H.M. Lee[1], and P. J. Stuckey[2]

[1] Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong SAR, China
{cwchoi,jlee}@cse.cuhk.edu.hk
[2] Deptartment of Computer Science & Software Engineering
University of Melbourne, 3010, Australia
pjs@cs.mu.oz.au

Abstract. Combining mutually redundant models with channelling constraints increases constraint propagation. However, the extra computation efforts of the additional variables and constraints may outweigh the gain of reduction in search space. In fact, many of the constraints in redundant modelling are not only logically redundant but also propagation redundant and hence cannot further reduce search space. We give general theorems for proving propagation redundancy of one constraint with respect to channelling constraints and constraints in the other model. We define a broad form of channelling constraints that are covered by our approach. We illustrate, using problems from CSPLib[3], how detecting and removing propagation redundant constraints can significantly speed up solving behaviour.

1 Introduction

Finding a good model of a constraint satisfaction problem (CSP) is a challenging task. A modeller must specify a set of constraints that capture the definitions of the problem, and the model should also have strong propagation. In other words, the model should be able to quickly reduce the domains of the variables of the problem, *and* the implementation of these propagators should be efficient, *and* the search space should not be too large.

A common technique to increase the propagation is to add redundant constraints, which are logically implied by the constraints of the model. Adding redundant constraints can be beneficial since the constraint solver may extract more information from these redundant constraints. However, not all logically redundant constraints will contribute additional propagation information to the constraint solver. Understanding whether the propagator for a redundant constraint will add useful propagation information is an interesting question. In this paper we show how to determine if a propagator

* We thank the anonymous referees for their constructive comments. The work described in this paper was substantially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region (Project no. CUHK4204/01E).

[3] Available at http://www.csplib.org/.

is *propagation redundant* with respect to a set of propagators, and hence will not add useful propagation information.

An important source of logically redundant constraints arises in redundant modelling [4]. A problem can be modelled differently from two viewpoints using two different sets of variables. By connecting the two different models with channelling constraints, which relates valuations in the two different models, stronger propagation behaviour can be observed. However, the additional variables and constraints impose extra computation overhead. Since each model is complete and only admits the solutions of the problem, each model is logically redundant with respect to the other model plus the channelling constraints. In many cases, some of the constraints are also propagation redundant with respect to the other constraints in the combined model.

Smith [7, 8] has examined the redundant models for a number of individual problems including n-Queens problem, Langford's problem and the social golfers problems. She empirically determined propagation redundancy for constraints in these problems. In this paper we give a theoretical framework which can determine propagation redundancy *apriori*.

Walsh [9] also compares the pruning behaviour of the different notions of consistency over the disequations, channelling constraints, and all-different constraints for permutation problems by introducing the measure of constraint tightness. Propagation redundancy can be seen as a more specific form of constraint tightness, that allows us to give generic approaches to proving better pruning behaviour, in particular about the other constraints in permutation problems.

In order to keep the benefits of redundant modelling without paying all the costs, we give theorems that allow us to show which constraints in a redundant model are not giving extra propagation and can be removed. In order to prove propagation redundancy, we introduce the notion of *propagation rules* which capture each possible propagation by a constraint and *channel functions* which relate these actions from one model to the other. Due to space limitations, we state the lemmas and theorems without proofs.[4] We give experimental results showing the benefits of detecting and removing propagation redundant constraints. Our earlier work [5] limits to the study of propagation redundancy in combining redundant models with permutation channels. This paper extends the study to cover other forms of channelling constraints.

2 Propagation Based Constraint Solving

In this paper we consider integer and set constraint solving with constraint propagation and tree search. The Boolean constraint solving is considered a special case of the integer constraint.

Constraints We consider a typed set of variables $\mathcal{V} = \mathcal{V}_I \cup \mathcal{V}_S$ made up of *integer* variables $\mathcal{V}_I$, for which we use lower case notation such as x and y, and *sets of integers*

[4] A longer version of this paper with the proofs of lemmas and theorems is available at `http://www.cse.cuhk.edu.hk/~cwchoi/cp03long.pdf`.

variables $\mathcal{V}_S$, for which we use upper case notation such as S and T. We use v to denote variables of either kind.

An *valuation* θ is a mapping of integer variables to integer values and set variables to sets of integer values, written $\{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n, S_1 \mapsto A_1, \ldots, S_m \mapsto A_m\}$. We extend the valuation θ to map expressions or constraints involving the variables in the natural way. Let *vars* be the function that returns the set of variables appearing in an expression, constraint or valuation.

A *primitive constraint* c defines a set of valuations $solns(c)$ each mapping the same set of variables $vars(c)$. We call $solns(c)$ the *solutions* of c. For a primitive constraint c defined by arithmetic expressions we define $solns(c) = \{\theta \mid vars(\theta) = vars(c) \wedge \models_\theta c\}$, that is θ makes the constraint c hold. Primitive constraints can also be defined directly, by giving the set (or table) $solns(c)$.

A *constraint* is a conjunction of primitive constraints, by abuse of notation, we will sometimes treat it as a set of primitive constraints. A constraint c is *logically redundant* with respect to a set of constraints C if $\models C \rightarrow c$.

Domains A *domain* D is a complete mapping from a fixed (countable) set of variables $\mathcal{V}$ to finite sets of integers (for the integer variables in $\mathcal{V}_I$) and to finite sets of finite sets of integers (for the set variables in $\mathcal{V}_S$). A *false domain* D is a domain with $D(v) = \emptyset$ for some v. The *intersection* of two domains D_1 and D_2, denoted $D_1 \sqcap D_2$, is defined by the domain $D_3(v) = D_1(v) \cap D_2(v)$ for all v. A domain D_1 is *stronger* than a domain D_2, written $D_1 \sqsubseteq D_2$, if D_1 is a false domain or $D_1(v) \subseteq D_2(v)$ for all variables v. A domain D_1 is equal to a domain D_2, denoted $D_1 = D_2$, if D_1 and D_2 are both false domains or $D_1(v) = D_2(v)$ for all variables v. We can understand a domain D as a constraint in the obvious way, $D \quad \leftrightarrow \quad \bigwedge_{v \in \mathcal{V}} \bigvee_{d \in D(v)} v = d$.

In an abuse of notation, we define a valuation θ to be an element of a domain D, written $\theta \in D$, if $\theta(v_i) \in D(v_i)$ for all $v_i \in vars(\theta)$.

We will be interested in determining the infimums and supremums of expressions with respect to some domain D. Define the *infimum* and *supremum* of an expression e with respect to a domain D as $\inf_D e = \inf \ \{\theta(e) | \theta \in D\}$ and $\sup_D e = \sup \ \{\theta(e) | \theta \in D\}$.

We will also use *range* notation: $[l \mathrel{..} u]$ denotes the set $\{d \mid l \leq d \leq u\}$ when l and u are integers, while $[L \mathrel{..} U]$ denotes the set of sets of integers $\{A \mid L \subseteq A \subseteq U\}$ when L and U are sets of integers.

We shall be interested in the notion of an *initial domain*, which we denote D_{init}. The initial domain gives the initial values possible for each variable. In effect an initial domain allows us to restrict attention to domains D such that $D \sqsubseteq D_{init}$.

Propagators A *propagator* f is a monotonically decreasing function from domains to domains, i.e. $D_1 \sqsubseteq D_2$ implies that $f(D_1) \sqsubseteq f(D_2)$, and $f(D) \sqsubseteq D$. A propagator f is *correct* for constraint c iff for all domains D

$$\{\theta \mid \theta \in D\} \cap solns(c) = \{\theta \mid \theta \in f(D)\} \cap solns(c)$$

This is a weak restriction since for example, the identity propagator is correct for all constraints c.

A *propagation solver* for a set of propagators F and current domain D, $solv(F, D)$, repeatedly applies all the propagators in F starting from domain D until there is no further change in resulting domain. In other words, $solv(F, D)$ returns a new domain defined by

$$iter(F, D) = \bigsqcap_{f \in F} f(D) \quad \text{and} \quad solv(F, D) = \text{gfp}(\lambda d.iter(F, d))(D)$$

where gfp denotes the greatest fixpoint w.r.t $\sqsubseteq$ lifted to functions.

Domain Consistency and Set Bounds Consistency A domain D is *domain consistent*[5] for a constraint c if D is the least domain containing all solutions $\theta \in D$ of c, i.e, there does not exist $D' \sqsubset D$ such that $\theta \in D \wedge \theta \in solns(c) \rightarrow \theta \in D'$.

A set of propagators F maintains *domain consistency* for a constraint c, if $solv(F, D)$ is always domain consistent for c.

Define the *domain consistency propagator* for a constraint c as

$$dom(c)(D)(v) = \{\theta(v) \mid \theta \in D \wedge \theta \in solns(c)\} \text{ where } v \in vars(c)$$
$$dom(c)(D)(v) = D(v) \text{ otherwise}$$

Example 1. Consider the constraint $c \equiv x_1 = 3x_2 + 5x_3$. Suppose domain $D(x_1) = \{2, 3, 4, 5, 6, 7\}$, $D(x_2) = \{0, 1, 2\}$, and $D(x_3) = \{-1, 0, 1, 2\}$. The solutions $\theta \in D$ of c are $\theta_1 = \{x_1 \mapsto 3, x_2 \mapsto 1, x_3 \mapsto 0\}$, $\theta_2 = \{x_1 \mapsto 5, x_2 \mapsto 0, x_3 \mapsto 1\}$, and $\theta_3 = \{x_1 \mapsto 6, x_2 \mapsto 2, x_3 \mapsto 0\}$. Hence, $dom(c)(D) = D'$ where $D'(x_1) = \{3, 5, 6\}$, $D'(x_2) = \{0, 1, 2\}$, and $D'(x_3) = \{0, 1\}$. D' is domain consistent with respect to c.

Domain consistency is prohibitive to compute for constraints involving set variables. For that reason, set bounds propagation is typically used where a domain maps a set variable to a lower bound set of integers and an upper bound set of integers.

We shall enforce this by restricting our attention to domains where the $D(S)$ is a range, that is $D(S) = \{A \mid \inf_D(S) \subseteq A \subseteq \sup_D(S)\}$. This is managed by only using set bounds propagators, which maintain this property.

We can define the domain and set bounds propagators $dsb(c)$ for a constraint c as follows:

$$dsb(c)(D)(v) = [\cap dom(c)(D)(v) \; .. \; \cup dom(c)(D)(v)] \text{ where } v \in vars(c) \cap \mathcal{V}_S$$
$$dsb(c)(D)(v) = dom(c)(D)(v) \text{ otherwise}$$

Note that as defined $dsb(c) = dom(c)$ when $vars(c) \subseteq \mathcal{V}_I$. From now on we shall restrict attention to dsb propagators.

3 Propagation Rules

In order to reason effectively about propagation, it will be useful to break down a propagator into the individual propagation steps that it can perform. That is the role of *propagation rules*.

[5] Equivalently, hyper-arc or generalized arc consistent [3].

An *atomic constraint* is one of $x_i = d$, $x_i \neq d$, $d \in S_i$ or $d \notin S_i$ where $x_i \in \mathcal{V}_I$, d is an integer, an $S_i \in \mathcal{V}_S$. An atomic constraint represents the basic changes in domain that occur during propagation, the elimination of a value from an integer domain, or the addition of a value to a lower bound, or removal of a value from an upper bound. Atomic constraints of the form $x_i = d$ are not strictly necessary. They are equivalent to removing all other values from the domain.

A *propagation rule* is of the form $C \rightarrowtail c$ where C is a conjunction of atomic constraints, c is an atomic constraint, and $\not\models C \rightarrow c$. Note our propagation rules when restricted to integer variables are similar to the "membership rules" [2] except we allow equations on the right hand side.

A propagator f *implements* a propagation rule $C \rightarrowtail c$ if for each $D \sqsubseteq D_{init}$ whenever $\models D \rightarrow C$, then $\models f(D) \rightarrow c$. A propagation rule $C \rightarrowtail c$ defines a propagator (for which we use the same notation) in the obvious way.

$$
\begin{array}{l}
(C \rightarrowtail c)(D)(v) = \{\theta(v) \mid \theta \in D \wedge \theta \in \mathit{solns}(c)\} \text{ if } vars(c) = \{v\} \text{ and } \models D \rightarrow C \\
(C \rightarrowtail c)(D)(v) = D(v) \text{ otherwise}
\end{array}
$$

A propagation rule $C_1 \rightarrowtail c_1$ *subsumes* a rule $C_2 \rightarrowtail c_2$ if $\models (D_{init} \wedge C_2) \rightarrow C_1$ and $\models (D_{init} \wedge c_1) \rightarrow c_2$. We can characterize a propagator f in terms of the propagation rules that it implements. Let $rules(f)$ be the set of rules implemented by f. Then $\mathit{prop}(f) \subseteq rules(f)$ are a set of propagation rules such that every $r \in rules(f)$ is subsumed by a rule $r' \in \mathit{prop}(f)$. The set $\mathit{prop}(f)$ can be automatically constructed by the approach of [1].

Example 2. For the propagator $f \equiv \mathit{dsb}(x_1 \neq x_2)$ for $D_{init}(x_1) = D_{init}(x_2) = \{1, 2, 3\}$, $\mathit{prop}(f)$ is

$$
\begin{array}{lll}
x_1 = 1 \rightarrowtail x_2 \neq 1 & x_1 = 2 \rightarrowtail x_2 \neq 2 & x_1 = 3 \rightarrowtail x_2 \neq 3 \\
x_2 = 1 \rightarrowtail x_1 \neq 1 & x_2 = 2 \rightarrowtail x_1 \neq 2 & x_2 = 3 \rightarrowtail x_1 \neq 3
\end{array}
$$

Note that f also implements e.g. $x_1 \neq 1, x_1 \neq 3 \rightarrowtail x_2 \neq 2$, which is subsumed by $x_1 = 2 \rightarrowtail x_2 \neq 2$.

Example 3. For the propagator $f \equiv \mathit{dsb}(S \subseteq T)$ for $D_{init}(S) = D_{init}(T) = \{\emptyset \ldots \{1, 2\}\}$. $\mathit{prop}(f)$ is

$$
\begin{array}{ll}
1 \in S \rightarrowtail 1 \in T & 2 \in S \rightarrowtail 2 \in T \\
1 \notin T \rightarrowtail 1 \notin S & 2 \notin T \rightarrowtail 2 \notin S
\end{array}
$$

A *key result* for domain and set bounds propagators $\mathit{dsb}(c')$, is that the propagation rules implemented are exactly those $C \rightarrowtail c$ where c' implies $C \rightarrow c$.

Lemma 4. *$\mathit{dsb}(c')$ implements $C \rightarrowtail c$ iff $\models (D_{init} \wedge c') \rightarrow (C \rightarrow c)$* □

4 Reasoning about Propagation

In this section we introduce the basic results for reasoning about propagators.

We say a set of propagators F_1 is *stronger* than a set of propagators F_2, written $F_1 \gg F_2$, if $solv(F_1, D) \sqsubseteq solv(F_2, D)$ for all domains $D \sqsubseteq D_{init}$. We say a set of propagators F_1 is *equivalent* to a set of propagators F_2, written $F_1 \approx F_2$, if $solv(F_1, D) = solv(F_2, D)$ for all domains $D \sqsubseteq D_{init}$. A propagator f is made *propagation redundant* by a set of propagators F if $F \gg \{f\}$. It is clear that a constraint c_2 that is logically redundant with respect to constraint c_1 is also propagation redundant with respect to c_1.

Lemma 5. *If* $\models D_{init} \wedge c_1 \rightarrow c_2$ *then* $\{dsb(c_1)\} \gg \{dsb(c_2)\}$. □

Typically though a logically redundant constraint c_2 will be made logically redundant by a conjunction of other constraints. It is well known that in general the domain (and set bounds) propagation of a conjunction of constraints is *not* equivalent to applying the domain (and set bounds) propagators individually.

Example 6. Consider the constraint $c_1 \equiv x_1 = 3x_2$, which is equivalent to $c_2 \wedge c_3$ where $c_2 \equiv x_1 \leq 3x_2$ and $c_3 \equiv x_1 \geq 3x_2$. If $D(x_1) = D(x_2) = \{0, 1, 2, 3, 4, 5, 6, 7\}$, then $D_1 = dsb(c_1)(D)$ and $D_2 = solv(\{dsb(c_2), dsb(c_3)\}, D)$ where $D_1(x_1) = \{0, 3, 6\}$ and $D_2(x_1) = \{0, 1, 2, 3, 4, 5, 6\}$. Hence $\{dsb(c_2), dsb(c_3)\} \not\gg \{dsb(c_1)\}$.

But there are cases where propagation of a conjunction is equivalent to propagation on the individual conjuncts.

Lemma 7. *Let* c_1 *and* c_2 *be two constraints sharing at most one variable* $x \in \mathcal{V}_I$*, then* $\{dsb(c_1), dsb(c_2)\} \approx \{dsb(c_1 \wedge c_2)\}$. □

Note the same result clearly does not hold when the shared variable is a set variable. Consider $c_1 = (S = \{1\} \vee S = \{2, 3\})$ and $c_2 = (S = \{2\} \vee S = \{1, 3\})$, then $solv(\{dsb(c_1), dsb(c_2), D) = D$ where $D(S) = [\emptyset \,..\, \{1, 2, 3\}]$, but $dsb(c_1 \wedge c_2)(D)$ is a false domain.

Propagation rules allow us to break up the consideration of a constraint into individual parts. That is the domain and set bounds propagator of a constraint is equivalent to the union of the propagation rules implemented by the propagator.

Lemma 8. $\{dsb(c')\} \approx \cup_{C \rightarrowtail c \in prop(dsb(c'))} \{C \rightarrowtail c\}$ □

5 Redundant Modelling and Channelling Constraints

Redundant modelling [4] models the problem from two different viewpoints. In general, the propagators defined for the two viewpoints act in different ways and discover information at different stages in the search. By joining the two models using channelling constraints, we can get the advantage of both sources of propagation. Of course, each model is logically redundant with respect to the other model plus the channelling constraints. However, in this section, we show cases in which propagation caused by some constraints in one model is subsumed by propagation induced from constraints in the other model through the channelling constraints.

Assume we have one model of the problem M_X using variables X, and another model M_Y using disjoint variables Y. Channelling constraints are used to join these two models together by relating X and Y. There is no real agreement on precisely what channelling constraints may be yet. For the purposes of our theorems we define a channelling constraints as follows.

Let A_X be the atomic constraints for D_{init} on variables X, and A_Y be the atomic constraints for D_{init} on variables Y. A *channel function* $\diamondsuit$ is a bijection from atomic constraints A_X to A_Y.

A *channelling constraint* (or simply *channel*) $C_\diamondsuit$ is the constraint

$$\bigwedge_{c \in A_X} (c \Leftrightarrow \diamondsuit(c))$$

The *channel propagator* $F_\diamondsuit$ is the set of propagation rules inferred from the channel function $\diamondsuit$.

$$F_\diamondsuit = \bigcup_{c \in A_X} \{c \rightarrowtail \diamondsuit(c), \diamondsuit(c) \rightarrowtail c\}$$

Note for channel function $\diamondsuit$, by definition $\diamondsuit^{-1}$ is also a channel function, and $C_\diamondsuit$ and $C_{\diamondsuit^{-1}}$, as well as $F_\diamondsuit$ and $F_{\diamondsuit^{-1}}$, are identical.

We now illustrate how common channels fit into this framework.

Permutation Channels A common form of redundant modelling is when we consider two viewpoints to a permutation problem. We can view the problem as finding a bipartite matching between two sets of objects of the same size. Assume the two viewpoints have the set of variables $X = \{x_0, \ldots, x_n\}$, and $Y = \{y_0, \ldots, y_n\}$ respectively.

The *permutation channel function* $\bowtie$ is defined as $\bowtie(x_i = j) = (y_j = i)$ and $\bowtie(x_i \neq j) = (y_j \neq i)$. The *permutation channel* $C_\bowtie$ is equivalent to the conjunction of constraints $\bigwedge_{i=0}^{n} \bigwedge_{j=0}^{n} (x_i = j \Leftrightarrow y_j = i)$.

Boolean Channels Another common redundant modelling is when we give both an integer and Boolean model. Suppose the integer variables are $X = \{x_0, \ldots, x_n\}$, where $D_{init}(x_i) = [0 \mathrel{..} k_i]$, and the Boolean variables are $Z = \{z_{ij} \mid 0 \leq i \leq n, 0 \leq j \leq k_i\}$

The *Boolean channel function* $\triangle$ is defined as $\triangle(x_i = j) = (z_{ij} = 1)$ and $\triangle(x_i \neq j) = (z_{ij} = 0)$. Note that the atomic constraints $z_{ij} \neq 1$ and $z_{ij} \neq 0$ are not needed for Boolean variables since they are equivalent (respectively) to $z_{ij} = 0$ and $z_{ij} = 1$. The *Boolean channel* $C_\triangle$ is equivalent to the conjunction of constraints $\bigwedge_{i=0}^{n} \bigwedge_{j=0}^{k_i} (x_i = j \Leftrightarrow z_{ij} = 1)$.

Set Channels A common form of redundant modelling is where one model deals with integer variables, and the other with variables over finite sets of integers, and the relation $x_i = j$ holds iff $i \in S_j$. This generalizes the assignment problem to where two or more integer variables can take the same value. Suppose the integer variables are $X = \{x_0, \ldots, x_n\}$, where $D_{init}(x_i) = [0 \mathrel{..} k]$, and the set variables are $\{S_0, \ldots, S_k\}$. The *set channel function* $\{\}$ is defined as $\{\}(x_i = j) = (i \in S_j)$ and $\{\}(x_i \neq j) = (i \notin S_j)$. The *set channel* $C_{\{\}}$ is equivalent to $\bigwedge_{i=0}^{n} \bigwedge_{j=0}^{k} (x_i = j \Leftrightarrow i \in S_j)$.

5.1 Proving Propagation Redundancy using Channels

We can now prove the fundamental theorem about propagation redundancy through channels. The core result is that a propagation rule that is mapped by a channel function to a rule subsumed by another propagation rule is propagation redundant. We extend channel functions to map conjunctions of constraints in the obvious manner $\diamondsuit(c_1 \wedge \cdots \wedge c_n) = \diamondsuit(c_1) \wedge \cdots \wedge \diamondsuit(c_n)$.

Lemma 9. *Let $C \rightarrowtail c$ be a propagation rule on Y variables, and $C' \rightarrowtail c'$ be a propagation rule on X variables. If $C' \rightarrowtail c'$ subsumes $\diamondsuit^{-1}(C) \rightarrowtail \diamondsuit^{-1}(c)$ then $\{C' \rightarrowtail c'\} \cup F_{\diamondsuit} \gg \{C \rightarrowtail c\}$.* □

We can straightforwardly lift this result to talk about propagation rules that are subsumed by the domain and set bounds propagator for a constraint, and then lift to set of propagation rules implemented by some propagator.

Lemma 10. *Let $C \rightarrowtail c$ be a propagation rule on Y variables, and c_X be a constraint on X variables. If $\models (D_{init} \wedge c_X \wedge \diamondsuit^{-1}(C)) \rightarrow \diamondsuit^{-1}(c)$, then $\{dsb(c_X)\} \cup F_{\diamondsuit} \gg \{C \rightarrowtail c\}$.* □

Corollary 11. *Let f_Y be a propagator on Y variables, and c_X be a constraint on X variables. If $\models (D_{init} \wedge c_X \wedge \diamondsuit^{-1}(C)) \rightarrow \diamondsuit^{-1}(c)$ for all $C \rightarrowtail c \in prop(f_Y)$, then $\{dsb(c_X)\} \cup F_{\diamondsuit} \gg \{f_Y\}$.* □

Often a single constraint does not capture all the propagation effects of a constraint on the other side of the permutation model. In that case we may need to find for each particular propagation rule, a constraint on the other side that causes the same propagation to occur.

Theorem 12. *Let f_Y be a propagator on Y variables. Suppose for each $r \equiv (C \rightarrowtail c) \in prop(f_Y)$, there exists constraint $imp(r)$ on X variables where $\models (D_{init} \wedge imp(r) \wedge \diamondsuit^{-1}(C)) \rightarrow \diamondsuit^{-1}(c)$, then $\cup_{r \in prop(f_Y)}\{dsb(imp(r))\} \cup F_{\diamondsuit} \gg \{f_Y\}$.* □

The framework just presented is closely related to Brand's approach [10] of identifying redundant rules in the compilation of constraints into rule-based constraint programs [2]. While Brand reasons redundancy at the rule level, we emply propagation rules as an analysis tool to detect redundancy at the constraint level.

5.2 Restrictive and Unrestrictive Channel Functions

The channels themselves may actually restrict the possible solutions in one or both models involved. We will concentrate on the X model, since the restrictions on the Y model can be seen by examining the inverse channel function.

A channel function $\diamondsuit$ is *restrictive* (on the variables X) if $\not\models D_{init} \rightarrow \exists Y C_{\diamondsuit}$, that is not all valuations on X variables are extensible to solutions of $C_{\diamondsuit}$.

Example 13. The $\bowtie$ channel function is restrictive, for example $\{x_1 = 2, x_2 = 2\}$ cannot be extended to be a solution of $C_{\bowtie}$, since it requires y_2 to take both values 1 and 2. The $\triangle$ channel function is unrestrictive. Any valuation on X variables can be extended to a solution of $C_{\triangle}$. However $\triangle^{-1}$ is restrictive, for example $\{z_{11} = 1, z_{12} = 1\}$ cannot be extended to a solution of $C_{\triangle}$ since it requires x_1 to be both 1 and 2.

Restrictive channel function can themselves make constraints propagation redundant.

Smith [7] first observes that the permutation channel makes each of the disequations between variables in either model propagation redundant. Walsh [9] proves this holds for other notions of consistency.

Lemma 14 ([9]). $F_{\bowtie} \gg \{dsb(x_i \neq x_k)\}$ □

Implicit in the Boolean channel is that each integer variable can take only one, and must take one value. This is represented in the Boolean model as the constraint $\sum_{j=0}^{k_i} z_{ij} = 1$. It is enforced by the restrictive channel function $\triangle^{-1}$.

Lemma 15. $F_{\triangle} \gg \{dsb(\sum_{j=0}^{k_i} z_{ij} = 1)\}$ *for all* $1 \leq i \leq n$. □

The channel function $\{\}^{-1}$ is restrictive, since each variable $x_i \in X$ can only take a single value j. It means that $S_j \cap S_{j'} = \emptyset$ for all $0 \leq j < j' \leq m$. It is clear that $F_{\{\}}$ makes these constraints propagation redundant.

Lemma 16. $F_{\{\}} \gg \{dsb(S_j \cap S_{j'} = \emptyset)\}$ *for all* $0 \leq j < j' \leq m$. □

Unrestrictive channel functions do not make any constraints (on X) propagation redundant. Interestingly in this case we can argue about propagation redundancy simply in terms of logical consequence.

Theorem 17. *Let* $\diamondsuit$ *be an unrestrictive channel function, let* c_Y *be a constraint on* Y *variables, and* c_X *a constraint on* X *variables. If* $\models (D_{init} \wedge c_X \wedge C_{\diamondsuit}) \rightarrow c_Y$, *then* $\{dsb(c_X)\} \cup F_{\diamondsuit} \gg \{dsb(c_Y)\}$. □

The reason the channel function must be unrestrictive for this result to hold is that the $\models (D_{init} \wedge c_X \wedge C_{\diamondsuit}) \rightarrow c_Y$ is too weak a condition in the general case.

Example 18. The permutation channel function is restrictive. Now $C \equiv x_0 + x_1 < 2 \wedge C_{\bowtie}$ is such that $\models C \rightarrow y_2 = 2$ since the only solutions of C are $\{x_0 \mapsto 0, x_1 \mapsto 1, x_2 \mapsto 2, y_0 \mapsto 0, y_1 \mapsto 1, y_2 \mapsto 2\}$ and $\{x_0 \mapsto 1, x_1 \mapsto 0, x_2 \mapsto 2, y_0 \mapsto 1, y_1 \mapsto 0, y_2 \mapsto 2\}$. But clearly it is not the case that $x_0 + x_1 < 2 \rightarrow x_2 = 2$. The problem is that the channel $C_{\bowtie}$ removes solutions of $x_0 + x_1 < 2$ like $\{x_0 \mapsto 0, x_1 \mapsto 0, x_2 \mapsto 0\}$ from consideration.

6 Example Problems

In the following, we give examples where the constraints in redundant modelling are propagation redundant.

6.1 All-Interval Series

The all-interval series problem, listed as "prob007" in CSPLib, from musical composition. The problem is to find a permutation of n numbers from 0 to $n-1$, such that the differences between adjacent numbers form a permutation from 1 to $n-1$.

There are two ways to model the problem. The first model, M_X, consists of n variables, $X = \{x_0, \dots, x_{n-1}\}$. Each x_i denotes the number in position i, and $D_{init}(x_i) = \{0, \dots, n-1\}$ for $i \in \{0, \dots, n-1\}$. We introduce auxiliary variables, $\{u_0, \dots, u_{n-2}\}$, that denote the difference between adjacent numbers. The constraints are:

- disequality constraints (IX1): $\forall 0 \le i < j \le n-1.\ x_i \ne x_j$ and $\forall 0 \le i < j \le n-2.\ u_i \ne u_j$.
- interval constraints (IX2): $\forall 0 \le i \le n-2.\ u_i = |x_i - x_{i+1}| - 1$.

The second model, M_Y, also consists of n variables, $Y = \{y_0, \dots, y_{n-1}\}$. Each y_i denotes the position for the number i, and $D_{init}(y_i) = \{0, \dots, n-1\}$ for $i \in \{0, \dots, n-1\}$. The auxilliary variables $\{v_0, \dots, v_{n-2}\}$ denote the position where the difference value of 1 to $n-1$ belongs. The constraints are:

- disequality constraints (IY1): $\forall 0 \le i < j \le n-1.\ y_i \ne y_j$ and $\forall 0 \le i < j \le n-2.\ v_i \ne v_j$.
- interval constraints (IY2): The constraints $\forall 0 \le i < j \le n-1.\ (y_i - y_j = 1) \Rightarrow (v_{j-i-1} = y_j)$ and $(y_j - y_i = 1) \Rightarrow (v_{j-i-1} = y_i)$ enforce that if y_i and y_j are adjacent, the position for their difference must be the smaller of them.

In the second model, we observe that only y_0 and y_{n-1} can lead to a difference value of $n-1$. Therefore, we can add the redundant constraints: (IY3) $(|y_0 - y_{n-1}| = 1) \wedge (v_{n-2} = min(y_0, y_{n-1}))$, to force y_0 and y_{n-1} to be adjacent.

The permutation channels for this problem are more interesting because we have two distinct kinds of variables in each model, each of which is related by a permutation channel. The channels are $x_i = j \Leftrightarrow y_j = i$ and $u_i = j \Leftrightarrow v_j = i$.

Example 19. Consider the constraint $c_Y \equiv (y_i - y_j = 1) \Rightarrow (v_{j-i-1} = y_j)$ of the all-intervals series problem. The propagation rules for $dsb(c_Y)$ have the forms

$$
\begin{array}{ll}
r1 & y_i = k+1 \wedge y_j = k \rightarrowtail v_{j-i-1} = k \\
r2 & v_{j-i-1} \ne k \wedge y_j = k \rightarrowtail y_i \ne k+1 \\
r3 & \qquad y_i = k+1 \wedge I \rightarrowtail y_j \ne k
\end{array}
$$

where in $r3$, I is any conjunction of disequations on v_{j-i-1} and y_j, not including $y_j \ne k$ ensuring that $v_{j-i-1} \ne y_j$. We can show for $imp(r1) \equiv imp(r2) \equiv imp(r3) \equiv (u_k = |x_k - x_{k+1}| - 1)$ that $\models (D_{init} \wedge imp(r1) \wedge x_{k+1} = i \wedge x_k = j) \rightarrow (u_k = j-i-1)$ and $\models (D_{init} \wedge imp(r2) \wedge u_k \ne j-i-1 \wedge x_k = j) \rightarrow (x_{k+1} \ne i)$. For the remaining propagation rules (r3), it is clear that I must contain $v_{j-i-1} \ne k$ since it does not contain $y_j \ne k$ and it must force the two to be different. We can show that $\models (D_{init} \wedge imp(r3) \wedge u_k \ne j-i-1 \wedge x_{k+1} = i) \rightarrow (x_k \ne j)$.

Hence the constraint is propagation redundant by Theorem 12. Similarly for the other (IY2) constraints $(y_j - y_i = 1) \Rightarrow (v_{j-i-1} = y_i)$. The disequality constraints (IY1) $y_i \ne y_j$ and $v_i \ne v_j$ are propagation redundant by Lemma 14. The only non-propagation redundant constraints in M_Y is (IY3) $(|y_0 - y_{n-1}| = 1) \wedge (v_{n-2} = min(y_0, y_{n-1}))$.

6.2 n-Queens Problem

In the n-queens problem, the task of which is to place n queens on an $n \times n$ chess board so that no two queens can attack each other.

The first model, M_X, consists of n variables, $X = \{x_0, \ldots, x_{n-1}\}$. Each x_i denotes the column position of the queen on row i, and $D(x_i) = \{0, \ldots, n-1\}$, for $i \in \{0, \ldots, n-1\}$. The constraints C_X enforce that no two queens can be on the same:

- column (QX1): $\forall 0 \leq i < j \leq n-1.\ x_i \neq x_j$.
- diagonal (QX2): $\forall 0 \leq i < j \leq n-1.\ x_i - i \neq x_j - j, x_i + i \neq x_j + j$.

The second model, M_Z, consists of $n \times n$ Boolean variables, $Z = \{z_{00}, \ldots, z_{0(n-1)}, \ldots, z_{(n-1)0}, \ldots, z_{(n-1)(n-1)}\}$. Each z_{ij} denotes whether we have a queen at row i column j or not. The constraints C_Z enforce that no two queens can be on the same:

- row (QZ1): $\forall 0 \leq i \leq n-1.\ \sum_{j=0}^{n-1} z_{ij} = 1$.
- column (QZ2): $\forall 0 \leq j \leq n-1.\ \sum_{i=0}^{n-1} z_{ij} = 1$.
- main diagonal (QZ3): $\sum_{i=0}^{n-1} z_{ii} \leq 1$, and $\sum_{i=0}^{n-1} z_{i(n-1-i)} \leq 1$.
- other diagonal (QZ4): $\forall 1 \leq k \leq n-1.\ \sum_{j=0}^{n-1-k} z_{j(j+k)} \leq 1, \sum_{j=0}^{n-1-k} z_{(j+k)j} \leq 1, \sum_{j=0}^{n-1-k} z_{j(n-1-j-k)} \leq 1, \sum_{j=0}^{n-1-k} z_{(j+k)(n-1-j)} \leq 1$.

We combine the two models using the Boolean channel $x_i = j \Leftrightarrow z_{ij} = 1$.

Example 20. In M_Z, the row constraints (QZ1) $\sum_{j=0}^{n-1} z_{ij} = 1$ are propagation redundant using Lemma 15.

Consider the main diagonal constraint (QZ3) $c_Z \equiv \sum_{i=0}^{n-1} z_{ii} \leq 1$. We can show that $c_X \equiv x_1 \neq x_i - i - 1 \wedge \cdots x_{i-1} \neq x_i - 1 \wedge x_{i+1} \neq x_i + 1 \wedge \cdots x_{n-1} \neq x_i + n - i - 1$ is such that $\models D_{init} \wedge c_X \wedge C_\triangle \rightarrow c_Z$. Now $dsb(c_X) \approx dsb(x_1 \neq x_i) \cup \cdots \cup dsb(x_{n-1} \neq x_i)$ by Lemma 7 since they share only one variable x_i. Since $\triangle$ is an unrestrictive channel function, by Theorem 17 we have that $dsb(c_Z)$ is propagation redundant. A similar argument applies to all other diagonal constraints (QZ4).

Note that the column constraints (QZ2) $\sum_{i=0}^{n-1} z_{ij} = 1$ are not propagation redundant, although the constraint $\sum_{i=0}^{n-1} z_{ij} \leq 1$ is (using a similar argument to the main diagonal constraints).

6.3 Balanced Academic Curriculum Problem

The problem "prob030" in CSPLib is to design a balanced academic curriculum. Following the description in [6], we can have both the integer model M_X and set model M_S.

Given m courses, and n periods, a, b are the minimum and maximum academic load allowed per period, c, d are the minimum and maximum number of courses allowed per period, t_i specifies the number of credits for course i, and R is a set of pairs $\langle i, j \rangle$ specifying that course i must be taken before course j.

We introduce a set of auxiliary variables l_j, which is shared by both models, to represent the academic load in period j as well as a variable u representing the maximum academic load in any period, i.e. $u = max\{l_j \mid 0 \leq j \leq n-1\}$. The objective function simply minimizes u. We also introduce another set of shared auxiliary variables q_j to represent the number of courses assigned to a period.

We have the following constraints that is common to both models (B1): $\forall 0 \leq j \leq n-1.\ a \leq l_j \leq b$ and $c \leq q_j \leq d$. We also add the following redundant constraints (B2): $\forall 0 \leq j \leq n-1.\ (\sum_{j=0}^{n-1} l_j) = (\sum_{i=0}^{m-1} t_i)$ and $(\sum_{j=0}^{n-1} q_j) = m$.

In the integer model, M_X, the variable x_i represents the period to which course i is assigned The constraints for the integer model M_X are:

- (BX1) $\forall 0 \leq j \leq n-1.\ (\sum_{i=0}^{m-1}((x_i = j) \times t_i)) = l_j$
- (BX2) $\forall 0 \leq j \leq n-1.\ (\sum_{i=0}^{m-1}(x_i = j)) = q_j$
- (BX3) $\forall \langle i, j \rangle \in R.\ x_i < x_j$

In the set model the set variables S_j representing the set of courses assigned to period j. The constraints for the set model M_S are:

- (BS1) $\forall 0 \leq i < j \leq n-1.\ S_i \cap S_j = \emptyset$
- (BS2) $\forall 0 \leq j \leq n-1.\ (\sum_{i \in S_j} t_i) = l_j$
- (BS3) $\forall 0 \leq j \leq n-1.\ |S_j| = q_j$
- (BS4) $\forall \langle i, j \rangle \in R. \forall 1 \leq k \leq n-1. \forall 0 \leq k' \leq k.\ (i \in S_k) \Rightarrow (j \notin S_{k'})$

We can use the set channels to combine the two models, $x_i = j \Leftrightarrow i \in S_j$.

Example 21. The (BS1) constraint $S_i \cap S_j = \emptyset$ is propagation redundant using Lemma 16. For the (BS4) constraint $c_S \equiv (i \in S_k) \Rightarrow (j \notin S_{k'})$ where $k' \leq k$ we can show that $\models (D_{init} \wedge x_i < x_j \wedge C_{\{\}}) \rightarrow c_S$. Hence since $\{\}$ is an unrestrictive channel function by Theorem 17 we have that $dsb(c_S)$ is propagation redundant.

Example 22. In an abuse of notation we use the "pseudo atomic constraint". $x \leq d$ to represent the conjunction $x \neq d+1, \ldots, x \neq \sup_{D_{init}}(x)$ and $x \geq d$ to represent the conjunction $x \neq \inf_{D_{init}}(x), \ldots, x \neq d-1$.

Consider the (BX2) constraint $c_X \equiv (\sum_{i=0}^{m-1}(x_i = j)) = q_j$, the propagation rules $C \rightarrowtail c$ for $dsb(c_X)$ are

$$\begin{aligned} q_j \leq d \wedge x_{i_1} = j \wedge \cdots \wedge x_{i_d} = j &\rightarrowtail x_i \neq j \\ x_{i_1} = j \wedge \cdots \wedge x_{i_d} = j &\rightarrowtail q_j \geq d \end{aligned}$$

for all $I = \{i_1, \ldots, i_d\} \subseteq \{0, \ldots, m-1\}$ and $i \in \{0, \ldots, m-1\} - I$; and

$$\begin{aligned} q_j \geq d \wedge x_{i_1} \neq j \wedge \cdots \wedge x_{i_{m-d}} \neq j &\rightarrowtail x_i = j \\ x_{i_1} \neq j \wedge \cdots \wedge x_{i_{m-d}} \neq j &\rightarrowtail q_j \leq d \end{aligned}$$

for all $I = \{i_1, \ldots, i_{m-d}\} \subseteq \{0, \ldots, m-1\}$ and $i \in \{0, \ldots, m-1\} - I$. Notice that all the atomic constraints involving q_j are mapped to themselves by $\{\}$, since q_j is shared by the two models. The rules are mapped to

$$\begin{aligned} q_j \leq d \wedge i_1 \in S_j \wedge \cdots \wedge i_d \in S_j &\rightarrowtail i \notin S_j \\ i_1 \in S_j \wedge \cdots \wedge i_d \in S_j &\rightarrowtail q_j \leq d \\ q_j \geq d \wedge i_1 \notin S_j \wedge \cdots \wedge i_{m-d} \notin S_j &\rightarrowtail i \in S_j \\ i_1 \notin S_j \wedge \cdots \wedge i_{m-d} \notin S_j &\rightarrowtail q_j \geq d \end{aligned}$$

We have that for $c_S \equiv |S_j| = q_j$, $\models (D_{init} \wedge c_S \wedge \{\}(C)) \rightarrow \{\}(c)$ for all the propagation rules above. Hence, $dsb(c_X)$ is propagation redundant using Corollary 11.

Similar reasoning applies to show that each constraint $(\sum_{i=0}^{m-1}((x_i = j) \times t_i)) = l_j$ of (BX1) is made propagation redundant by (BS2) $(\sum_{i \in S_j} t_i) = l_j$.

7 Experiments

We can take advantage of the reasoning about propagation redundancy to eliminate propagators that are propagation redundant. We then get a model with exactly the same propagation behaviour but with less propagators. This can translate into *faster* propagation.[6] In the following experiments, All the benchmarks were executed using ILOG Solver 4.4 on Sun Ultra 5/400 workstation running Solaris 8.

7.1 All-Interval Series

We compare the different models for solving the all-interval series problem. We search for all solutions in order to fairly compare the propagation strengths and use a first-fail heuristic for variables selection, and least to greatest value selection heuristic.

The models under comparison include the single models: M_X and M_Y, the *full* combined model M_X + $C_{\bowtie}$ + M_Y, and an *opt*imized combined model IX2 + $C_{\bowtie}$ + IY3 as discussed in Example 19. Puget and Régin, in their note[7], show that all the solutions can be found more efficiently by replacing (IX1) by (IX1') `alldifferent` constraints on x and u. The *pr* model uses $IX1' + IX2$. The *pr full* model is the combination of *pr* and M_Y, IX1' + IX2 + $C_{\bowtie}$ + M_Y. The *pr opt* model is the optimized combination of *pr* and M_Y, IX1' + IX2 + $C_{\bowtie}$ + IY3 since the same reasoning applies.

Table 1 gives the results of the comparison. We show the results using three sets of search variables X, Y and $X \cup Y$. Entries with a "—" mean unable to solve the problem after one hour of execution time. Compared with the single models M_X and M_Y, clearly the *full* and *pr full* model reduces the number of fails significantly. The *opt* model maintains the same number of fails as the *full* model and is the fastest for the smaller instance 12. The *pr opt* model maintains the same number of fails as the *pr full* model, and is the fastest for larger instances 13, 14 and 15, as the `alldifferent` constraints is too expensive for the smaller instance. Note that the optimized models *opt* and *pr opt* can solve the size 15 instance much faster than *pr*, and no other models can solve this instance within the time limit.

7.2 Balanced Academic Curriculum Problem

Table 2 shows the result of finding the optimal solution and proving optimality for some smaller instances derived from the problem instances posted in CSPLib. We use the first-fail heuristic for the search on the integer variables X, and naive enumeration for search on the set variables S. The table entry with value "—" means that Solver cannot solve the problem after one hour of execution time.

[6] Note there is no guarantee since e.g. the number of propagation steps may have increased.

[7] Available at `http://www.csplib.org/prob/prob007/puget.pdf`.

	Search	$n = 12$		$n = 13$		$n = 14$		$n = 15$	
Model	Vars	fails	(sec)	fails	(sec)	fails	(sec)	fails	(sec)
pr	X	38778	(24.32)	156251	(105.26)	674346	(530.47)	3045037	(2328.57)
M_X	X	880112	(260.92)	4914499	(1589.83)	—	—	—	—
full	X	39241	(222.07)	158368	(1048.19)	—	—	—	—
opt	X	39241	(36.34)	158368	(157.84)	685301	(770.57)	—	—
pr full	X	38461	(236.42)	155183	(1088.91)	—	—	—	—
pr opt	X	38461	(42.77)	155183	(188.94)	670045	(910.90)	—	—
M_Y	Y	—	—	—	—	—	—	—	—
full	Y	16280	(70.81)	62949	(303.61)	266130	(1458.74)	—	—
opt	Y	16280	**(6.36)**	62949	(26.00)	266130	(108.54)	1275661	(553.45)
pr full	Y	**12296**	(62.96)	**43681**	(260.90)	**164841**	(1127.64)	—	—
pr opt	Y	**12296**	(7.91)	**43681**	**(25.78)**	**164841**	**(101.42)**	**704097**	**(458.12)**
full	$X \cup Y$	39195	(222.42)	158282	(1065.77)	—	—	—	—
opt	$X \cup Y$	39195	(36.36)	158282	(158.40)	684592	(783.01)	—	—
pr full	$X \cup Y$	38447	(230.65)	155176	(1094.61)	—	—	—	—
pr opt	$X \cup Y$	38447	(42.47)	155176	(198.36)	669950	(898.66)	—	—

Table 1. Comparing the different models of All-Interval Series Problem

The *full* model represents the full combined model between the integer and set model as discussed in Section 6.3, while the *opt* model represents the reduced combined model after removing the redundant propagators as discussed in Example 21 and 22, that is B1 + B2 + BX3 + $C_{\{\}}$ + BS2 + BS3. In [6], the authors reported that it is difficult to find the optimal solution and prove optimality with propagation based solving alone. However, by adding redundant constraints (B2), we were able to solve all the problem instances with M_X alone. The row *CPLEX* gives the runtime for solving the problem instances with ILOG CPLEX 8.0 using an integer linear programming (ILP) model in [6]. The row *Hybrid* implements the hybrid ILP and CP model described in [6] together with the redundant constraints (B2) using ILOG Hybrid 1.3. Clearly, the *full* model is substantially better in terms of number of fails when compared with the single model (M_X or M_S), The *hybrid* model gives the least number of fails, but suffer from the overhead of invoking two solvers. The *opt* model is more efficient and can solve all the instances in less than 1 second.

8 Conclusion

It is clear that reasoning about propagation redundancy can lead to significantly faster models, that do not increase the search space. Although we have illustrated the use of the theorems herein by hand, the approach can clearly be automated. To use Theorem 12 we can straightforwardly define the propagation rules for many constraints (parametrically in D_{init}) or construct them automatically using the approach of [1]. Given the propagation rules, we can individually check those that are subsumed by constraints in the other model. If we have a parametric definition, then this check can also be parametric, rather than needing to consider every individual propagation rule. We can use Theorem 17 to prove propagation redundancy without considering propagation rules.

Model	Search Variables	8 Periods fails	(sec)	10 Periods fails	(sec)	12 Periods fails	(sec)
CPLEX	n/a	n/a	(1.80)	n/a	(2.27)	n/a	(20.32)
Hybrid	X	**101**	(0.61)	468	(2.20)	58442	(146.47)
Hybrid	Boolean	219	(0.76)	**277**	(1.03)	**315**	(2.09)
M_X	X	**101**	**(0.04)**	468	(0.25)	33602	(11.62)
full	X	**101**	(0.24)	470	(1.80)	33530	(192.62)
opt	X	**101**	(0.08)	470	(0.68)	33530	(38.54)
M_S	S	—	—	—	—	—	—
full	S	1577	(2.83)	323	(0.81)	882	(4.56)
opt	S	1577	(0.94)	323	**(0.24)**	882	**(0.95)**

Table 2. Comparing the different models for solving the balanced academic curriculum problem

There are clearly many important future directions for this line of work. Modern set bounds propagation solvers (including ILOG Solver 4.4) implement slightly stronger propagators than $dsb(c)$, by including cardinality reasoning. We can model this extra propagation using cardinality variables and propagation rules. For a set constraint c to be propagation redundant we need to prove their redundancy too. For the examples in this paper this is straightforward. We plan to extend the theorems for the general case.

Similarly, many integer constraint solvers use integer bounds propagation. Clearly we can extend the notion of propagation rules to integer bounds propagators using the atomic constraints $x_i \leq d$ and $x_i \geq d$. The only complication arises in formalizing what the bounds propagators are for an individual constraint. Usually bounds propagators do not have a completeness property like Lemma 4.

References

[1] S. Abdennadher and C. Rigotti. Automatic generation of rule-based solvers for intentionally defined constraints. *IJAIT* 11(2):283–302, 2002.

[2] K. Apt and E. Monfroy. Constraint programming viewed as rule-based programming. *Theory and Practice of Logic Programming*, 1(6):713–750, 2001.

[3] C. Bessiére and J. Régin. Arc consistency for general constraint networks: preliminary results. In *IJCAI-97* pages 398–404, 1997.

[4] B. Cheng, K. Choi, J. Lee, and J. Wu. Increasing constraint propagation by redundant modelling: an experience report. *Constraints*, 4(2):167–192, 1999.

[5] C.W. Choi, J.H.M. Lee, and P. J. Stuckey Propagation Redundancy for Permutation Channels In *IJCAI-03*, to appear.

[6] B. Hnich, Z. Kiziltan, and T. Walsh. Modelling a balanced academic curriculum problem. In *CP-AI-OR'02*, pages 121–131, 2002.

[7] B. M. Smith. Modelling a permutation problem. Research Report 2000.18, School of Computer Studies, University of Leeds, 2000.

[8] B. M. Smith. Dual models in constraint programming. Research Report 2001.02, School of Computer Studies, University of Leeds, 2001.

[9] T. Walsh. Permutation problems and channelling constraints. In *LPAR2001*, pages 377–391, 2001.

[10] S. Brand. A note on redundant rules in rule-based constraint programming. In *Recent Advances in Constraints*, pages 109 – 120, 2003.