Removing Propagation Redundant Constraints in Redundant Modeling

C.W. CHOI and J.H.M. LEE
The Chinese University of Hong Kong
and
P. J. STUCKEY
NICTA Victoria Laboratory and the University of Melbourne

A widely adopted approach to solving constraint satisfaction problems combines systematic tree search with various degrees of constraint propagation for pruning the search space. One common technique to improve the execution efficiency is to add redundant constraints, which are constraints logically implied by others in the problem model. However, some redundant constraints are *propagation redundant* and hence do not contribute additional propagation information to the constraint solver. Redundant constraints arise naturally in the process of redundant modeling where two models of the same problem are connected and combined through channeling constraints. In this paper, we give general theorems for proving propagation redundancy of one constraint with respect to channeling constraints and constraints in the other model. We illustrate, on problems from CSPlib (`http://www.csplib.org/`), how detecting and removing propagation redundant constraints in redundant modeling can speed up search by several order of magnitudes.

Categories and Subject Descriptors: D.3.3 [**Programming Languages**]: Language Constructs and Features—*Constraints*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Logic and constraint programming*

General Terms: Performance, Theory

Additional Key Words and Phrases: Constraint Propagation, Redundant Constraints, Redundant Modeling

1. INTRODUCTION

Finite domain constraint programming combines backtracking tree search with constraint propagation to solve *constraint satisfaction problems* (CSPs) [Mackworth 1977]. *Constraint propagation* removes infeasible values from the domains of variables to reduce the search space. This propagation-based constraint solving framework is realized in modern constraint programming systems such as ECL^{i}PS^e [Cheadle et al. 2003], ILOG Solver [1999], and SICStus Prolog [2003], which have been successfully applied to many real-life industrial applications.

Author's address: C. W. Choi and J. H. M. Lee, Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong, email: `{cwchoi,jlee}@cse.cuhk.edu.hk`; P. J. Stuckey, NICTA Victoria Laboratory, Department of Computer Science & Software Engineering, University of Melbourne, 3010, Australia, email: `pjs@cs.mu.oz.au`.

There is usually more than one way of modeling a problem as a CSP. By modeling a problem as a CSP, we mean the process of determining the variables, the associated domains of the variables, and the expressions of the constraints. Finding a good model of a CSP is a challenging task. A modeler must specify a set of constraints that capture the definitions of the problem, but this is not enough. The model should also have *strong* propagation: that is, it should be able to quickly reduce the domains of the variables of the problem. Moreover, the implementation of *propagators* to perform constraint propagation should be efficient. Last but not least, the choice of variables and the associated domains should lead to a smaller search space than others.[1]

A common technique to increase propagation strength is to add *redundant constraints*,[2] which are logically implied by the constraints of the model. An early and significant use of redundant constraints appears in the work of Carlier and Pinson [1989] for solving job-shop scheduling problems. Adding redundant constraints can be beneficial since the constraint solver may extract more information from these redundant constraints. However, some logically redundant constraints are *propagation redundant*, and hence do not contribute additional propagation information to the constraint solver. Generally, we only want to add redundant constraints that are not propagation redundant in order to reduce the search space.

Example 1. Consider the following constraints,

$$x_1 \geq x_2,\ x_2 \geq x_3,\ x_1 \geq x_3.$$

Suppose the domain for x_1 is $\{-2, -1, 0, 1\}$, and the domains for x_2 and x_3 are both $\{-2, -1, 0, 1, 2\}$. During constraint propagation, the constraint solver checks each constraint in turn and removes infeasible values from the domains. This process is repeatedly applied until there are no further changes in the resulting domains.

(1) We check $x_1 \geq x_2$ and remove 2 from the domain of x_2 since it is infeasible to form a solution of $x_1 \geq x_2$ with $x_2 = 2$. Now, the domain of x_2 is $\{-2, -1, 0, 1\}$.
(2) We check $x_2 \geq x_3$ and remove 2 from the domain of x_3 since it is infeasible to form a solution of $x_2 \geq x_3$ with $x_3 = 2$. Now, the domain of x_3 is $\{-2, -1, 0, 1\}$.
(3) We check $x_1 \geq x_3$ and do nothing. This checking is redundant since anything that can be removed by this constraint will be removed by the other two. The constraint is *propagation redundant.*

Clearly, the constraint $x_1 \geq x_3$ is logically implied by the constraints $x_1 \geq x_2$ and $x_2 \geq x_3$. Note that if $x_1 \geq x_3$ were the first constraint checked, it would indeed remove the value 2 from the domain of x_3. However, it is easy to verify that by removing $x_1 \geq x_3$ from the model, we still obtain exactly the same resulting domains. Hence, $x_1 \geq x_3$ does not (really) contribute additional domain reduction to the model. ■

Note that logical redundancy does not imply propagation redundancy.

[1]For example, the search space of a problem model using integer variables is usually smaller than that using Boolean variables.

[2]Redundant constraints are also known as *implied constraints* in some CSP literature [Smith et al. 2000; Frisch et al. 2004].

Example 2. Consider the following constraints,

$$x_1 - x_2 \geq 0, x_1 + x_2 \geq 0, x_1 \geq 0$$

Suppose the domain of x_1 and x_2 is $\{-2, -1, 0, 1, 2\}$. During constraint propagation, the constraint solver checks each constraint in turn as follows:

(1) Checking $x_1 - x_2 \geq 0$ removes no values from any domain.

(2) Checking $x_1 + x_2 \geq 0$ again removes no values from any domain.

(3) Checking $x_1 \geq 0$ removes the values -2 and -1 from the domain of x_1.

Note that $x_1 \geq 0$ is logically redundant with respect to $x_1 - x_2 \geq 0 \wedge x_1 + x_2 \geq 0$. Clearly it is not propagation redundant. ■

Depending on the order in which constraints are checked, propagation redundant constraints may or may not remove values from the domain (as illustrated above). Hence there certainly is a runtime cost associated with propagation redundant constraints. Removing propagation redundant constraints leads us to exactly the same domains after constraint propagation, but with significantly less cost for the propagation, as we shall see later in the experiments.

An important source of logically redundant constraints is in *redundant modeling* [Cheng et al. 1999]. A problem can be modeled differently from two viewpoints using two different sets of variables. By connecting the two different models with *channeling constraints*, which relate valuations in the two different models, stronger propagation behavior can be achieved in the combined model. However, the additional variables and constraints impose extra computation overhead. Given each model is complete and only admits the solutions of the problem then each model is logically redundant with respect to the other model plus the channeling constraints. In many cases, some of the constraints are also propagation redundant with respect to the other constraints in the combined model. By reasoning about propagation redundancy, we can improve redundant modeling by just keeping the constraints which give beneficial *new* propagation.

In this paper, we introduce the notion of *restrictive and unrestrictive channel functions* to characterize channeling constraints. We study the propagation behavior of constraints based on the notion of *propagation rules*, which capture each possible propagation by a constraint. This allows us to systematically determine if a propagator is redundant with respect to the propagators of a set of constraints through simple implication tests. We give general theorems for proving propagation redundancy of constraints involved in redundant models.

We focus on propagators that perform (the combination of) two popular propagation techniques, namely *domain propagation* [Van Hentenryck et al. 1998] and *set bounds propagation* [Gervet 1997], including global constraints that implement these approaches (see Example 26). The underlying machinery we use can express any propagators that only deal with integer domains and set bounds, and some of our results are directly applicable to such propagators. Although we do not consider stronger set based propagators that reason more about cardinalities [Azevedo and Barahona 2000; Müller 2001]), we can understand stronger cardinality reasoning as additional constraints using implicit cardinality variables. Hence, we can model common cardinality reasoning using additional propagators.

We illustrate, on problems from CSPLib (`http://www.csplib.org/`), how detecting and removing propagation redundant constraints can significantly speed up solving behavior. This paper is a revised and extended version of our earlier work [Choi et al. 2003a; 2003b].

The remainder of the paper is organized as follows. In Section 2, we introduce propagation-based constraint solving and *propagation rules*, a way of enumerating the different propagation behaviors of a propagator. In Section 3, we give theorems that are useful in determining propagation redundant constraints. In Section 4, we define a broad form of channeling constraints that are covered by our approach. In Section 5, we give theorems that allow us to show which constraints in a redundant model are not causing extra propagation and can be removed. In Section 6, we give experimental results showing the benefits of detecting and removing propagation redundant constraints. In Section 7, we discuss related work. In Section 8, we summarize our contributions and shed light on future directions of research.

2. BACKGROUND

In this paper, we consider integer and set constraint solving with constraint propagation and tree search. In an abuse of notation, we refer to arithmetic constraints over Boolean variables as Boolean constraints, they are often called pseudo-Boolean constraints. Hence, Boolean constraint solving is considered as a special case of integer constraint solving. Our notations, although different from the conventional CSP literatures, allow us to express the theoretical framework in a simpler manner.

2.1 Variables and Domains

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

Each variable is associated with a finite set of possible values, defined by the domain of the CSP. 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. A *singleton* domain D is such that $|D(v)| = 1$ for all $v \in \mathcal{V}$. 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 $D_1(v) \subseteq D_2(v)$ for all v. A domain D_1 is equal to a domain D_2, denoted $D_1 = D_2$, if $D_1(v) = D_2(v)$ for all v. 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 $D \sqsubseteq D_{init}$. We also use *range* notation whenever possible: $[\, l \,..\, u \,]$ denotes the set $\{d \mid l \leq d \leq u\}$ when l and u are integers, while $[\, L \,..\, U \,]$ denotes the set of sets of integers $\{A \mid L \subseteq A \subseteq U\}$ when L and U are sets of integers.

2.2 Valuations, Infima and Suprema

A *valuation* θ is a mapping of integer variables ($x_i \in \mathcal{V}_I$) to integer values and set variables ($S_i \in \mathcal{V}_S$) to sets of integer values, written

$$\{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n, S_1 \mapsto A_1, \ldots, S_m \mapsto A_m\}$$

where $d_i \in D(x_i)$ and $A_j \in D(S_j)$. Let *vars* be the function that returns the set of variables appearing in an expression, constraint or valuation. Given an expression e, $\theta(e)$ is obtained by replacing each $v \in \mathit{vars}(e)$ by $\theta(v)$ and calculating the value of the resulting variable free expression. 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 $v_i \in \mathit{vars}(\theta)$. The *projection* of a valuation θ onto variables V, denoted $\theta|_V$ is the valuation $\{v \mapsto \theta(x) \mid v \in (V \cap \mathit{vars}(\theta))\}$.

Define the *infimum* and *supremum* of an expression e with respect to a domain D as $\inf_D e = \inf\{\theta(e) \mid \theta \in D\}$ and $\sup_D e = \sup\{\theta(e) \mid \theta \in D\}$. The ordering $\preceq$ used by inf and sup depends on the type of the expression. If e has integer type then $d_1 \preceq d_2$ iff $d_1 \leq d_2$, while if e has set of integer type then $d_1 \preceq d_2$ iff $d_1 \subseteq d_2$. Note that these values may not exist for arbitrary domains and set of integer type expressions. Later we shall restrict ourselves to domains and expression where infimum and supremum always do exist.

2.3 Constraints and CSPs

A constraint places restriction on the allowable values for a set of variables and is usually written in well understood mathematical syntax. More formally, a *constraint* c is a relation expressed using the available function and relation symbols in a specific constraint language. For the purpose of this paper, we assume the usual (integer) interpretation of arithmetic constraints, set operators such as $\in$ and $\subseteq$, and logical operators such as $\neg$, $\wedge$, $\vee$, $\Rightarrow$, and $\Leftrightarrow$. We define

$$\mathit{solns}(c) = \{\theta \mid \mathit{vars}(\theta) = \mathit{vars}(c) \wedge \models_\theta c\},$$

that is the set of θ that make the constraint c hold true. We call $\mathit{solns}(c)$ the *solutions* of c. In some cases, constraints can also be defined directly by giving the set (or table) $\mathit{solns}(c)$. We sometimes treat an integer constraint c as an expression with value 1 if true and 0 if false. We can understand a domain D as a constraint in the obvious way,

$$D \quad \leftrightarrow \quad \bigwedge_{v \in \mathcal{V}} \bigvee_{d \in D(v)} v = d.$$

A CSP consists of a set of constraints read as conjunction. A solution to a CSP is a valuation θ that makes each constraint of a CSP holds true, i.e. $\theta|_{\mathit{vars}(c)} \in \mathit{solns}(c)$ for all constraint c of a CSP. A constraint c is *logically redundant* with respect to a constraint c' if $\models c' \rightarrow c$, that is c holds whenever c' holds. Adding logically redundant constraints to a CSP does not change the solutions of a CSP.

2.4 Propagators and Propagation Solvers

In the context of propagation-based constraint solving, a constraint specifies a propagator, which gives the basic units of propagation. 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

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

This is a weak restriction since for example, the identity propagator is correct for all constraints c. We assume that a propagator f for a constraint c is *checking*, that is, if D is a singleton domain, then $f(D) = D$ iff there exists $\theta \in D$ and $\theta \in \mathit{solns}(c)$. A checking propagator correctly determines the satisfiability of the constraint c for singleton domains.

A *propagation solver* for a set of propagators F and current domain D, $\mathit{solv}(F, D)$, repeatedly applies all the propagators in F starting from domain D until there is no further change in resulting domain. $\mathit{solv}(F, D)$ is the largest domain $D' \sqsubseteq D$ which is a fixpoint (i.e. $f(D') = D'$) for all $f \in F$. In other words, $\mathit{solv}(F, D)$ returns a new domain defined by

$$\begin{aligned} \mathit{iter}(F, D) &= \bigsqcap_{f \in F} f(D), \\ \mathit{solv}(F, D) &= \mathrm{gfp}(\lambda d.\mathit{iter}(F, d))(D). \end{aligned}$$

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

2.5 Domain and Set Bounds Propagators

Propagators are often (but not always) linked to implementing some notion of *local consistency*. The most well studied consistency notion is *arc consistency* [Mackworth 1977] which ensures that for each binary constraint, every value in the domain of the first variable, has a supporting value in the domain of the second variable which satisfied the constraint. Arc consistency can be naturally extended to constraints of more than two variables. This extension has been called *generalized arc consistency* [Mohr and Masini 1988], as well as *domain consistency* [Van Hentenryck et al. 1998] (which is the terminology we will use), and *hyper-arc consistency* [Marriott and Stuckey 1998].

A domain D is *domain consistent* 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 $\theta \in D \wedge \theta \in \mathit{solns}(c) \rightarrow \theta \in D'$.

Definition 1. Define the *domain propagator* for a constraint c as

$$\mathit{dom}(c)(D)(v) = \begin{cases} \{\theta(v) \mid \theta \in D \wedge \theta \in \mathit{solns}(c)\} & \text{where } v \in \mathit{vars}(c) \\ D(v) & \text{otherwise.} \end{cases}$$

Note that $\mathit{dom}(c)(D)$ makes D domain consistent for c.

Example 3. 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 of c are:

$$\{x_1 \mapsto 3, x_2 \mapsto 1, x_3 \mapsto 0\}, \quad \{x_1 \mapsto 5, x_2 \mapsto 0, x_3 \mapsto 1\}, \quad \{x_1 \mapsto 6, x_2 \mapsto 2, x_3 \mapsto 0\}.$$

Hence, $\mathit{dom}(c)(D) = D'$ where $D'(x_1) = \{3, 5, 6\}$, $D'(x_2) = \{0, 1, 2\}$, and $D'(x_3) = \{0, 1\}$. Clearly, D' is domain consistent with respect to c. ■

Set bounds propagation [Gervet 1997] 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. The set bounds propagator returns the smallest set range which includes the result returned by the domain propagator.

Definition 2. Define the *set bounds propagator* for a constraint c where $vars(c) \subseteq \mathcal{V}_S$ as

$$sb(c)(D)(v) = \begin{cases} [\,\cap(dom(c)(D)(v)) \mathinner{..} \cup(dom(c)(D)(v))\,] & \text{where } v \in vars(c) \\ D(v) & \text{otherwise.} \end{cases}$$

Example 4. Consider the constraint $c \equiv S_1 \subseteq S2$. Suppose the domain D where $D(S_1) = [\,\{1\} \mathinner{..} \{1,2,3,4\}\,]$, $D(S_2) = [\,\emptyset \mathinner{..} \{1,2,3\}\,]$. Then, $D' = sb(c)(D)$ where $D'(S_1) = D'(S_2) = [\,\{1\} \mathinner{..} \{1,2,3\}\,]$. ■

A constraint can involve both integer and set variables. In such case, we use domain propagation for the integer variables and set bounds propagation for the set variables.

Definition 3. Define the *domain and set bounds propagator* $dsb(c)$ for a constraint c as:

$$dsb(c)(D)(v) = \begin{cases} sb(c)(D)(v) & \text{where } v \in vars(c) \cap \mathcal{V}_S \\ dom(c)(D)(v) & \text{otherwise.} \end{cases}$$

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.

Example 5. Consider the constraint $c \equiv |S| = x$. Suppose $D(x) = \{2\}$ and $D(S) = [\,\emptyset \mathinner{..} \{1,5,8\}\,]$. The solutions of c are:

$$\{x \mapsto 2, S \mapsto \{1,5\}\}, \quad \{x \mapsto 2, S \mapsto \{1,8\}\}, \quad \{x \mapsto 2, S \mapsto \{5,8\}\}.$$

Hence, applying the domain propagator, $D' = dom(|S| = x)(D)$, gives $D'(S) = \{\{1,5\},\{1,8\},\{5,8\}\}$. The domain and set bounds propagator instead determines $dsb(c)(D) = D$ since $\cap\{\{1,5\},\{1,8\},\{5,8\}\} = \emptyset$ and $\cup\{\{1,5\},\{1,8\},\{5,8\}\} = \{1,5,8\}$. ■

2.6 Atomic Constraints and Propagation Rules

An *atomic constraint* represents the basic changes in domain that occur during propagation. For integer variables, the atomic constraints represent the elimination of values from an integer domain, i.e. $x_i \neq d$ or $x_i = d$ where $x_i \in \mathcal{V}_I$ and d is an integer.[3] For set variables, the atomic constraints represent the addition of a value to a lower bound set of integer or the removal of a value from an upper bound set of integer, i.e. $d \in S_i$ or $d \notin S_i$ where d is an integer and $S_i \in \mathcal{V}_S$.

[3]Atomic constraints of the form $x_i = d$ are not strictly necessary for propagation rules. They are equivalent to removing all other values from the domain. However, they would become useful in the later parts of the paper.

Definition 4. Define a *propagation rule* as $C \rightarrowtail c$ where C is a conjunction of *atomic constraints*, and c is a single atomic constraint such that $\not\models C \rightarrow c$.

For notational convenience we shall write extended rules $C \rightarrowtail C'$ where C' is a conjunction of atomic constraints as a shorthand for a set of rules $\{C \rightarrowtail c \mid c \in C'\}$. A propagation rule $C \rightarrowtail c$ defines a propagator (for which we use the same notation) in the obvious way.

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

In another word, $C \rightarrowtail c$ defines a propagator that removes values from D based on c only when D implies C. We can characterize an arbitrary propagator f in terms of the propagation rules that it implements.

Definition 5. A propagator f *implements* a propagation rule $C \rightarrowtail c$ iff

$$\models D \rightarrow C \text{ implies } \models f(D) \rightarrow c$$

for all $D \sqsubseteq D_{init}$.

Example 6. The propagator $f \equiv \mathit{dsb}(x_1 \neq x_2)$ for $D_{init}(x_1) = D_{init}(x_2) = \{1, 2, 3\}$ implements the rules

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

■

Example 7. The propagator $f \equiv \mathit{dsb}(S \subseteq T)$ for $D_{init}(S) = D_{init}(T) = \{\emptyset \ldots \{1, 2\}\}$. implements rules

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

■

Let Φ_f be the set of all possible rules implemented by f. This definition of f is often unreasonably large. In order to reason more effectively about propagation rules for a given propagator, we need to have a minimal representation.[4]

Definition 6. A set of propagation rules implemented by f, $\Pi_f \subseteq \Phi_f$, is *minimal* iff

— $\mathit{solv}(\Pi_f, D) = \mathit{solv}(\Phi_f, D)$, and

— there does not exist a $\Pi'_f \subset \Pi_f$ such that $\mathit{solv}(\Pi'_f, D) = \mathit{solv}(\Phi_f, D)$ for all $D \sqsubseteq D_{init}$.

That is, all propagation caused by f is also caused by the Π_f. Notice that Π_f is not unique.

[4]Both Brand [2003] and Abdennadher and Rigotti [2002] give effective methods for creating minimal representations of any constraints in terms of propagation rules.

Example 8. Consider the Boolean constraint $c \equiv z_1 = z_2 = z_3$ where $D_{init}(z_1) = D_{init}(z_2) = D_{init}(z_3) = [\,0\,..\,1\,]$. A minimal set of propagation rules implemented by $dsb(c)$ consists of the rules:

$$\begin{array}{rl@{\qquad}rl}
(r1) & z_1 = 1 \rightarrowtail z_2 = 1 & (r2) & z_1 = 0 \rightarrowtail z_2 = 0 \\
(r3) & z_2 = 1 \rightarrowtail z_3 = 1 & (r4) & z_2 = 0 \rightarrowtail z_3 = 0 \\
(r5) & z_3 = 1 \rightarrowtail z_1 = 1 & (r6) & z_3 = 0 \rightarrowtail z_1 = 0
\end{array}$$

Another minimal set of propagation rules implemented by $dsb(c)$ consists of the rules:

$$\begin{array}{rl@{\qquad}rl}
(r7) & z_1 = 1 \rightarrowtail z_3 = 1 & (r8) & z_1 = 0 \rightarrowtail z_3 = 0 \\
(r9) & z_2 = 1 \rightarrowtail z_1 = 1 & (r10) & z_2 = 0 \rightarrowtail z_1 = 0 \\
(r11) & z_3 = 1 \rightarrowtail z_2 = 1 & (r12) & z_3 = 0 \rightarrowtail z_2 = 0
\end{array}$$

Note that propagation rules for constraints with Boolean domain $\{0, 1\}$ can be represented using only atomic constraints involving equations since $\models D_{init} \rightarrow ((z = b) \leftrightarrow (z \neq (1 - b)))$ for $b \in \{0, 1\}$. ■

3. PROPAGATION REDUNDANT CONSTRAINTS

We shall be interested in reasoning about redundancy with respect to sets of 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 $F \gg \{f\}$. Our main aim is to discover and eliminate propagation redundant constraints and/or propagators. Before we can determine propagation redundant constraints, we need to establish some theorems.

The interest in characterizing a propagator in terms of the propagation rules is revealed by the following lemma. The propagation rules implemented by $dsb(c)$ of constraint c are exactly those $C \rightarrowtail c'$ where c implies $C \rightarrow c'$.

LEMMA 1. *Given a constraint c, $dsb(c)$ implements $C \rightarrowtail c'$ iff*

$$\models (D_{init} \wedge c) \rightarrow (C \rightarrow c').$$

PROOF. See the appendix. □

Lemma 1 enables us to relate logical redundancy of constraints with propagation redundancy of the propagators. A constraint c_2 that is logically redundant with respect to constraint c_1, is also propagation redundant with respect to c_1.

THEOREM 2. *Given constraints c_1 and c_2,*

$$\textit{if } \models (D_{init} \wedge c_1) \rightarrow c_2 \textit{ then } \{dsb(c_1)\} \gg \{dsb(c_2)\}$$

PROOF. Follows immediately from Lemma 1. □

Example 9. Consider the constraints $c_1 \equiv x_2 = x_1 + 2$ and $c_2 \equiv x_1 \neq x_2$ for $D_{init}(x_1) = D_{init}(x_2) = \{0, \ldots, 5\}$. Clearly, $\models D_{init} \wedge c_1 \rightarrow c_2$, the condition of Theorem 2 holds and we know that $dsb(c_2)$ is propagation redundant w.r.t. $dsb(c_1)$. For example, for $D \subseteq D_{init}$ where $D(x_1) = \{2\}$ and $D(x_2) = \{2, \ldots, 5\}$, we have $dsb(c_1)(D) \sqsubseteq dsb(c_2)(D)$ where $dsb(c_1)(D)(x_2) = \{4\}$ and $dsb(c_2)(D)(x_2) = \{3, \ldots, 5\}$. ■

Typically though a logically redundant constraint is made logically redundant by a *conjunction* of other constraints. However, 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.

LEMMA 3. *Given constraints c_1 and c_2, $\{dsb(c_1 \wedge c_2)\} \gg \{dsb(c_1), dsb(c_2)\}$.*

PROOF. Suppose to the contrary that there exists a variable $y \in vars(c_1 \wedge c_2)$ such that

$$solv(\{dsb(c_1 \wedge c_2)\}, D)(y) \not\subseteq solv(\{dsb(c_1), dsb(c_2)\}, D)(y)$$

for certain $D \sqsubseteq D_{init}$. Assume $y \in \mathcal{V}_I$. then there exists an integer $d \in D(y)$ such that $d \in dsb(c_1 \wedge c_2)(D)(y)$ and $\{dsb(c_1), dsb(c_2)\}$ eliminates d from y. By definition of propagation solver, there can be no solutions θ which satisfies $c_1 \wedge c_2$ in D where $\theta(y) = d$. By the definition, $\{dsb(c_1 \wedge c_2)\}$ must also eliminate d from y. Hence, $d \notin dsb(c_1 \wedge c_2)(D)(y)$, contrary to the hypothesis. Similar arguments apply for the case $y \in \mathcal{V}_S$. □

If a constraint c is logically redundant w.r.t. a conjunction of constraints c_1 and c_2, then $\{dsb(c)\}$ is propagation redundant w.r.t. $\{dsb(c_1 \wedge c_2)\}$ using Theorem 2. However, constraint programming system normally implements a *separate* propagator for each individual constraint. Because of Lemma 3, $\{dsb(c)\}$ is *not necessarily* propagation redundant w.r.t. the propagators of the individual constraints collectively, i.e.$\{dsb(c_1), dsb(c_2)\}$. Hence, it is difficult (in general) to determine whether a constraint that is logically redundant with respect to a conjunction of constraints, is propagation redundant or not. Interestingly, there is a case where propagation of a conjunction of constraints is equivalent to propagation on the individual conjuncts.

THEOREM 4. *If c_1 and c_2 are two constraints sharing at most one integer variable, $x \in \mathcal{V}_I$, then $\{dsb(c_1), dsb(c_2)\} \approx \{dsb(c_1 \wedge c_2)\}$.*

PROOF. See the appendix. □

Example 10. Consider again the integer constraints of Example 1, $c_1 \equiv x_1 \geq x_2$, $c_2 \equiv x_2 \geq x_3$, and $c_3 \equiv x_1 \geq x_3$, where $D_{init}(x_1) = [-2\,..\,1]$ and $D_{init}(x_2) = D_{init}(x_3) = [-2\,..\,2]$. It is clear that $\models D_{init} \wedge c_1 \wedge c_2 \rightarrow c_3$. By Theorem 2, we have $\{dsb(c_1 \wedge c_2)\} \gg \{dsb(c_3)\}$. Note that c_1 and c_2 share only one integer variable x_2. By Theorem 4, we have $\{dsb(c_1), dsb(c_2)\} \approx \{dsb(c_1 \wedge c_2)\}$. Hence, we show that $\{dsb(c_3)\}$ is propagation redundant w.r.t. $\{dsb(c_1), dsb(c_2)\}$ as demonstrated in Example 1.

Note that for Example 2 the constraints $x_1 - x_2 \geq 0$ and $x_1 + x_2 \geq 0$ share more than one variable, hence $\{dsb(x_1 - x_2 \geq 0 \wedge x_1 + x_2 \geq 0)\} \not\approx \{dsb(x_1 - x_2 \geq 0), dsb(x_1 + x_2 \geq 0)\}$. Thus, while $x_1 \geq 0$ is logically redundant w.r.t $x_1 - x_2 \geq 0 \wedge x_1 + x_2 \geq 0$, it is not propagation redundant. ■

Note that Theorem 4 does *not* hold when the single variable shared is a *set* variable, because we only apply set bounds propagation. If we did use *set domain* propagators the result readily extends to the case where a single shared variable is a set variable.

Example 11 *(Counter-example).* Consider the constraints $c_1 \equiv S \in \{\{1\}, \{2, 3\}\}$ and $c_2 \equiv S \in \{\{2\}, \{1, 3\}\}$ where $D(S) = \{\emptyset..\{1, 2, 3\}\}$. Now, $dsb(c_1)(D) = dsb(c_2)(D) = D$, but $dsb(c_1 \wedge c_2)(D)$ is a false domain since $c_1 \wedge c_2$ is unsatisfiable. Hence, Theorem 4 does *not* hold when the shared variable is a set variable and we use set bounds propagators.

However, if we use *set domain* propagators, then $dom(c_1)(D) = D_1$ where $D_1(S) = \{\{1\}, \{2, 3\}\}$ and $dom(c_2)(D_1)(S) = \emptyset$. Hence, Theorem 4 holds when the shared variable is a set variable and we use set domain propagators. ■

4. CHANNELING CONSTRAINTS

Redundant modeling [Cheng et al. 1999] models a problem from more than one viewpoint. By joining two models using channeling constraints, we can get the advantage of both sources of propagation.

Assume we have one model of the problem M_X using variables X, and another model M_Y using disjoint variables Y. Channeling constraints can be used to join these two models together by relating X and Y. There is no real agreement, as yet, as to precisely what channeling constraints are. For the purposes of our theorems we define a channeling constraint 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. We extend channel functions to map conjunctions of atomic constraints in the obvious way,

$$\diamondsuit(c_1 \wedge \cdots \wedge c_n) = \diamondsuit(c_1) \wedge \cdots \wedge \diamondsuit(c_n)$$

where $c_1, \ldots, c_n$ are atomic constraints.

Definition 7. A *channeling constraint* (or simply *channel*) $C_\diamondsuit$ is the constraint

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

Definition 8. 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 that 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.

4.1 Permutation Channels

A common form of redundant modeling is when we consider two viewpoints to a *permutation problem* [Geelen 1992]. In a permutation problem, the objective is to find a bipartite matching between two sets of objects $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$ satisfying all other problem specific constraints. Generally, we can model a permutation problem from two different viewpoints. In the first viewpoint, we assign objects from B to A. We use the set of variables $X = \{x_1, \ldots, x_n\}$ to denote objects in A, and the domain $D(x_i) = \{1, \ldots, n\}$, for all $1 \leq i \leq n$ to denote objects in B. The second viewpoint swaps the role between A and B, i.e. assign objects from A to B. We use the set of variables $Y = \{y_1, \ldots, y_n\}$ to denote objects in B, and the domain $D(y_j) = \{1, \ldots, n\}$, for all $1 \leq j \leq n$ to denote objects in A.

The *permutation channel function* $\bowtie$ is defined as $\bowtie(x_i = j) = (y_j = i)$ and $\bowtie(x_i \neq j) = (y_j \neq i)$ for all $1 \leq i, j \leq n$. The *permutation channel* $C_{\bowtie}$ is equivalent to the conjunction of constraints

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} (x_i = j \Leftrightarrow y_j = i)$$

Example 12. **Langford's Problem** The problem "prob024" in CSPLib is an example of permutation problem. The problem is to find an $(m \times n)$-digit sequence which includes the digits 1 to n, with each digit occurring m times. There is one digit between any consecutive pair of the digit 1, two digits between any consecutive pair of the digit 2, ..., and n digits between any consecutive pair of the digit n.

Smith [2000] suggests two ways to model the Langford's problem. We use the (3×9) instance to illustrate the two models. In the first model, M_X, we use 27 variables $X = \{x_1, \ldots, x_{27}\}$, which we can think of as $1_1, 1_2, 1_3, 2_1, \ldots, 9_2, 9_3$. Here, 1_1 represents the first digit 1 in the sequence, 1_2 represents the second digit 1, and so on. The initial domain of these variables, $D_{init}(x_i) = \{1, \ldots, 27\}$ for $1 \leq i \leq 27$, represents the positions of the digit x_i in the sequence. We enlist the constraints of Smith's model as follows:

— (LX1) disequality constraints: $\forall 1 \leq i < j \leq 27.\ \ x_i \neq x_j$

— (LX2.1) separation constraints: $\forall 1 \leq i \leq 9.\ \ x_{3i-1} = x_{3i-2} + (i+1)$

— (LX2.2) separation constraints: $\forall 1 \leq i \leq 9.\ \ x_{3i} = x_{3i-1} + (i+1)$

In the second model, M_Y, we again use 27 variables $Y = \{y_1, \ldots, y_{27}\}$ to represent each position in the sequence. The initial domain of these variables, $D_{init}(y_i) = \{1, \ldots, 27\}$ for $1 \leq i \leq 27$, corresponds to the digits $1_1, 1_2, 1_3, 2_1, \ldots, 9_2, 9_3$ in position y_i of the sequence. The constraints are:

— (LY1) disequality constraints: $\forall 1 \leq i < j \leq 27.\ \ y_i \neq y_j$

— (LY2.1) separation constraints: $\forall 1 \leq i \leq 9. \forall 1 \leq j \leq 27 - 2(i+1).\ \ y_j = 3i - 2 \Leftrightarrow y_{j+(i+1)} = 3i - 1$

— (LY2.2) separation constraints: $\forall 1 \leq i \leq 9. \forall 1 \leq j \leq 27 - 2(i+1).\ \ y_j = 3i - 2 \Leftrightarrow y_{j+2(i+1)} = 3i$

— (LY3) separation constraints: $\forall 1 \leq i \leq 9. \forall (28 - 2(i+1)) \leq j \leq 27.\ \ y_j \neq 3i - 2$

The permutation channel for the two models is simply $x_i = j \Leftrightarrow y_j = i$ for all $1 \leq i, j \leq 27$. ■

Example 13. **All Interval Series Problem** The problem "prob007" in CSPLib is from musical composition. The problem is to find a permutation of n numbers from 1 to n, such that the differences between adjacent numbers form a permutation from 1 to $n - 1$. We give two ways to model the problem. The first model derives from the model suggested by Puget and Régin [2001], and the the second model slightly modifies the model suggested by Choi and Lee [2002].

The first model, M_X, consists of n variables, $X = \{x_1, \ldots, x_n\}$. Each x_i denotes the number in position i, and $D_{init}(x_i) = [\,1\,..\,n\,]$ for $1 \leq i \leq n$. We introduce auxiliary variables, $\{u_1, \ldots, u_{n-1}\}$ that denote the difference between adjacent numbers, where $D_{init}(u_i) = [\,1\,..\,n-1\,]$ for $1 \leq i \leq n-1$. The constraints are:

—(IX1.1) disequality constraints: $\forall 1 \leq i < j \leq n.\ \ x_i \neq x_j$

—(IX1.2) disequality constraints: $\forall 1 \leq i < j \leq n-1.\ \ u_i \neq u_j$

— (IX2) interval constraints: $\forall 1 \leq i \leq n-1.\ \ u_i = |x_i - x_{i+1}|$

The second model, M_Y, also consists of n variables, $Y = \{y_1, \ldots, y_n\}$. Each y_i denotes the position for the number i, and $D_{init}(y_i) = [\,1\,..\,n\,]$ for $1 \leq i \leq n$. The auxiliary variables $\{v_1, \ldots, v_{n-1}\}$ denote the position where the difference value of 1 to $n-1$ belongs, and $D_{init}(v_i) = [\,1\,..\,n-1\,]$ for $1 \leq i \leq n-1$. The constraints are:

— (IY1.1) disequality constraints: $\forall 1 \leq i < j \leq n.\ \ y_i \neq y_j$

— (IY1.2) disequality constraints: $\forall 1 \leq i < j \leq n-1.\ \ v_i \neq v_j$

— (IY2.1) interval constraints: $\forall 1 \leq i < j \leq n.\ \ (y_i - y_j = 1) \Rightarrow (v_{j-i} = y_j)$

— (IY2.2) interval constraints:$\forall 1 \leq i < j \leq n.\ \ (y_j - y_i = 1) \Rightarrow (v_{j-i} = y_i)$

The (IY2.1) and (IY2.2) constraints 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, observe the fact that only the numbers 1 and n can give us the difference of $n-1$. Therefore, we can add the following redundant constraints:

$$\text{(IY3)}:\ (|y_1 - y_n| = 1) \wedge (v_{n-1} = \min(y_1, y_n)),$$

which requires y_1 and y_n 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 $x_i = j \Leftrightarrow y_j = i$ for all $1 \leq i, j \leq n$ and $u_i = j \Leftrightarrow v_j = i$ for all $1 \leq i, j \leq n-1$. ■

4.2 Boolean Channels

Another common form of redundant modeling is when we give both an integer and Boolean models. Suppose we have an integer model using the integer variables $X = \{x_1, \ldots, x_n\}$ and the domain $D_{init}(x_i) = [\,1\,..\,k\,]$. We can have a corresponding Boolean model using the Boolean variables $Z = \{z_{ij} \mid 1 \leq i \leq n, 1 \leq j \leq k\}$. Each variable z_{ij} encodes the proposition that $x_i = j$.

The *Boolean channel function* $\triangle$ is defined as $\triangle(x_i = j) = (z_{ij} = 1)$ and $\triangle(x_i \neq j) = (z_{ij} = 0)$ for all $1 \leq i \leq n, 1 \leq j \leq k$. 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

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{k} (x_i = j \Leftrightarrow z_{ij} = 1)$$

Example 14. **n-Queens Problem** This well-known problem is to place n queens on an $n \times n$ chess board so that no two queens can attack each other. There are two common ways to model this problem, i.e., an integer model and a Boolean model.

The integer model, M_X, consists of n variables, $X = \{x_1, \ldots, x_n\}$. Each x_i denotes the column position of the queen on row i, and $D_{init}(x_i) = \{1, \ldots, n\}$, for $1 \leq i \leq n$. The constraints are:

— (QX1) column constraints: $\forall 1 \leq i < j \leq n.\ \ x_i \neq x_j$

— (QX2.1) diagonal constraints: $\forall 1 \leq i < j \leq n.\ \ x_i - i \neq x_j - j$

— (QX2.2) diagonal constraints: $\forall 1 \leq i < j \leq n.\ \ x_i + i \neq x_j + j$

The Boolean model, M_Z, consists of $n \times n$ Boolean variables, $Z = \{z_{11}, \ldots, z_{1n}, \ldots, z_{n1}, \ldots, z_{nn}\}$. Each Boolean variable z_{ij} denotes whether we have a queen at row i column j or not. The constraints are:

— (QZ1) row constraints: $\forall 1 \leq i \leq n.\ \ \sum_{j=1}^{n} z_{ij} = 1$

— (QZ2) column constraints: $\forall 1 \leq j \leq n.\ \ \sum_{i=1}^{n} z_{ij} = 1$

— (QZ3.1) diagonal constraints: $\forall 0 \leq k \leq n - 1.\ \ \sum_{i=1}^{n-k} z_{i(i+k)} \leq 1$

— (QZ3.2) diagonal constraints: $\forall 1 \leq k \leq n - 1.\ \ \sum_{i=1}^{n-k} z_{(i+k)i} \leq 1$

— (QZ3.3) diagonal constraints: $\forall 0 \leq k \leq n - 1.\ \ \sum_{i=1}^{n-k} z_{i(n-i-k+1)} \leq 1$

— (QZ3.4) diagonal constraints: $\forall 1 \leq k \leq n - 1. \sum_{i=1}^{n-k} z_{(i+k)(n-i+1)} \leq 1$

We combine the two models using the Boolean channel $x_i = j \Leftrightarrow z_{ij} = 1$ for all $1 \leq i \leq n, 1 \leq j \leq k$. ■

4.3 Set Channels

Another common form of redundant modeling 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 permutation problem to where two or more integer variables can take the same value. Suppose the integer variables are $X = \{x_1, \ldots, x_n\}$, where $D_{init}(x_i) = [\,1\,..\,k\,]$ for all $1 \leq i \leq n$, and the set variables are $S = \{S_1, \ldots, S_k\}$ where $D_{init}(S_j) = [\,\emptyset\,..\,\{1, \ldots, n\}\,]$ for all $1 \leq j \leq k$.

The *set channel function* $\{\}$ is defined as $\{\}(x_i = j) = (i \in S_j)$ and $\{\}(x_i \neq j) = (i \notin S_j)$ for all $1 \leq i \leq n, 1 \leq j \leq k$. The *set channel* $C_{\{\}}$ is equivalent to

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{k} (x_i = j \Leftrightarrow i \in S_j)$$

Example 15. **Social Golfers Problem** The problem "prob010" in CSPLib is to arrange $n = g \times s$ players into g groups of s players each week, playing for w weeks, so that no two players play in the same group twice. Smith [2001] suggests two ways to model this problem.

In the first model we use variables $X = \{x_{lk} | 1 \leq l \leq n, 1 \leq k \leq w\}$ to denote the group which player l plays on week k, and $D_{init}(x_{lk}) = [\,1\,..\,g\,]$ for all $1 \leq l \leq n, 1 \leq k \leq w$.

The constraints of the problem are expressed as:

— (GX1) each group has s players: $\forall 1 \leq i \leq g. \forall 1 \leq k \leq w.\ \ \sum_{l=1}^{n}(x_{lk} = i) = s$

— (GX2) two players only play in the same group in one week:

$$\forall 1 \le k_1 < k_2 \le w. \forall 1 \le l_1 < l_2 \le n.\ \neg(x_{l_1 k_1} = x_{l_2 k_1} \wedge x_{l_1 k_2} = x_{l_2 k_2})$$

The second model uses set variables $S = \{S_{ik} | 1 \le i \le g, 1 \le k \le w\}$ to denote the set of players play in group i on week k. and $D_{init}(S_{ik}) = [\emptyset \mathinner{..} \{1, \ldots, n\}]$ for all $1 \le i \le g, 1 \le k \le w$. The constraints are expressed as:

— (GS1) no groups in the same week have a player in common:

$$\forall 1 \le k \le w. \forall 1 \le i_1 < i_2 \le g.\ S_{i_1 k} \cap S_{i_2 k} = \emptyset$$

— (GS2) each group has s players: $\forall 1 \le i \le g. \forall 1 \le k \le w.\ |S_{ik}| = s$

— (GS3) no different groups have more than one player in common:

$$\forall 1 \le i_1 \ne i_2 \le g. \forall 1 \le k_1 < k_2 \le w.\ |S_{i_1 k_1} \cap S_{i_2 k_2}| \le 1$$

We can use the set channels to combine the two models, $x_{lk} = i \Leftrightarrow l \in S_{ik}$ for all $1 \le l \le n, 1 \le k \le w, 1 \le i \le g$. ■

Example 16. **Balanced Academic Curriculum Problem** The problem, listed as "prob030" in CSPLib, is to design an academic curriculum aiming to balance the loads in each academic period. Following the description in Hnich *et al.* [2002], we can have both an 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 prerequisite 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. $u = \max\{l_j \mid 1 \le j \le n\}$. 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 $D_{init}(u) = D_{init}(l_j) = [0 \mathinner{..} \sum_{i=1}^{m} t_i]$ and $D_{init}(q_j) = [1 \mathinner{..} m]$.

We have the following constraints that are common to both models:

— (B1.1) load allowed per period: $\forall 1 \le j \le n.\ a \le l_j \le b$

— (B1.2) number of courses allowed per period: $\forall 1 \le j \le n.\ c \le q_j \le d$

We also add the following redundant constraints:

— (B2.1) all the credits must be fulfilled: $(\sum_{j=1}^{n} l_j) = (\sum_{i=1}^{m} t_i)$

— (B2.2) all the courses must be taken: $(\sum_{j=1}^{n} q_j) = m$

In the integer model, M_X, the variables $X = \{x_i | 1 \le i \le m\}$ represent the period to which course i is assigned and $D_{init}(x_i) = [1 \mathinner{..} n]$ for all $1 \le i \le m$. The constraints for the integer model M_X are:

— (BX1) l_j is the load taken in period j: $\forall 1 \le j \le n.\ (\sum_{i=1}^{m}((x_i = j) \times t_i)) = l_j$

— (BX2) q_j is the number of courses in period j: $\forall 1 \le j \le n.\ (\sum_{i=1}^{m}(x_i = j)) = q_j$

— (BX3) courses are taken respecting prerequisites: $\forall \langle i, j \rangle \in R.\ x_i < x_j$

c	$\bumpeq(c)$
$S_i = \emptyset$	$\{z_{ij} = 0 \mid 1 \leq j \leq k\}$
$S_a \subseteq S_b$	$\{z_{aj} \leq z_{bj} \mid 1 \leq j \leq k\}$
$S_a \cap S_b = \emptyset$	$\{z_{aj} + z_{bj} \leq 1 \mid 1 \leq j \leq k\}$
$S_a = S_b \cup S_c$	$\{z_{bj} \leq z_{aj} \mid 1 \leq j \leq k\} \cup$
	$\{z_{cj} \leq z_{aj} \mid 1 \leq j \leq k\} \cup$
	$\{z_{aj} \leq z_{bj} + z_{cj} \mid 1 \leq j \leq k\}$
$S_a = S_b \cap S_c$	$\{z_{aj} \leq z_{bj} \mid 1 \leq j \leq k\} \cup$
	$\{z_{aj} \leq z_{cj} \mid 1 \leq j \leq k\} \cup$
	$\{z_{bj} + z_{cj} \leq z_{aj} + 1 \mid 1 \leq j \leq k\}$
$S_a = S_b - S_c$	$\{z_{aj} \geq z_{bj} - z_{cj} \mid 1 \leq j \leq k\} \cup$
	$\{z_{aj} \leq z_{bj} \mid 1 \leq j \leq k\} \cup$
	$\{z_{aj} + z_{cj} \leq 1 \mid 1 \leq j \leq k\}$
$\lvert S_i \rvert = m$	$\{m = \sum_{j=1}^{k} z_{ij}\}$

Fig. 1. Mapping of Common Set Constraints to Boolean Constraints

In the set model, the set variables $S = \{S_j | 1 \leq j \leq n\}$ represent the set of courses assigned to period j and $D_{init}(S_j) = [\,\emptyset \,..\, \{1, \ldots, m\}\,]$ for all $1 \leq j \leq n$. The constraints for the set model M_S are:

— (BS1) No course is taken twice: $\forall 1 \leq i < j \leq n.\ \ S_i \cap S_j = \emptyset$

— (BS2) l_j is the load in period j: $\forall 1 \leq j \leq n.\ \ (\sum_{i \in S_j} t_i) = l_j$

— (BS3) q_j is the number or courses in period j: $\forall 1 \leq j \leq n.\ \ |S_j| = q_j$

— (BS4) courses are taken respecting prerequisites:

$$\forall \langle i, j \rangle \in R. \forall 1 \leq k \leq n-1. \forall 1 \leq k' \leq k.\ \ (i \in S_k) \Rightarrow (j \notin S_{k'})$$

We can use the set channels to combine the two models, $x_i = j \Leftrightarrow i \in S_j$ for all $1 \leq i \leq m, 1 \leq j \leq n$ ■

4.4 Channels between Set and Boolean Models

A very uncommon form of redundant modeling is when we give a set model and a Boolean version of this model. The reason it is uncommon is that there is no natural gain in expressiveness in moving to the Boolean model.

Suppose the set variables are $\{S_1, \ldots, S_n\}$. where $D_{init}(S_i) = [\,\emptyset \,..\, \{1, \ldots, k\}\,]$, and the Boolean variables are $z_{ij}, 1 \leq i \leq n, 1 \leq j \leq k$. The *set2bool channel function* $\bumpeq$ is defined as $\bumpeq(j \in S_i) = (z_{ij} = 1)$ and $\bumpeq(j \notin S_i) = (z_{ij} = 0)$. The *set2bool channel* $C_{\bumpeq}$ is equivalent to

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{k} (j \in S_i \Leftrightarrow z_{ij} = 1)$$

With the $\bumpeq$ channel, we can map common set constraints (c) to Boolean constraints $(\bumpeq(c))$ as given in Figure 1. We shall prove that set bounds propagation of set constraints (c) is equivalent to domain propagation for the corresponding Boolean constraints $(\bumpeq(c))$.

5. PROPAGATION REDUNDANT CONSTRAINTS IN REDUNDANT MODELING

In redundant modeling, each model is logically redundant with respect to the other model plus the channeling constraints. In general, the propagators defined for two viewpoints act in different ways and discover information at different stages in the search. However, we show two possibilities in which propagation caused by some constraints in one model can be made redundant by: (a) propagation induced from constraints in the other model through channels and (b) propagation of the channels themselves. For brevity, we shall concentrate on one model when stating some of the lemmas and theorems. The restrictions on the other model can be seen easily by examining the inverse channel function.

5.1 Propagation Redundancy Through Channels

In order to show that the propagation caused by some constraints in one model is subsumed by propagation induced from constraints in the other model through channels, we often need to break up the consideration of propagator into individual propagation rules. Therefore, we need the following lemma to ensure that the domain and set bounds propagator of a constraint is equivalent to the union of the propagation rules implemented by the propagator.

LEMMA 5. *Consider a minimal set of propagation rules, $\Pi_{dsb(c)}$, implemented by $dsb(c)$ for constraint c. Then $\{dsb(c)\} \approx \Pi_{dsb(c)}$.*

PROOF. See the appendix. $\square$

Next, we need to define formally the notion of subsumption.

Definition 9. A propagation rule $C_1 \rightarrowtail c_1$ *directly subsumes* a rule $C_2 \rightarrowtail c_2$ iff

$$\models (D_{init} \land C_2) \rightarrow C_1 \quad \text{and} \quad \models (D_{init} \land c_1) \rightarrow c_2.$$

A channel function enables us to map a propagation rule r from one model to the other model. If the mapped propagation rule is directly subsumed by another propagation rule r' in the other model, then the following lemma tells us that r is propagation redundant w.r.t. the channel propagator and r'.

LEMMA 6. *Let $C \rightarrowtail c$ be a propagation rule on Y variables, and $C' \rightarrowtail c'$ be a propagation rule on X variables. If $C' \rightarrowtail c'$ directly subsumes $\diamondsuit^{-1}(C) \rightarrowtail \diamondsuit^{-1}(c)$, then $(\{C' \rightarrowtail c'\} \cup F_{\diamondsuit}) \gg \{C \rightarrowtail c\}$.*

PROOF. Consider the case that $\models D \rightarrow C$ for all $D \sqsubseteq D_{init}$. Applying $F_{\diamondsuit}$ to D, $D_1 = solv(F_{\diamondsuit}, D)$, we have $\models D_1 \rightarrow \diamondsuit^{-1}(C)$ using the definition of $F_{\diamondsuit}$. By the condition of the lemma and Definition 9, we have that $\models D_1 \rightarrow C'$. By applying $C' \rightarrowtail c'$ to D_1, $D_2 = (C' \rightarrowtail c')(D_1)$, we have $\models D_2 \rightarrow c'$. By the condition of the lemma and Definition 9, we have $\models D_2 \rightarrow \diamondsuit^{-1}(c)$. Applying $F_{\diamondsuit}$ to D_2, $D_3 = solv(F_{\diamondsuit}, D_2)$, we have $\models D_3 \rightarrow c$ using the definition of $F_{\diamondsuit}$. Since we have show that $\models solv(\{C' \rightarrowtail c'\} \cup F_{\diamondsuit}, D) \rightarrow c$ for $\models D \rightarrow C$, $\{C' \rightarrowtail c'\} \cup F_{\diamondsuit}$ implements $C \rightarrowtail c$. By definition, $(\{C' \rightarrowtail c'\} \cup F_{\diamondsuit}) \gg \{C \rightarrowtail c\}$. $\square$

We can straightforwardly lift the results of Lemma 6 to talk about propagation rules that are directly subsumed by the domain and set bounds propagator for a constraint.

Theorem 7. *Let c_X be a constraint on X variables and $C \rightarrowtail c$ be a propagation rule on Y variables. If*

$$\models (D_{init} \wedge c_X \wedge \diamondsuit^{-1}(C)) \rightarrow \diamondsuit^{-1}(c),$$

then $\{dsb(c_X)\} \cup F_{\diamondsuit} \gg \{C \rightarrowtail c\}$.

Proof. By Lemma 5, we have that $dsb(c_X)$ implements the propagation rule $\diamondsuit^{-1}(C) \rightarrow \diamondsuit^{-1}(c)$. Hence by Lemma 6 the result holds. □

A corollary of Theorem 7 is that if every propagation rule in a minimal set of propagation rules implemented by $dsb(c_Y)$ is subsumed by $dsb(c_X)$ through the channel function, then $dsb(c_Y)$ is propagation redundant w.r.t. the channel propagator and $dsb(c_X)$.

Corollary 8. *Let c_X be a constraint on X variables, c_Y be a constraint on Y variables, and $\Pi_{dsb(c_Y)}$ be a minimal set of propagation rules implemented by $dsb(c_Y)$. If*

$$\models (D_{init} \wedge c_X \wedge \diamondsuit^{-1}(C)) \rightarrow \diamondsuit^{-1}(c) \textit{ for all } (C \rightarrowtail c) \in \Pi_{dsb(c_Y)},$$

then $\{dsb(c_X)\} \cup F_{\diamondsuit} \gg \{dsb(c_Y)\}$.

Example 17. Consider the (LY2.1) constraints of the Langford's Problem (Example 12),

$$c_Y \equiv y_j = 3i - 2 \Leftrightarrow y_{j+(i+1)} = 3i - 1$$

for all $1 \leq i \leq 9$ and $1 \leq j \leq 27 - 2(i + 1)$. A minimal set of propagation rules $\Pi_{dsb(c_Y)}$ for $dsb(c_Y)$ consists of the rules:

$$\begin{array}{lrcl}
(r1) & y_j = 3i-2 & \rightarrowtail & y_{j+(i+1)} = 3i-1 \\
(r2) & y_{j+(i+1)} = 3i-1 & \rightarrowtail & y_j = 3i-2 \\
(r3) & y_j \neq 3i-2 & \rightarrowtail & y_{j+(i+1)} \neq 3i-1 \\
(r4) & y_{j+(i+1)} \neq 3i-1 & \rightarrowtail & y_j \neq 3i-2
\end{array}$$

Using the channel function $\bowtie^{-1}$, the propagation rule (r1) is mapped to

$$x_{3i-2} = j \rightarrowtail x_{3i-1} = j + i + 1.$$

Now, it is straightforward to show that

$$\models (D_{init} \wedge c_X \wedge x_{3i-2} = j) \rightarrow x_{3i-1} = j + i + 1$$

where $c_X \equiv x_{3i-1} = x_{3i-2} + (i + 1)$ of (LX2.1). Similar arguments apply for the other propagation rules (r2), (r3) and (r4). Hence, using Corollary 8, $dsb(c_Y)$ is propagation redundant w.r.t $F_{\bowtie}$ and $dsb(c_X)$.

For the (LY2.2) constraints, $c'_Y \equiv y_j = 3i - 2 \Leftrightarrow y_{j+2(i+1)} = 3i$ where $1 \leq i \leq 9$ and $1 \leq j \leq 27 - 2(i + 1)$, we can similarly show that $dsb(c'_Y)$ is propagation redundant w.r.t. $F_{\bowtie}$ and $dsb(c_X \wedge c'_X)$ where $c'_X \equiv x_{3i} = x_{3i-1} + (i+1)$ of (LX2.2). Although model M_X does not include the propagator $dsb(c_X \wedge c'_X)$, we can still show propagation redundancy since $\{dsb(c_X), dsb(c'_X)\} \approx \{dsb(c_X \wedge c'_X)\}$ by Theorem 4.

Similar arguments apply for the (LY3) constraints $y_j \neq 3i - 2$, where $1 \leq i \leq 9$ and $(28 - 2(i + 1)) \leq j \leq 27$, is propagation redundant w.r.t. $C_{\bowtie}$ and c_X. ■

For brevity we shall introduce pseudo atomic constraints $x \leq d$ equivalent to the conjunction $x \neq d+1, \ldots, x \neq \sup_{D_{init}}(x)$ and $x \geq d$ equivalent to the conjunction $x \neq \inf_{D_{init}}(x), \ldots, x \neq d-1$, to discuss the next example.

Example 18. Consider the (BX2) constraints of the balanced academic curriculum problem (Example 16),

$$c_X \equiv (\sum_{i=1}^{m}(x_i = j)) = q_j$$

for all $1 \leq j \leq n$. A minimal set of propagation rules $\Pi_{dsb(c_X)}$ consists of the rules:

$$\begin{array}{lrcl}
(r1) & x_{i_1} = j \wedge \cdots \wedge x_{i_d} = j & \rightarrowtail & q_j \geq d \\
(r2) & x_{i_1} \neq j \wedge \cdots \wedge x_{i_{m-d}} \neq j & \rightarrowtail & q_j \leq d \\
(r3) & q_j \leq d \wedge x_{i_1} = j \wedge \cdots \wedge x_{i_d} = j & \rightarrowtail & x_k \neq j \\
(r4) & q_j \geq d \wedge x_{i_1} \neq j \wedge \cdots \wedge x_{i_{m-d}} \neq j & \rightarrowtail & x_l = j
\end{array}$$

$\forall d \in \{1, \ldots, m\}$, $\forall k \in (\{1, \ldots, m\} - K)$, and $\forall l \in (\{1, \ldots, m\} - L)$ where $K = \{i_1, \ldots, i_d\} \subseteq \{1, \ldots, m\}$ and $L = \{i_1, \ldots, i_{m-d}\} \subseteq \{1, \ldots, m\}$. All the atomic constraints involving q_j are mapped to themselves by the channel function $\{\}$ since q_j is shared by the two models, e.g. the propagation rule (r1) is mapped to:

$$i_1 \in S_j \wedge \cdots \wedge i_d \in S_j \rightarrowtail q_j \geq d$$

Now, it is straightforward to show that

$$\models (D_{init} \wedge c_S \wedge i_1 \in S_j \wedge \cdots \wedge i_d \in S_j) \rightarrow q_j \geq d$$

where $c_S \equiv |S_j| = q_j$ of (BS3). Similarly arguments apply for the other propagation rules (r2), (r3) and (r4). Hence, using Corollary 8, $dsb(c_X)$ is propagation redundant w.r.t. to $F_{\{\}}$ and $dsb(c_S)$.

Similar arguments apply to show that for constraint (BX1), $c'_X \equiv (\sum_{i=1}^{m}((x_i = j) \times t_i)) = l_j$ for all $1 \leq j \leq n$, $dsb(c'_X)$ is made propagation redundant by $F_{\{\}}$ and $dsb(c'_S)$ where $c'_S \equiv (\sum_{i \in S_j} t_i) = l_j$ of (BS2). ■

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 9. *Let c_Y be a constraint on Y variables and $\Pi_{dsb(c_Y)}$ be a minimal set of propagation rules implemented by $dsb(c_Y)$. If there exists a constraint c_r on X variables for each $(r \equiv (C \rightarrowtail c)) \in \Pi_{dsb(c_Y)}$ such that*

$$\models (D_{init} \wedge c_r \wedge \diamondsuit^{-1}(C)) \rightarrow \diamondsuit^{-1}(c),$$

then

$$\bigcup_{r \in \Pi_{dsb(c_Y)}} \{dsb(c_r)\} \cup F_{\diamondsuit} \gg \{dsb(c_Y)\}.$$

PROOF. The proof follows straightforwardly from Lemma 5 and Theorem 7. □

Example 19. Consider the (IY2.1) constraints of the all intervals series (Example 13),

$$c_Y \equiv (y_i - y_j = 1) \Rightarrow (v_{j-i} = y_j)$$

for all $1 \leq i < j \leq n$. A minimal set of propagation rules $\Pi_{dsb(c_Y)}$ for $dsb(c_Y)$ is of the forms,

$$\begin{array}{lrcl} (r1) & y_i = k+1 \wedge y_j = k & \rightarrowtail & v_{j-i} = k \\ (r2) & v_{j-i} \neq k \wedge y_j = k & \rightarrowtail & y_i \neq k+1 \\ (r3) & y_i = k+1 \wedge I & \rightarrowtail & y_j \neq k \end{array}$$

where I (of $r3$) is any conjunction of disequations on v_{j-i} and y_j, excluding $y_j \neq k$, ensuring that $v_{j-i} \neq y_j$. To apply Theorem 9, we look at each of the propagation rules:

— For the propagation rule (r1), we can show that

$$\models (D_{init} \wedge c_{r1} \wedge x_{k+1} = i \wedge x_k = j) \rightarrow (u_k = j - i)$$

where $c_{r1} \equiv (u_k = |x_k - x_{k+1}|)$ of (IX2).

— For the propagation rule (r2), we can show that

$$\models (D_{init} \wedge c_{r2} \wedge u_k \neq j - i \wedge x_k = j) \rightarrow (x_{k+1} \neq i).$$

where $c_{r2} \equiv (u_k = |x_k - x_{k+1}|)$ of (IX2).

— For the propagation rule (r3), I must contain $v_{j-i} \neq k$ since it does not contain $y_j \neq k$ and it must force the two to be different. We can show that

$$\models (D_{init} \wedge c_{r3} \wedge u_k \neq j - i \wedge x_{k+1} = i) \rightarrow (x_k \neq j).$$

where $c_{r3} \equiv (u_k = |x_k - x_{k+1}|)$ of (IX2).

Note that even though each example propagation rule (r1) to (r3) is made propagation redundant by $C_{\bowtie}$ and the same $u_k = |x_k - x_{k+1}|$ of (IX2), we indeed require a different constraint for each different value of k.

Similar arguments apply to show that the other (IY2.2) constraints $(y_j - y_i = 1) \Rightarrow (v_{j-i} = y_i)$ is propagation redundant w.r.t. $C_{\bowtie}$ and constraints of (IX2).

Note that the logically redundant constraint $(|y_1 - y_n| = 1) \wedge (v_{n-1} = \min(y_1, y_n))$ of (IY3) is not propagation redundant. ■

5.2 Propagation Redundancy Caused by Channels

The channels themselves may actually restrict the possible solutions in one or both models involved.

Definition 10. A channel function $\diamondsuit$ is *restrictive* (on the variables X) iff

$$\not\models D_{init} \rightarrow \exists Y \; C_{\diamondsuit}$$

that is not all valuations on X variables are extensible to solutions of $C_{\diamondsuit}$.

Example 20. The permutation channel functions $\bowtie$ 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 Boolean channel function $\triangle$ 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.

Similarly the set channel function $\{\}$ is unrestrictive while $\{\}^{-1}$ is restrictive. For example $S_1 = \{1\}, S_2 = \{1\}$ cannot be extended to a solution of $C_{\{\}}$ since it requires x_1 to be both 1 and 2.

The set2bool channel $\doteqdot$ is clearly unrestrictive in both directions. ■

5.2.1 *Restrictive Channel Functions.* Restrictive channel functions can themselves make constraints propagation redundant. Smith [2000] first observed that the permutation channel makes each of the disequations between variables in either model propagation redundant. Walsh [2001] proves this holds for other notions of consistency.

THEOREM 10 (WALSH [2001]). *$F_{\bowtie} \gg \{dsb(x_i \neq x_j)\}$ for all $1 \leq i < j \leq n$.* □

Example 21. Using Theorem 10, the permutation channel makes the following constraints propagation redundant: the (LX1) and (LY1) constraints of the Langford's Problem (Example 12); and the (IX1.1), (IX1.2), (IY1.1) and (IY1.2) constraints of the all intervals series (Example 13) ■

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 $\sum_{j=0}^{k} z_{ij} = 1$. It is enforced by the restrictive channel function $\triangle^{-1}$.

THEOREM 11. *$F_\triangle \gg \{dsb(\sum_{j=1}^{k} z_{ij} = 1)\}$ for all $1 \leq i \leq n$.*

PROOF. A minimal set of propagation rules for $dsb(\sum_{j=1}^{k} z_{ij} = 1)$ consist of the rules:

$$\begin{array}{lrl} (r1) & z_{ij} = 1 & \rightarrowtail\ z_{ij'} = 0, \text{ for all } j' \neq j \\ (r2) & z_{i1} = 0, \ldots, z_{i(j-1)} = 0, z_{i(j+1)} = 0, \ldots, z_{ik} = 0 & \rightarrowtail\ z_{ij} = 1. \end{array}$$

We show that $F_\triangle$ implements both (r1) and (r2).

— For the rule (r1), suppose $D \sqsubseteq D_{init}$ where $D(z_{ij}) = \{1\}$. Let $D_1 = solv(F_\triangle, D)$. By the rule $(z_{ij} = 1 \rightarrowtail x_i = j) \in F_\triangle$, we have $D_1(x_i) = \{j\}$. Hence, $j' \notin D_1(x_i)$ for all $j' \neq j$. By the rule $(x_i = j' \rightarrowtail z_{ij'} = 1) \in F_\triangle$, we have $1 \notin D_1(z_{ij'})$ for all $j' \neq j$.

— For the rule (r2), suppose $D \sqsubseteq D_{init}$ where $D(z_{ij'}) = \{0\}$ for all $1 \leq j' \neq j \leq k$. Let $D_1 = solv(F_\triangle, D)$. By the rule $(z_{ij'} = 0 \rightarrowtail x_i \neq j') \in F_\triangle$ for all $j' \neq j$, we have $D(x_i) \cap \{1, \ldots, j-1, j+1, \ldots, k\} = \emptyset$. Hence, $D(x_i) = \{j\}$. By the rule $(x_i = j \rightarrowtail z_{ij} = 1) \in F_\triangle$, we have $0 \notin D(z_{ij})$.

By Lemma 1 and Lemma 5, we have $F_\triangle \gg \{dsb(\sum_{j=1}^{k} z_{ij} = 1)\}$. □

Example 22. The (QZ1) constraints of the n-Queens Problem (Example 14) are propagation redundant w.r.t. the Boolean channel using Theorem 11. ■

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.

THEOREM 12. *$F_{\{\}} \gg \{dsb(S_j \cap S_{j'} = \emptyset)\}$ for all $1 \le j < j' \le m$.*

PROOF. A minimal set of propagation rules for $dsb(S_j \cap S_{j'} = \emptyset)$, where $j < j'$, consists of the rules:

$$\begin{array}{ll} (r1) & i \in S_j \rightarrowtail i \notin S_{j'} \\ (r2) & i \in S_{j'} \rightarrowtail i \notin S_j \end{array}$$

We show that $F_{\{\}}$ implements both (r1) and (r2).

— For the rule (r1), suppose $D \sqsubseteq D_{init}$ where $i \in \inf_D(S_j)$. Let $D_1 = solv(F_{\{\}}, D)$. By the rule $(i \in S_j \rightarrowtail x_i = j) \in F_{\{\}}$, we have $D_1(x_i) = \{j\}$ and $j' \notin D_1(x_i)$. By the rule $(x_i \neq j' \rightarrowtail i \notin S_{j'}) \in F_{\{\}}$, we have $i \notin \sup_{D_1}(S_{j'})$.

— For the rule (r2), suppose $D \sqsubseteq D_{init}$ where $i \in \inf_D(S_{j'})$. Let $D_1 = solv(F_{\{\}}, D)$. By the rule $(i \in S_{j'} \rightarrowtail x_i = j') \in F_{\{\}}$, we have $D_1(x_i) = \{j'\}$ and $j \notin D_1(x_i)$. By the rule $x_i \neq j \rightarrowtail i \notin S_j$, we have $i \notin \sup_{D_1}(S_j)$.

By Lemma 1 and Lemma 5, we have $F_{\{\}} \gg \{dsb(S_j \cap S_{j'} = \emptyset)\}$. □

Example 23. Using Theorem 12, the set channel makes both the (GS1) constraints of the social golfers problem (Example 15) and (BS1) constraints of the balanced academic curriculum problem (Example 16) propagation redundant. ■

5.2.2 *Unrestrictive Channel Functions.* 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. If a constraint c_X logically implies another constraint c_Y through an unrestrictive channel, then $dsb(c_X)$ subsumes all the propagation rules implemented by $dsb(c_Y)$.

LEMMA 13. *Let c_X a constraint on X variables, c_Y be a constraint on Y variables and $\Pi_{dsb(c_Y)}$ be a minimal set of propagation rule implemented by $dsb(c_Y)$. If $\diamondsuit$ be an unrestrictive channel function and*

$$\models (D_{init} \wedge c_X \wedge C_{\diamondsuit}) \rightarrow c_Y,$$

then

$$\models (D_{init} \wedge c_X \wedge \diamondsuit^{-1}(C)) \rightarrow \diamondsuit^{-1}(c)) \text{ for all } (C \rightarrowtail c) \in \Pi_{dsb(c_Y)}.$$

PROOF. Suppose to the contrary that for some rule $(C \rightarrowtail c) \in \Pi_{dsb(c_Y)}$ there exists a solution θ_X of $D_{init} \wedge c_X \wedge \diamondsuit^{-1}(C)$ but not a solution of $\diamondsuit^{-1}(c)$. Since θ_X is a solution of $D_{init} \wedge c_X$ and $\diamondsuit$ is unrestrictive, we can map θ_X to θ_Y using $\diamondsuit$ and $\theta = \theta_X \cup \theta_Y$ is a solution of $D_{init} \wedge c_X \wedge C_{\diamondsuit}$. By the condition of the lemma, we have that θ is a solution of c_Y. In particular, $\theta_Y \subset \theta$ is a solution of c_Y since $vars(c_Y)$ contains only Y variables. By construction, θ_Y is also a solution of C since θ_X is a solution of $\diamondsuit^{-1}(C)$. Similarly, θ_Y is not a solution of c since θ_X is not a solution of $\diamondsuit^{-1}(c)$. Using Lemma 1, $dsb(c_Y)$ does not implement $C \rightarrowtail c$, contrary to the hypothesis. □

We can straightforwardly lift the results of Lemma 13 to determine propagation redundancy of constraints simply in terms of logical implication through unrestrictive channel.

THEOREM 14. *Let c_X a constraint on X variables and c_Y be a constraint on Y variables. If $\diamondsuit$ be an unrestrictive channel function and*

$$\models (D_{init} \wedge c_X \wedge C_{\diamondsuit}) \rightarrow c_Y,$$

then $\{dsb(c_X)\} \cup F_{\diamondsuit} \gg \{dsb(c_Y)\}$.

PROOF. The proof follows straightforwardly from Lemma 13 and Corollary 8. □

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 24 *(Counter-example)*. The permutation channel function is restrictive. Now $\models C \rightarrow y_3 = 3$, where $C \equiv x_1 + x_2 < 4 \wedge C_{\bowtie}$ since the only solutions of C are

$$\begin{array}{ll} \{x_1 \mapsto 1, x_2 \mapsto 2, x_3 \mapsto 3, y_1 \mapsto 1, y_2 \mapsto 2, y_3 \mapsto 3\} & \text{and} \\ \{x_1 \mapsto 2, x_2 \mapsto 1, x_3 \mapsto 3, y_1 \mapsto 2, y_2 \mapsto 1, y_3 \mapsto 3\}. & \end{array}$$

However, it is not the case that $x_1 + x_2 < 4 \rightarrow x_3 = 3$. The problem is that the channel $C_{\bowtie}$ removes solutions of $x_1 + x_2 < 4$ like $\{x_1 \mapsto 1, x_2 \mapsto 1, x_3 \mapsto 1\}$ from consideration. ■

We can use Theorem 14 to prove propagation redundancy of many of the propagators in our examples.

Example 25. Consider the (QZ3.1) constraint of the n-Queens Problem (Example 14) for $k = 0$,

$$c_Z \equiv \sum_{i=1}^{n} z_{ii} \leq 1.$$

It is clear that

$$\begin{array}{rcl} c_X & \equiv & (x_1 - 1 \neq x_i - i) \wedge \cdots \wedge (x_{i-1} - (i-1) \neq x_i - i) \;\; \wedge \\ & & (x_{i+1} - (i+1) \neq x_i - i) \wedge \cdots \wedge (x_n - n \neq x_i - i) \end{array}$$

of (QX2.1) satisfies $\models D_{init} \wedge c_X \wedge C_{\triangle} \rightarrow c_Z$. We also have

$$\begin{array}{rcl} \{dsb(c_X)\} & \approx & \{dsb(x_1 - 1 \neq x_i - i), \ldots, dsb(x_{i-1} - (i-1) \neq x_i - i), \\ & & dsb(x_{i+1} - (i+1) \neq x_i - i), \ldots, dsb(x_n \neq x_i + n - i)\} \end{array}$$

by Theorem 4. Since $\triangle$ is an unrestrictive channel function, $dsb(c_Z)$ is propagation redundant w.r.t. $F_{\triangle}$ and the propagators of constraints (QX2.1) by Theorem 14.

Similar arguments apply to show that the other constraints of (QZ3.1) and (QZ3.2) are propagation redundant w.r.t. $C_{\triangle}$ and the propagators of (QX2.1). Also, the constraints of (QZ3.3) and (QZ3.4) are propagation redundant w.r.t. to $C_{\triangle}$ and the propagators of (QX2.2).

Note that the (QZ2) constraints $\sum_{i=1}^{n} z_{ij} = 1$, where $1 \leq j \leq n$, are not propagation redundant. However, we can split (QZ2) into two constraints:

— (QZ2.1) $\sum_{i=1}^{n} z_{ij} \leq 1$
— (QZ2.2) $\sum_{i=0}^{n-1} z_{ij} \geq 1$.

Using similar arguments to c_Z, we can show that constraint (QZ2.1) is propagation redundant w.r.t. $C_{\triangle}$ and (QX1). ■

The following example demonstrates that our approach is also applicable to propagators for global constraints. The use of the `alldifferent` global constraints in the n-Queens problem can make constraints (QZ2) propagation redundant.

Example 26. Consider the (QX1) constraints of the n-Queens Problem (Example 14). Rather than using a set of separate disequality constraints, we can use a single `alldifferent` global constraint:

$$\text{(QX1')}\ \texttt{alldifferent}([x_1, \ldots, x_n]).$$

The propagator $dsb(\texttt{alldifferent}([x_1, \ldots, x_n]))$ is equivalent to

$$dsb\Big(\bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} x_i \neq x_j\Big)$$

and has an efficient implementation [Régin 1994].

Consider the (QZ2) constraints, $c_Z \equiv \sum_{i=1}^{n} z_{ij} = 1$, where $1 \leq j \leq n$. It is straightforward to verify that

$$\models D_{init} \wedge \texttt{alldifferent}([x_1, \ldots, x_n]) \wedge C_{\triangle} \rightarrow c_Z.$$

Since $\triangle$ is an unrestrictive channel function, $dsb(c_Z)$ is propagation redundant w.r.t. $F_{\triangle}$ and the propagator $dsb(\texttt{alldifferent}([x_1, \ldots, x_n]))$ by Theorem 14, ■

Example 27. Consider the (GS2) constraints of the social golfers problem (Example 15),

$$c_{S_1} \equiv |S_{ik}| = s$$

where $1 \leq i \leq g$ and $1 \leq k \leq w$. It is clear that $c_{X_1} \equiv \sum_{l=1}^{n}(x_{lk} = i) = s$ of (GX1) satisfies $\models D_{init} \wedge c_{X_1} \wedge C_{\{\}} \rightarrow c_{S_1}$. Since $\{\}$ is an unrestrictive channel function, by Theorem 14, $dsb(c_{S_1})$ is propagation redundant w.r.t. $F_{\{\}}$ and $dsb(c_{X_1})$.

We can similarly show that the (GS3) constraints are propagation redundant w.r.t. $C_{\{\}}$ and the (GX2) constraints. ■

Example 28. Consider the (BS4) constraints of the balanced academic curriculum problem (Example 16),

$$c_S \equiv (i \in S_k) \Rightarrow (j \notin S_{k'})$$

where $\langle i, j \rangle \in R$, $1 \leq k \leq n-1$ and $1 \leq k' \leq k$. It is clear that the (BX3) constraint, $c_X \equiv x_i < x_j$, satisfies $\models (D_{init} \wedge c_X \wedge C_{\{\}}) \rightarrow c_S$. Since $\{\}$ is an unrestrictive channel function, by Theorem 14, $dsb(c_S)$ is propagation redundant w.r.t. $F_{\{\}}$ and $dsb(c_X)$. ■

In part because the $\doteq$ channel is unrestrictive in both directions, we can prove that set bounds propagation provide the same propagation strength as the mapping of set constraints to Booleans.[5]

THEOREM 15. *Let $dsb(c)$ be the set bounds propagator for set constraint c and $\doteq(c)$ be the Boolean equivalent of c. Then*

[5]Set bounds propagation, however, does still provide a more efficient implementation.

Channel Function	Type	Applicable Theorems
$\bowtie$ / $\bowtie^{-1}$	Restrictive	Corollary 8 (Example 17) Theorem 9 (Example 19) Theorem 10 (Example 21)
$\triangle$	Unrestrictive	Theorem 14 (Examples 25 and 26)
$\triangle^{-1}$	Restrictive	Corollary 8 Theorem 9 Theorem 11 (Example 22)
$\{\}$	Unrestrictive	Theorem 14 (Examples 27 and 28)
$\{\}^{-1}$	Restrictive	Corollary 8 (Example 18) Theorem 9 Theorem 12 (Example 23)
$\doteq$ / $\doteq^{-1}$	Unrestrictive	Theorem 15

Table I. A Summary of Results

(a). $\{dsb(c)\} \cup F_{\doteq} \gg \{dsb(c') \mid c' \in \doteq(c)\}$ *and*
(b). $\{dsb(c') \mid c' \in \doteq(c)\} \cup F_{\doteq} \gg \{dsb(c)\}$.

Proof. See the appendix. $\square$

For ease of referencing, Table I summarizes the results presented in this section. This serves as a guide for problem modelers to quickly identify which theorems are related to their problem.

6. 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 strength but with less propagators. This can translate into *faster* propagation.[6] We verify empirically the improvement of removing propagation redundant constraints for the problems in Section 4, *except* for the n-Queens problem, the reason being that there exist better single models for the n-Queens problem using the global `alldifferent` constraint so that redundant modeling is not worthwhile.

In the following experiments, all the benchmarks are executed using ILOG Solver 4.4 on Sun Ultra 5/400 workstations running Solaris 8. The first column of each table indicates the problem instances. The second column describes the models under comparison. The third column indicates the choices of search variables. In the case of combined models, we have the choice of searching the variables for just one model, or from both models together. However, the question of choosing the "best" set of search variables that gives the smallest search space is out of the scope of this paper. To compare the performance of the different models, we measure the total number of fails (fourth column), total memory used in kilobytes (fifth column), and CPU time in seconds (sixth column). Table entries marked with a "—" mean failure to solve the problem after one hour of execution. To highlight the benefits of removing propagation redundant constraints, we place the

[6]Note there is no guarantee since the number of propagation steps may have increased depending on the order and the events which propagators are processed.

Table II. Finding the First Solution of the Langford's Problem

Instance	Model	Search	Fails	KBytes	Seconds
3×9	M_X	X	192	142	0.08
	full/*opt*	X	77/77	2079/142	0.56/0.07
	M_Y	Y	—	—	—
	full/*opt*	Y	48/48	2362/142	0.36/0.05
	full/*opt*	$X \cup Y$	42/42	2075/142	0.40/0.05
3×10	M_X	X	569	161	0.27
	full/*opt*	X	217/217	3002/165	2.24/0.21
	M_Y	Y	—	—	—
	full/*opt*	Y	22/22	3166/161	0.28/0.03
	full/*opt*	$X \cup Y$	116/116	2978/165	1.55/0.13

figures for the *full* combined model and the *opt*imized combined model on the same cell separated by the symbol "/". To improve the efficiency of combined models, we use the `IlcInverse` global constraint in ILOG Solver to implement the permutation channel. However, ILOG Solver does not provide such a global constraint implementation for the other two channels. Hence, we have implemented our own global constraints for the Boolean channel and set channel to make the results more consistent.

6.1 Langford's Problem

Table II compares the different models for finding the first solution of the Langford's Problem. 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 (LX2.1 + LX2.2 + $C_{\bowtie}$) as discussed in Examples 17 and 21. We use the smallest domain first (i.e. `IlcChooseMinSizeInt` in ILOG Solver) variable ordering heuristic and order values in the domain from the least to the greatest.

Our *opt* model corresponds to the *minimal combined model* of Smith [2000]. Smith empirically shows that using the minimal combined model with search variables $X \cup Y$ is more efficient in solving this problem. Our results agree with those presented by Smith, where the *opt* model is faster and maintains the same number of fails as the *full* model for all three sets of search variables, and the *opt* model with search variables $X \cup Y$ is the fastest among all the models under comparison. In addition to time and number of fails, the amount of memory consumption needed to solve a problem is also an important measure of performance (which is not presented by Smith). The presence of propagation redundant constraints consumes a lot of *unnecessary* memory spaces. From Table II, we can see that the *opt* model requires a lot less memory than the corresponding *full* model.

The benefits of removing propagation redundant constraints are more apparent when we solve for all the solutions of the Langford's Problem (see Table III). Problem instances "4×14" and "4×15" are infeasible and have no solutions. For all instances, the experiment confirms that the *opt* model has the same search space as the *full* model. As the problem size increases, the *opt* model leads to a more significant saving of time and memory consumption over the *full* model.

Table III. Finding All the Solutions of the Langford's Problem

Instance	Model	Search	Fails	KBytes	Seconds
3×9	M_X	X	938	142	0.38
	full/opt	X	432/432	2220/142	3.11/0.35
	M_Y	Y	—	—	—
	full/opt	Y	348/348	2413/142	2.77/0.28
	full/opt	$X \cup Y$	251/251	2260/142	2.32/0.25
3×10	M_X	X	3114	161	1.39
	full/opt	X	1318/1318	3166/169	13.51/1.16
	M_Y	Y	—	—	—
	full/opt	Y	1059/1059	3575/169	11.22/0.88
	full/opt	$X \cup Y$	768/768	3190/169	9.52/0.78
4×14	M_X	X	83068	475	89.03
	full/opt	X	20885/20885	21574/491	1494.09/46.74
	M_Y	Y	—	—	—
	full/opt	Y	8139/8139	25206/487	704.30/21.01
	full/opt	$X \cup Y$	6553/6553	22870/487	640.13/16.87
4×15	M_X	X	351126	538	399.14
	full/opt	X	—/78556	—/550	—/176.24
	M_Y	Y	—	—	—
	full/opt	Y	25270/25270	32957/550	2440.22/66.57
	full/opt	$X \cup Y$	20526/20526	29243/546	2270.67/55.63

Table IV. Finding All the Solutions of the All Interval Series Problem

Instance	Model	Search	Fails	KBytes	Seconds
12	M_X	X	880112	189	265.91
	full/opt	X	39241/39241	2587/169	213.31/33.37
	M_Y	Y	—	—	—
	full/opt	Y	16280/16280	2846/173	76.58/6.13
	full/opt	$X \cup Y$	39195/39195	2587/173	216.03/32.23
13	M_X	X	4914499	228	1632.54
	full/opt	X	158383/158383	3494/200	1016.96/142.78
	M_Y	Y	—	—	—
	full/opt	Y	62949/62949	3859/200	310.22/24.14
	full/opt	$X \cup Y$	158297/158297	3494/200	1008.84/177.39
14	M_X	X	—	—	—
	full/opt	X	—/685301	—/228	—/696.12
	M_Y	Y	—	—	—
	full/opt	Y	266130/266130	5127/240	1473.22/107.39
	full/opt	$X \cup Y$	—/684592	—/228	—/696.25
15	M_X	X	—	—	—
	full/opt	X	—/3096868	—/267	—/3415.98
	M_Y	Y	—	—	—
	full/opt	Y	—/1275661	—/271	—/521.63
	full/opt	$X \cup Y$	—/3091947	—/267	—/3444.73

6.2 All Interval Series

Finding the first solution for the All Interval Series problem is an easy problem. The challenge is to find all the solutions. Table IV compares the different models for finding all the solutions of the all interval series problem. 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 Examples 19 and 21. We use the smallest domain first variable ordering heuristic and order values in the domain from the least to the greatest.

Given the same set of search variables (X or Y), the *full* models reduce the number of fails significantly as compared to the single models (M_X and M_Y). However, the overhead of redundant modeling surpasses the gains from the reduction in search space. By removing propagation redundant constraints from the combined models, the *opt* model with search variables Y is the fastest. The experiment confirms that the *opt* model maintains the same number of fails as the *full* model for all test cases. As the problem size increases, the benefit of the *opt* model is more apparent for both time and memory consumption, and it is the only model which could solve instance $n = 15$ within the time limit. The amount of memory consumption for the *opt* models is competitive to the single models (M_X, M_Y), as opposed to huge memory overhead for the *full* model.

6.3 Social Golfers Problem

The social golfers problem has a large number of symmetric solutions and it is impractical to search for all the solutions.[7] Table V compares the different models for finding the first solution to the social golfers problem. The problem instances are indicated using the parameters *g*-*s*-*w* as described in Example 15. The models under comparison include the single models: M_X and M_S, the *full* combined model (M_X + $C_{\{\}}$ + M_S), and an *opt*imized combined model (GX1 + GX2 + $C_{\{\}}$) as discussed in Examples 23 and 27. The following heuristics are used for variable ordering. For search variables X, we use ascending order of variables indices and variables are ordered players by weeks. Barnier and Brisset [2002] show that this heuristic can solve the 8-4-9 instances efficiently. For search variable S and $X \cup S$, we simply use smallest domain first with values ordered from the least to the greatest.

The experiment confirms that the *opt* model has the same number of fails as the *full* model and, at the same time, speeds up the search and reduces the memory consumption for all test cases. The combined models (*full* and *opt*) with search variables S reduce the number of fails when compared to the single model M_S, but this is not the case when compared to M_X with search variables X. In terms of runtime, no one model dominates the others. The *opt* model with search variables S is the fastest for instance 4-3-4, *opt* model with search variables $X \cup S$ is the fastest for instance 7-2-13, and the single model M_X is the fastest for instances 8-4-9 and 9-2-17.

[7]Although many effective (and often sophisticated) symmetries breaking techniques have been studied (e.g. Puget [2002]), we restrain from doing so since the focus of this paper is on removing propagation redundant constraints. To have a fair comparison, we also avoid the addition of symmetry constraints because the same constraint might be easy to express in one model but not the other [Smith 2001].

Table V. Finding the First Solution of the Social Golfers Problem

Instance	Model	Search	Fails	KBytes	Seconds
4-3-4	M_X	X	1509146	240	395.31
	full/*opt*	X	1509146/1509146	330/244	592.41/459.66
	M_S	S	2389	142	0.41
	full/*opt*	S	1102/1102	326/244	0.41/0.27
	full/*opt*	$X \cup S$	4587/4587	326/244	1.81/1.33
7-2-13	M_X	X	63860	2587	33.56
	full/*opt*	X	63860/63860	6048/2627	122.96/37.62
	M_S	S	37158	3598	30.44
	full/*opt*	S	27998/27998	6064/2603	34.87/13.38
	full/*opt*	$X \cup S$	361/361	6079/2627	1.37/0.42
8-4-9	M_X	X	32	5952	0.32
	full/*opt*	X	32/32	8519/6015	0.89/0.35
	M_S	S	—	—	—
	full/*opt*	S	—/—	—/—	—/—
	full/*opt*	$X \cup S$	—/—	—/—	—/—
9-2-17	M_X	X	7355	7039	8.00
	full/*opt*	X	7355/7355	17474/7114	29.39/8.46
	M_S	S	74098	10718	111.78
	full/*opt*	S	51444/51444	17531/7055	134.26/45.68
	full/*opt*	$X \cup S$	6788/6788	17580/7114	26.67/7.43

6.4 Balanced Academic Curriculum Problem

Hnich *et al.* [2002] report that it is difficult to find the optimal solution of the balance academic curriculum problem with propagation-based constraint solver alone. The challenge of this problem is to find the optimal solution and prove optimality. Table VI compares the different models for finding the optimal solution and proving optimality for the three problem instances of the balanced academic curriculum problem posted in CSPLib. The models under comparison include the single models: M_X and M_S, the *full* combined model (M_X + $C_{\{\}}$ + M_S), and an *opt*imized combined model (B1.1 + B1.2 + B2.1 + B2.2 + BX3 + $C_{\{\}}$ + BS2 + BS3) as discussed in Examples 18, 23 and 28. The following heuristics are used for variable ordering. We use smallest domain first for search variables X and $X \cup S$, and ascending order of variable indices for search variable S. Values are ordered from the least to the greatest.

It is interesting to note that we were able to solve all the problem instances with M_X alone after adding redundant constraints (B2.1) and (B2.2). The experiment confirms that the *opt* model has the same number of fails as the *full* model for all three sets of search variables. The *opt* model is faster and consumes less memory than the *full* model. It is interesting to note that for this problem, the amount of memory consumption for the *opt* model is minimal, even less than the single model M_X. The performance of the *opt* model is clearly superior to the other models. The *opt* model with search variable X is the fastest for instances with 8 periods, the *opt* model with search variables $X \cup S$ is the fastest and has the least number of fails for instances with 10 periods, and the *opt* model with search variables S is the fastest and has the least number of fails for the instance with 12 periods.

Table VI. Finding the Optimal Solution and Proving Optimality of the Balanced Academic Curriculum Problem

Instance	Model	Search	Fails	KBytes	Seconds
8 Periods	M_X	X	101	377	0.08
	full/opt	X	101/101	589/79	0.19/0.02
	M_S	S	—	—	—
	full/opt	S	1577/1577	589/79	2.16/0.19
	full/opt	$X \cup S$	118/118	589/79	0.21/0.03
10 Periods	M_X	X	468	432	0.50
	full/opt	X	470/470	766/87	1.62/0.12
	M_S	S	—	—	—
	full/opt	S	323/323	766/83	0.63/0.07
	full/opt	$X \cup S$	149/149	766/87	0.32/0.02
12 Periods	M_X	X	33602	801	27.47
	full/opt	X	33530/33530	1692/102	147.46/4.97
	M_S	S	—	—	—
	full/opt	S	882/882	1669/98	3.78/0.19
	full/opt	$X \cup S$	—/10541901	—/102	—/1393.21

7. RELATED WORK

Smith [2000; 2001] has examined the redundant models for a number of individual problems including the n-Queens problem, Langford's problem and the social golfers problems. Smith empirically demonstrates that some constraints in the redundant models can be removed without increasing the search space. Smith points out that for these problems the so-called *minimal combined model*, which combine the first model and only the variables of the second model (without the constraints) using channeling constraints, produces the same search behavior as combining the models in full. This is proved in an ad hoc manner by Choi and Lee [2002]. In this paper, we aim for a theoretical framework which can determine propagation redundancy of a particular constraint involved in redundant models *a priori.*

Apt and Monfroy [2001] develop "membership rules" as a way of building propagators for any constraints. Propagation rules are similar to the "membership rules" when restricted to integer variables. However, we develop propagation rules as a method for reasoning about the parts of a propagator's behavior.

Brand [2003] gives a general theorem to determine when a rule is propagation redundant with respect to a set of rules in rule-based constraint programming, and illustrates the applicability using "membership rules." In fact, our definition of a propagation rule satisfies the required properties of Brand's theorem. Hence, we can apply Brand's theorem to determine when a propagation rule is propagation redundant with respect to a set of propagation rules. In this paper, we are interested in propagation redundancy beyond the individual propagation rules, but propagation redundancy of the constraint as a whole. We also generalize the notion of propagation redundancy of a propagation rules through a channel function.

Hnich *et al.* [2004] and Walsh [2001] introduce the notion of *constraint tightness* as a measure to compare the propagation strength of different permutation constraints. Their work focuses on comparing the propagation strength of the different notions of consistency over the disequations, channeling constraints, and `alldifferent` global constraints in redundant modeling of *only* permutation problems and injection problems. Our comparison measure is similar to constraint tightness except that constraint tightness is parameterized by a local consistency property. However, in existing constraint solvers, there are propagators which implement *none* of the (established) local consistency properties. An example is the multiplication constraint $x = y \times z$ over integer domain as discussed in Apt [2003, pages 219–220]. In such cases where the local consistency property of a constraint is unknown, our comparison measure would still be applicable. In this paper, we are not only interested in studying the propagation of the permutation constraints, but also the other constraints in redundant models. We also cover a broader class of channeling constraints beyond the permutation channels.

Walsh [2003] proves that "bounds consistency" on set (multiset) variables is equivalent to bounds consistency on the corresponding occurrence representation. This result is related to Theorem 15 since the occurrence representation of set variables corresponds to Boolean variables described in Section 4.4. However, existing constraint solvers break Boolean constraints into parts and propagate each part separately. We prove the theorem based on this more realistic assumption.

A corollary of Theorem 4 is that we can determine domain consistency of an entire integer CSP with tree structure just using the individual domain propagators, since we can repeatedly apply the above lemma to break the conjunction of constraints into individual constraints. This is highly related to the "backtrack-free" approach to solving CSPs with tree structure of Freuder [1982].

8. CONCLUSION

The contributions of this paper are three-fold. First, we define channeling constraints in terms of channel functions which allow us to cover a broad form of redundant modeling. By breaking up a propagator into individual propagation rules, we reason that constraints in one model can be made propagation redundant by constraints in the other model through channels. Second, we introduce the notion of restrictive and unrestrictive channel functions to characterize channeling constraints. Restrictive channel functions can themselves make a constraint in the combined model propagation redundant. Unrestrictive channel functions allow the detection of propagation redundancy of a constraint in one model with respect to a constraint in the other model plus the channels simply in terms of logical consequence. Third, benchmarking results confirm that removals of propagation redundant constraints from combined model can often lead to a faster implementation with the same search space and consuming less memory. As explained in Section 7, this paper extends related work by covering a broader form of redundant modeling and reasoning about the propagation redundancy of all the constraints in the redundant models.

Although we have concentrated on domain and set bounds propagators, many of our results can be used for other propagators. Lemma 6 can be applied for any propagator, since it only relies on the propagation rules. We can use Theorems 7, 9, 14, and Corollary 8 to prove the weaker propagators for c than $dsb(c)$ are propagation redundant, or that stronger propagators for c than $dsb(c)$ make another propagator redundant.

Our work prompts a number of important future directions for research. It is interesting to investigate if the process of removing propagation redundant constraints can be (semi-)automated. To use Theorem 9 we can straightforwardly define the propagation rules for many constraints (parametrically in D_{init}) or construct them automatically using the approach of Abdennadher and Rigotti [2002]. The number of propagation rules for most constraints, however, are exponential. A naive implementation would be computationally too expensive and impractical for more complex real-life applications. A possible approach is to consider *parameterized propagation rules*, which denotes a set of propagation rules, so that the number of rules is vastly reduced. We can also try to use Theorem 14 to prove propagation redundancy without considering propagation rules.

The amount of computation overhead induced by propagation redundant constraints depends on the order and on the events which constraints are processed during constraint propagation. Our experimental platform, ILOG Solver, is a proprietary constraint programming library which does not provide access to such information. It would be interesting to study how these factors affect the performance of constraint solving after propagation redundant constraints are removed from the model.

Our existing approach analyzes the propagation behavior of the redundant constraints in the model statically before search. It is interesting to investigate for an alternative approach of analyzing dynamically the propagation behavior of redundant constraints during search. Based on the results of dynamic analysis, the constraint solver should avoid (as much as possible) processing the propagation redundant constraints during constraint propagation. This would minimize the computation overhead incurred by the propagation redundant constraints even if they are present in the model.

Redundant modeling gives rise to the need to decide which variables to label during search. As demonstrated in Section 6, the choice of search variables can greatly affect the size of the search space. For example, Geelen [1992], Smith [2000; 2001] and Hnich *et al.* [2004], also show that certain choices of search variables do lead to a smaller search space. Therefore, it is interesting to study and establish criteria in choosing the better set of search variables.

APPENDIX

We present the longer proofs, in full, in this appendix, to improve the readability of the main body of the text.

LEMMA 1. *Given a constraint c, $dsb(c)$ implements $C \rightarrowtail c'$ iff*

$$\models (D_{init} \wedge c) \rightarrow (C \rightarrow c').$$

PROOF. To prove the if direction ($\Rightarrow$), suppose to the contrary that $dsb(c)$ implements $C \rightarrowtail c'$ and $\not\models (D_{init} \wedge c) \rightarrow (C \rightarrow c')$. Then, there exists a solution $\theta \in D_{init}$ such that θ satisfies $c \wedge C \wedge \neg(c')$. Now, we build a domain, $D_\theta \sqsubseteq D_{init}$, from θ as follows:

$$D_\theta(v) = \begin{cases} \{\theta(v)\} & \text{for all } v \in vars(\theta), \\ D_{init}(v) & \text{otherwise} \end{cases}$$

Since θ satisfies c and $\neg(c')$, we have $\models D_\theta \rightarrow C$ and $\not\models D_\theta \rightarrow c'$. Since θ is a solution of c, we have $dsb(c)(D_\theta) = D_\theta$. Thus, $\not\models dsb(c)(D_\theta) \rightarrow c'$. By Definition 5, $dsb(c)$ does not implements $C \rightarrowtail c'$, contrary to the hypothesis.

For the only if direction ($\Leftarrow$), suppose to the contrary that $\models (D_{init} \wedge c) \rightarrow (C \rightarrow c')$ and $dsb(c)$ does not implement $C \rightarrowtail c'$. Then, there exists $D \sqsubseteq D_{init}$ such that $\models D \rightarrow C$ but $\not\models dsb(c)(D) \rightarrow c'$. For each form of c', we show that there exists a solution $\theta \in D$ of c.

— If $c' \equiv x \neq d$ where $x \in vars(c)$ and $d \in D(x)$, then $\not\models dsb(c)(D) \rightarrow c'$ means that $d \in dsb(c)(D)$. By the definition of $dsb(c)$, there exists a solution $\theta \in D$ of c where $\theta(x) = d$.

— If $c' \equiv x = d$ where $x \in vars(c)$ and $d \in D(X)$, then $\not\models dsb(c)(D) \rightarrow c'$ means that $d' \in dsb(c)(D)$ where $d' \neq d$ and $d' \in D(X)$. By the definition of $dsb(c)$, there exists a solution $\theta \in D$ of c where $\theta(x) = d'$.

— If $c \equiv d \in S$ where $X \in vars(c)$ and $d \notin \inf_D(S)$, then $\not\models dsb(c)(D) \rightarrow c'$ means that $d \notin \inf_{D'}(S)$ where $D' = dsb(c)(D)$. By the definition of $dsb(c)$, there exists a solution $\theta \in D$ of c where $d \notin \theta(S)$.

— If $c \equiv d \notin S$ where $X \in vars(c)$ and $d \in \sup_D(S)$, then $\not\models dsb(c)(D) \rightarrow c'$ means that $d \in \sup_{D'}(S)$ where $D' = dsb(c)(D)$. By the definition of $dsb(c)$, there exists a solution $\theta \in D$ of c where $d \in \theta(S)$.

Now, we know that $\theta \in dsb(c)(D)$ since θ is a solution of c. Using $\not\models dsb(c)(D) \rightarrow c'$, we have that θ is not a solution of c'. However, we also know that θ satisfies $D_{init} \wedge c \wedge C$ since $D \sqsubseteq D_{init}$ and $\models D \rightarrow C$. Hence, $\not\models (D_{init} \wedge c) \rightarrow (C \rightarrow c')$, contrary to the hypothesis. $\square$

THEOREM 4. *If c_1 and c_2 are two constraints sharing at most one integer variable, $x \in \mathcal{V}_I$, then $\{dsb(c_1), dsb(c_2)\} \approx \{dsb(c_1 \wedge c_2)\}$.*

PROOF. We have $\{dsb(c_1 \wedge c_2)\} \gg \{dsb(c_1), dsb(c_2)\}$ by Lemma 3. To show $\{dsb(c_1), dsb(c_2)\} \gg \{dsb(c_1 \wedge c_2)\}$, suppose to the contrary that there exists a variable $y \in vars(c_1 \wedge c{\cdot}2)$ such that

$$solv(\{dsb(c_1), dsb(c_2)\}, D)(y) \not\subseteq solv(\{dsb(c_1 \wedge c_2)\}, D)(y)$$

for certain $D \sqsubseteq D_{init}$. Let $D_1 = solv(\{dsb(c_1), dsb(c_2)\}, D)$ and $D_2 = solv(\{dsb(c_1 \wedge c_2)\}, D)$. Assume w.l.o.g. that $y \in vars(c_1)$. For each type of domain changes by $dsb(c_1 \wedge c_2)$, we show that it leads to a contradiction.

— If $dsb(c_1 \wedge c_2)$ eliminates a value d from $D(y)$ where $y \in \mathcal{V}_I$, then $d \in D_1(y)$ and $d \notin D_2(y)$. By definition of $dsb(c_1)$, there exists a solution $\theta_1 \in D_1$ of c_1 such that $\theta(y) = d$ since $d \in D_1(y)$. Now if there exists a solution $\theta_2 \in D$ of c_2 where $\theta_2(x) = \theta_1(x)$ then we have a contradiction, since $\theta_1 \cup \theta_2 \in D$ is a solution of $c_1 \wedge c_2$. Otherwise there is no such θ_2, hence $dsb(c_2)(D)$ eliminates the value $\theta_1(x)$ from $D(x)$. Hence $\theta_1(x) \notin D_1(x)$. But then $\theta_1 \notin D_1$ which contradicts the hypothesis.

— If $dsb(c_1 \wedge c_2)$ adds a value d to $\inf_D(y)$ where $y \in \mathcal{V}_S$, then $d \notin \inf_{D_1}(y)$ and $d \in \inf_{D_2}(y)$. By definition of $dsb(c_1)$, there exists a solution $\theta_1 \in D_1$ of c_1

such that $d \notin \theta_1(y)$ since $d \notin \inf_{D_1}(y)$. Now if there exists a solution $\theta_2 \in D$ of c_2 where $\theta_2(x) = \theta_1(x)$ then we have a contradiction, since $\theta_1 \cup \theta_2 \in D$ is a solution of $c_1 \wedge c_2$ which gives a solution where $y = d$. Otherwise there is no such θ_2, hence $dsb(c_2)(D)$ eliminates the value $\theta_1(x)$ from the domain of x. Hence $\theta_1(x) \notin D_1(x)$. But then $\theta_1 \notin D_1$ which contradicts the hypothesis.

— If $dsb(c_1 \wedge c_2)$ eliminates a value d from $\sup_D(y)$ where $y \in \mathcal{V}_S$, then $d \in \sup_{D_1}(y)$ and $d \notin \sup_{D_2}(y)$. By definition of $dsb(c_1)$, there exists a solution $\theta_1 \in D_1$ of c_1 such that $d \in \theta_1(y)$ since $d \in \sup_{D_1}(y)$. Now if there exists a solution $\theta_2 \in D$ of c_2 where $\theta_2(x) = \theta_1(x)$ then we have a contradiction, since $\theta_1 \cup \theta_2 \in D$ is a solution of $c_1 \wedge c_2$ which gives a solution where $y = d$. Otherwise there is no such θ_2, hence $dsb(c_2)(D)$ eliminates the value $\theta_1(x)$ from the domain of x. Hence $\theta_1(x) \notin D_1(x)$. But then $\theta_1 \notin D_1$ which contradicts the hypothesis.

$\square$

LEMMA 5. *Consider a minimal set of propagation rules, $\Pi_{dsb(c)}$, implemented by $dsb(c)$ for constraint c. Then $\{dsb(c)\} \approx \Pi_{dsb(c)}$.*

PROOF. We have $\{dsb(c)\} \gg \Pi_{dsb(c)}$ by Lemma 1 and Theorem 2. To show that $\Pi_{dsb(c)} \gg \{dsb(c)\}$, suppose to the contrary that there exists a variable $x_i \in vars(c)$ where $1 \leq i \leq n$ such that

$$solv(\Pi_{dsb(c)}, D)(x_i) \not\subseteq solv(\{dsb(c)\}, D)(x_i)$$

for certain $D \sqsubseteq D_{init}$. Let $D_1 = solv(\Pi_{dsb(c)}, D)$ and $D_2 = solv(dsb(c), D)$. For each type of domain changes by $dsb(c)$, we show that it leads to a contradiction.

— If $dsb(c)$ eliminates a value d from $D(x_i)$ where $x_i \in \mathcal{V}_I$, then $d \in D_1(x_i)$ and $d \notin D_2(x_i)$. Clearly, $dsb(c)$ implements a rule $r \in \Phi_{dsb(c)}$ such that

$$r \equiv \Big(\bigwedge_{j=1}^{n} \bigwedge_{d' \in A} x_j \neq d' \Big) \rightarrowtail x_i \neq d$$

where $A = (D_{init}(x_j) - D(x_j))$. Now $\Pi_{dsb(c)} \gg \{r\}$ by the definition of $\Pi_{dsb(c)}$. Hence, $d \notin D_1(x_i)$, contrary to the hypothesis.

— If $dsb(c)$ adds a value d to $\inf_D(x_i)$ where $x_i \in \mathcal{V}_S$, then $d \notin \inf_{D_1}(x_i)$ and $d \in \inf_{D_2}(x_i)$. Clearly, $dsb(c)$ implements a rule $r \in \Phi_{dsb(c)}$ such that

$$r \equiv \Big(\bigwedge_{j=1}^{n} \bigwedge_{d' \in A} d' \in x_j \ \bigwedge_{k=1}^{n} \bigwedge_{d'' \in B} d'' \notin x_k \Big) \rightarrowtail d \in x_i.$$

where $A = \inf_D(S_j) - \inf_{D_{init}}(S_j)$ and $B = \sup_{D_{init}}(S_k) - \sup_D(S_k)$. Now $\Pi_{dsb(c)} \gg \{r\}$ by the definition of $\Pi_{dsb(c)}$. Hence, $d \in \inf_{D_1}(x_i)$, contrary to the hypothesis.

— If $dsb(c)$ eliminates a value d from $\sup_D(x_i)$ where $x_i \in \mathcal{V}_S$, then $d \in \sup_{D_1}(x_i)$ and $d \notin \sup_{D_2}(x_i)$. Clearly, $dsb(c)$ implements a rule $r \in \Phi_{dsb(c)}$ such that

$$r \equiv \Big(\bigwedge_{j=1}^{n} \bigwedge_{d' \in A} d' \in x_j \ \bigwedge_{k=1}^{n} \bigwedge_{d'' \in B} d'' \notin x_k \Big) \rightarrowtail d \notin x_i.$$

where $A = \inf_D(S_j) - \inf_{D_{init}}(S_j)$ and $B = \sup_{D_{init}}(S_k) - \sup_D(S_k)$. Now $\Pi_{dsb(c)} \gg \{r\}$ by the definition of $\Pi_{dsb(c)}$. Hence, $d \notin \sup_{D_1}(x_i)$, contrary to the hypothesis. $\square$

In order to prove Theorem 15, we need to introduce the notion of nogood constraints. A *nogood constraint* c is an integer constraint where every valuation in D_{init} except one valuation θ is a solution of c. We call the non-solution valuation θ of c as the *nogood*.

Consider a conjunction of a nogood constraint c_1 and an integer constraint c_2, such that $vars(c_2) \subseteq vars(c_1)$. The following lemma tells us about two useful properties of the nogood of c_1 if a value is removed by $\{dsb(c_1 \wedge c_2)\}$ and not by $\{dsb(c_1), dsb(c_2)\}$.

LEMMA 16. *Let c_1 be a nogood constraint with $vars(c_1) = \{x_1, \ldots, x_n\} \subseteq \mathcal{V}_I$ with the nogood θ, and c_2 be a constraint with $vars(c_2) \subseteq vars(c_1)$. Suppose $D_1 = solv(\{dsb(c_1), dsb(c_2)\}, D)$ and $D_2 = solv(\{dsb(c_1 \wedge c_2)\}, D)$ for $D \sqsubseteq D_{init}$. If there exists a value $d \in D(x_k)$, $1 \leq k \leq n$ such that $d \in D_1(x_k)$ and $d \notin D_2(x_k)$, then*

(a). $\theta \in D$ and

(b). $\theta(x_k) = d$.

PROOF. For (a), suppose to the contrary that $d \in D_1(x_k)$, $d \notin D_2(x_k)$ and $\theta \notin D$, then there must exists $1 \leq i \leq n$ such that $\theta(x_i) \notin D(x_i)$. Since $d \in D_1(x_k)$, by definition, $d \in dsb(c_2)(D)(x_k)$. Thus there exists a solution $\theta_1 \in D$ of c_2 where $vars(\theta_1) = vars(c_1)$ and $\theta_1(x_k) = d$. Since $\theta_1(x_i) \neq \theta(x_i)$, clearly θ_1 is also a solution of c_1. Hence, $d \in D_2(x_k)$ by the definition of $dsb(c_1 \wedge c_2)$, contrary to the hypothesis.

For (b), suppose to the contrary that $d \in D_1(x_k)$, $d \notin D_2(x_k)$, and $\theta(x_k) \neq d$, by definition, $d \in dsb(c_2)(D)(x_k)$. Thus there exists a solution $\theta_1 \in D$ of c_2 where $vars(\theta_1) = vars(c_1)$ and $\theta_1(x_k) = d$. Since $\theta_1(x_k) \neq \theta(x_k)$, clearly θ_1 is also a solution of c_1. Hence, $d \in D_2(x_k)$ by the definition of $dsb(c_1 \wedge c_2)$, contrary to the hypothesis. $\square$

The following lemma identifies the condition where propagation of the conjunction of a nogood constraint c_1 and an integer constraint c_2, such that $vars(c_2) \subseteq vars(c_1)$, is equivalent to propagation on the individual conjuncts. The condition requires that each valuation $\theta' \in D_{init}$ differing from the nogood θ of c_1 by only one assignment must be a solution of c_2.

LEMMA 17. *Let c_1 be a nogood constraint with the nogood θ, $vars(c_1) = \{x_1, \ldots, x_n\} \subseteq \mathcal{V}_I$, and c_2 be a constraint with $vars(c_2) \subseteq vars(c_1)$. If for all valuations $\theta' \in D_{init}$, such that there exists $1 \leq j \leq n$ and $\theta'(x_i) = \theta(x_i)$ for all $1 \leq i \neq j \leq n$, are solutions of c_2, then $\{dsb(c_1), dsb(c_2)\} \approx \{dsb(c_1 \wedge c_2)\}$.*

PROOF. By Lemma 3 we have that $\{dsb(c_1 \wedge c_2)\} \gg \{dsb(c_1), dsb(c_2)\}$. To show $\{dsb(c_1), dsb(c_2)\} \gg \{dsb(c_1 \wedge c_2)\}$, suppose to the contrary that there exists a variable x_k where $1 \leq k \leq n$ such that

$$solv(\{dsb(c_1), dsb(c_2)\}, D)(x_k) \not\subseteq solv(\{dsb(c_1 \wedge c_2)\}, D)(x_k)$$

for certain $D \sqsubseteq D_{init}$. Let $D_1 = solv(\{dsb(c_1), dsb(c_2)\}, D)$ and $D_2 = solv(\{dsb(c_1 \wedge c_2)\}, D)$, then there exists a value $d \in D(x_k)$ such that $d \in D_1(x_k)$ and $d \notin D_2(x_k)$. By the definition of *solv*, $d \in dsb(c_1)(D)(x_k)$ since $d \in D_1(x_k)$. Now it is not the case that $D(x_i) = \{\theta(x_i)\}$ for $1 \leq i \leq n$, $i \neq k$ (otherwise, $d \notin dsb(c_1)(D)(x_k)$ since θ is the nogood of c_1.) Thus, there must exists $1 \leq j \leq n$ and $j \neq k$ such that $|D(x_j)| \geq 2$. Hence, there exists $d_j \in D(x_j)$ such that $d_j \neq \theta(x_j)$. By Lemma 16, we have that $\theta \in D$ and $\theta(x_k) = d$. Consider the valuation θ' defined as $\theta'(x_j) = d_j$ and $\theta'(x_i) = \theta(x_i)$ for all $1 \leq i \leq n$ where $i \neq j$. Note that $\theta'(x_k) = \theta(x_k) = d$. By construction, $\theta' \in D$ and θ' is a solution of c_1. By the condition of the lemma, θ' is also a solution of c_2. Hence, $d \in D_2(x_k)$ using definition of $dsb(c_1 \wedge c_2)$, contrary to the hypothesis. $\Box$

THEOREM 15. *Let $dsb(c)$ be the set bounds propagator for set constraint c and $\doteqdot(c)$ be the Boolean equivalent of c. Then*

(a). $\{dsb(c)\} \cup F_{\doteqdot} \gg \{dsb(c') \mid c' \in \doteqdot(c)\}$ and

(b). $\{dsb(c') \mid c' \in \doteqdot(c)\} \cup F_{\doteqdot} \gg \{dsb(c)\}$.

PROOF. For (a), since $\doteqdot$ is an unrestrictive channel, this immediately gives us that $\{dsb(c)\} \cup F_{\doteqdot} \gg \{dsb(c') \mid c' \in \doteqdot(c)\}$ using Theorem 14 and Theorem 2.

For (b), since $\doteqdot^{-1}$ is also an unrestrictive channel, we also have that

$$\{dsb(\wedge\{c' \mid c' \in \doteqdot(c)\})\} \cup F_{\doteqdot} \gg \{dsb(c)\}$$

using Theorem 14. It remains to show that

$$\{dsb(\wedge\{c' \mid c' \in \doteqdot(c)\})\} \approx \{dsb(c') \mid c' \in \doteqdot(c)\}$$

for each of the constraints c in Figure 1.

For c of the form: $S_i = \emptyset$, $S_a \subseteq S_b$, $S_a \cap S_b = \emptyset$, and $|S_i| = m$, no two constraints in $\doteqdot(c)$ share a variable. Hence, the results hold by Theorem 4.

For the remaining constraints, $S_a = S_b \cup S_c$, $S_a = S_b \cap S_c$, $S_a = S_b - S_c$, we show the proof for $S_a = S_b - S_c$ the others are similar.

Consider the 3 Boolean constraints in $\doteqdot(S_a = S_b - S_c)$ for a particular j: $c_1 \equiv z_{aj} \geq z_{bj} - z_{cj}$, $c_2 \equiv z_{aj} \leq z_{bj}$, and $c_3 \equiv z_{aj} + z_{cj} \leq 1$. Note that c_1 is a nogood constraint with nogood $\{z_{aj} \mapsto 0, z_{bj} \mapsto 1, z_{cj} \mapsto 0\}$, and the valuations: $\{z_{aj} \mapsto 1, z_{bj} \mapsto 1, z_{cj} \mapsto 0\}$, $\{z_{aj} \mapsto 0, z_{bj} \mapsto 0, z_{cj} \mapsto 0\}$, and $\{z_{aj} \mapsto 0, z_{bj} \mapsto 1, z_{cj} \mapsto 1\}$ are all solutions of $c_2 \wedge c_3$. By Lemma 17, we have that $\{dsb(c_1 \wedge (c_2 \wedge c_3))\} \approx \{dsb(c_1), dsb(c_2 \wedge c_3)\}$. Now c_2 and c_3 share only one Boolean variable z_{aj}, by Theorem 4, $\{dsb(c_2 \wedge c_3)\} \approx \{dsb(c_2), dsb(c_3)\}$. Hence, $\{dsb(c_1 \wedge c_2 \wedge c_3)\} \approx \{dsb(c_1), dsb(c_2), dsb(c_3)\}$. Since for any two constraints in $\doteqdot(c)$ with different values of j do not share any variables, the results hold by Theorem 4. $\Box$

ACKNOWLEDGMENTS

The work described in this paper was substantially supported by grants from the Research Grants Council of the Hong Kong Special Administrative Region. (Project nos. CUHK4219/04E and CUHK4183/00E)

REFERENCES

Abdennadher, S. and Rigotti, C. 2002. Automatic generation of rule-based solvers for intentionally defined constraints. *International Journal of Artificial Intelligence Tools 11,* 2, 283–302.

Apt, K. 2003. *Principles of Constraint Programming.* Cambridge University Press.

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

Azevedo, F. and Barahona, P. 2000. Modelling digital circuits problems with set constraints. In *Proceedings of the 1st International Conference on Computational Logic (CL 2000).* 414–428.

Barnier, N. and Brisset, P. 2002. Solving the Kirkman's schoolgirl problem in a few seconds. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP 2002).* 477–491.

Brand, S. 2003. A note on redundant rules in rule-based constraint programming. In *Recent Advances in Constraints, Joint ERCIM/CologNet International Workshop on Constraint Solving and Constraint Logic Programming.* 109 – 120.

Carlier, J. and Pinson, E. 1989. An algorithm for solving the job-shop problem. *Management Science 35,* 2, 164–176.

Cheadle, A., Harvey, W., Sadler, A., Schimpf, J., Shen, K., and Wallace, M. 2003. ECL^{i}PS^e: An introduction. Technical Report IC-Parc-03-1, IC-Parc, Imperial College London.

Cheng, B. M. W., Choi, K. M. F., Lee, J. H. M., and Wu, J. C. K. 1999. Increasing constraint propagation by redundant modeling: an experience report. *Constraints 4,* 2, 167–192.

Choi, C. W. and Lee, J. H. M. 2002. On the pruning behaviour of minimal combined models for permutation CSPs. In *Proceedings of the International Workshop on Reformulating Constraint Satisfaction Problems.* 3–17.

Choi, C. W., Lee, J. H. M., and Stuckey, P. J. 2003a. Propagation redundancy for permutation channels. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 03).* 1370–1371.

Choi, C. W., Lee, J. H. M., and Stuckey, P. J. 2003b. Propagation redundancy in redundant modelling. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP 2003).* 229–243.

Freuder, E. C. 1982. A sufficient condition for backtrack-free search. *Journal of the ACM 29,* 1, 24–32.

Frisch, A. M., Jefferson, C., and Miguel, I. 2004. Symmetry breaking as a prelude to implied constraints: A constraint modelling pattern. In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI 2004).* 171–175.

Geelen, P. A. 1992. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI 92).* 31–35.

Gervet, C. 1997. Interval propagation to reason about sets: Definition and implementation of a practical language. *Constraints 1,* 3, 191–244.

Hnich, B., Kiziltan, Z., and Walsh, T. 2002. Modelling a balanced academic curriculum problem. In *Proceedings of the 4th International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR 2002).* 121–131.

Hnich, B., Walsh, T., and Smith, B. M. 2004. Dual modelling of permutation and injection problems. *Journal of Artificial Intelligence Research 21,* 357–391.

ILOG. 1999. *ILOG Solver 4.4: Reference Manual.*

Mackworth, A. K. 1977. Consistency in networks of relations. *Artificial Intelligence 8,* 1, 99–118.

Marriott, K. and Stuckey, P. J. 1998. *Programming with Constraints: an Introduction.* The MIT Press.

Mohr, R. and Masini, G. 1988. Good old discrete relaxation. In *Proceedings of the 8th European Conference on Artificial Intelligence (ECAI 88).* 651–656.

Müller, T. 2001. Constraint propagation in mozart. Ph.D. thesis, Universität des Saarlandes, Naturwissenschaftlich-Technische Fakultät I, Fachrichtung Informatik.

Puget, J.-F. 2002. Symmetry breaking revisited. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP 2002)*. 446–461.

Puget, J.-F. and Régin, J.-C. 2001. Solving the all interval problem. Available from `http://4c.ucc.ie/~tw/csplib/prob/prob007/puget.pdf`.

Régin, J.-C. 1994. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI 94)*. 362–367.

SICStus Prolog. 2003. *SICStus Prolog User's Manual, Release 3.10.1*.

Smith, B., Stergiou, K., and Walsh, T. 2000. Using auxiliary variables and implied constraints to model non-binary problems. In *Proceedings of the 17th National Conference on Artificial Intelligence (AAAI 2000)*. 182–187.

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

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

Van Hentenryck, P., Saraswat, V., and Deville, Y. 1998. Design, implementation and evaluation of the constraint language cc(FD). *Journal of Logic Programming 37,* 1–3, 139–164.

Walsh, T. 2001. Permutation problems and channelling constraints. In *Proceedings of the 8th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR 2001)*. 377–391.

Walsh, T. 2003. Consistency and propagation with multiset constraints: A formal viewpoint. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP 2003)*. 724–738.

Received December 2004; revised July 2005; accepted October 2005