Hybrid Set Domains to Strengthen Constraint Propagation and Reduce Symmetries

Andrew Sadler, Carmen Gervet

Hybrid Set Domains to Strengthen Constraint Propagation and Reduce Symmetries

Andrew Sadler and Carmen Gervet

IC–Parc, Imperial College London, SW7 2AZ, U.K.
{ajs2,cg6}@icparc.ic.ac.uk

Abstract. In CP literature combinatorial design problems such as sport scheduling, Steiner systems, error-correcting codes and more, are typically solved using Finite Domain (FD) models despite often being more naturally expressed as Finite Set (FS) models. Existing FS solvers have difficulty with such problems as they do not make strong use of the ubiquitous set cardinality information. We investigate a new approach to strengthen the propagation of FS constraints in a tractable way: extending the domain representation to more closely approximate the true domain of a set variable. We show how this approach allows us to reach a stronger level of consistency, compared to standard FS solvers, for arbitrary constraints as well as providing a mechanism for implementing certain symmetry breaking constraints. By experiments on Steiner Systems and error correcting codes, we demonstrate that our approach is not only an improvement over standard FS solvers but also an improvement on recently published results using FD 0/1 matrix models as well.

1 Introduction

Combinatorial designs have applications in areas as diverse as error-correcting codes, sport scheduling, Steiner systems and more recently networking and cryptography (e.g. see [1] for a survey). While a combinatorial design problem is defined in terms of discrete points, or sets, in the CLP framework it is modeled as a constraint satisfaction problem (CSP) with variables representing the points or sets and having a domain of values. Conceptually these domains are sets of possible instantiations but in practice it is often a requirement that the domains be approximated for efficiency reasons. A common approach to approximating variable domains is to use upper and lower bounds (where "upper" and "lower" are defined by some appropriate ordering on domain elements) which are known to enclose the actual domain. Finite Set (FS) domains are ordered by inclusion (the subset ($\subseteq$) order) and have bounds which are ground sets e.g. $X \in [\{1\}, \{1,2,3\}]$. The lower bound, denoted $glb(X)$, contains the definite elements of the set $\{1\}$ while the upper bound $lub(X)$, contains in addition the potential elements $\{2,3\}$. The constraint reasoning is based on local bound consistency techniques extended to handle set constraints [2] and solvers of this sort have been embedded in a growing number of CP languages (e.g. ECL^{i}PS^e, ILOG, CHOCO, Facile, BProlog). The bounds representation is compact and benefits

from interval reasoning techniques which allow us to remove at a minimal cost set values that can never be part of any solution. However it does not guarantee in general, that all the values from a domain are locally consistent (true for the set cardinality constraint) and it does not provide any form of global reasoning.

Because of these weaknesses, many of the recent proposals to tackle combinatorial design problems efficiently assume a FD model where the domain elements are naturally ordered ($\leq$) and the bounds are the min/max value. Extensive research towards improving the efficiency of FD consistency algorithms (e.g. global constraints [3, 4]) and search (e.g. symmetry breaking approaches) [5, 6] has made this a powerful and general scheme. However many combinatorial design problems are more naturally expressed in FS models, we investigated ways to achieve better efficiency for such models.

In this paper we discuss briefly our work on global filtering for n-ary set constraints over fixed cardinality sets and present in much more detail a new domain representation for set variables. Experimental results are shown on Steiner systems and error-correcting code problems.

The paper is structured as follows. In Sect. 2, we give background on consistency notions for finite set constraint systems. Section 3 addresses the problem of global set constraints. Section 4 introduces the new set interval representation. Section 5 defines the hybrid domain and in Sect. 6 we experimentally evaluate our approach.

2 Background

The solving of a CSP is handled by interleaving constraint propagation (domain reduction) and search. The constraint propagation can be formally defined by the level of consistency enforced for each constraint or system of constraints. We recall the different consistency notions used in this paper.

Given a finite domain representation, we say that a constraint is Generalized Arc Consistent (GAC) **iff** any value assigned to a variable from its domain can be extended to a complete assignment to the constraint [7]. GAC generalizes arc consistency (AC) defined for binary constraints.

Maintaining GAC can be costly however and so when dealing with domains which are approximated by bounds it is often easier/more efficient to only ensure that the bounds of the domain, when assigned to the variable, can be extended to a complete assignment. This notion of "bounds consistency" is used in many FD solvers where bounds are the min/max domain elements mentioned above.

When dealing with FS domains represented as bounds ordered by the $\subseteq$ relation, the bounds (glb/lub) cannot, in general, be extended to a complete assignment in the problems that we consider because of the presence of cardinality restrictions. e.g. $X \subseteq \{1, 2, 3, 4\}, |X| = 2$, not all subsets of $\{1, 2, 3, 4\}$ have 2 elements. [2] introduces a local consistency notion for various binary and ternary set relations that ensures the ordering and depending on the constraint relation a certain level of consistency is reached (e.g. AC for the set inclusion). [8] extends the consistency notions of [2] to multi-sets (sets where an element may

occur more than once) and combines them with the standard bounds consistency notions for FD into a level of consistency called BC which can be applied to constraints involving all the three types of variables. For FD variables the definition is exactly that of standard FD "bounds consistency" and for (multi-)set variables the "bounds" which are required to be "consistent" (i.e. extendable to complete assignment) are *not* the glb/lub but correspond to the bounds on the number of times any given element may occur within a set. For simple sets these bounds are always 0..1.

3 Global Set Constraints

To strengthen set constraint propagation in the presence of set cardinality information we first investigated global set constraints, seeking tractable and effective global filtering algorithms for n-ary set constraint over fixed cardinality sets. For practical modelling reasons FS solvers provide a number of n-ary constraints like `all_disjoint`, `all_union` which are syntactic abstractions for a collection of respectively binary and ternary constraints ($X_1 \cap X_2 = \emptyset$, $X_1 \cup X_2 = X_{12}$). The constraint reasoning is based on local bounds consistency.

[8] shows that BC on n-ary `all_disjoint` is equivalent to BC on the decomposition. This holds because any set can be assigned the empty set. However, when the set cardinalities are constrained (and not zero), which is ubiquitous in combinatorial design problems, the equivalence no longer holds.

Using some standard results from design theory we derived four global conditions which must hold for disjoint sets of fixed cardinality. Using an extension of Hall's theorem[9] we proved that these conditions, if satisfied, were sufficient to ensure BC, and were able to convert the proof procedure into an efficient polynomial time algorithm[1] to enforce said consistency level. Interestingly this implementation corresponds closely to the GAC algorithm for the Global Cardinality Constraint of Régin[10]. Owing to space restrictions we refer the reader to [11] for details.

Other Global Constraints Despite the existence of a polynomial filtering algorithm for the global disjoint constraint, we believe it unlikely that such BC algorithm exist for the more general case of global cardinality-intersection constraints, like the `atmost1` constraint of [12]. Despite large amount of work being done on the problem, to date, it is not known whether some relatively small instance of Steiner systems (which the `atmost1` constraint models) exist or not, e.g. $S(4, 5, 17)$ and $S(5, 6, 17)$.[2] These open instances lend weight to our belief.

Another approach is thus necessary to make active use of the cardinality information for arbitrary set constraints in an efficient and effective manner.

[1] $O(ncv\sqrt{nc})$ where n=num vars, c=cardinality and v=size of largest lub
[2] See Sect. 6 for explanation of notation

4 Lexicographic Bounds - The FD Analogy

The motivation that lead us to consider lexicographic bounds to represent set variables is two fold: 1) to keep a compact representation for set variables, 2) to build upon the analogy with bounds reasoning for integer variables and its efficient and effective constraint propagation, e.g. [13].

If we think of a FD variable as a FS variable constrained to have exactly 1 element, then the domain of the FD variable corresponds directly to the lub of the FS variable. The min/max bounds of the FD domain are the smallest/largest elements in the lub. Extending the idea of min/max bounds to FS variables with arbitrary (and non fixed) cardinalities will require a suitable total order on the FS domain elements (as $\leq$ totally orders the FD domain elements).

We propose a new bounds representation for set domains based on an ordering different from the set inclusion (subset order). The ordering is lexicographic and we define *lexicographic bounds* denoted $\langle \inf, \sup \rangle$. This ordering relation defines a *total* order on sets of natural numbers, in contrast to the *partial* order $\subseteq$. We use the symbols $\prec$ (and $\preceq$) to denote a total strict (respectively non-strict) lexicographic order.

Definition 1. *Let $\preceq$ be a total order on sets of integers defined as follows*

$$X \preceq Y \textit{ iff } X = \emptyset \vee x < y \vee \big(x = y \wedge X \setminus \{x\} \preceq Y \setminus \{y\}\big) \quad \textit{where } x = max(X) \textit{ and } y = max(Y) \quad (1)$$

Example 1. Consider the sets $\{1,2,3\}, \{1,3,4\}, \{1,2\}, \{3\}$, the list that orders these sets w.r.t. $\preceq$ is $[\{1,2\}, \{3\}, \{1,2,3\}, \{1,3,4\}]$.

This lexicographic ordering for sets is not the only possible definition, nor is it, perhaps, the most common when talking about sets. We use this definition for two reasons: 1) for sets of cardinality 1 it is equivalent to the $\leq$ ordering of FD variables and 2) usefully, it extends the $\subseteq$ ordering and we have:

Theorem 1. $\forall X, Y \in \mathcal{P}(\mathbb{N}) : X \subseteq Y \Rightarrow X \preceq Y$

Proof. If $X \subseteq Y$ then either $X = \emptyset$ in which case $X \preceq Y$ for all Y, or $\emptyset \subset X \subseteq Y$ in which case consider the max elements of X and Y (namely $x = max(X)$ and $y = max(Y)$ resp.). Since $X \subseteq Y$ we have that $x \leq y$ because X contains no elements greater than those in Y, so if $x < y$ then clearly by definition $X \leq Y$ and we are done. If $x = y$ then we consider the next largest elements in each set and our arguments hold recursively (since sets are finite). □

Theorem 1 will be used in the hybrid domain to make inferences between the two bounds representations for set variables (we also use this equivalent implication with the direction reversed $\forall X, Y \in \mathcal{P}(\mathbb{N}) : X \not\subseteq Y \Leftarrow X \not\preceq Y$).

This ordering defined on ground sets of integers is not new; it is simply the standard arithmetic ordering ($\leq$) on the natural numbers written in binary, where the binary number corresponds to the 0/1 characteristic vector representation of the ground set. More explicitly, for any two ground sets of integers

X and Y, and their corresponding characteristic binary numbers $\boldsymbol{X}$ and $\boldsymbol{Y}$, we have the following equivalence

$$X \preceq Y \textbf{ iff } \boldsymbol{X} \leq \boldsymbol{Y} \tag{2}$$

Example 2. Consider $X = \{4, 2\}$ and $Y = \{4, 3, 2\}$, equiv $\boldsymbol{X} = [0, 1, 0, 1, 0]$ and $\boldsymbol{Y} = [0, 1, 1, 1, 0]$.

Clearly we have $X \preceq Y$ and $\boldsymbol{X} \leq \boldsymbol{Y}$.

A common use of this ordering is in search problems to break symmetries (e.g. [14] on SAT clauses or [5, 6] on vectors of FD variables). It is important to understand that this is *not* the use to which we put the ordering here. We use this order on *ground* sets as a means to approximate the domain of a Finite Set variable by upper and lower bounds w.r.t. this order. We will show in a later section how we can implement a constraint to enforce the order between FS vars.

Figure 1 shows the relationship between the partial inclusion order and our total lexicographic order.

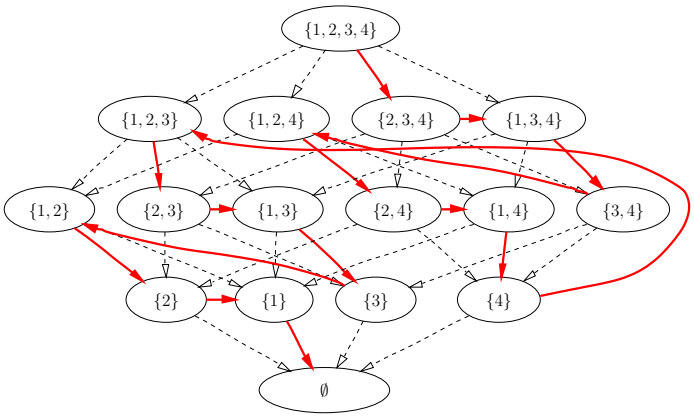

Fig. 1. A graph showing the $\prec$ order (solid) superimposed onto the standard $\subset$ lattice (dashed)

Intuitively, the relationship between the two ordering relations is best viewed by moving downwards from the top of the lattice. Each horizontal line represents sets incomparable with $\subset$ relation. On the other hand, one can follow the directed arc in bold starting from $\{1, 2, 3, 4\}$ to create the totally ordered list of sets under $\prec$ from the greatest to the smallest:[$\{4, 3, 2, 1\}$, $\{4, 3, 2\}$, $\{4, 3, 1\}$, $\{4, 3\}$, $\{4, 2, 1\}$, $\{4, 2\}$, $\{4, 1\}$, $\{4\}$, $\{3, 2, 1\}$, $\{3, 2\}$, $\{3, 1\}$, $\{3\}$, $\{2, 1\}$, $\{2\}$, $\{1\}, \emptyset$].

Note that the sets above have been written with their elements in arithmetic *de*creasing order and observe that all sets "beginning" with a common sequence (e.g. all sets beginning with $\{4, 3\}$) are to be found together. Similarly all beginning with $\{3\}$ are together, though not all the sets *containing* $\{3\}$. It is this

grouping property of the lex order, combined with its extension of the $\subseteq$ order (Theorem 1) that will form the basis of the hybrid inference rules in the next section.

The following table summarizes, the different domain approximations at hand. We use $[glb, lub]$ to denote the set of all sets which contain glb and are contained in lub. We use $\langle inf, sup \rangle$ to represent the set of all sets which come after inf and before sup in the $\preceq$ order.

type	domain	order	minimal	maximal
FD	$\mathbb{N}$	$\leq$ (total)	min	max
FS	$\mathcal{P}(\mathbb{N})$	$\subseteq$ (partial)	glb	lub
FS (lex)	$\mathcal{P}(\mathbb{N})$	$\preceq$ (total)	inf	sup

Given that the lexicographic order embeds the partial inclusion order (Theorem 1), one could wonder whether it can replace it altogether.

Pros The lex bound domain overcomes one major weakness of the subset bound domain, in that it allows us to make more active use of the cardinality constraint. Since the lex bounds are valid instantiations of the set variable, then any condition which must hold for the set variable (e.g. constraints on the set variable, like the cardinality constraint) can be enforced on the lex bounds.

Example 3. A set X is known to take two or three elements from $\{5, 4, 3, 2, 1\}$. The subset bounds representation can not yield tighter bounds when considering the cardinality constraint, i.e. $X \in [\emptyset, \{5, 4, 3, 2, 1\}]$. However, with the lex bound representation, we can prune the bounds. Let the initial bounds describe the same initial domain $X \in \langle \emptyset, \{5, 4, 3, 2, 1\} \rangle$ ($2^5 = 32$ unique sets). When propagating the cardinality constraints, we are able to tighten the domain to $\langle \{2, 1\}, \{5, 4, 3\} \rangle$, (26 unique sets). If now cardinality is bound to be exactly 2 then we have bounds $\langle \{2, 1\}, \{5, 4\} \rangle$ which corresponds to only $\binom{5}{2} = 10$ sets.

Cons Despite its success allowing cardinality constraint to filter the domain more actively, the lex bound representation is unable to always represent certain critical constraints. Primary amongst these constraints is the inclusion or exclusion of a single element. Such constraints are not always representable in the domain because the lex bounds represent possible set *instances* and not *definite* and *potential* elements of a set.

Example 4. Consider the bound constraint $X \in \langle \emptyset, \{4, 3, 2, 1\} \rangle$. The constraint $1 \in X$ yields new bounds of $X \in \langle \{1\}, \{4, 3, 2, 1\} \rangle$, unfortunately not all sets which lie in this range contain the element 1 (eg. $\{3, 2\}$). Note however that the constraint $4 \in X$ allows us to prune the bounds to $X \in \langle \{4\}, \{4, 3, 2, 1\} \rangle$ where all the sets in the range *do* contain 4 (see Fig. 1 for a visual proof).

It is the inability to capture such fundamental constraints efficiently in the domain which lead us to consider a hybrid domain of both subset and lexicographic bounds.

5 Hybrid Domain

In this section we extend the subset domain representation with extra bounds representing the lexicographically smallest and largest instantiations of the set, as well as bounds for the cardinality. We give extra rules to be used in addition to those given for subset domains in [15]. Taken together these rules are necessary and sufficient to maintain consistent hybrid domains w.r.t. the constraint store.[3]

We represent the bounds which constitute the domain of a variable as $X \in [a_X, b_X]\,|c_X, d_X|\langle e_X, f_X\rangle$, where: a_X, b_X are lower,upper bound w.r.t. $\subseteq$, c_X, d_X are lower,upper bound w.r.t. $|X|$ (cardinality), e_X, f_X are lower,upper bound w.r.t. $\preceq$. We will in fact for the sake of brevity, overload the $\in$ symbol further and use $X \in [a_X, b_X]\,|c_X, d_X|$ to indicate that the variable X lies within the lattice $[a_X, b_X]$ *and* has cardinality in the range $c_X..d_X$.

We use the above naming convention, where the letters a, b, c, d, e, f are suffixed by the set variable names (which will be one of X, Y, Z). When we refer to numeric elements of the domain we use the lowercase letter x, when we refer to set values from the domain we use the lowercase letter s. We adopt the operational semantics style of [16] and present our inferences as rewrite rules operating on a constraint store. The rewrite rules have the form

$$\frac{\text{inference}}{\{\text{Old store}\} \longmapsto \{\text{New store}\}}$$

however to save space we may omit the domain constraints from the stores (e.g. $X \in [a_X, b_X]\,|c_X, d_X|\langle e_X, f_X\rangle$) and adopt the notation that any "primed" bound (e.g. a'_X) appearing in the inference indicates the new value of that bound in the new store. Furthermore when the old and new stores contain the same constraints we will give only the inference, with the common store being shown in the section heading. Finally, the special constraint $tell(\ldots)$ is used to indicate the addition of a new constraint, allowing special actions to be performed when constraints are setup.

5.1 Intra-Domain Consistency - $\{X \in [a_X, b_X]\,|c_X, d_X|\langle e_X, f_X\rangle\}$

There follows a number of inference rules designed to keep the various bounds of our hybrid domain mutually consistent. There will be one rule associated with each of the six bounds, followed by one rule indicating failure.[4]

`IR 1.` $a'_X = a_X \cup \{x \big| x \in e_X \cap f_X \wedge \forall_{x' \in (e_X \cup f_X) \setminus (e_X \cap f_X)}\ x' < x\}$

`IR 1` states, in essence, that any elements which form a common "beginning" to both lex bounds (see Sect. 4), should be part of the glb.

`IR 2.` $b'_X = b_X \setminus \big\{x \big| \{x\} \cup a_X \succ f_X \vee \big(d_X - |a_X| = 1 \wedge \{x\} \cup a_X \prec e_X\big)\big\}$

[3] Note that the way the new bounds are actually computed is not presented here. This depends on ones choice of data structures and generic fixed point algorithm.

[4] Instantiation when $e_X = f_X$ is guaranteed by `IR 1`, `IR 2` and the rules of [15].

`IR 2` tells us when elements can never be part of the set because their inclusion would violate the lex bounds. There are two such cases, indicated by the disjunction in the definition of the set of elements to exclude.

- Firstly, no element can be included, which if added to the *glb* would cause it to be greater than ($\succ$) the lex upper bound f_X. This follows from Theorem 1.
- The second case arises when there is at most one more element which *could* be added to the set (i.e. when $d_X - |a_X| = 1$), in such a situation any potential element if added to the *glb* must not cause it to be less than ($\prec$) the lex lower bound e_X.

`IR 3.` $c'_X = \begin{cases} max(|a_X|, c_X) & \text{if } a_X = e_X \\ max(|a_X|+1, c_X) & \text{otherwise} \end{cases}$

When e_X and a_X coincide (are equal), then the cardinality is at least the number of elements in a_X. When they do not, then we know that X must contain a_X, but cannot be exactly a_X (since $a_X \prec e_X$), hence the cardinality is at least the number of elements in $a_X + 1$.

`IR 4.` $d'_X = \begin{cases} min(|b_X|, d_X) & \text{if } b_X = f_X \\ min(|b_X|-1, d_X) & \text{otherwise} \end{cases}$

A similar argument holds for `IR 4` as holds for `IR 3`.

`IR 5.` $e'_X = \inf\left(\{s | s \in [a_X, b_X]\,|c_X, d_X| \wedge s \succeq e_X\}\right)$

`IR 6.` $f'_X = \sup\left(\{s | s \in [a_X, b_X]\,|c_X, d_X| \wedge s \preceq f_X\}\right)$

Together, `IR 5` and `IR 6` ensure that the lex bounds of the domain can only undergo monotonic reduction.

`IR 7.` $$\frac{e_X \succ f_X \vee e_X = \inf(\emptyset) \vee f_X = \inf(\emptyset)}{\left\{\, X \in [a_X, b_X]\,|c_X, d_X|\langle e_X, f_X\rangle \,\right\} \longmapsto \left\{\, \texttt{fail} \,\right\}}$$

If the domain becomes empty, or no values exist for the new lex bounds then clearly we should fail.

5.2 Constraints

Inclusion - $\{X \subseteq Y\}$ Strict inclusion ($\subset$) requires strict total orders ($\prec$ and $<$).

`IR 8.` $$\frac{}{\left\{\, tell(X \subseteq Y) \,\right\} \longmapsto \left\{\, X \subseteq Y, X \preceq Y, c_X \leq c_Y, d_X \leq d_Y \,\right\}}$$

Intersection - $\{Z = X \cap Y\}$ Similar rules exist for the variable Y.

`IR 9.` $$\frac{}{\left\{\, tell(Z = X \cap Y) \,\right\} \longmapsto \left\{\, Z = X \cap Y, tell(Z \subseteq X), tell(Z \subseteq Y) \,\right\}}$$

`IR 10.` $e'_X = \inf(\{s | s \in [a_X, b_X]\,|c_X, d_X| \wedge |s \cap a_Y| \leq d_Z \wedge |s \cap b_Y| \geq c_Z\})$

`IR 11.` $f'_X = \sup(\{s | s \in [a_X, b_X]\,|c_X, d_X| \wedge |s \cap a_Y| \leq d_Z \wedge |s \cap b_Y| \geq c_Z\})$

Union - $\{Z = X \cup Y\}$ Similar rules exist for the variable Y.

IR 12. $\dfrac{}{\{\, tell(Z = X \cup Y)\,\} \longmapsto \{\, Z = X \cup Y, tell(X \subseteq Z), tell(Y \subseteq Z)\,\}}$

IR 13. $e'_X = \inf(\{s | s \in [a_X, b_X]\, |c_X, d_X| \wedge |s \cup a_Y| \leq d_Z \wedge |s \cup b_Y| \geq c_Z\})$

IR 14. $f'_X = \sup(\{s | s \in [a_X, b_X]\, |c_X, d_X| \wedge |s \cup a_Y| \leq d_Z \wedge |s \cup b_Y| \geq c_Z\})$

Difference - $\{Z = X \setminus Y\}$

IR 15. $\dfrac{}{\{\, tell(Z = X \setminus Y)\,\} \longmapsto \{\, Z = X \setminus Y, tell(Z \subseteq X)\,\}}$

IR 16. $e'_X = \inf(\{s | s \in [a_X, b_X]\, |c_X, d_X| \wedge |s \setminus b_Y| \leq d_Z \wedge |s \setminus a_Y| \geq c_Z\})$

IR 17. $f'_X = \sup(\{s | s \in [a_X, b_X]\, |c_X, d_X| \wedge |s \setminus b_Y| \leq d_Z \wedge |s \setminus a_Y| \geq c_Z\})$

IR 18. $e'_Y = \inf(\{s | s \in [a_Y, b_Y]\, |c_Y, d_Y| \wedge |a_X \setminus s| \leq d_Z \wedge |b_Y \setminus s| \geq c_Z\})$

IR 19. $f'_Y = \sup(\{s | s \in [a_Y, b_Y]\, |c_Y, d_Y| \wedge |a_X \setminus s| \leq d_Z \wedge |b_Y \setminus s| \geq c_Z\})$

Ordering Constraint - $\{X \preceq Y\}$

IR 20. $e'_Y = inf(\{s | s \in [a_Y, b_Y]\, |c_Y, d_Y| \langle e_Y, f_Y\rangle \wedge s \succeq e_X\})$

IR 21. $f'_X = sup(\{s | s \in [a_X, b_X]\, |c_X, d_X| \langle e_X, f_X\rangle \wedge s \preceq f_Y\})$

6 Experiments and Comparisons

We implemented our lex bound inferences atop the `ic_sets` library in ECL^{i}PS^e[17], using a list of integers in decreasing order to represent each lex bound. This allows all required new bounds to be calculated in $O(|b_X|)$ time which we believe is close to optimal given the nature of our lex ordering and the presence of cardinality bounds.[5] To illustrate the benefits of our hybrid domain over the conventional subset domain for reasoning about set problems in the presence of cardinality information, we look at error correcting codes and the commonly referenced problem of finding Steiner systems. A 2GHz Pentium 4 with 1GB of RAM was used for all experiments.

Unless otherwise stated the search procedure used in the following problems is the following: Each set variable is fully instantiated before moving to the next, in a fixed order. Each set is instantiated by first trying to include, then on backtracking, exclude the largest unassigned element from its domain.

Definition 2 (Binary error correcting codes). *A binary error correcting code is a collection of bit-strings (vectors of 0s and 1s of length (n)), called codewords, with the property that the* distance *between any two codewords is at least some number (d). The distance between two codewords is defined to be the number of positions in which the two bit-strings vary. This distance function is called the* Hamming *distance.*

[5] Note that subset bounds alone can be updated in $O(1)$ time, e.g. in `ic_sets`.

A variant of this problem is to find codes which have a fixed weight (w), where the weight of a code is defined as the number of 1s that each codeword contains. Each codeword must contain the same number of 1s.

We can model this problem using set variables (S_i) to represent the codewords (C_i), with the correspondence that the codeword represents the characteristic function of the set (i.e. the element x is in the set S_i **iff** the code C_i has a 1 at position x). Using the set model, the distance between two codewords can be defined as the cardinality of the symmetric difference of the two sets. From the basic set constraints presented in this paper (and present in most set solvers), we can define the symmetric difference in a number of ways, but in keeping with [18] we define the distance between two sets as

$$distance(S_i, S_j) = n - |S_i \cap S_j| - |\{1 \dots n\} \setminus (S_i \cup S_j)|$$

As an optimization problem then, the task is to find (and prove) the maximum size (number of codewords $= a(n, d, w)$) for a binary error correcting code with given parameters (n, d) and optionally fixed weight (w).

We solve the optimization problem by simply trying to find increasingly larger codes, and proving optimality by failing to find one larger. We increase the code size by one codeword each iteration.

Figure 2 shows the backtracks and runtimes required to find and prove the optimal code size for non-trivial (i.e. non-zero size) instances of the constant weight error correcting codes optimization problem with parameters $n \in \{6, 7, 8, 9, 10\}$, $d \in \{4, 6, 8, 10, 12\}$ and $w \in \{3, 4, 5, 6, 7, 8\}$. The graphs show those 48 problems which were proved to optimality in under 240 seconds by both subset and hybrid solvers. The results were ordered by the number of backtracks required using the subset domain, thus the problems exhibit a general trend of greater difficulty as the problem number increases.

We observe that the backtracks for our hybrid domain, though occasionally the same as the subset domain in the easier problems, are in general significantly lower.[6] Whilst the runtimes can be seen to be comparable overall, they show

[6] Note the logarithmic scale used on both graphs.

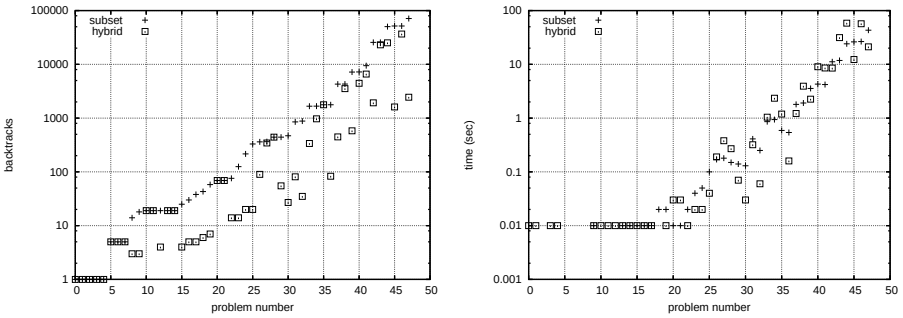

Fig. 2. Backtracks and runtimes for solved fixed weight binary error correcting codes

fluctuations whereby in some instances our approach is slower, and in others it is faster. The former occurs when the reduction in backtracks is insufficient to outweigh the overheads associated with maintaining the stronger consistency, and is entirely expected. The point is well illustrated by the final two pairs of data points on each graph. The final two points on the backtrack graph represent a 96.6% reduction in the number of backtracks, whereas the previous two represent a "mere" 29.2% reduction. The corresponding runtime changes are 50.9% faster and 1114.5% slower. Better CPU times can be expected by investigating different data structures for the hybrid domain bounds and an integrated solver for the whole hybrid domain.

Not shown in the graphs are the optimality proofs that only our hybrid domain solver was able to find in the 240 second time limit. $a(9,4,7) = 4$ was found with 21779 backtracks in 201.71 seconds and $a(10,6,7) = 3$ was found with 14619 backtracks in 111.14 seconds.

Definition 3 (Steiner Systems). *A steiner system $S(t,k,v)$ is a set A of v points and a family of subsets of size k of A (called blocks) such that any t points in A appear in exactly one block.*

To demonstrate the benefits of our approach on existing models we adopt the common Steiner system set model of $\binom{v}{t}/\binom{k}{t}$ set variables representing the blocks of the design, constrained such that the pairwise intersection contains *at most* 1 element. We call this the *primal* model.

Another way to model Steiner systems, and design problems in general using set variables is to employ what we call a *dual* model. Instead of modelling the blocks themselves as set variables, we instead number the blocks $1..b$ and have a set variable corresponding to each point of the base set, which contains the block numbers in which the element occurs. In [12] a global constraint `atmost1` is proposed which does strictly more than just constraining the size of the dual-sets. We find however that the full inferences of this constraint are costly to attain and instead, in our experiments, we settle for a simple redundant constraint that constrains the number of times an element may appear in the design to be exactly r. This second model we refer to as the *+dual sum* model as it can be easily implemented by summing vectors of reified inclusion Booleans. Table 1 clearly shows the benefit that our hybrid domain brings in reducing the size of the search space. In many cases removing backtracks altogether and in others reducing the number by as much as 159 times.

Table 2 shows the computational cost of maintaining this higher level of consistency. In many cases the time taken to find the solution actually increases. This is especially pronounced when the search space is large and solutions are relatively easy to find. Consider the $S(2,3,31)$ system which contains 155 blocks, each of which can be instantiated to one of $\binom{31}{3} = 4495$ values, this constitutes quite a large search space out of which 930 backtracks is a relatively small number. With the "+dual sum" model this instance can be solved without backtracks using the simple subset domain representation and so the extra mechanism for reasoning with the hybrid domain can only add overhead.

Table 1. Backtracks to find first soln.

	backtracks			
	primal		+dual sum	
$S(t,k,v)$	subset	hybrid	subset	hybrid
$S(2,3,07)$	6	0	0	0
$S(2,3,09)$	4521	384	2398	15
$S(2,3,15)$	90	0	0	0
$S(2,3,31)$	930	0	0	0
$S(2,4,13)$	19	0	1	0
$S(2,5,21)$	40	0	0	0
$S(3,4,08)$	60	2	8	2
$S(3,4,16)$	4136	132	240	132
$S(3,6,22)$	3048	42	92	42

Table 2. Time to find first soln.

	time (s)			
	primal		+dual sum	
$S(t,k,v)$	subset	hybrid	subset	hybrid
$S(2,3,07)$	0.01	0.01	0.01	0.01
$S(2,3,09)$	2.95	1.63	3.23	0.13
$S(2,3,15)$	0.41	1.01	0.18	1.06
$S(2,3,31)$	31.3	100.9	6.83	99.63
$S(2,4,13)$	0.04	0.14	0.02	0.14
$S(2,5,21)$	0.16	2.97	0.1	2.83
$S(3,4,08)$	0.05	0.07	0.03	0.08
$S(3,4,16)$	41.59	59.7	7.11	54.69
$S(3,6,22)$	15.29	77.48	2.47	54.98

However, when considering *harder* problems such as the $S(2,3,09)$ instance, 12 blocks each with $\binom{9}{3} = 84$ possible values, the 4521 backtracks is a more significant proportion of the search space. The reduction of this number to 384 by the hybrid domain results in a 44.7% reduction in the runtime. With the "+dual sum" model, the reduction of the backtracks by 99.3% results in runtime reduction of 96.0%.

To investigate whether we had simply been "lucky" or "unlucky" to find (resp. not find) solutions quickly we ran experiments to find *all* solutions to the various designs. Due to the large numbers of (symmetric) solutions that exist for steiner systems, we were only able to find all solutions to the $S(2,3,7)$ in a reasonable time. Table 3 shows, for the primal model, that the overheads associated with our hybrid domains is almost exactly balanced by the reduced search space (87.1% fewer backtracks and 2.5% less runtime). For the "+dual sum" model we observe a 52.4% reduction in backtracks which, given the current implementation, does not come with a reduction in runtime. However, the results are promising.

Table 3. Time and backtracks taken to find all 151200 solutions of $S(2,3,7)$

domain	model	time(s)	backtracks	bt/sol
subset	primal	609	1557048	10.30
hybrid	primal	594	200507	1.33
subset	+dual sum	378	410479	2.71
hybrid	+dual sum	462	195349	1.29

6.1 Symmetry

Much work has been done recently to improve the efficiency of searching for solutions of highly symmetric problems. In this section we compare our work with developments in one particular family of symmetry breaking techniques, the lex-ordered symmetry breaking constraint in matrix models.

In [19] the authors show how existing symmetry breaking techniques like lex[5, 6] can be combined with more conventional constraint like the `sum` constraint to both increase the amount of pruning and (in some instances) reduce the time taken to solve problems. The authors demonstrate their technique on finding and proving the in-existence of a number of small Steiner systems.

The model they choose is a 2D matrix of 0/1 FD variables where rows correspond to the characteristic function of a block, and columns therefore correspond to the dual sets mentioned earlier. A constraint on the magnitude of the scalar product between any two rows, corresponds to the restriction that two sets may intersect in at most 1 element. They compare the effect of posting lex constraints on both the rows and the columns ($>_{lex}$ R $\geq_{lex}$ C), with posting lex on the columns ($\geq_{lex}$ C) and a specialized constraint called `LexGreaterAndSum` on the rows, we will denote this specialized constraint which combines the lex ordering with the sum constraint as ($>^{\sum}_{lex}$ R) for brevity.

For comparison, our model is the same as that presented in the previous section where set variables correspond to rows, with the addition of dual sets (corresponding to the columns). Simple channelling constraints maintain the correspondence between the sets. The lex constraints are enforced locally between adjacent rows and adjacent columns using the inference rules `IR 20` and `IR 21`. The dual sets are not constrained to have a fixed cardinality since no such constraints existed on the columns in the matrix model. We implement the exact same labelling strategies, row-wise and column-wise, as used in [19] by channelling to a matrix of reified inclusion Booleans.

In tables 4 and 5 we duplicate and extend the results of [19], adding for comparison the final column showing how our model performs. Note that the third column contains the backtrack values from the original paper, with the runtimes being scaled by the same factor as the runtimes for the previous column for which we were able to duplicate backtrack counts. From these results our hybrid domain model not only out performs the plain double-lex constrained matrix model in terms of search space reduction and runtimes, but also outperforms the specialized `LexGreaterAndSum` constrained model as well; providing, in the hardest of the problems, a 95.0% backtrack reduction compared to the double-lex model and further 48.5% compared to the specialized `LexGreaterAndSum` model. Runtimes drop by 85.2% and 23.8% respectively as well.

Conclusion The hybrid domain provides a natural data structure for the lex ordering constraints ($\prec$ and $\preceq$) and the set constraints of the problem ($\cap$ and $|\ \ |$) to interact effectively. By keeping the set based model, but enhancing the domain representation and local inferences, we can reason at least as strongly and efficiently as less intuitive FD matrix models and without the need to identify and invent specialized global constraint propagation algorithms.

Table 4. Comparison with table 1 of [19]. Row-wise labelling.

Prob	No sym breaking		$>_{lex} R \geq_{lex} C$		$>^{\sum}_{lex} R \geq_{lex} C$		$\succ R \succeq C$	
$S(t,k,v)$	btracks	time(s)	btracks	time(s)	btracks est	time(s)	btracks	time(s)
$S(2,3,6)$	6194	2.7	13	0.0	11	*0.0*	7	0.0
$S(2,3,7)$	6	0.4	2	0.0	1	*0.0*	0	0.0
$S(2,3,8)$	-	>16hr	740	0.7	390	*0.7*	58	0.3
$S(2,3,9)$	4521	5.6	336	0.5	250	*0.5*	12	0.2
$S(2,3,10)$	-	>16hr	723209	1339.8	433388	*1136.4*	12346	167.8

Table 5. Comparison with table 3 of [19]. Column-wise labelling.

Prob	No sym breaking		$>_{lex} R \geq_{lex} C$		$>^{\sum}_{lex} R \geq_{lex} C$		$\succ R \succeq C$	
$S(t,k,v)$	btracks	time(s)	btracks	time(s)	btracks est	time(s)	btracks	time(s)
$S(2,3,6)$	26351	9.8	46	0.0	27	*0.0*	22	0.0
$S(2,3,7)$	585469	340.6	151	0.1	52	*0.1*	42	0.2
$S(2,3,8)$	-	>16hr	6837	5.5	1962	*3.1*	1314	3.7
$S(2,3,9)$	-	>16hr	90561	98.0	8971	*14.0*	5232	18.0
$S(2,3,10)$	-	>16hr	37861490	48789.8	3701480	*9478.1*	1906918	7226.4

7 Related Work

The most closely related work to ours is that of [16], which extended the inference rules of [15] with extra rules operating on set cardinalities. In our framework we are clearly free to implement all the extra inferences of [16] and hence can reach at least the same degree of pruning. We can however make important inferences that cannot be made in [16].

Example 5. $Z = X \cap Y$, $X \in [\emptyset, \{5,4,3,2,1\}]\,|3,3|$, $Y \in [\emptyset, \{4,3,2,1\}]\,|3,4|$, $Z \in [\emptyset, \{4,3,2,1\}]\,|3,3|$. These domains are a fixed point for both traditional set solvers and [16]. However with our lex bounds added (i.e. for X, we have $\langle\{3,2,1\}, \{5,4,3\}\rangle$), `IR 11` reduces this to $\langle\{3,2,1\}, \{4,3,2\}\rangle$ and `IR 2` gives us $X \in [\emptyset, \{4,3,2,1\}]$.

8 Conclusion

We have analysed Finite Sets solvers in the light of modern techniques and advances in efficiently modeling and solving combinatorial design problems. We presented a new and novel hybrid domain for FS variables which allows us to strengthen the level of consistency that we reach in a tractable and efficient manner. It is clearly stronger than BC achieved by standard set solvers since we prune more but defining the level of consistency reached for such multi-bounded domains is an open problem.

We showed how our implementation prototype is able to improve not only the state of the art in FS solving for a class of combinatorial design problems, but also improves on recently published results that use FD 0/1 matrix models and specialized global constraints. Though sometimes slower than existing FS techniques, our approach is typically faster at solving problems with cardinality constraints especially when there are no (or few) solutions as well as at proving optimality. Furthermore we believe that the CPU times can be improved further for all cases by considering a fully integrated set solver built specifically for hybrid domains (i.e. not atop an existing solver).

References

1. Colbourn, Dinitz, Stinson: Applications of combinatorial designs to communications, cryptography, and networking. In: Surveys in Combinatorics London Mathematical Society Lecture Note Series 187. Cambridge University Press (1999)
2. Gervet, C.: Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language. CONSTRAINTS journal **1(3)** (1997) 191–244
3. Beldiceanu, N., Contejean, E.: Introducing Global Constraints in CHIP. In: Mathematical Computation Modelling. Volume 20(12). (1994)
4. Régin, J.C.: A filtering algorithm for constraints of difference in csps. In: Proc. AAAI-94. (1994)
5. Flener, P., Frisch, A., Hnich, B., Kiziltan, Z., Miguel, I., Pearson, J., Walsh, T.: Breaking row and column symmetries in matrix models. In: Proc. CP'02. LNCS, Springer (2002)
6. Gent, I.P., Prosser, P., Smith, B.M.: A 0/1 encoding of the gaclex for pairs of vectors. In: ECAI/W9 Modelling and Solving Problems with Constraints. (2002)
7. Mackworth, A.: On reading sketch maps. In: IJCAI'77. (1977) 598–606
8. Walsh, T.: Consistency and propagation with multiset constraints: A formal viewpoint. In: Proc. CP-2003. (2003)
9. Hall, P.: On Representatives of Subsets. J. of London Math. Soc. **10** (1935) 26–30
10. Régin, J.C.: Generalized arc consistency for global cardinality constraint. In: Proc. AAAI-96. (1996)
11. Sadler, A., Gervet, C.: Global filtering for the disjoint constraint on fixed cardinality sets. Technical Report IC-PARC-04-2, Imperial College, London (2004)
12. Sadler, A., Gervet, C.: Global reasoning on sets. In: FORMUL'01 workshop in conjunction with CP-01. (2001)
13. Puget, J.F.: A fast algorithm for the bound consistency of alldiff constraints. AAAI (1998)
14. Crawford, J., Ginsberg, M., Luks, E.M., Roy, A.: Symmetry breaking predicates for search problems. In: Fifth Int. Conf. on Knowledge Rep. and Reasoning. (1996)
15. Gervet, C.: Conjunto: constraint logic programming with finite set domains. In: Proc. ILPS-94. (1994)
16. Azevedo, F.: Cardinal: an extended set solver. Computational Logic (2000)
17. Schimpf, J., Cheadle, A.M., Harvey, W., Sadler, A., Shen, K., Wallace, M.: ECL^{i}PS^e. Technical Report 03-1, IC-Parc, Imperial College London (2003)
18. Müller, T., Müller, M.: Finite set constraints in Oz. In: 13. Workshop Logische Programmierung. (1997)
19. Hnich, B., Kiziltan, Z., Walsh, T.: Combining symmetry breaking with other constraints: lexicographic ordering with sums. In: Proc. SymCon. (2003)