Enhancing Set Constraint Solvers with Lexicographic Bounds

Andrew Sadler and Carmen Gervet

Imperial College, London SW7 2AZ, U.K.

Abstract. Since their beginning in constraint programming, set solvers have been applied to a wide range of combinatorial search problems, such as bin-packing, set partitioning, circuit design, and Combinatorial Design Problems. In this paper we present and evaluate a new means towards improving the practical reasoning power of Finite Set (FS) constraint solvers to better address such set problems with a particular attention to the challenging symmetrical set problems often cast as Combinatorial Design Problems (CDPs). While CDPs find a natural formulation in the language of sets, in constraint programming literature, alternative models are often used due to a lack of efficiency of traditional FS solvers. We first identify the main structural components of CDPs that render their modeling suitable to set languages but their solving a technical challenge. Our new prototype solver extends the traditional subset variable domain with lexicographic bounds that better approximate a set domain by satisfying the cardinality restrictions applied to the variable, and allow for active symmetry breaking using ordering constraints. Our contribution includes the formal and practical framework of the new solver implemented on top of a traditional set solver, and an empirical evaluation on benchmark CDPs.

Keywords: Constraint Programming, Artificial Intelligence, modelling languages.

Introduction

Combinatorial Design Problems are NP-hard problems which have applications in areas as diverse as error-correcting codes, sport scheduling, Steiner systems and more recently networking and cryptography (e.g. see [13] for a survey). They deal essentially with the search for families of sets with certain properties subject to constraints including intersection and cardinality restrictions. Such problems find a natural and concise formulation in the language of Finite Sets but they exhibit properties that make them hard to solve with existing Finite Set solvers. While a combinatorial design problem is defined in terms of discrete points, or sets, in the Constraint Logic Programming 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.

Consider the domain of a set variable known to be a subset of $\{1, ..., 100\}$. There are $2^{100} = 1267650600228229401496703205376$ possible values for this set. Clearly an explicit exhaustive enumeration of domain values is space prohibitive.

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 set domain attached to X specified by the interval $[glb(X), lub(X)]$ describes the set of values: $\{\{1\}, \{1, 2\}, \{1, 3\}, \{1, 2, 3\}\}$.

The constraint reasoning is based on local bound consistency extended to handle set constraints [43, 21]. Set constraints are built on the set inclusion relation together with set operations symbols $\cap, \cup, \setminus, ||$ (cardinality) applied to set variables. We refer to such finite set constraint systems as "traditional subset bound solvers", which are basically in the `Conjunto` style described in [23]. The subset 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 can cause important inferences to be missed and in general the bounds are not possible solution values. *This is one of the three main deficiencies we have identified in traditional subset bound solvers.*

The second one concerns the reasoning with graded functions, such as the set cardinality or weight constraints. The basic propagation rules of such solvers did not make active use of the cardinality information. The cardinality function ($||$) maps from a set of values to a single value. In the context of subset bound solvers it allows the modeller to tackle a wider range of problems than Finite Set constraints alone including optimization problems. For example if we consider the system of constraints: $X, Y \subseteq \{a, b, c, d\}, |X| = 2, Z = Y \setminus X$, one should be able to infer that $|Z| \leq 2$, which is not the case in traditional subset bound solvers. By separating, for example, the cardinality information from the domain of the set variable, the system is depriving itself of potentially valuable information which could be used to speed the search for a solution.

The third one is strangely also one of their advantages. By choosing to model problems with finite set variables, as opposed to many other possible approaches, one is often avoiding the introduction of symmetries into the model itself. For instance, instead of having a single variable s_1 to represent a 3 element set, one could model the set as a vector of FD variables (x_1, x_2, x_3) with each variable v_i being used to hold a single element of the set when it becomes known. In such a model however, the vectors $(1, 2, 4)$ and $(4, 2, 1)$ would both represent the same set. This is an example of a symmetry induced by the model. Clearly, when sets are core to the problem structure a set model is at once close to the high-level natural description and less prone to modelling induced symmetries. However,

there are other types of symmetries such as sets-of-sets symmetries which often occur in the problems that we consider. Traditional subset bound solvers do not allow set variables to contain other set variables as elements. Hence such modelling symmetries can still exist at the set variable level.

In this paper, we present a novel approach to strengthen constraint propagation and reduce symmetries actively. Our enhanced Finite Set solver features a "hybrid" domain representation. The idea, first presented in [49], consists of strengthening the propagation of FS constraints in a tractable way, by extending the concept of subset bound domain. We introduce lexicographic bounds to more closely approximate the true domain of a set variable, specially by guaranteeing that the lexicographic bounds satisfy the cardinality restrictions (unlike the subset bounds), and thus strengthen constraint propagation. This total ordering also allows us to curtail set symmetries in a simple and effective manner using lexicographic constraints (unlike subset bounds).

This paper is mainly based on the Ph.D. dissertation of A. Sadler [46]. It is structured as follows. In section 2 we survey recent advances in FS solvers, that address some of the deficiencies we described, often in ways complementary or orthogonal to our approach. Section 3 defines the class of problems we address. Section 4 introduces the concept of hybrid set domain that enriches traditional subset bounds with lexicographic bounds. In section 5 and 6 we describe the operational semantics and implementation of the new solver. Section 7 evaluates the approach. We give some directions for further developments in section 8 and conclude in section 9.

1 Related work

Subset bound solvers have shown their flexibility and modelling abstraction in representing discrete optimization problems (e.g. [2, 4, 15]). The solver removes inconsistent set values by pruning elements of the subset domain bounds that can never be part of any feasible solution [43, 21, 23]. As the use of subset bound solvers has increased and new application domains have emerged as candidates for set solvers, their limitations have become clearer. These are chiefly due to the poor domain approximation, the ineffective use of set cardinality information, and the difficulty of curtailling set symmetries.

Though implementation details vary, at their core the finite set solvers of `SOLVER` [29], ECL^{i}PS^e [50], `FACILE` [3], `MOZART-OZ` [39, 40], `B-Prolog` [59] and `CHOCO` [33], all have the subset bounds as domain representation. In recent years, much research has focused on improving finite set solver effectiveness. It explores i) new set cardinality inferences, ii) the development of global set constraint, iii) the search for more expressive set domain representations, iv) the application of symmetry breaking techniques to sets.

The `Cardinal` solver of Azevedo is a subset bound solver, which offers enhancements intended to strengthen the use of the cardinality information [1]. Traditional set solvers such as `Conjunto` have a cardinality component but use the cardinality information (and other graded functions like weight) in a uni-

directional way, meaning that when a set domain gets refined its cardinality is pruned. Azevedo's work exploits new inferences with application to digital circuit diagnosis and warehouse location [2, 1]. The main strength of `Cardinal` is the stronger pruning of the cardinality component through additional cardinality inferences associated with each basic set operation. The solver combines arc consistency and bound consistency on the set cardinality. While this enables to detect unfeasibility in cases and reduces the cardinality bounds further, it affects the set bounds essentially upon instantiation. The system cannot create a strong interaction between the cardinality component and the set bounds component because of the set domain representation (sets of equal size are incomparable). A total lexicographic ordering addresses this point in particular by representing the cardinality information in the set domain itself (i.e. lex bounds). Also, the cardinality inferences do not address the issue of active domain pruning using symmetry breaking constraints.

The work on global set constraint propagators is also motivated by constraint reasoning in the presence of cardinality information together with improving domain approximation. It focuses mainly on deriving effective global propagators for symbolic constraints which are most common in discrete search problems involving sets, such as `atmost1, distinct`, stating respectively that n sets of known cardinality should intersect pairwise in atmost one element, or not be equal [47], or the `disjoint` and `partition` constraints for sets of known cardinality [29, 48, 7]. Theoretical results helped determine whether such global constraints could infer anything more than their decompositions [58], as well as determine the tractability of a range of global set constraints [7]. Their practical effectiveness yet remains to be shown.

Regarding the effectiveness of set domain representation, Hawkins, Lagoon and Stuckey propose a radically different approach to the standard subset bound domains [34]. They show that Reduced Ordered Binary Decision Diagrams (ROBDDs) [9] can be used to represent full domains efficiently. Basically they consider a set domain with all its possible values and use existing efficient C libraries to represent such a domain with ROBDDs and demonstrate techniques for combining ROBDDs in ways that correspond to basic FS constraints (e.g. $\in, ||, \subseteq, \cap, \cup$). Recently, the ROBDD approach was evaluated on different set domain representations showing the flexibility of the approach to implement any set domain representation [26], including the lexicographic bounds first introduced in [49]. The authors carried out a comprehensive and comparative evaluation of the different domain representations using ROBDDs on several standard benchmark set problems [26]. The high performance results when compared with common set solvers demonstrate, in particular, the synergy between the ROBDD data structure and most domain representations including lexicographic bounds. Note that this synergy is strongly dependent on the relative ordering of ROBDD variables when representing set constraints. An optimal ordering of the boolean variables guarantees a linear representation for binary and ternary set constraints except for the cardinality which requires a quadratic number of ROBDD nodes [26].

The identified weaknesses of traditional subset bound solvers have been partially overcome via these extensions. While we first researched towards deriving global set constraints to overcome some of these weaknesses [47, 48], the analysis of the characteristics and behaviour of the global constraints lead us to propose a more general and extensible approach to achieving our aims. The approach presented in this article does not alter the set based constraint model and is complementary to the `Cardinal` approach (the rules of Cardinal can be added if necessary), and orthogonal to the ROBDD approach and global set constraint approach.

Finally, it is worth mentioning recent advances in symmetry breaking techniques to sets [4, 16, 51]. In particular, [51] prove that a successful and tractable symmetry breaking technique during search over integer domains based on dominance detection, becomes NP-hard for set models that contain set variables and value symmetry. This could suggest that adding static symmetry breaking constraints to the model can be a practical venue for set variable symmetries. Our approach goes in this direction.

2 Challenging problems

To this date, set solvers have been applied to tackle small and large size combinatorial search problems ranging from bin packing [21], set partitioning [41, 23], digital circuit [2], and more recently combinatorial design problems [4, 34, 26, 46]. We focus here on CDPs, because they have application in very diverse areas, find a natural formulation in the language of sets, but are especially hard to solve with existing set solvers. We identify the key properties they feature, as a preliminary step towards improving set solvers suitability.

Combinatorial Design Problems (CDP) find a natural formulation in the language of sets: "The concept of configuration is a mapping of a set of objects into a finite abstract set with a given structure" [6]. When the finite abstract set is composed of subsets each of fixed size k, these subsets are known as blocks. Traditional instances of CDPs that involve blocks are often found in combinatorial and design theory [12], but real world problems such as tournament and conference scheduling [10], and cryptography [13], also frequently exhibit this structure. Their significance is felt acutely in real world applications such as optical network design and routing (WDM, DWDM, SONET). CDPs are amongst the most challenging problems to solve due to their structural properties and symmetric setting. While the approach presented in this article is suitable for any set problem, it was first motivated by the concise formulation, yet inherent difficulty set solvers have when addressing combinatorial design problems. In this section we study the structural properties of CDPs from a set modelling perspective pointing at the challenges they raise for set solvers.

Combinatorial design falls under the combinatorics branch of discrete mathematics. It concerns the enumeration side of combinatorial design analysis, where it is not sufficient to just know that a configuration exists, but it is also necessary

to see the configuration explicitly instantiated. That is, to see the arrangements of objects in the finite abstract set.

Definition 1. *(Combinatorial Design Problem - CDP) Characterised by the base set of objects (A) and the collection of constraints (C) that define the structure of an abstract set (S), find (A, S), a mapping (or assignment) of elements from A into S such that C is satisfied.*

A solution to a CDP therefore, is an assignment of base objects into the structured set such that the constraints are satisfied. The following network design example illustrates, in more concrete terms, the type of problem that can be classified as a CDP.

The SONET problem arises from the deployment of transmission technology over optical fibre networks .

> The network contains a number of client nodes and there are known demands (in terms of numbers of channels) between pairs of nodes [55, 54]. A SONET ring joins a number of nodes; a node is installed on a SONET ring using a 'add-drop multiplexer' (ADM) that is capable of adding and dropping the traffic (quite prohibitive). Each node can be installed on more than one ring, and traffic can be routed between a pair of client nodes only if they are both installed on the same ring: there is no traffic allowed between rings. There are capacity limits on the rings (in terms of both nodes and channels). The objective is to minimise the total number of ADMs required, whilst satisfying all demands.

For a given number of ADMs, the problem becomes one of satisfaction i.e. can the given traffic demands be met, and how many rings are required. In this satisfaction problem, the base set of objects are the client nodes, which must be assigned to the structured set of SONET rings. The CDP constraints are simply the constraints of the problem mapped in the natural way. Using this CDP as an example we will now briefly introduce four key properties which CDPs exhibit and which can be exploited to more efficiently solve them. Not all CDPs have these properties, and those that have some may not have all. However, the following characteristics are to be found in most CDP problems.

Let us first clarify what we mean by the term *structured set* as it has bearing on the properties we define. A set s is structured when it contains semantically significant subsets that correspond naturally to objects in the problem domain (e.g. one subset per ring, in the above example). Given this structure, a solution to a CDP can be viewed in one of two ways: 1) as a mapping from base objects to named (uniquely identified) subsets of s, or 2) as a mapping from named subsets to base objects (i.e. the direction of mapping is reversed). In the above example, we could express the solution assignment as a mapping from client nodes to named rings, or named rings to client nodes. This observation leads to the simple notion of duality.

Dualisable. Informally, the dual of a CDP is another CDP, where the role of base objects and named sub-structures have been reversed. The base objects of the former (primal) CDP take the place of named sub-structures in the structured set of the dual CDP. Similarly the named sub-structures of the primal CDP's structured set become the base objects of the dual CDP. Since the primal and the dual are CDPs, we can model both naturally with set constraints, and use either to our advantage when searching for a solution.

Cardinality restrictions. Another consequence of having identified semantically significant sub-structures is that the constraints of the problem will almost always impose restrictions on the sizes of these sub-structures, for example, the maximum number of ADMs on any given ring. These problem constraints translate to cardinality constraints in the set model, either fixing or otherwise constraining the cardinality of set variables. As has been mentioned earlier, existing solvers do not reason strongly with these sorts of graded function constraints.

Intersection constraints. Often the relationship between sub-structures will be defined in terms of the elements that they have in common. For example if there is a traffic demand between two nodes, then the rings over which that traffic is routed must be accessible to both source and destination nodes. In the dual model for the above example, the rings on which two routers sit (i.e. the dual sub-structures) must have a non empty intersection if there is a traffic demand between them. Such intersection constraints are often combined with cardinality constraints (e.g. `non` empty intersection:: $|s \cap s_1| \neq 0$) and, as mentioned above, such propagation can be improved.

Symmetries. Finally, the problem of inherent problem symmetries such as set-of-sets symmetries apply to many CDPs. For example in the above problem, we can permute the ring sub-structures of any valid assignment and we would still get a valid assignment.

Existing computer-based approaches to tackle CDPs with *complete solvers* were proposed early nineties and are essentially based on Integer Programming (IP) and finite domain Constraint Programming (CP). Each paradigm requires a high level of expertise to derive a good model, and often relies on hybridization to improve effectiveness. Both use 0-1 matrices based on the characteristic vector representation of a set (e.g. [24, 55, 37, 53, 54]).

It is our initial motivation to be able to express CDPs by natural and intuitive Finite Set models and have these models solved effectively. To achieve this we need to improve the ability of FS solvers to reason with the four commonly occurring CDP properties mentioned above: dualisable, cardinality and intersection restrictions, and value and set symmetries. Our contribution is to actually do so by enriching the expressiveness of a set domain.

3 Hybrid set Domain

Notations. We use the following 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. Throughout the paper, we use the abbreviation FD for finite (integer) domain and FS for finite set domain.

This section presents the new concept of hybrid set domain: a new type of bounds for the approximation of FS variable domains. In addition to approximating the actual domain, these bounds represent actual set instances which satisfy the *cardinality restrictions*. To the best of our knowledge, this is the first time that a domain approximation with this property has been used for Finite Set solvers.

3.1 Lexicographic Bounds - The FD Analogy

The use of lexicographic bounds to represent set variables allows us to: 1) keep a compact representation for set variables, 2) build upon the analogy with bounds reasoning for integer variables and its efficient and effective constraint propagation e.g. the `all_different` constraint of [44]. Indeed, 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). The bounds representation $[min, max]$ for integer variables is space and time efficient for simple operations.

Recently, new algorithms for global constraints showed how Generalized Bounds Consistency (bounds consistency for n-ary constraints) offers a great balance between effectiveness in constraint propagation and efficiency (e.g [45]). This has been possible for two main reasons:

1. There is a total order on the elements of the interval domain, allowing for the definition of a total order among the variables based on their bound representation.
2. Integer variables take as assignment a single value, thus the bounds represent potential assignments (unlike subset bounds).

We now investigate which of those strengths we can expect to inherit with a proper representation of totally ordered set bounds. We propose a new bounds representation for set domains based on an ordering different from the subset order. The ordering is lexicographic and we define lexicographic bounds denoted $\langle inf, sup \rangle$. This new 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 2. *Let $\preceq$ be a total order on sets of integers defined as follows*

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

max denotes the arithmetically largest element of X or Y.

Example 1. Consider the sets $\{1,2,3\}, \{1,3,4\}, \{1,2\}, \{3\}$, the list that orders these sets with respect to $\preceq$ 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. Its use comes from 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. $\forall X, Y \in \mathcal{P}(N) : X \subseteq Y \Rightarrow X \preceq Y$

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 $x \leq y$. If $x = y$ then we consider the next largest elements in each set and our arguments hold recursively (since sets are finite). □

Theorem 1 is subsequently 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 $\forall X, Y \in \mathcal{P}(N) : x \not\subseteq Y \Leftarrow X \not\preceq Y$). Consider the following example, as an illustration of what can be inferred.

Example 2. Suppose we know that a particular set X is lexicographically bounded upwards by the set $\{3\}$ (i.e. $X \preceq \{3\}$). The above inference tells us that any set strictly greater than $\{3\}$ can never be a value for X. By definition, any set containing any number greater than 3 ($>$) is lexicographically greater ($\succ$) than $\{3\}$, and hence we can conclude that such numbers can never occur in X and should be removed from the *lub*.

If we view sets from their characteristic functions (i.e. 1 if an element is in the set, 0 otherwise), our ordering can be considered the same as the decreasing lexicographic order applied to the zero-padded 0/1 characteristic vector representation of a ground set.

More explicitly, for any two sets of integers X and Y, and their corresponding characteristic vector representation $\mathbf{X}$ and $\mathbf{Y}$ in decreasing order, we have the following equivalence:

$$X \preceq Y \textbf{ iff } \mathbf{X} \leq_{lex} \mathbf{Y}$$

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

Clearly we have $X \preceq Y$ and $\mathbf{X} \leq_{lex} \mathbf{Y}$

A common use of this ordering is in search problems to break symmetries (e.g. on SAT clauses [14] or on vectors of FD variables [17, 19]). It is important to note that this is *not* the use to which we put the ordering here. We use this ordering on *ground* sets as a means to approximate the domain of a FS 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 variables.

Multisets In the context of multisets, the lexicographic ordering can be extended naturally by considering variants of the "lexicographic elements list" approach (i.e. write out the multiset in decreasing order and compare), or equivalently the "lexicographic occurrence vectors" approach (i.e. instead of occurrence value being simply 0 or 1, it defines the number of times an element appears). This leads to an ordering of multisets based on the ordering of the elements within the multisets ($<$) [31]. Since the element order is simply the natural order ($<$) on integers, this ordering coincides with the lexicographic order we use.

3.2 Comparing Lexicographic and Subset Orderings

Let us consider the subset domain specifying the powerset $\mathcal{P}(\{1,2,3,4\})$. Sets of equal size are incomparable under $\subset$ relation. On the other hand, with the $\prec$ relation we can create the totally ordered list of sets 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 decreasing order, and 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 motivated the use of a co-lexicographic or decreasing ordering, because it is at the heart of the hybrid inference rules.

If the *increasing* lex ordering $\preceq_{inc}$ were considered instead, we would loose the fundamental property of Theorem 1 and there would be no relationship between the subset and lex bounds. For instance, the totally ordered list of sets with this ordering from the greatest to the smallest is: $[\{4\}, \{4,3\}, \{3\}, \{4,2\}, \{4,3,2\}, \{3,2\}, \{2\}, \{4,1\}, \{4,3,1\}, \{3,1\}, \{4,2,1\}, \{4,3,2,1\}, \{3,2,1\}, \{2,1\}, \{1\}, \emptyset]$. We have

$$\{1,3\} \subseteq \{1,2,3\} \not\Rightarrow \{1,3\} \preceq_{inc} \{1,2,3\}$$

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

Pros The lexicographic bounds overcome two major weaknesses of the subset bounds. They enable the active use of 1) the cardinality constraints, and that of 2) symmetry breaking constraints, with a strong and effective interaction between both.

Cardinality reasoning The lexicographic bounds describe set values that satisfy the cardinality constraints, unlike the subset bounds. The core reason is that sets of equal size ordered lexicographically are comparable, which is not the case under set inclusion. Thus the subset bounds can not be pruned subject to cardinality changes, except upon instantiation or unification with another set variable.

Example 4. Let the set X take 2 or 3 elements from $\{5,4,3,2,1\}$. The subset bounds representation can not yield tighter bounds when considering the cardinality restriction. We have $X \in [\emptyset, \{5,4,3,2,1\}]$. However, with the lex bound representation, we can directly 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 $\langle \{1,2\}, \{3,4,5\} \rangle$ (26 unique sets). If the set cardinality becomes exactly 2 then we have the lex bounds $\langle \{1,2\}, \{4,5\} \rangle$ which contains only 22 sets.

Symmetry breaking We illustrate below that applying symmetry breaking constraints between sets using a total ordering that coincides with the set domain ordering guarantees effective propagation. The inclusion (partial) ordering on the other hand, does not exploit symmetry breaking constraints actively.

Example 5. Let the sets X, Y range over a subset domain $[\{\}, \{1,2,3,4,5\}]$. An ordering constraint between X and Y, $X \preceq Y$ (using Definition 2) will not prune the subset bounds essentially because the domain ordering is partial and does not coincide with the ordering constraint. On the other hand, with lex bounds the initial bounds would be identical and the ordering constraint yields the new domains: $X \in \langle \{\}, \{2,3,4,5\} \rangle$, $Y \in \langle \{1\}, \{1,2,3,4,5\} \rangle$. We deduce that X has atmost 4 elements, and Y contains atleast 1 element.

From this example, one can also see the interaction between the ordering constraint and the set cardinality when reasoning upon lexicographic bounds.

Cons Despite its success allowing cardinality and symmetry breaking constraints to filter the set 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 values* and not *definite* and *potential elements* of a set. Thus the subset bound information needs to be kept.

Example 6. Consider the lex bound constraint $X \in \langle \emptyset, \{1,2,3,4\} \rangle$. The constraint $1 \in X$ yields new lex bounds of $X \in \langle \{1\}, \{1,2,3,4\} \rangle$. Unfortunately not all sets which lie in this range contain the element 1 (eg. $\{2,3\}$). However the constraint $4 \in X$ allows us to prune the bounds to $X \in \langle \{4\}, \{1,2,3,4\} \rangle$ where all the sets in the range do contain 4 by definition of the lex ordering.

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

3.3 Domain representation

The hybrid set domain extends the subset domain representation with extra bounds representing the lexicographically smallest and largest instantiations of the set, and bounds for the cardinality of the set.

The following table summarises 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	N	$\leq$ (total)	min	max
FS	$(P)(N)$	$\subseteq$ (partial)	glb	lub
FS (lex)	$(P)(N)$	$\preceq$ (total)	inf	sup

Formally, inf and sup denote the unique meet and join operators in the totally ordered lattice $(\mathcal{P}(\mathcal{N}), \preceq)$ where $\mathcal{P}(\mathcal{X})$ is the powerset of $\mathcal{X}$ such that:

Property 1.
$s \preceq s_1 \Leftrightarrow s = \inf(\{s, s_1\})$
$s \preceq s_1 \Leftrightarrow s_1 = \sup(\{s, s_1\})$

We represent the bounds which constitute the domain of a set 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 the smallest lower/greatest 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 powerset lattice $[a_X, b_X]$ *and* has cardinality in the range $c_X..d_X$.

Definition 3. *A hybrid domain* $D =< [a_X, b_X], |c_X, d_X|, \langle e_X, f_X \rangle >$ *is such that:*

1. $a_X \subseteq e_X \subseteq b_X \ , \quad a_X \subseteq f_X \subseteq b_X$*: The definite elements* (a_X) *are in both lex bounds, and these bounds do not contain any element which is not a possible one* (b_X)
2. $|a_X| \leq c_X, \quad d_X \leq |b_X|$*: The size of the subset bounds is an outerbox for the set cardinality bounds.*
3. $|e_X|, |f_X| \in [c_X..d_X]$*: The size of the lex bounds belongs to the range specified by the cardinality bounds.*

The hybrid domain specifies the set $\{s \mid a_X \subseteq s \subseteq b_X, c_X \leq |s| \leq d_X, e_X \preceq s \preceq f_X\}$.

4 Execution model

The execution model describes the constraint solving in the elected constraint domain. It is a top-down execution model which defines the operational semantics of the system. The model describes how the constraints are processed and what they lead to. In our case, it basically extends the subset bound constraint model presented in [23] to deal with lexicographic bounds in a hybrid domain. The idea consists in (1) constraining each set variable to range over a hybrid set domain, and (2) enforcing local consistency, i.e. removing some values of the set domains that can never be part of any feasible solution by pruning the various interval bounds. This is achieved by making use of local consistency adapted to the handling of the finite set constraints over a hybrid set domain.

4.1 Preliminaries

A finite set model, formulates a CDP as a set constraint satisfaction problem (set-CSP) with variables representing the points or sets and having a domain of values. A classical *constraint satisfaction problem* (CSP) models combinatorial problems in terms of relations (constraints) specified over a set of variables with corresponding domains of possible values. A set-CSP is a CSP where the variables can be set variables or integer variables. 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. When dealing with domains which are approximated by bounds, the common cost effective approach ensures 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. 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 because, among others, 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. A notion of Set Bounds Consistency describes the consistency level achieved. It is recalled below.

Let the function symbol $\texttt{sol}$ denote a particular solution to a set-CSP such that $s_i = \texttt{sol}(X_i)$ should be read as s_i is the value of the set variable X_i in the solution $\texttt{sol}$ with support in the other variables. The set of all solutions to a set-CSP is represented by the symbol $\texttt{sol}^+$. Let $C(S)$ denote a constraint relation C applied to a set of set variables S.

Definition 4. (SBC) *For a general constraint C(S), such that* $\texttt{sol} \in \texttt{sol}^+_{C(S)}$*, to be Set Bounds Consistent (SBC), the following two conditions must be met:*

1. *for* $X_i \in S$, *for* $x \in lub(X_i)$ $\exists\ \texttt{sol} \in \texttt{sol}^+_{C(S)} : x \in \texttt{sol}(X_i)$
2. *for* $X_i \in S$, $(\forall\ \texttt{sol} \in \texttt{sol}^+_{C(S)}(x \in \texttt{sol}(X_i))) \Rightarrow x \in glb(X_i)$

The constraint must be satisfiable. All elements in the upperbounds of set variable domains must occur in at least one solution (1). Any element which occurs in a given set in all solutions must be in the lowerbound of that set variable domains (2).

Definition 5. *A* primitive set constraint *is a set constraint that involves only one set constraint or a set operation in relational form (e.g. $X = Y \cap Z$).*

Definition 6. *An* admissible system *of constraints is a system of constraints such that every set variable X ranges over a set domain.*

4.2 SBC$^+$

The notion of SBC needs to be revised for our hybrid set domain. The main issue is that the hybrid set domain uses multiple bounds to specify a set, and associates to it its cardinality bounds.

Let us consider the hybrid set domain as a *two dimensional box* where one dimension corresponds to the set value and the second to the cardinality value. This representation views the hybrid domain as one constraint relation subset of the cartesian product $[c, d] \times [e, f]$ commonly called a box (the lex bounds are an inner approximation of the subset bounds). Ideally we wish to derive the tightest possible box that satisfies a constraint such that the lex and cardinality bounds can belong to a solution, but the best that can be guaranteed effectively for all set constraints is a smaller box such that the subset bounds are SBC and the other bounds enclose the solution. Therefore we extend the SBC notion above to SBC^+ with a fourth condition.

Definition 7. *(**SBC**$^+$) Given a constraint C, over the set of variables S, C is SBC^+ if and only if:*

- *There exists* $\texttt{sol} \in \texttt{sol}^+_{C(S)}$ *for C(S) to be SBC, and*
- *for* $X_i \in S, (\texttt{sol}(X_i), |\texttt{sol}(X_i)|) \subseteq [inf(X_i), sub(X_i)] \times [min(|X_i|), max(|X_i|)]$

A set solution value must be between its lex bounds and its cardinality between the cardinality bounds.

A set-CSP is SBC^+ if and only if each constraint in it is SBC^+.

In other words, SBC^+ ensures that for a satisfiable constraint, all elements in the subset lower bounds belong to all solutions, elements in subset upper bound belong to atleast a solution, and the lex and cardinality bounds contain the solution value.

4.3 Enforcing local consistency

The consistency notions define conditions to be satisfied by the different bounds in the hybrid domain so that a set constraint is SBC^+. This is achieved by repeatedly removing unsupported values from the hybrid domains of its variables.

The essential point is that a refinement of all bounds allows us to prune a domain. Reducing the set of possible values a set could take can be achieved either by reducing the subset bounds (extending the collection of *definite* elements of a set *i.e.* or by reducing the collection of *possible* elements *i.e.*) or pruning the lex bounds or cardinality bounds in the usual FD sense, w.r.t. their respective orderings. All computations are deterministic.

Approach We define the operational semantics of the new hybrid FS solver as a system of logical transformation rules which must all be satisfied. To avoid unnecessary duplication of content, and to focus precisely on *our* contributions to the state of FS solvers, we assume that some mechanism is already in place to enforce:

- SBC over a standard subset bound solver for the subset bounds of our hybrid domain, and
- Standard FD bounds consistency for the cardinality bounds and arithmetic constraints.

The relevant inference rules can be found in [23]. We present essentially the new inference rules *pertaining to the lexicographic bounds* with respect to the domain and primitive constraints.

Since this is a constraint system, it is possible to have a system which is unsatisfiable. In the event such a system is detected, we introduce the syntactic constraint `fail` to indicate that the store in unsatisfiable. Also for clarity we 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 state.

For the hybrid domain constraint Consider the domain constraint $X \in [a_X, b_X]|c_X, d_X|\langle e_X, f_X\rangle$. Inferring its local consistency, SBC^+, amounts to possibly pruning each interval present in the domain such that the various bounds of our hybrid domain hold according to our representation and semantics of a hybrid domain as given in Definition 3, and that the conditions stated in Definition 7 hold. In particular we must ensure that i) any set solution value X_i must be between its lex bounds, and ii) $|X_i|$ between its cardinality bounds. This is enforced by pruning the subset bounds from the lex bounds (rules 1–2), the cardinality bounds from the subset and lex bounds (rules 3–4), and the lex bounds from the subset and cardinality bounds (rules 5–6).

It derives that enforcing SBC^+ yields *tighter* subset bounds than SBC because the lex bounds enable the addition of new elements to the *glb* and the removal of elements from the *lub*. This is depicted by the following rewrite rules describing the conditions to be satisfied by each bound (and its potential new value):

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 from the max values, should be part of the glb. Thus changes

to the lex bounds can impact the glb. For instance if $e_X = \{5,4,3\}$ and $f_X = \{5,4,2\}$ then we infer that 5 and 4 are required elements to be included in a_X since any solution set value inbetween e_X and f_X contains those elements.

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

`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 is greater than the max of f_X. If added to the *glb* this would cause it to be greater than ($\succ$) the lex upper bound f_X which is not possible. This follows from Theorem 1.
- The second case arises when there is at most one more element which *could* be added to the set (ie. 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.

The reasoning goes as follows to remove from the lub (b_X) all the elements that cannot appear in *any* sets within the lex bounds. We consider all the different cases for a value $y \in b_X$. (case 1) If $y \in b_X \setminus a_X$ and $\{y\} \cup a_X \succ f_X$, then $y \notin X$ since we require $X \preceq f_X$, thus we infer $y \notin b_X$. If $y \in f_X \cup e_X$ it can not be excluded from X as it is in at least one lex bound. (case 2) If $y \in b_X \setminus (f_X \cup e_X)$, and $\{y\} \cup a_X$ instantiates X ($|a_X| \leq d_X - 1$) such that $X \preceq e_X$ then we require $y \notin b_X$ otherwise such a set value has no support in the domain constraint. In the last case y can not be forbidden: $y \in b_X \setminus (f_X \cup e_X)$ and $|a_X| < d_X - 2$, then we can always find a value $z \in f_X$ (*e.g.* $z = \boldsymbol{f}_{X_n}$) such that $s = a_X \cup \{y, z\}$ is consistent.

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

By definition of the hybrid domain and the subset ordering between the different bounds, we have $|a_X| \leq c_X \leq |e_X|, |f_X| \leq d_X \leq |b_X|$. The lower cardinality bound relates to the size of $|a_X|$ and $|e_X|$ since e_X is the smallest lex bound that contains a_X and whose size is between c_X and d_X.

If $a_X = e_X$ then necessarily $|a_X| = |e_X|$ thus $c_X \geq |a_X|$. Otherwise we have $a_X \subset e_X$ and since $e_X \preceq X$ clearly $a_X \subset X$ which implies $|X| \geq |a_X| + 1 \Rightarrow c_X \geq |a_X| + 1$.

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

The upper cardinality bound relates to the size of $|b_X|$ and $|f_X|$ since f_X is the greatest lex bound subset of b_X that contains a_X and whose size is between c_X and d_X.

We have either $f_X = b_X$ or $f_X \subset b_X$. In the first case the lub sets the maximum number of element any instance of X could have, thus $d_X \leq |b_X|$. In the second case $f_X \subset b_X$ implies that $X \subset b_X$ thus by definition of the partial inclusion ordering we have $|X| \leq |b_X| - 1$.

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

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

`IR 5`. defines the new lex-inf as the smallest successor under $\preceq$ of the current inf such that the domain constraint holds. In a dual manner, `IR 6`. defines the new lex-sup as the greatest predecessor of the current sup under $\preceq$ such that the domain constraint holds. This relates to the following property, essential for the contracting, inclusion monotone and idempotent properties of the inference rules.

Property 2. (lex domain inclusion) We have $\langle e, f\rangle \subseteq \langle e', f'\rangle$ **iff** $e \preceq e' \wedge f' \preceq f$

Proof.

$$\begin{aligned}\langle e, f\rangle \subseteq \langle e', f'\rangle &\Leftrightarrow \forall s \in \langle e, f\rangle \Rightarrow s \in \langle e', f'\rangle \\ &\Leftrightarrow \forall s, e \preceq s \preceq f \Rightarrow e' \preceq s \preceq f' \\ &\Leftrightarrow e \preceq e' \wedge f \preceq f'\end{aligned}$$

Together, `IR 5`. and `IR 6`. ensure that the lex bounds of the domain can only undergo monotonic reduction. They derive respectively the smallest and greatest lex bounds that satisfy the hybrid domain constraint. The rewrite rules give a declarative *definition* of the functions. The operational description is given when describing our implementation. This declarative setting is also used below when describing the new lex bounds of expressions like inf $(\{s \mid s \in [a_X, b_X]|c_X, d_X|\langle e_X, f_X\rangle \wedge s \succeq e_X\})$. The reason is that the actual algorithms depend heavily on the choice of data structures and since there are many choices, we give the conditions in a data structure agnostic form.

For primitive set constraints We express the logical inference rules as rewrite rules which operate on a conceptual constraint store when some conditions are met. The conditions relate to the domain bounds and as such the rules constitute a data-driven operational description of the solver. Where it is unambiguous to do so, we omit the *domain* constraints (i.e. constraints of the form $X \in [a_X, b_X]|c_X, d_X|\langle e_X, f_X\rangle$) from the constraint store. The transformation rules have the following form signifying that "the changes occur when the conditions hold":

$$\frac{\text{Conditions}}{\{\text{ Changes to the constraint store }\}}$$

Finally, we use the syntax $\mathit{tell}(C)$ which indicates that the constraint C has been newly added to the store. It allows the definition of actions which are operationally associated with such additions.

The hybrid domain constraints must be consistent or the solver fails:

IR 7 $$\frac{e_X \succ f_X \vee b_X \subseteq e_X \vee b_X \subseteq f_X \vee a_X \supseteq f_X}{\{\, X \in [a_X, b_X] | c_X, d_X | \langle e_X, f_X \rangle \,\} \longmapsto \{\, \texttt{fail} \,\}}$$

If the domain becomes empty then clearly we should fail. Again, the failure conditions that deal solely with the subset or cardinality bounds are not presented here (see [23]).

The hybrid domain constraint is solved with respect to the lex bounds if:

IR 8 $$\frac{e_X = f_X}{\{\, X \in [a_X, b_X] | c_X, d_X | \langle e_X, f_X \rangle \,\} \longmapsto \{\, X = e_X \,\}}$$

The following rules present the additional inferences to enforce SBC^+ on each primitive constraint with respect to the lex bounds.

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

IR 9 $$\frac{e'_Y = \inf(\{s \mid s \in [a_Y, b_Y] | c_Y, d_Y | \langle e_Y, f_Y \rangle \ \wedge s \succeq e_X\})}{\{\, X \preceq Y \,\} \longmapsto \{\, X \preceq Y \,\}}$$

IR 10 $$\frac{f'_X = \sup(\{s \mid s \in [a_X, b_X] | c_X, d_X | \langle e_X, f_X \rangle \ \wedge s \preceq f_Y\})}{\{\, X \preceq Y \,\} \longmapsto \{\, X \preceq Y \,\}}$$

The same inferences exist for the strict ordering constraints and it requires *strict* total order.

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

IR 11 $$\frac{}{\{\, tell(X \subseteq Y) \,\} \longmapsto \{\, X \subseteq Y, tell(X \preceq Y), tell(|X| \leq |Y|) \,\}}$$

Intersection ($X \cap Y = Z$) Similar rules exist for the variable Y.

IR 12 $$\frac{}{\{\, tell(Z = X \cap Y) \,\} \longmapsto \{\, Z = X \cap Y, tell(Z \subseteq X), tell(Z \subseteq Y) \,\}}$$

IR 13 $$\frac{e'_X = \inf(\{s \mid s \in [a_X, b_X] | c_X, d_X| \ \wedge |s \cap a_Y| \leq d_Z \ \wedge |s \cap b_Y| \geq c_Z\})}{\{\, Z = X \cap Y \,\} \longmapsto \{\, Z = X \cap Y \,\}}$$

IR 14 $$\frac{f'_X = \sup(\{s \mid s \in [a_X, b_X] | c_X, d_X| \ \wedge |s \cap a_Y| \leq d_Z \ \wedge |s \cap b_Y| \geq c_Z\})}{\{\, Z = X \cap Y \,\} \longmapsto \{\, Z = X \cap Y \,\}}$$

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

IR 15 $$\frac{}{\{\, tell(Z = X \cup Y) \,\} \longmapsto \{\, Z = X \cup Y, tell(X \subseteq Z), tell(Y \subseteq Z) \,\}}$$

IR 16 $$\frac{e'_X = \inf(\{s \mid s \in [a_X, b_X] | c_X, d_X| \ \wedge |s \cup a_Y| \leq d_Z \ \wedge |s \cup b_Y| \geq c_Z\})}{\{\, Z = X \cup Y \,\} \longmapsto \{\, Z = X \cup Y \,\}}$$

IR 17 $$\frac{f'_X = \sup(\{s \mid s \in [a_X, b_X] | c_X, d_X| \ \wedge |s \cup a_Y| \leq d_Z \ \wedge |s \cup b_Y| \geq c_Z\})}{\{\, Z = X \cup Y \,\} \longmapsto \{\, Z = X \cup Y \,\}}$$

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

IR 18 $$\frac{}{\left\{ tell(Z = X \setminus Y) \right\} \longmapsto \left\{ Z = X \setminus Y, tell(Z \subseteq X) \right\}}$$

IR 19 $$\frac{e'_X = \inf(\{s \mid s \in [a_X, b_X] | c_X, d_X| \ \wedge |s \setminus b_Y| \leq d_Z \ \wedge |s \setminus a_Y| \geq c_Z\})}{\left\{ Z = X \setminus Y \right\} \longmapsto \left\{ Z = X \setminus Y \right\}}$$

IR 20 $$\frac{f'_X = \sup(\{s \mid s \in [a_X, b_X] | c_X, d_X| \ \wedge |s \setminus b_Y| \leq d_Z \ \wedge |s \setminus a_Y| \geq c_Z\})}{\left\{ Z = X \setminus Y \right\} \longmapsto \left\{ Z = X \setminus Y \right\}}$$

IR 21 $$\frac{e'_Y = \inf(\{s \mid s \in [a_Y, b_Y] | c_Y, d_Y| \ \wedge |a_X \setminus s| \leq d_Z \ \wedge |b_Y \setminus s| \geq c_Z\})}{\left\{ Z = X \setminus Y \right\} \longmapsto \left\{ Z = X \setminus Y \right\}}$$

IR 22 $$\frac{f'_Y = \sup(\{s \mid s \in [a_Y, b_Y] | c_Y, d_Y| \ \wedge |a_X \setminus s| \leq d_Z \ \wedge |b_Y \setminus s| \geq c_Z\})}{\left\{ Z = X \setminus Y \right\} \longmapsto \left\{ Z = X \setminus Y \right\}}$$

Arbitrary Predicate Satisfaction Constraint (`satisfies`) The satisfies constraint relation allows the modeller to enforce arbitrary unary constraints which can be specified as predicates which must hold for valid values of the set. Rather than have the modeller create their own bespoke filtering algorithms for their arbitrary constraints, we provide the following simple unary constraint which takes a predicate and ensures that the predicate holds for the lex bounds e_X and f_X.

IR 23 $$\frac{e'_X = \inf(\{s \mid s \in [a_X, b_X] | c_X, d_X | \langle e_X, f_X \rangle \ \wedge pred(s))}{\left\{ X \texttt{satisfies } pred \right\} \longmapsto \left\{ X \texttt{satisfies } pred \right\}}$$

IR 24 $$\frac{f'_X = \sup(\{s \mid s \in [a_X, b_X] | c_X, d_X | \langle e_X, f_X \rangle \ \wedge pred(s))}{\left\{ X \texttt{satisfies } pred \right\} \longmapsto \left\{ X \texttt{satisfies } pred \right\}}$$

These two inference rules allow any arbitrary predicate defined i) to evaluate to *true* for ground sets, ii) to act as a constraint and iii) to interact with the other more conventional constraints in order to trigger further propagation and domain pruning.

Properties of the inference rules In a traditional subset bound solver, the inference rules are proven correct (all possible solutions are kept), contracting (final domains are subset of the initial domains), and idempotent (the smallest domains have been computed the first time). The proof is performed by representing each inference rule as a mapping from a Cartesian product of domains to another Cartesian product of domains [23].

The main difference here is that each domain is not one single interval but a *box* (Cartesian product of intervals with their respective partial/total orderings). The reasoning on one interval extends naturally to that over a box where each interval within it satisfies the constraints.

Property 3. Boxes specifying hybrid domains are partially ordered by set inclusion.

Proof. By definition we have $[a,b] \subseteq [a',b']$ iff $\forall x \in [a,b] \Rightarrow x \in [a',b']$. For subset bound domains this is equivalent to $a \subseteq a'$ and $b \subseteq b'$. For lex bounds this is equivalent to $a \succeq a'$ and $b \preceq b'$ (property 2), and for the cardinality bounds this is equivalent to $a \geq a'$ and $b \leq b'$. Thus in the case of a box describing a hybrid domain we have: $[a,b]|c,d|\langle e,f\rangle \subseteq [a',b']|c',d'|\langle e',f'\rangle$ iff $[a,b] \subseteq [a',b'] \wedge |c,d| \subseteq |c',d'| \wedge \langle e,f\rangle \subseteq \langle e',f'\rangle$. □

Furthermore as each interval is convex the box is convex. As a consequence, the same approach and results hold as each box is a convex closure of a hybrid domain. The inference rules above are i) *correct* since only irrelevant values are removed from the domains, ii) *contracting* since the domains can only get refined, and iii) *idempotent* since every element that can be removed has been removed the first time. The inference rules are *inclusion monotone* since smaller initial domains yield smaller final domains.

4.4 Operational semantics

The inference rules described so far can be applied to individual constraints. The operational semantics shows how to check and infer the consistency of an admissible system of constraints. The consistency of such a system results from the consistency of each constraint appearing in it. The operational semantics is described by a standard relaxation or fixed point algorithm [35, 5], which can be seen as an adaptation of the `AC_3` algorithm [38] where domains are specified by intervals. Our algorithm is similar to the one used for a subset bound set solver. The only difference between the algorithms lies in the inference rules applied.

Theorem 2. *The fixed point algorithm satisfies the standard properties of: termination, existence of a unique fixed point independent of the constraint ordering, and correctness.*

It follows directly from the properties of the inference rules (contractance of the inference rules relates to termination and unicity, idempotence to unicity, and monotonicity to correctness and unicity properties). The computation of the fixed point which is unique and independent of the constraint ordering relates to the structure of the boxes:: propagation methods based on the AC-3 algorithm compute a unique fixed point independent of the ordering of the inference rules, if the states of the iteration process can be ordered within a lattice and if the inference rules applied are contracting, idempotent and inclusion monotone [42]. Older and Vellino show that the contractance and idempotence properties guarantee the existence of a fixed point. In addition, due to the monotonicity of the inference rules, the fixed point is unique and independent of the ordering of the inference rules. In our case, the only things that change during our iteration process are the bounds within the boxes. Thus the states can be characterized by the set of boxes. The boxes are partially ordered by the set inclusion within the

lattice of set and integer domains. Additionally, the contractance, idempotence and inclusion monotone properties are satisfied by our inference rules. Thus, the generic algorithm has a unique fixed point independent of the ordering of the inference rules.

5 Practical Framework

The execution model has given us the structure of the hybrid set domain system whose solver is based on consistency techniques. It constitutes the basis of the design of a prototype solver. Its functionalities (apart from those of a logic-based language like Prolog) are set operations and relations from set theory together with the set cardinality which is partly handled by a finite domain solver. In this part, we describe the implementation of our prototype which raises among others the issues of (1) dynamic handling of a system of constraints by means of delay mechanisms, (2) specific set data structure required to attach all the relevant information related to a hybrid set domain, (3) algorithms to implement the inference rules effectively.

We implemented our prototype hybrid-domain solver in the constraint logic programming system ECL^{i}PS^{e}using, as a base, the `ic-sets` library. Therefore, we begin with a brief overview of the ECL^{i}PS^{e}system in general and this library in particular.

5.1 ECLiPSe

Historically the ECLiPSe language is derived from the Prolog family of logical languages, but has been extended in many directions to make the modelling and solving of combinatorial problems easier [50]. Perhaps the most important extensions as far as writting constraint solvers is concerned, is the addition to the Prolog language of attributed variables [30, 28] and suspended goals. ECLiPSe implements a general delayed computation scheme whereby goals (computations) may be suspended (delayed) until some conditions are met at which time they are re-awakened[1]. This essentially allows computations to progress in a data-driven manner, where changes to some data structure can trigger computation. The typical data structures which trigger such delayed computations are the attributes of constrained logic variables. ECLiPSe allows an arbitrary number of domains to be attached to a single variable in the form of attributes which consist of a domain representation and a collection of suspension lists into which suspended computations may be stored. Using this general suspension mechanism, a simple constraint solver can awake the goals when the domain of a variable changes. Such a mechanism offers a natural and convenient method of implementing logical inference rules, such as those used to describe the hybrid set domain solver.

[1] Many other constraint programming systems support both attributed variables and suspended goals, e.g. `Ciao, Sicstus, SWI-Prolog`.

As well as simple data-driven computation, the ECLiPSe language provides many solvers, facilities for user-defined constraints and the ability to build new solvers on top of existing ones. The ECLiPSe system facilitates writing constraint solvers by providing a wide range of efficiently implemented data structures (including hash-tables, lists (ordered and unordered), heaps, multi-dimensional arrays and graphs), along with algorithms which operate on them (sorting, searching, finding shortest paths etc.). These features make it quick and easy to implement many (though not all) algorithms efficiently. In addition ECLiPSe boasts a wealth of existing constraint solvers and search support libraries, which are easy to integrate with.

5.2 Extending ic-sets

`ic_sets` uses attributed variables where an array of 0/1 variables represents the subset-bounds of the set domain, which is restricted to integers. Notification of domain changes is performed using the general suspension mechanisms of ECLiPSe with the addition of a pair of notification ports where notice of element addition (resp. removal) is given. The addition of these notification ports allows for a more efficient implementation of many operations on the subset domain representation, than could be achieved without. Indeed most operations take constant time for each element included/excluded from a variable's domain.

The ground representation of sets in the `ic_sets` library as shipped with ECLiPSe is a list of integers in ascending order. In order to simplify computations between ground sets and our lex bounds, we altered `ic_sets` so that the ground representation was a list of integers in descending order. This change does not affect the complexity or efficiency of the `ic_sets` solver in any way. This choice of ordered lists makes it easy to work with ground sets in the context of a Prolog system, but does incur an overhead when mixing ground set values and non-ground set variables, as the system must convert between the array and list representations. For the subset bounds domain, such occasions are rare in practise and the overhead is slight. This is not the case with the lex bounds however, as typically these will need to be compared against ground sets much more often. It is partially because of this increased frequency of comparison that we chose to use the same (ordered list) representation for our lex bounds.

Ground Set Representation We represent ground sets as a list sorted in *decreasing* element order, following our lexicographic ordering. Since we deal exclusively with ordered lists of numbers, we can optimise certain set operations w.r.t. our lists. The worst case complexity for set operations on this ground set representation is linear in the size of the set. Clearly this could be optimized using a binary tree or BDDs data structures. However, this would have meant reconsidering the `ic_set` solver itself. In the scope of evaluating the pruning gain from a hybrid domain we first built a prototype on top of `ic_sets`.

Hybrid Domain Data Structure As mentioned earlier, in order to evaluate the new concept of the hybrid set domain and its applicability, we chose to adapt

the existing `ic_sets` solver by adding the extra inference rules to it. To support these inference rules we needed to extend the data structure which described the set domains.

In addition to the array of logical variables which represent the glb and lub subset bounds and the FD cardinality variable (with its associated min and max bounds), we add two ground sets representing the *inf* and *sup* lexicographic bounds. These two ground sets are represented by the decreasing ordered lists described above.

Also, in addition to the existing suspension lists of the `ic_sets` attribute, namely added, removed, card min and card max, we add two new suspension lists: *lex min* and *lex max*. Goals suspended in one of these two lists will be woken whenever the inf or sup bound changes respectively.

So as to minimise the performance penalty for models which do not make use of the cardinality information or lex bounds, we retain the 0/1 variable array representation for the subset bounds that `ic_sets` uses. However, there are many occasions when we need access to the glb and lub separately in list form (i.e. not encoded in the 0/1 array), since computations using ground sets (such as the lex bounds) are more efficient if performed using a list representation. To this end we investigated a number of ways that this could be achieved. We considered the following approach: **caching**. Use a *valid* flag in the domain to indicate that the pre-generated list representations are a valid representation of the current 0/1 array. When the lists are required, if they are *invalid*, then re-generate them, store them and set the flag to valid. If they are valid then simply use them. Any modifications to the 0/1 array results in the flag being set to *invalid*. The caching method proves to be efficient in practice because it interacts nicely with the priority based data-driven implementation of our inferences rules. It mitigates the performance overheads of building the hybrid solver on top of an existing solver whose implementation has been optimised based on assumptions that are no longer valid.

ECLiPSe gives the algorithm designer the ability to specify the relative importance of constraints (inferences) by means of a rudimentary priority scheme. At an implementation level, when adding goals back into the resolvent, the order is dependent on a priority which is attached to the suspended goal (1 to 6, 1 being the highest priority). The effect of the priority scheme is that all woken goals with a higher priority will execute before any lower priority ones. As a general rule, it is suggested that goals which terminate quickly be given higher priorities than longer running ones.

With this in mind, in our implementation we leave the existing subset bound propagators at priorities 3 and 4, as was the case in `ic_sets`, and add our various lex bound propagators at priorities 5 and 6. If the subset inferences alone are sufficient to detect an unsatisfiable constraint store, then the priorities will ensure that our lex inference rules computation will never be started and hence the conversion between 0/1 array and lists will not be performed.

Unification Procedure When constraining two set variables X and Y to be equal, we intersect their domains as shown below and merge all suspension lists attached

to each attribute. Clearly we need to maintain SBC^+ for the new domains. The new bounds resulting from the unification of X and Y are given below:

$$a'_X \equiv a'_Y = a_X \cap a_Y \qquad b'_X \equiv b'_Y = bX \cup b_Y \tag{1}$$

$$c'_X \equiv c'_Y = max(c_X, c_Y) \;\; d'_X \equiv d'_Y = min(d_X, d_Y) \tag{2}$$

$$e'_X \equiv e'_Y = sup(\{e_X, e_Y\}) \; f'_X \equiv f'_Y = inf(\{f_X, f_Y\}) \tag{3}$$

5.3 Implementing the Inference Rules

As was detailed above, a number of computations must be made, values calculated and operations performed in order to implement the inference rules of our hybrid domain.

In the rules which describe our solver, many bound updates are specified using the inf and sup functions. Recall that these are analogous to the min and max functions of integer arithmetic and an efficient implementation is essential to the efficient propagation of the associated inference rules.

As used in the definition of our inference rules, the inf and sup functions allow us to *succinctly* describe the new values of lex bounds. They do not, however, translate simply into any naturally efficient algorithm for computing the new value.

Example 7. By analogy with FD bounds, consider the following definition for the new finite domain lower bound (c'_Y) of the FD variable (Y) involved in the FD constraint $X = Y + 3$.

$$c'_Y = min(\{y \mid c_Y \leq y \wedge y \leq d_Y \wedge y + 3 \geq c_X \wedge y + 3 \leq d_X\})$$

This can be read as defining the new bound (c'_Y) to be the minimum value y which is in the variable's range and which has a supporting value in the domain of X.

Though this is a perfectly valid definition of the relationship between c_Y , d_Y, c_X and d_X, it gives no clue as to how efficiently calculate c'_Y. We know that an efficient method (assuming that we can efficiently compute numeric subtraction and comparison) is to define c'_Y as

$$c'_Y = \begin{cases} c_X - 3 & if \;\; c_Y < c_X - 3 \\ c_Y & if \;\; c_Y \leq d_X - 3 \\ fail & \text{otherwise} \end{cases}$$

An efficient implementation of the above example is possible because we know:

- How to algebraically rewrite some of the conditions.
- That the resulting expressions are functions (i.e. have only a single output).
- That the min can be replaced by conditional if statements.
- That there is an efficient implementation of subtraction (in hardware).

In order to duplicate this feat of efficient computation for our *inf* and *sup* functions we would require the equivalent of lex-bound set algebraic manipulation. Since we are not aware of such an algebra, we define an algorithm that computes the *inf* and *sup* functions. Algorithm 1 computes the smallest lex bound that satisfies the domain constraint.

The basic approach taken to compute the *inf* bound (i.e. the lexicographically smallest value satisfying some condition) is to intelligently enumerate possible set values in a lexicographically increasing order until one is found that satisfies the conditions: satisfies the cardinality (it records the number of elements added to the result and the remaining *glb* and *lub* elements) and subset bounds restrictions of the domain constraints. It terminates the "for" loop when these limits are reached (line 5 and 8). The first value returned will be the lexicographically smallest value in a hybrid set domain, corresponding to the value of $inf(\{s \mid s \in [a_X, b_X] | c_X, d_X | \langle e_X, f_X \rangle\})$. The *glb* inclusion is performed line 12, the $e_X = s \oplus [y \mid ys] \wedge x < y$ and $lexgreat$ tests line 14 can be performed in constant time by having a flag indicate if the partially generated set (s) is necessarily within the bounds and by removing any preceeding elements from the bounds if s is a prefix of either. These tests ensure that the partially built inf bound remains within the existing lex bounds.

Algorithm 1 Computing the smallest lex bound

Input: Domain bounds $[a_X, b_X] | c_X, d_X | \langle e_X, f_X \rangle$
Output: s

```
1:  glbs ← |a_X|;
2:  lubs ← |b_X|;
3:  s ← [];
4:  foreachtail [x|xs] ⊆ b_X do
5:      if |s| + glbs = d_X then
6:          s ← s ⊎ a_X; {only remaing glb elements can be included}
7:          return s
8:      if |s| + lubs = c_X then
9:          s ← s ⊕ [x|xs]; {all remaining lub elements must be included}
10:         return s
11:     if x ∈ a_X then
12:         s ← s ⊕ [x];
13:         glbs ← glbs − 1;
14:     else if e_X = s ⊕ [y | ys] ∧ x < y or lexgreat(s ⊕ [x], f_X) then
15:         exclude x
16:     else
17:         choose
18:         exclude x
19:         or
20:         include x
21:         s ← s ⊕ [x];
22:     lubs ← lubs − 1
```

Note the use, on line 17, of a non-deterministic choice statement. The operational semantics of this statement is that the first branch is taken, and only if some failure happens in subsequent computations does the computation return to the state immediately before the choice and resume after taking the second branch. Such a notion will be familiar to Prolog programmers as the disjunction connective (;), but also exists in other high level modelling languages as it allows for a more compact representation of algorithms which perform search. The reason for this choice point is that the new inf bound does not build on the existing one in terms of values (except if they are required elements) it just has to be a successor of it, the first satisfiable one.

The related algorithm for computing the *sup* function (i.e. the lexicographically largest value first satisfying the domain conditions) is very similar and differs only in lines 18 and 20 which are exchanged [46]. Each of these algorithms allows us to compute the new lex bounds satisfying the domain transformation rules `IR 5` and `IR 6` in $O(v)$ time, with $v = |b_X|$.

Example 8. Let us assume we have $X \in [\{4\}, \{1,2,3,4,5\}]|3,3|\langle\{4,2,1\},\{2,4,5\}\rangle$. We now add the constraint $3 \in X$. The glb bound is updated to $\{3,4\}$ and the lex bounds need updating too. The inf bound is updated using Algorithm 1. It proceeds from the lub list representation $[5,4,3,2,1]$ by determining which element starting from 5 is required (lines 9,12) or excluded (line 14) or either. 5 is excluded at the choice point. 4 and 3 are included (line 12), 2 is excluded at the choice point (line 18), and 1 is included because the test line 8 is satisfied ($|\{4,3\}| + 1 = c_X = 3$). The new inf bound is returned $\{4,3,1\}$. The sup will also be updated to $\{4,3,2\}$ ($\{5,4,3\}$ would be greater than the current sup).

The complexity of re-establishing SBC^+ (inference rules `IR 1, IR 2, IR 3, IR 4, IR 5` and `IR 6`) on a hybrid domain constraint until a fixed point is reached takes in the worst case $O(v)$ using Algorithm 1 and its decreasing lex order counterpart to update the lex bounds.

We prove this by considering the different bounds in order:

`IR 1, IR 2`:: Additions to a_X (or removals from b_X) may result in the constant time modification of c_X (resp. d_X) and then the $O(v)$ time modification of e_X.

`IR 3, IR 4`:: Incrementing c_X can only trigger $O(v)$ time updates to e_X and f_X or an instantiation of the set if $c_X = |b_X|$. Decrementing b_X will similarly cause e_X and f_X updates, but may also trigger the removal of elements from b_X. Since there are atmost v elements to be removed from b_X, this process can be repeated no more than $O(v)$ times. By delaying the updates off e_X and f_X until b_X and d_X have stabilized, we can maintain the $O(v)$ overall worst case complexity.

`IR 5, IR 6`:: Advancing e_X may also lead to updates of a_X and b_X. These in turn may cause e_X updates, but since there can be atmost v subset bound updates, the process must terminate in $O(v)$ steps. And similarly for retreating f_X.

5.4 Predictor Function

Algorithm 1 and its counterpart allow us to quickly compute (in order) new potential lex bounds. However in a system of constraints they need to be tested against the other criteria of the inference rules for an arbitrary set constraint. Though this proved adequate for small domains, the time to compute bounds satisfying even very simple conditions can grow exponentially as the actual domain sizes do. To combat this problem we extended Algorithm 1 in a general way to incorporate the other conditions of the inference rules at each step of the algorithm.

Algorithm 2 `INTPRED` Predictor routine for the $Z = X \cap Y$

Input: X Domain bound $[a_X, b_X]|c_X, d_X|\langle e_X, f_X\rangle$
Input: Y Domain bound $[a_Y, b_Y]|c_Y, d_Y|\langle e_Y, f_Y\rangle$
Input: X Domain bound $[a_Z, b_Z]|c_Z, d_Z|\langle e_Z, f_Z\rangle$
Input: Loop variables s, x, xs from Algorithm 1
Output: The symbolic values `include, exclude` or `either`

1: $temp1 \leftarrow |s \cap b_Y|$; {number of potential intersection elements in bounds}
2: $temp2 \leftarrow c_Z - |temp1|$; {number of extra intersection elements required}
3: **if** $temp1 \geq c_Z \vee (|xs \cap b_Y| \geq temp2 \wedge temp2 \leq d_X - |s|)$ **then**
4: **if** $x \in a_Y$ **then**
5: $temp3 \leftarrow |s \cap a_Y|$;
6: **if** $temp3 = d_Z$ **then**
7: **return** `exclude`
8: **else if** $temp3 < d_Z$ **then**
9: **return** `either`;
10: **else**
11: Fail
12: **else**
13: $temp4 \leftarrow |xs| - |xs \cap a_Y|$; {number of non-intersecting elements remaining}
14: $temp5 \leftarrow c_X - |s| - temp4$; {number of intersecting elements required}
15: **if** $temp4 < c_X \wedge temp5 + |(a_X \setminus s) \cap a_Y| + |s \cap a_Y| > d_Z$ **then**
16: **return** `include`
17: **else**
18: **return** `either`
19: **else**
20: **return** `include`

We achieved this by allowing constraints to pass in predictor functions which determine, using the current state of the algorithm variables, whether the current element *must* or *must not* be part of the result. The lex-bound test of the original algorithms can be seen as a special case of these predictor functions. In general, for the constraints listed, these tests which are called at most once per potential element, can be computed in $\mathcal{O}(|b_X|)$ time thus giving us a $\mathcal{O}(|b_X|^2)$ time algorithm for computing the new lex bounds for a system of constraints.

We illustrate the concept by giving the predictor function `INTPRED` used for the intersection constraint, in Algorithm 2. Intersection constraints form a core part of many CDPs. The predictor function is called within the "foreach" loop of Algorithm 1 and uses the terms s, x, xs from this loop as input to its test in a way similar to the lex bound tests on line 14 of Algorithm 1.

The inclusion (resp. exclusion) of an element occurs in the following conditions: An element must be included if 1) it is in a_X, 2) omission would necessarily violate e_X, 3) *all* remaining lub elements (including this one) are required to meet c_X, or additionally: i) Any set of size c_X made *without* this element must exceed the intersection size d_Z, ii) no set of size d_X made *without* this element can reach the intersection size c_Z. An element must be *excluded* if: 1) it is not in b_X, 2) inclusion would necessarily violate f_X, 3) the cardinality bound d_X has been reached, or additionally: i) Any set of size c_X made with this element must exceed the intersection size d_Z, ii) No set of size d_X made with this element can reach the intersection size c_Z. Function `INTPRED` shows how to compute these conditions. The possible return values of the function are `include` (meaning that the element must be included in the bound), `exclude` (meaning that the element must not be included in the bound) and `either` (meaning that it does not matter whether the element is included or excluded and the decision should be left up to the default branching rule of the generate algorithm). The predictor functions for other constraints would follow a similar pattern. Table 1 shows the times taken using this *enumeration with predictor* function approach and compares it with the Algorithm 1 and its counterpart.

c	Alg. 1 + counterpart (ms)	with predictor (ms)
1	2.77	2.76
2	3.89	3.78
3	4.16	4.10
4	6.67	4.21
5	8.86	4.40
6	13.40	4.47
7	19.56	4.81
8	31.67	5.06
9	49.07	4.22
10	75.28	2.36
11	107.81	4.82

Table 1. Time taken to reach fixed point for $X \in [\emptyset, \{16, ..., 1\}|c| \wedge |X \cap \{6, 5, 4, 3, 2, 1\}| \leq 1$ using "Algorithm 1 and its counterpart" and "predictor function"

Multisets A brief note on extending this hybrid domain representation and inf and sup computation scheme to deal with multisets. The hybrid set domain presented, and its associated implementation data structures (i.e. ordered lists

for the glb, lub, inf and sup bounds), can be extended to multisets in the obvious way (i.e. the multisets glb, lub, inf and sup bounds represented with ordered lists containing duplicates). Both the lex-order generating algorithm 1 and the above predictor functions can be applied directly and with trivial modifications to hybrid multiset domains. However, if instead of an explicit fully enumerated list representation for multisets (e.g. $\{4, 4, 3, 1, 1, 1\}$) a more compact *run-length encoded list* was used (e.g. $\{4^2, 3^1, 1^3\}$) then the predictor functions could be modified to return a numeric range indicating how many of the current elements must be present in the result and how many at most may be present. The symbolic return values of `include, exclude` and `either` would be replaced with a pair of numbers $((o, p))$ indicating the minimum number of elements that must be included (o) and the maximum that *may* be (p). A range of (0, 0) would signify that the result must not contain the current element at all.

6 Experimental Results

To illustrate the benefits of our hybrid domain over the traditional subset bound solver and closely related FD models (where a block is specified using the characteristic vector of a finite subset) we considered a class of CDPs from design theory and combinatorics as well as a network design problem. In this paper we present our results on the SONET problem and benchmarks on Steiner systems. More results can be found in [48, 46]. These problems were chosen because they each have natural and intuitive FS model which exhibit different constraints and structures and they have been addressed before which allows us to draw comparisons. A 2GHz Pentium 4 with 1GB of RAM was used for all our experiments.

Unless otherwise stated the search procedure used in the following problems is simple: 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 as this strategy best exploits the lex ordering.

The purpose of the experiments that we run is twofold:

- Firstly, we seek to evaluate the effectiveness of our inference rules at strengthening the propagation phase and therefore reducing the search phase of solving CDPs using a FS solver.
- Secondly, we are evaluating our implementation of these rules in the prototype solver to better understand how a hybrid domain set solver *should/could* be implemented.

6.1 The SONET problem

As we mentioned in section 3, this problem is a CDP that involves minimizing the number of ADMs while satisfying the demand constraints (partitioning among a set of channels).

Model For our experiments we implemented a simple primal/dual set model for the topology of the network, with the assignment of traffic handled by simple FD sum constraints tied to the set variables with reified inclusion constraints.

The problem instances are specified by a weighted undirected demand graph $G = \langle V, E \rangle$ where the weight of each edge $((f, t, c)_k \in E)$ signifies the number of channels (c) required between the two nodes (f and t), and a demand k to be satisfied over this edge. The demand is seen as a number of channels that is split among the different rings that contain the pair of nodes f, t. In addition, we have the set of available rings (R), the maximum number of nodes which may be attached to a single ring $(RingSize)$, the number of nodes $(|V|)$ and the maximum number of channels each ring can accommodate $(MaxChannels)$. In our FS model, primal sets (X_i) represent the nodes assigned to rings, and dual sets (X'_j) represent the rings on which a node sits. The model can now be specified as follows.

$$\text{for } i \in R \;\; X_i \subseteq V \tag{4}$$

$$\text{for } i \in R \;\; |X_i| \leq RingSize \tag{5}$$

$$\text{for } j \in V \;\; X'_j \subseteq R \tag{6}$$

$$\text{for } j \in V \;\; X'_j \supset \emptyset \tag{7}$$

$$\text{for } (f, t, \star) \in E \;\; X'_f \cap X'_t \supset \emptyset \tag{8}$$

$$\text{for } i \in R, j \in V \;\; j \in X_i \iff i \in X'_j \tag{9}$$

$$\sum_{i \in R} |X_i| = \sum_{j \in V} |X'_j| \tag{10}$$

Equations (9) and (10) bind the primal and dual variables and Equation (8) requires that nodes which have demands between them must lie on at least one common ring. The above model captures the connectivity of the network design but fails to ensure sufficient capacities on the rings. To handle this, we have a finite domain channel allocation variable Y_{ik} for each ring i and demand k. This FD variable represents the number of channels of the demand which are sent over the given ring.

$$\text{for } (f, t, c)_k \in E \;\; Y_{ik} \geq 0 \tag{11}$$

$$\text{for } (f, t, c)_k \in E \;\; \sum_{i \in R} Y_{ik} = c \tag{12}$$

$$\text{for } i \in R \sum_{(f,t,c)_k \in E} Y_{ik} \leq \; MaxChannels \tag{13}$$

We tie the two models together with the following simple constraints which ensure that demands are only routed across rings if they contain both nodes.

$$\text{for } (f, t, c)_k \in E, i \in R \;\; Y_{ik} > 0 \Rightarrow (f \in X_i \wedge \; t \in X_i) \tag{14}$$

Finally, since the rings are indistinguishable, we post a lexicographic ordering between ring variables so as to reduce redundant search.

$$\text{for } 0 \leq i < |R| \;\; X_i \preceq X_{i+1} \tag{15}$$

The number of ADMs required by a solution is equal to the sum of the ring cardinalities and so we seek to minimise this value:

$$\text{minimise} \sum_{i \in R} |X_i| \tag{16}$$

Operationally, we use the continue mode of the general `lib(branch and bound)` library of ECL^{i}PS^e, which proceeds by incrementally reducing the upperbound of the objective function and searching for a new solution.

Experiments We make use of several benchmark instances of the SONET problem first studied by Sherali and Smith [55, 52] and more recently by B. Smith [53, 54]. These benchmarks are split into three categories: small, medium and large. The small test instances were easy to solve and are omitted here for space reasons. Medium instances have 10 nodes, 6 rings available, each ring can accomodate 5 ADMs and 25 traffic channels. There are 15 demand pairs. Large instances have 13 nodes, 7 rings available, each ring can accomodate 5 ADMs and 40 traffic channels. There are 25 demand pairs.

We actually present two versions of the SONET problem here — the full problem with all the above constraints, and a simplified problem where the bandwidth limit for each ring is relaxed (i.e. rings can carry unlimited bandwidth). We do this for two reasons:

1. Without the bandwidth limits, the problem can be naturally modelled entirely using set variables and constraints, as it simplifies to a problem of defining the network topology. Hence we can more clearly see the performance benefits of our extra hybrid inferences, and the problem instances are less tightly constrained.
2. With the bandwidth limits, we see how FD and FS models can be naturally combined to provide concise models which, with powerful solvers, can be made to propagate efficiently.

We compare our hybrid solver against a traditional subset bound solver, `ic-sets`, using the same simple model and simple search strategy and against the constraint model presented in [53]. We simply label the primal set variables (X_i) first using the standard biggest element first, in-before-out scheme, then (if required) label the demand variables (Y_{ik}) trying small domain values first.

Results Table 2 shows the results for the medium-sized instances of the problem ignoring ring capacities. The columns headed "ADMs" show the minimum number of ADMs for which a solution was found; those marked with an asterisk (*) were *proved optimal* in the time allowed (3000 seconds = 50 minutes). Note that the final row of each table shows the average backtracks and runtimes,

which are only lowerbounds for the actual values if timeouts occurred. It is clear that the extra inferences of the hybrid domain have a massive impact on both the number of backtracks and the runtimes. This is thanks to the combination of cardinality inferences on the lex bounds, and symmetry breaking constraints that prune actively the lex bounds. Indeed the subset bound solver was unable to even find the optimal number of ADMs in 2 instances within the time limit and could not prove optimality for 4 of them. The hybrid solver finds and proves optimality in 4.86 seconds and 1338 backtracks on average, an improvement of strictly more than 99.59% and 99.89% respectively.

	HYBRID DOMAIN			SUBSET BOUND		
INSTANCE	ADMs	bt	time(sec)	ADMs	bt	time(sec)
1	*14	1791	6.57	*14	2560365	2445.1
2	*14	803	3.25	*14	1186455	1187.07
3	*14	740	3.18	*14	702488	715.53
4	*13	518	2.58	*13	386263	388.92
5	*15	2265	8.48	*15	1437099	1314.69
6	*14	1130	4.39	*14	1554749	1425.94
7	*13	218	0.95	*13	45639	47.64
8	*14	548	2.09	*14	646413	636.58
9	*15	3662	12.56	17	2973240	2997.2
10	*14	802	3.18	*14	366644	351.54
11	*12	229	1.0	*12	75777	70.46
12	*15	1782	5.8	15	3145512	3005.67
13	*15	1921	6.4	16	3125485	3003.48
14	*15	2353	8.06	15	3350260	3505.26
15	*15	1315	4.38	*15	1397653	1266.53
Average		1338	4.86		1213741	1177.04

Table 2. Medium-sized uncapacitied SONET instances

Similar results can be seen for the harder problem when capacities are taken into account. Table 3 shows an average reduction in runtime of strictly more than 98.33% and a drop in backtracks of strictly more than 99.43%.

Results for larger instances shown in Tables 4 and 5 display similar improvements in both search space and runtime. The hybrid solvers strengthens a traditional subset bound solvers with dramatic improvements both in backtracks and CPU time. In cases only the hybrid solver could reach and prove optimality in the give time limit.

The benefits of our hybrid solver are clearly demonstrated on this problem. A natural, concise, simple, in fact minimal specification of the problem can be effectively used to solve even the hardest of instances, *without the need for specialised search strategies or redundant constraints.* Also given that the dual of a CDP is a CDP itself, a natural set formulation can be used for both the primal and the dual with basic channeling constraints.

	HYBRID DOMAIN			SUBSET BOUND		
INSTANCE	ADMs	bt	time(sec)	ADMs	bt	time(sec)
1	*16	8688	35.29	18	2177051	3005.82
2	*16	3990	15.73	16	2122083	3001.64
3	*14	740	4.14	*14	702488	1102.43
4	*14	910	5.25	*14	604631	1005.52
5	*16	5414	22.92	16	2182592	3007.28
6	*17	35507	153.9	18	2277675	2998.23
7	*14	823	3.85	*14	284440	427.26
8	*16	2988	10.99	16	2053975	3012.57
9	*17	48736	207.04	20	1857591	3020.47
10	*16	10633	49.93	17	2127676	3004.50
11	*16	8268	33.13	16	2193006	3003.46
12	*17	16072	56.62	18	2097853	3003.23
13	*15	2745	11.26	17	2178804	3003.77
14	*15	3209	14.5	17	1874238	2999.95
15	*15	1315	5.4	*15	1396372	2051.25
Average		10002	42.00		>1742031	>2509.83

Table 3. Medium-sized capacitied SONET instances

	HYBRID DOMAIN			SUBSET BOUND		
INSTANCE	ADMs	bt	time(sec)	ADMs	bt	time(sec)
1	*22	477920	1608.27	34	2239681	3000.02
2	24	872261	3000.05	33	1888859	3000.02
3	*22	65108	196.64	30	2299899	3000.02
4	*23	481504	1510.02	33	2016097	3000.02
5	23	872223	3000.05	33	2098161	3000.01
6	27	883061	3000.05	34	1850383	3000.02
7	*20	214200	770.22	34	2102120	3000.02
8	*20	10105	36.61	29	2314607	3000.02
9	22	767895	3000.06	33	2051824	3000.02
10	27	965648	3000.05	34	2005265	3000.02
11	25	873919	3000.05	35	2099437	3000.02
12	*20	102717	396.44	32	1969604	3000.02
13	*21	255590	1024.30	32	1878671	3000.02
14	32	958159	3000.05	35	1966647	3000.02
15	30	1010305	3000.05	35	1944136	3000.02
Average		587374.33	1969.53		2048359.40	3000.02

Table 4. Large-sized uncapacitied SONET instances

	HYBRID DOMAIN			SUBSET BOUND		
INSTANCE	ADMs	bt	time(sec)	ADMs	bt	time(sec)
1	*22	532065	2248.68	34	1483657	3000.02
2	25	701311	3000.06	34	1317386	3000.02
3	*22	65039	227.71	31	1468889	3000.02
4	*23	476205	1767.82	34	1418783	3000.02
5	23	717284	3000.06	33	1384010	3000.02
6	27	710189	3000.06	34	1288818	3000.02
7	*22	270310	1163.94	34	1460390	3000.02
8	*20	11688	54.73	29	1482336	3000.02
9	23	648634	3000.06	33	1368335	3000.02
10	27	728733	3000.06	34	1322042	3000.02
11	27	697775	3000.06	35	1465305	3000.02
12	22	653016	3000.06	32	1255272	3000.02
13	*21	255590	1300.91	33	1254258	3000.02
14	32	727303	3000.06	-	-	3000.02
15	31	787821	3000.06	35	1324310	3000.02
Average		532197.53	2250.96		1378127.93	3000.02

Table 5. Large-sized capacitied SONET instances

The results for the capacitied instances, where the FS model is used in conjunction with a FD model of the capacities, show that our hybrid FS solver can interact well with FD models and solvers. While the results for the large instances demonstrate that the prototype implementation scales well. Our stronger inferences and tighter domain approximation allow these problems to be solved many orders of magnitude faster than subset bound solvers.

Comparison with 0-1 CP models An alternative CP approach to the SONET problem was proposed in [53, 54]. The first approach ignores the demand capacities of the rings due to their additional complexity. It considers a 0-1 matrix formulation and derives tailored approaches to break symmetries in order to derive and prove optimality efficiently. The decision variables are 0/1 variables x_{ik} that take the value 1 if the node i is assigned to ring k. Set variable are used to express constraint (8). The search is driven by the 0-1 variables. The problem has been addressed using the "armoury of CP modelling" quoting the author in order to derive efficient results. Given the effort spent on re-modelling, we will compare against the initial models that resemble most our simple primal/dual set model and basic heuristic search technique.

Table 6, taken directly from [53], gives the results of the 0-1 CP model enriched with implied constraints, variable and value ordering heuristics, and dynamic symmetry breaking (SBDS). It showcases results of the instances given in Table 2 without optimization. The results are based on only 4 rings instead of the 6 given in the benchmarks since as the author says "they are sufficient for all optimal solutions". This clearly reduces further the time to derive and prove optimality.

		With SBDS			No symmetry breaking		
Instance	Value	F	P	time(sec)	F	P	time(sec)
1	14	1	15	0.11	1	15	0.07
2	14	1	25,044	11.6	1	188,664	52.4
3	14	24	103	0.14	24	324	0.21
4	13	1,618	27,264	12.8	3,922	172,315	58.2
5	15	1	82,531	40.6	1	745,525	287
6	14	1	36,901	16.8	1	279,085	91.6
7	13	922	933	0.81	2,126	2,137	1.33
8	14	1,657	28,206	12.4	4,180	187,902	52.9
9	15	36,395	71,960	41.3	133,684	507,068	196
10	14	374	435	0.36	983	1,256	1.03
11	12	320	331	0.35	751	762	0.91
12	15	1,022	117,855	62.3	4,976	1,292,875	553
13	15	1,870	89,117	47.0	10,770	1,018,949	433
14	15	9	71,127	35.5	9	908,497	387
15	15	4	53,851	26.0	5	468,969	180

Table 6. Table of [53]: Solving medium-sized uncapacitied SONET instances, allowing only 4 rings, with and without symmetry breaking, using ILOG Solver. 'Value' is the minimum number of ADMs required. F is the number of backtracks (fails) to find the optimal solution, P is the total number of backtracts to prove optimality. Time is the cpu time in sec. on a 600MHz Celeron PC.

The 0-1 and set models are quite similar (primal and dual models with finite subsets represented as sets or 0-1 variables) but differ in their resolution. The 0-1 model is solved by branching on the 0-1 variables, adding implied constraints, heuristics, and dynamic symmetry breaking techniques; whereas the set model branches on the set variables, uses a simple labelling strategy and breaks symmetries by adding stating lexicographic ordering constraints. The main observation is that the 0-1 CP approach is not robust: its behaviour varies a lot among the instances while the set-CSP model and hybrid solver (see table 2) is reliable, suggesting that instances 5, 9 and 14 are the hardest. This can be due to the role of the variable ordering heuristics which suits some instances and not others in the 0-1 CP model. Finally, bearing in mind that the set approach considers the full problems (i.e. 6 rings) it still outperforms the 0-1 CP model in 10 out of 15 cases, shown in bold in Table 6.[2]

To improve performance the 0-1 CP model was strengthened with numerous redundant constraints and constraints which (though not strictly speaking redundant) will hold true for at least one optimal solution. In an attempt to find solutions more quickly, in [53], the author also employs dynamic symmetry breaking, handcrafted search heuristics and specialised code to detect optimality. This extra remodelling effort is rewarded with fast optimal solutions for medium-sized instances of the problem ignoring ring capacities with an average

[2] Though the experiments were performed on a slower computer, they were performed using Solver which performs basic FS and FD operations much faster than our prototype solver.

time of 1.4 seconds. The model was further improved together with its performances, in [54] by considering the number of ADMs needed by each node as decision variables (instead of the assignment of a node to a ring), together with three different variable ordering heuristics. The outcome was an improvement of few orders of magnitude towards the simpler approach, with solutions to the capacitied instances (using Solver 6.0 on a 1.7GHz Pentium M PC).

6.2 t-designs

Definition 8. *(t-design). Let $A = \{1, 2, ..., v\}$. Let S be a collection of distinct k-element subsets of A. The pair (A, S) is a $t-(v, k, \lambda)$ design* ***iff*** $0 < t \leq k < v, \lambda > 0$ *and every t-element subset of A is contained in exactly λ of the blocks in S.*

In general $t-(v, k, \lambda)$ designs are referred to as t-designs, A is known as the *base set* and the elements of S are called *blocks*. Note that in the above definition the pair (A, S) is a configuration satisfying the constraints of the design, and we refer to S as an instance of the design.

We first present generic models for any $t-(v, k, \lambda)$ design and will look experimentally at Steiner systems (ie. $t-(v, k, 1)$ designs) [36]. We consider three models (primary, dual and primal+dual), with and without symmetry breaking ordering constraints. Recall that the dual model is an alternative yet still natural and intuitive model for many CDPs which may allow certain problem constraints to be expressed more concisely and/or to improve propagation in conjunction with the primal model.

Generic t-design Models We will give three complete models for t-designs, all of which are parameterised by the t parameters: the number of elements (v) in the base set ($V = \{1, ..., v\}$), the cardinality of each block in the design (k), the number of blocks (λ) that any t element subset of V must appear in. Recall also the packing number $D_\lambda(v, k, t)$ which defines the number of blocks in a design. For convenience, we number each block from 1 to $D_\lambda(v, k, t)$ and denote the set of all such block numbers as $B = \{1, ..., D_\lambda(v, k, t)\}$.

Primal Model Primal set variables X_i correspond directly to blocks in the design.

$$\text{for } i \in B \qquad X_i \subseteq V \tag{17}$$

$$\text{for } i \in B \qquad |X_i| = k \tag{18}$$

$$\text{for } is \subseteq B, |is| = \lambda + 1 \; | \bigcap_{i \in is} X_i | < t \tag{19}$$

Equation 19 ensures that any subset common to more than λ blocks must have strictly fewer than t elements, thus ensuring that the blocks form at least a packing. The packing is known to be a design because of the number of blocks.

Optionally we can remove some of the symmetries by enforcing an order between the primal variables.

$$\text{for } 1 \leq i < |B| \; X_i \preceq X_{i+1} \tag{20}$$

In some cases the $\preceq$ constraint can be strengthened to $\prec$, for example when $\lambda = 1$.

Dual Model Dual set variables X'_j correspond to the set of block identifiers for blocks containing the element j.

$$\text{for } j \in V \qquad X'_j \subseteq B \tag{21}$$

$$\text{for } j \in V \; |X'_j| = D_\lambda(v-1, k-1, t-1) \tag{22}$$

$$\text{for } js \subseteq V, |js| = t \qquad |\textstyle\bigcap_{j \in js} X'_j| = \lambda \tag{23}$$

Equation 23 is a direct translation of the "t element subset occurs in exactly λ blocks" condition for t-designs.

As with the primal model, we can post ordering constraints to break symmetries here as well.

$$\text{for } 1 \leq j < |V| \; x'_j \preceq X'_{j+1} \tag{24}$$

Primal+Dual Model Though the above two models are both complete descriptions of the t-design problem, they can be combined to provide stronger propagation. Such a combination is a simple matter of adding channelling constraints to tie the primal and dual set variables together.

$$\text{for } i \in B, j \in V \; j \in X_i \iff i \in X'_j \tag{25}$$

Experiments - Steiner Systems ($\boldsymbol{\lambda = 1}$) Recall that Steiner Systems are t-designs where the λ parameter is 1.

Definition 9 (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 the *dual* model. Instead of modelling the blocks themselves as set variables we 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. A global constraint `atmost1` is defined and does

strictly more than just constraining the size of the dual-sets [47]. 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 7. 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 8. Time to find first soln.

	time (s)			
	primal		+dual sum	
	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

Table 7 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 8 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.

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 9 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. It appears that the performance results on Steiner systems are caused by cases where the system of constraints would "stall" in a way similar to FD constraints ($x < y, y < x$). This is discussed in Section 8 with some directions to address this point.

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

Table 9. Time and backtracks taken to find all 151200 solutions of $S(2, 3, 7)$

6.3 Symmetry

Much work has been done recently to improve the efficiency of searching for solutions of highly symmetric problems. We compare our work with developments in one particular family of symmetry breaking techniques, the lex-ordered symmetry breaking constraint in 0-1 incidence matrix models.

The idea put forward in [27], consists of showing how a 0-1 matrix model for Steiner systems can be enhanced by the use of a specialised global constraint which combines symmetry breaking and the sum constraint. An experimental evaluation is carried out on small instances of Steiner systems. The authors show how existing symmetry breaking techniques like lex [17, 19] 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. They demonstrate their technique on finding and proving the non-existence of a number of small Steiner systems.

The model chosen 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. The authors 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 9.` and

`IR 10`. 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 [27] by channelling to a matrix of reified inclusion Booleans.

Table 10. Comparison with table 1 of [27]. 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 11. Comparison with table 3 of [27]. 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

In tables 10 and 11 we duplicate and extend the results of [27], adding for comparison the final column showing how our model performs. Note that the third column contains the backtrack values from the original paper[3], 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 outperforms 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.

Results 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 a high level set-CSP 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 create specialized global constraint propagation algorithms. Our hybrid domain and inferences subsume the inferences of this global constraint

[3] The experiments were run using ILOG Solver 5.3 on a 1GHz pentium III processor with 256 Mb RAM under Windows XP

for these problems. The main strength is that the lex ordering of the symmetry breaking constraints coincides with the lex bounds ordering, and combined with cardinality restrictions leads to powerful pruning.

7 Improving Performance

Without any alterations to the set model, our hybrid domain solver is able to significantly reduce the search space for many problems compared to traditional subset bound solvers. However the performance results are not all similar. There is a discrepancy between the spectacular improvements on the SONET problem viz. the subset bound solver and the more moderate ones on the Steiner systems. We have identified two performance issues during our further experimental work on the Steiner systems [46]. They are presented here with directions towards addressing them successfully.

7.1 Implementation of a Bespoke Hybrid Solver

The implementation of our solver on top of `ic_sets` solver gives modularity and flexibility to implement the algorithms in any CLP system equipped with a subset bound and FD solver. However, solvers like `ic_sets` use a domain structure which is optimised according to the assumption that all domain updates (subset bounds) can be performed using single element insertions or deletions. Explicitly enumerating the subset bounds is assumed to be a rare operation and therefore an array of variables is used which allows constant time insertions and deletions at the expense of subset bounds enumeration.

In our hybrid solver, enumeration of the subset bounds is an essential operation, more essential (i.e. more frequently applied) than single element insertions or deletions. As such, the hybrid inferences use a list structure to make enumeration efficient at the expense of insertions/deletions. In our prototype we maintain both structures simultaneously, but even allowing for the caching method described in section 6, there is a time overhead to update them both as well as the space overhead of storing them both. A bespoke hybrid domain solver should use a common data structure tuned to make enumeration and comparison as efficient as possible. We also note that `ic-sets` does not use the cardinality inference rules presented in [2]. The current prototype does not use them either, but with an optimized bespoke hybrid solver, it will be interesting to evaluate their effectiveness.

Bitmaps Depending on the existing ground set representations within a given set solver, the following equivalence between the characteristic vector and the characteristic binary number order can be very useful if ground sets are represented within a RAM machine as bitmaps, since most CPUs contain hardware to perform the $\leq$ test in constant time (for bitmaps up to a certain size). By treating the sets characteristic 0/1 vector as a binary number, we can see that the lexicographic order corresponds directly to the natural $\leq$ order on these

numbers (as also noted in [14]). Recall that a characteristic number ($\mathcal{N}_X$) for the ground set X is defined by: $\mathcal{N}_X \equiv \Sigma_{x \in X} 2^x$. Using this notation we get the following equivalence:

$$X \preceq Y \textbf{ iff } \mathcal{N}_X \leq \mathcal{N}_Y$$

BDD An alternative approach is the use of Binary Decision Diagrams. Recent results on the use of Reduced Ordered Binary Decision Diagrams (ROBDD) have shown their flexibility and efficiency in propagating over full set domains but also lex bounds, subset bounds, cardinality bounds [26]. In particular it illustrates the synergy between ROBDD data structures and lex bounds, in that it requires a smaller number of nodes to store the bounds compared to a full domain or cardinality bounds. Such an efficient implementation with an optimal ordering of the boolean variables guarantees a linear representation of most primitive set constraints, and can outperform set solvers such as `ic-sets` and MOZART on a set of combinatorial problems. Efficient libraries exist such as CUDD for manipulating such data structure [56].

7.2 Pathological cases

The second performance issue was less straightforward to identify. Despite the ability to compute individual lex bounds and achieve SBC^+ on the hybrid domain constraint in time polynomial in the size of the lub, we have detected cases where our approach can suffer from requiring an exponential amount of time to reach a fixed point for a system of FS constraints.

This problem is highlighted by the following simple constraint system, which has obvious analogies with pathological cases in FD bounds reasoning.

Example 9. $X \prec Y \wedge Y \prec X$

The actions of the generic rules `IR 25` and `IR 26` on the lex bounds of the set domains is to iteratively reduce the actual domain size by two until the domains become empty. As mentioned above this is clearly analogous to the action of bounds consistent propagation of the corresponding $X < Y \wedge Y < X$ system of FD constraints. But unlike the FD case which takes $\mathcal{O}(min(|b_X|, |b_Y|))$ steps to terminate as the domains are $|b_X|$ in size, the set domains may take $\mathcal{O}(min(|2^{|b_X|}, 2^{|b_Y|}))$ time to exhaust since their domains may be much larger.

Example 9 is an extreme case where the constraint system is unsatisfiable, and is a common problem for any bounds reasoning system. During the analysis of our experimental results, we discovered conditions under which our prototype solver would exhibit similar behaviour when the constraints system is satisfiable as in Example 10. Interestingly this point was also raised when evaluating the benefits of ROBDDs to model lex bound constraint propagators, but the reasons were not identified.

Example 10. Consider the task of pruning the lex lower bounds of X, given the system of constraints in Table 12. The table shows the different potential values of e_X which will be generated by the inference rules for the two intersection constraints. Each potential value taking $O(|b_X|)$ steps to generate.

	Initial	(C1.)	(C2.)	Fixed Point
	{3, 2, 1}			
		{5, 3, 1}	{6,4,2}	
		{7, 3, 1}	{7,4,2}	
		{7,5, 1}	{7,6,2}	
		{8, 3, 1}	{8,4,2}	
$X \subseteq \{9,8,7,6,5,4,3,2,1\}$		{8,5, 1}	{8,6,2}	
$\lvert X \rvert = 3$		{8, 7, 1}	{8,7,2}	
$\lvert X \cap \{9,8,7,,5,3,1\} \rvert \geq 3$ (C1.)		{8,7,3}	{8,7,4}	
$\lvert X \cap \{9,8,7,6,4,2\} \rvert \geq 3$ (C2.)		{8,7,5}	{8,7,6}	
		{9,3,1}	{9,4,2}	
		{9,5,1}	{9,6,2}	
		{9,7,1}	{9,7,2}	
		{9,7,3}	{9,7,4}	
		{9,7,5}	{9,7,6}	
		{9,8, 1}	{9,8,2}	
		{9,8,3}	{9,8,4}	
		{9,8,5}	{9,8,6}	
				{9,8,7}

Table 12. Exponential time to reach fixed point for simple system of two intersection constraints C1. and C2.

In our implementation of the inference rules we revise the lex bounds according to applicable inference rules on an individual basis until a fixed point is reached. This means that it is possible for two (or more) inference rules to repeatedly reduce the lex bounds of a number of variables until one of the domains has been exhausted. It is this worst case behaviour that causes the hybrid solver to "stall" on some of the Steiner instances mentioned earlier. The potentially exponential number of steps required to reach a fixed point stems from the treatment of each constraint individually (or locally). A more powerful approach could take a global view. For the problem of calculating lex-bounds which simultaneously satisfy a number of simple constraints, as in the above example, we could make use of the predictor functions introduced in Section 6.5. Since the predictor functions determine a priori whether a given element must be included or excluded from a bound in order to ensure that a single simple constraint can be satisfied, we could combine the results from multiple predictor functions in parallel.

For the above example this would result in the production of the correct fixed point value $\{9,8,7\}$ with a single call to the parallel bound computation function, taking only 21 steps of the generation algorithm (v.s. the $33 \times 9 = 297$ (values-returned $\times$ steps-per-value) steps of the non-parallel case).

8 Contribution and Perspectives

We have presented the formal and practical framework of an enhanced FS solver with lexicographic bounds and inference rules. It is able to reason more strongly

about FS models *without requiring any changes to the set-CSP model*. The new solver is based on a much tighter approximation of the set variable's domain, whereby the lex bounds satisfy the cardinality restrictions and the associated lex ordering allows to break symmetries effectively using lex ordering constraints. These properties of a set domain are novel and facilitate the stronger reasoning and interaction between constraints. The benefits are demonstrated experimentally illustrating how the structural properties of CDPs we have extracted can be modeled and tackled effectively, with spectacular improvements over subset bound solvers specially on the SONET problem. When a 0-1 CP model is chosen instead of the set model and more elaborate symmetry breaking techniques are used —compared to a simple lex ordering among set variables—, our hybrid domain and inferences subsume the inferences of those approaches, and is more robust on the instances presented.

Future work involves investigating new means to render the interaction between those three components (subset bounds, lex bounds and cardinality bounds) even stronger, as well as taking the steps identified in the previous section to improve performance.

Acknowledgements We thank the anonymous reviewers for their insightful comments.

References

1. F. Azevedo. *Constraint Solving over Multi-Valued Logics. Application to Digital Circuits*. Frontiers in Artificial Intelligence and Applications, 2003.
2. F. Azevedo and P. Barahona. Cardinal: an extended set solver. in *Proceedings of Computational Logic*, 2000.
3. N. Barnier and P. Brisset. Facile: A Functional Constraint Library. In *CICLOPS'01* workshop, help alongside with CP-2001.
4. N. Barnier and P. Brisset. Solving the Kirkman's Schoolgirl Problem in a Few Seconds. In M. Wallace, editor, *Proceedings of CP*-2004.
5. F. Benhamou. Interval Constraint Logic Programming. In A. Podelski, editor, *Constraint Programming: Basics and Trends*, LNCS 910, 1995.
6. C. Berge, *Principles of Combinatorics*. New York: Academic Press, 1971.
7. C. Bessière, B. Hnich, E. Hébrard, and T. Walsh. Disjoint, Partition and Intersection Constraints for Sets and Multiset Variables. In M. Wallace, editor, *Proceedings of CP*-2004, LNCS 3258.
8. C. Bessière, B. Hnich, E. Hébrard, and T. Walsh. The Tractability of Global Constraints. In M. Wallace, editor, *Proceedings of CP*-2004, LNCS 3258.
9. R.E. Bryant. *Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams*. ACM Comput. Surv., 24(3), 293–318, 1992.
10. R. Campbell. A minimum distances basketball scheduling problem, 1976.
11. J.G. Cleary. Logical arithmetic. *In Future Generation Computing Systems*, chapter 2(2),1987.
12. C. J. Colbourn and J. H. Dinitz, editors. *The CRC Handbook of Combinatorial Designs*, CRC Press, 1998.

13. C. J. Colbourn, J.H. Dinitz, and 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.
14. J. Crawford, M. Ginsberg, E.M. Luks, and A. Roy. Symmetry breaking predicates for search problems. In *Fifth Int. Conf. on Knowledge Rep. and Reasoning*, 1996.
15. A. Eremin, F. Ajili, and R. Rodosek . A Set-based Approach to the Optimal IGP Weight Setting Problem. In *Proceedings of INOC*-2005.
16. T. Fahle, S. Schamberger, M. Sellman. Symmetry breaking. In *Proceedings of CP'01*, pp 93–07, Springer 2001.
17. P. Flener, A. Frisch, B. Hnich, Z. Kiziltan, I. Miguel, J. Pearson, T. Walsh. Breaking row and column symmetries in matrix models. In *Proceedings of CP02*, Springer 2002.
18. I. Gent,W. Harvey, T. Kelsey and S. Linton. Generic SBDD Using Computational Group Theory. *In Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming (CP'03)*, Springer, 2003.
19. I.P. Gent, P. Prosser, B.M. Smith. A 0/1 encoding of the gaclex for pairs of vectors. In *ECAI-W9 Modelling and Solving Problems with Constraints*, 2002.
20. C. Gervet. New Structures of Symbolic Constraint Objects: Sets and Graphs. In *Third Workshop on Constraint Logic Programming (WCLP'93)*, 1993.
21. C. Gervet. Conjunto : Constraint Logic Programming with Finite Set Domains. In M. Bruynooghe, editor, *Proceedings of ILPS*-1994.
22. C. Gervet. *Set Intervals in Constraint Logic Programming: Definition and Implementation of a Language*. PhD thesis, Université de Franche-Comté, France, September 1995. European thesis, in English.
23. C. Gervet. Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language. *In CONSTRAINTS journal* 1(3), 1997.
24. C.D. Godsil. Linear Algebra and designs. Manuscript (229p.), 1995.
25. P. Hawkins, V. Lagoon, and P.J. Stuckey. Set bounds and (split) set domain propagation using ROBDDs. In G. Webb and X. Yu, editors, *Proceedings of AI'04: Australian Joint Conference on Artificial Intelligence*, LNCS 3339, 2004.
26. P. Hawkins, V. Lagoon, and P. Stuckey. Solving Set Constraint Satisfaction Problems using ROBDDs. *Journal of Artificial Intelligence Research* 24, 2005.
27. B. Hnich, Z. Kiziltan, T. Walsh. Combining symmetry breaking with other constraints: lexicographic ordering with sums. In *Proceedings of SymCon workshop* held alongside CP'03, 2003.
28. C. Holzbaur. Metastructures versus attributed variables in the context of extensible unification. In M. Bruynooghe and M. Wirsing, editors, *Proceedings of PLILP'92*. vol.631 LNCS, Springer, 1992.
29. Ilog. User's manual. ILOG Solver 6.0 Sept., 2003.
30. S. Le Huitouze. A New Datastructure for Implementing Extensions to Prolog. In *Proceedings of PLILP*-1990, LNCS 456.
31. Z. Kiziltan. Symmetry Breaking Ordering Constraints. *PhD thesis, Uppsala University*, Sweden, 2004.
32. Z. Kiziltan and T. Walsh. Constraint Programming with Multisets. In *Proceedings of the SymCon-02 workshop*, held alongside CP-2002.
33. F. Laburthe. CHOCO: Implementing a CP Kernel. http://choco.sourceforge.net/, 2000. In *Proceedings of TRICS*, held alongside CP-2000.
34. V. Lagoon and P.J. Stuckey. Set domain propagation using ROBDDs. In M. Wallace, editor, *Proceedings CP-2004*, LNCS 3258.

35. J.H.M. Lee and H. van Emden. Interval computation as deduction in CHIP. In *Journal of Logic Programming*, 16(3–4):255-276, 1993.
36. C.C. Lindner and A. Rosa. *Topics on Steiner Systems*, volume 7 of *Annals of Discrete Mathematics*. North Holland, 1980.
37. H. Liu, F. Tobagi. A novel efficient technique for traffic Grooming in WDM SONET with Multiple Line Speeds. *In Proceedings of IEEE ICC*. 2004.
38. A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 1977.
39. Mozart/Oz, `http://www.moxart-oz.org/`.
40. T. Müller and M. Müller. Finite Set Constraints in Oz. In *Workshop Logische Programmierung*, Burkhard Freitag and Dietmar Seipel, editors, 13, 1997.
41. T. Müller. Solving Set Partitioning Problems with Constraint Programming. In *Proceedings of PAPPACT*-1998.
42. W. Older and A. Vellino. Constraint Arithmetic on Real Intervals. In F. Benhamou and A. Colmerauer, editors, *Constraint Logic Programming: Selected Papers*. MIT Press, 1993.
43. J-F. Puget. PECOS a High Level Constraint Programming Language In *Proceedings of Spicis*, 1992.
44. J-F. Puget. A fast algorithm for the bound consistency of alldiff constraints. In *Proceedings of AAAI'98*, 1998.
45. C-G. Quimper, A. Lopez-Ortiz, P. van Beek, and A. Golynski. Improved algorithms for the global cardinality constraint. *In Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming*, Toronto, September, 2004.
46. A. Sadler. *Strengthening Finite Set Constraint Solvers through Active Use of Problem Structure, Symmetries and Cardinality Information*. PhD thesis, University of London, Imperial College, 2005.
47. A. Sadler and C. Gervet. Global Reasoning on Sets. In *FORMUL'01 workshop on modelling and problem formulation* held alongside CP-2001.
48. A. Sadler and C. Gervet. Global Filtering for the Disjointness Constraint on Fixed Cardinality Sets. Technical report ICPARC-04-02, March 2004.
49. A. Sadler and C. Gervet. Hybrid Set Domains to Strengthen Constraint Propagation and Reduce Symmetries. In M. Wallace, editor, *Proceedings of CP*-2004, LNCS, 2004.
50. J. Schimpf, A. Cheadle, W. Harwey, A. Sadler, K. Shen, and M. Walllace. ECL^{i}PS^e Technical report 03-1, IC-Parc, Imperial College London, 2003.
51. M. Sellman and P. van Hentenryck. Structural Symmetry Breaking. *In Proceedings of IJCAI'05*, 2005.
52. H. D. Sherali and J. C. Smith. *Improving discrete model representations via symmetry considerations.* Management Science, 47:1396–1407, 2001.
53. B.M. Smith. Search strategies for optimization: Modelling the sonet problem. In 2nd International Workshop on Reformulating CSPs, 2003.
54. B.M. Smith. Symmetry and Search in a Network Design Problem. In *Proceedings of CP'AI-OR'05*, LNCS 3524, Springer, 2005.
55. J.C. Smith. Tight discrete formulations to enhance solvability with applications to production, telecommunications and air transportation problems. *PhD thesis, Blacksburg, Virginia*, 2000.
56. F. Somenzi. CUDD: Colorado University Decision Diagram package. `http:`/vlsi.colorado.edu/ fabio/CUDD/.
57. P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* Logic Programming Series. The MIT Press, 1989.

58. T. Walsh. Consistency and Propagation with Multiset Constraints: A Formal Viewpoint. In *Proceedings of CP*-2003, LNCS.
59. N.F. Zhou. B-Prolog http://www.probp.com/.