

# Evaluation of Length-Lex Set Variables

Justin Yip and Pascal Van Hentenryck

Department of Computer Science, Brown University, USA

**Abstract.** This paper presents the first experimental evaluation of the length-lex domain for set variables. The implementation is based on bound-consistency algorithms proposed in earlier work and two novel technical contributions: a generic filtering algorithm which automatically pushes ordering constraints into symmetric binary constraints with only a logarithmic overhead and an adaptation of symmetry-breaking constraints from 0/1 matrices to the length-lex ordering. The experimental results indicate that the length-lex representation for set variables is very effective and robust on traditional set-CSPs benchmarks.

## 1 Introduction

Set variables for constraint programming were proposed in the early 1990s [1] but they have received increasing attention in recent years. The main issue in set variables is that their domain may contain an exponential number of sets and an explicit representation is often too costly. Hence, most of the research has focused on representations that either approximate the set domain or are as compact as possible. The traditional set domain is the subset-bound representation [1, 2], which maintains a pair of sets  $(R, P)$  to denote the domain  $\{s \mid R \subseteq s \subseteq P\}$ . The addition of a cardinality component to the subset-bound representation, and the associated pruning rules, was proposed in [3, 4]. This richer domain was further enhanced by a lexicographic component in [5, 6] since many set constraint satisfaction problems contain many symmetries which can be broken by lexicographic constraints. 0/1 matrix models can also be used to encode the characteristic function of the subset-bound domain, while cardinality and lexicographic restrictions are expressed by posting constraints [7, 8]. An exact representation of set domains using binary decision diagrams was proposed in [9]. It enables the solver to enforce arc consistency. The representation may require exponential space in the worst case, but convincing experimental results were given on a variety of benchmarks.

This paper considers the length-lex representation recently proposed in [10]. In theory, the length-lex representation offers three fundamental advantages. First, contrary to the subset-bound representation, it features a total ordering on sets, which makes it possible to enforce bound consistency. Second, the length-lex representation directly captures cardinality and lexicographic information. Finally, the representation often makes it possible to enforce bound consistency in polynomial time. However, despite these potential theoretical benefits, the length-lex domain was never evaluated experimentally.

The contributions of this paper are threefold. First, the paper presents the first experimental evaluation of the length-lex representation for set variables. The implementation, which consists of about 18,000 lines of code, was integrated inside the COMET system and evaluated on the traditional benchmarks for set variables. The experimental results on this preliminary implementation indicate that the length-lex representation for set variables is extremely effective on traditional benchmarks and the filtering algorithms reduce the search space by orders of magnitude compared to most representations. The models used for the benchmarks also feature two technical contributions. The first technical contribution is algorithmic and shows how to push lexicographic ordering constraints into (symmetric) set constraints. The idea of pushing lexicographic constraints into other constraints was proposed before (e.g., [11, 10, 20]), as it typically achieves a stronger propagation. The novelty in this paper is a generic routine that pushes the length-lex ordering into any symmetric binary constraint with only an  $O(\log n)$  overhead. The second technical contribution pertains to dual modeling and symmetric breaking in primal/dual set-CSPs [7, 8, 13–16]. This paper shows that the traditional techniques used in 0/1 matrices to break variables and value symmetries also apply to the length-lex ordering. This is particularly significant since the length-lex representation capture lexicographic information directly, allowing a strong pruning of the search space. The rest of this paper is organized as follows. Section 2 gives an overview of the length-lex representation. Section 3 introduces the filtering algorithms to handle the combination of a symmetric constraint and a length-lex ordering constraint. Section 4 presents how to adapt the dual modeling technique to the length-lex domain. Section 5 presents the empirical results and Section 6 concludes the paper.

## 2 The Length-Lex Representation of Set Variables

*Notations:* For simplicity, we assume that sets take their values in a universe  $U(n)$  of integers  $\{1, \dots, n\}$  equipped with traditional set operations.  $n, m$  are integers denoting the size of a universe, the number of variables, or both.  $S, T, P$  and  $Q$ , possibly subscripted, are set variables.  $X$  and  $Y$  are the length-lex domains of  $S$  and  $T$  respectively. Elements of  $U(n)$  are denoted by the letter  $f$ , possibly subscripted and modified as  $\check{f}$ ,  $\dot{f}$  and  $\hat{f}$  to denote the minimum, mean and maximum value respectively. Sets are denoted by  $lb, ub, s, t, w, x, y, z$ . A subset  $s$  of  $U(n)$  of cardinality  $c$  is called  $c$ -set and is denoted as  $\{s_1, s_2, \dots, s_c\}$  where  $(s_1 < s_2 < \dots < s_c)$ . The notation  $s_{i..j}$  is a shorthand for  $\{s_i, s_{i+1}, \dots, s_j\}$ .

*Length-Lex Representation:* The length-lex ordering  $\preceq$ , proposed in [10], totally orders sets first by cardinality and then lexicographically.

**Definition 1 (Length-Lex Ordering).** *The length-lex ordering is defined by*

$$s \preceq t \text{ iff } s = \emptyset \vee |s| < |t| \vee |s| = |t| \wedge (s_1 < t_1 \vee s_1 = t_1 \wedge s \setminus \{s_1\} \preceq t \setminus \{t_1\})$$

*Its strict version is defined by  $s \prec t$  iff  $s \preceq t \wedge s \neq t$ .*

*Example 1 (Length-Lex Ordering).* Given  $U(4) = \{1, \dots, 4\}$ , we have  $\emptyset \prec \{1\} \prec \{2\} \prec \{3\} \prec \{4\} \prec \{1, 2\} \prec \{1, 3\} \prec \{1, 4\} \prec \{2, 3\} \prec \{2, 4\} \prec \{3, 4\} \prec \{1, 2, 3\} \prec \{1, 2, 4\} \prec \{1, 3, 4\} \prec \{2, 3, 4\} \prec \{1, 2, 3, 4\}$ .

**Definition 2 (Length-Lex Interval).** Given a universe  $U(n)$ , a length-lex interval is a pair of sets  $\langle lb, ub \rangle$ . It contains all sets (inclusively) between  $lb$  and  $ub$  in the length-lex ordering, i.e.,  $\{s \subseteq U(n) \mid lb \preceq s \preceq ub\}$ .

*Example 2 (Length-Lex Interval).* Given  $U(6)$ , the interval  $\langle \{1, 3, 4\}, \{1, 5, 6\} \rangle$  denotes the set  $\{\{1, 3, 4\}, \{1, 3, 5\}, \{1, 3, 6\}, \{1, 4, 5\}, \{1, 4, 6\}, \{1, 5, 6\}\}$ .

Because the length-lex ordering defines a total order on sets, it is possible to enforce bound consistency on set constraints.

**Definition 3 (Bound Consistency for Binary Constraint).** A constraint  $\mathcal{C}$  over two set variables  $S$  and  $T$  with respective domains  $X = \langle lb_X, ub_X \rangle$  and  $Y = \langle lb_Y, ub_Y \rangle$  is bound consistent if

$$\exists y \in Y : \mathcal{C}(lb_X, y) \wedge \exists y \in Y : \mathcal{C}(ub_X, y) \wedge \exists x \in X : \mathcal{C}(x, lb_Y) \wedge \exists x \in X : \mathcal{C}(x, ub_Y).$$

Consistency algorithms for the length-lex representation have been studied intensively. Reference [10] presented algorithms to enforce bound consistency on many unary constraints in time  $\tilde{O}(c)$ , where  $c$  is the maximum cardinality of the set variable, and provided the first propagator for set domains whose running time is independent of the universe size  $n$  (usually  $n \gg c$ ). A generic algorithm that enforces bound consistency on binary constraints and only relies on a simple feasibility routine was proposed in [17]. The generic algorithm can easily be generalized to a propagation routine for a  $k$ -arity constraint. That paper also demonstrated a specialized algorithm for binary disjoint constraint running in time  $O(c^3)$ . An  $O(n^4/\epsilon)$  propagator for approximated bound consistency of a combination of two knapsack constraints was proposed in [19], while an algorithm for enforcing bound consistency on a knapsack constraint running in time  $\tilde{O}(c)$  with an one-time preprocessing cost  $O(c^2n)$  was proposed in [18]. The same paper also gave an amortization scheme that reduces the running time for the binary disjoint constraint from  $O(c^3)$  to  $O(c^2)$ .

*PF-interval:* [17] proposed the concept of PF-closed interval (hereinafter called *PF-interval* for short) in order to simplify the design of propagators. The key idea is that any length-lex interval can be partitioned into a linear number of PF-intervals, a special class of length-lex interval that enjoys some elegant properties of the subset-bound representation with cardinality. Informally, a PF-interval denotes all  $c$ -sets that begin with the same prefix, immediately followed by one element  $f$  of a set  $F$  (we call it the F-set), the rest being filled by elements greater than  $f$ . The F-set is critical for the efficient inference of lexicographic order, since it determines the most significant element after the required prefix. Moreover, the generic algorithm in [17] guarantees that the F-set is always a range. This paper exploits this property and formally redefines PF-interval.

**Definition 4 (PF-Interval).** Let  $P$  be a set and  $\check{f}$ ,  $\hat{f}$ ,  $n$  and  $c$  be integers. A PF-interval  $pf\langle P, \check{f}, \hat{f}, n, c \rangle$  satisfies

$$(\max(P) < \check{f}) \wedge (\check{f} \leq \hat{f}) \wedge (n - \hat{f} + 1 \geq c - |P|)$$

and denotes the set of sets

$$\left\{ P \uplus \{f\} \uplus s \mid \check{f} \leq f \leq \hat{f} \wedge s \subseteq \{f+1, \dots, n\} \wedge |P \uplus \{f\} \uplus s| = c \right\}.$$

*Example 3 (PF-interval).* Consider the length-lex interval in example 2. It contains all sets that begin with  $\{1\}$ , and immediately followed by 3, 4 or 5, and filled by elements in  $\{3, \dots, 6\}$ . It can be expressed as a PF-interval  $pf\langle \{1\}, 3, 5, 6, 3 \rangle$ .

*Example 4 (PF-interval).* The PF-interval  $pf\langle \{1, 2\}, 5, 6, 8, 4 \rangle$  denotes of set of sets  $\{\{1, 2, 5, 6\}, \{1, 2, 5, 7\}, \{1, 2, 5, 8\}, \{1, 2, 6, 7\}, \{1, 2, 6, 8\}\}$ .

The structure of PF-interval makes its inferences almost equivalent to the subset-bound+cardinality representation. The inferences are based on a feasibility routine  $hs$  that takes two intervals and return whether there is a solution.

**Specification 1 (Feasibility Routine  $hs\langle \mathcal{C} \rangle$ ).** Given a constraint  $\mathcal{C}$  and length-lex intervals  $X$  and  $Y$ ,  $hs\langle \mathcal{C} \rangle(X, Y) \equiv \exists x \in X, y \in Y : \mathcal{C}(x, y)$ .

Consider the binary disjoint constraint  $\mathcal{D}$ . Given two PF-intervals, the feasibility routine  $hs\langle \mathcal{D} \rangle$  checks whether two prefixes (which corresponds to required sets in the subset-bound domain) are disjoint and whether there are enough *free* elements (possible sets) to satisfy the cardinality requirements. In addition, the routine must check if both PF-intervals can pick different elements from their F-set. If these three conditions hold, there is a solution. The successor(predecessor) construction algorithms are based on this feasibility routine and they greedily pick the smallest(largest) element one at a time that satisfies the feasibility routine. The whole process takes  $O(c)$  time.

However, not all length-lex intervals enjoy these elegant properties. This problem can be remedied by observing any length-lex interval can be partitioned into  $O(c)$  PF-intervals. With this decomposition technique available, we can enforce bound consistency on binary length-lex constraint by first partitioning both length-lex intervals into  $O(c)$  PF-intervals, and then performing pair-wise comparisons. The total runtime is  $O(c^3)$ . See [17] for more details.

### 3 Pushing Length-Lex Ordering into Binary Constraints

This section shows how to push a length-lex constraint into a binary symmetric constraint. The idea of pushing lexicographic constraints into other constraints was proposed before (e.g., [11, 10, 20]). In particular, [11] presented an algorithm to enforce the lexicographic ordering and sum constraint for two vectors of variables simultaneously. [20] proposed an generic algorithm to push the lexicographic order into a global constraint by invoking  $O(m)$  calls to the domain-consistent global constraint propagator,  $m$  being the number of variables.

---

**Algorithm 1**  $bc\langle\mathcal{C}_{\preceq}\rangle(X = \langle lb_X, ub_X \rangle, Y = \langle lb_Y, ub_Y \rangle)$

---

- 1:  $(lb_X, ub_X) \leftarrow (succ_X\langle\mathcal{C}_{\preceq}\rangle(X, Y), pred_X\langle\mathcal{C}_{\preceq}\rangle(X, Y))$
  - 2:  $(lb_Y, ub_Y) \leftarrow (succ_Y\langle\mathcal{C}_{\preceq}\rangle(X, Y), pred_Y\langle\mathcal{C}_{\preceq}\rangle(X, Y))$
  - 3: **return**  $lb_X \neq \perp \wedge ub_X \neq \perp \wedge lb_Y \neq \perp \wedge ub_Y \neq \perp$
- 

$\check{X}:\langle\{1, 4, 5\}, \{6, 7, 9\}\rangle$	$\check{Y}:\langle\{2, 3, 4\}, \{6, 8, 9\}\rangle$	$\check{X}_{pf}^1: pf\langle\{1\}, 4, 8, 9, 3\rangle$	:
$\check{X}:\langle\{1, 4, 5\}, \{1, 8, 9\}\rangle$	$\check{Y}:\emptyset$	$\check{X}_{pf}^1: pf\langle\{\}, 2, 5, 9, 3\rangle$	$\check{Y}_{pf}^1: pf\langle\{\}, 2, 5, 9, 3\rangle$
$\check{X}:\langle\{2, 3, 4\}, \{6, 7, 9\}\rangle$	$\check{Y}:\langle\{2, 3, 4\}, \{6, 7, 9\}\rangle$	$\check{X}_{pf}^2: pf\langle\{6, 7\}, 8, 9, 9, 3\rangle$	$\check{Y}_{pf}^2: pf\langle\{6, 7\}, 8, 9, 9, 3\rangle$
$\check{X}:\emptyset$	$\check{Y}:\langle\{6, 8, 9\}, \{6, 8, 9\}\rangle$	:	$\check{Y}_{pf}^1: pf\langle\{6, 8\}, 9, 9, 9, 3\rangle$

**Table 1.** The Slicing (Left) and Decomposition (Right) of Length-Lex Domains.

In this section, we give a generic bound consistency algorithm  $bc\langle\mathcal{C}_{\preceq}\rangle$  that pushes the length-lex ordering constraint into a symmetric binary constraint<sup>1</sup>, only assuming a feasibility routine  $hs\langle\mathcal{C}\rangle$ . The algorithm  $bc\langle\mathcal{C}_{\preceq}\rangle$  enforces bound consistency with time  $O(\alpha c^2 \log^2 n)$ , which is a  $O(\log n)$  overhead incurred over the generic bound consistency algorithm  $bc\langle\mathcal{C}\rangle$ . Indeed,  $bc\langle\mathcal{C}\rangle$  is an algorithm presented in [17] which enforces bound consistency in  $O(\alpha c^2 \log n)$  time and only relies on a feasibility routine  $hs\langle\mathcal{C}\rangle$  over two PF-intervals (that takes  $O(\alpha)$  time).  $hs\langle\mathcal{C}\rangle$  is usually computationally inexpensive. For instance, for binary disjoint constraint  $\mathcal{D}$ ,  $hs\langle\mathcal{D}\rangle$  takes  $O(c)$  and therefore  $bc\langle\mathcal{D}\rangle$  takes  $O(c^3 \log n)$ , which is only a  $O(\log n)$  overhead compared to the specialized  $O(c^3)$  algorithm.

**Definition 5 (Symmetric Constraint).** A binary constraint  $C$  over two set variables  $S$  and  $T$  is symmetric if and only if  $C(S, T) \Leftrightarrow C(T, S)$ .

The generic algorithm  $bc\langle\mathcal{C}_{\preceq}\rangle$  is depicted in Algorithm 1 and simply computes the predecessor and successor for the two sets. What is interesting is their implementation which is based on two key observations. First, if the greatest set in  $X$  is smaller than the smallest set in  $Y$ , the length-lex ordering constraint is entailed and we can simply apply  $bc\langle\mathcal{C}\rangle$ . Second, when two PF-intervals are identical ( $X = Y$ ) and  $C$  is symmetric, if  $hs\langle C\rangle(X, Y)$  holds, then  $hs\langle\mathcal{C}_{\preceq}\rangle(X, Y)$  also holds. As a consequence of the first property, the algorithms start by slicing the representation in several pieces, which we first illustrate on an example before defining it formally.

*Example 5.* Suppose we have  $U(9)$ ,  $X = \langle\{1, 4, 5\}, \{6, 7, 9\}\rangle$ ,  $Y = \langle\{2, 3, 4\}, \{6, 8, 9\}\rangle$ .  $X$  can be sliced into two length-lex intervals, the first interval  $\check{X}$  contains all sets smaller than  $\min Y = \{2, 3, 4\}$  and the second interval  $\check{X}$  contains all sets in  $Y$ . Similarly,  $Y$  can also be sliced. Table 1 shows the partitions. The table also shows the decomposition of the resulting intervals in PF-intervals. Note that all sets in  $\check{X}$  and  $\check{Y}$  satisfy the ordering constraint.

<sup>1</sup> The restriction to symmetric constraint is natural, since otherwise the symmetry would already be broken by the constraint itself.

**Specification 2 (3Slices).** Given two length-lex intervals  $X$  and  $Y = \langle lb_Y, ub_Y \rangle$ .  $3Slices(X, Y)$  returns three intervals  $\check{X}$ ,  $\dot{X}$ , and  $\hat{X}$  such that

$$\check{X} \equiv \{x \in X \mid x \prec lb_Y\} \text{ and } \dot{X} \equiv \{x \in X \mid x \in Y\} \text{ and } \hat{X} \equiv \{x \in X \mid ub_Y \prec x\}$$

**Lemma 1.** Given two length-lex intervals  $X$  and  $Y$ .  $3Slices(X, Y) = \check{X}, \dot{X}, \hat{X}$  and  $3Slices(Y, X) = \check{Y}, \dot{Y}, \hat{Y}$ , we have  $\dot{X} = \dot{Y}$ .

The algorithm  $bc\langle C_{\preceq} \rangle$  first decomposes a length-lex interval into two parts. One part can ignore the length-lex ordering because the ordering constraint is already satisfied and calls the existing  $bc\langle C \rangle$  algorithm. The other part deals with two identical intervals and exploits the following symmetric property.

**Lemma 2.** If a binary constraint  $C$  is symmetric,  $hs\langle C \rangle(X, X) = hs\langle C_{\preceq} \rangle(X, X)$ .

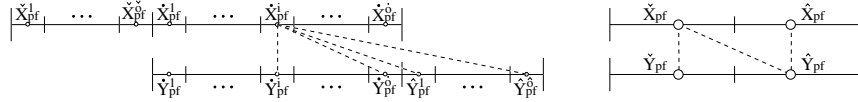
*Proof.* Suppose  $C(s, t)$  holds for  $s, t \in X$ . By symmetry,  $C(t, s)$  holds and  $s \preceq t \vee t \preceq s$  also holds.  $\square$

### 3.1 $pred_X$ for Length-Lex Intervals

We now demonstrate the generic algorithm (Algorithm 2) for finding the new upper bound for  $X$ .

**Specification 3 ( $pred_X$ ).** For a constraint  $C$  and two length-lex intervals  $X$  and  $Y$ , function  $pred_X\langle C \rangle(X, Y)$  returns the greatest set  $x \in X$  in the length-lex ordering which belongs to a solution of the constraint or  $\perp$  if there is no solution:

$$pred_X\langle C \rangle(X, Y) \equiv \begin{cases} \max\{x \in X \mid \exists y \in Y : C(x, y)\} & \text{if } hs\langle C \rangle(X, Y) \\ \perp & \text{otherwise} \end{cases}$$



**Fig. 1.** Key Observations.  $pred_X\langle C_{\preceq} \rangle(X, Y)$  (Left).  $pred_X\langle C_{\preceq} \rangle(X_{pf}, Y_{pf})$  (Right)

The algorithm first slices the length-lex intervals (lines 1–2) and then decompose the results into PF-intervals (lines 3–4).  $\hat{X}$  and  $\check{Y}$  are not considered as they violate the ordering constraints. The algorithm then locates the largest PF-interval that contains the new bound and constructs the new bound (lines 5–10). The routine for locating the PF-interval is similar to  $bc\langle C \rangle$ , except that we must pay some attention to the ordering constraint. The intuition is captured in Figure 1. When considering PF-intervals  $\dot{X}_{pf}^i$ , we only consider the PF-intervals in  $Y$  no smaller than  $\dot{X}_{pf}^i$ . Two cases must be distinguished. First, there is exactly one PF-interval in  $Y$  identical to  $\dot{X}_{pf}^i$  and the algorithm must call  $pred_X\langle C_{\preceq} \rangle$  to take into account the ordering constraint (line 8 and the vertical line in Figure 1 (Left)). Second, the remaining PF-intervals are greater than  $\dot{X}_{pf}^i$  and the algorithm simply calls  $pred_X\langle C \rangle$  (line 7 and the diagonal lines in Figure 1 (Left)). If no PF-interval in  $\dot{X}$  contain a new upper bound, we will try  $\check{X}$ . As all sets in  $\check{X}$  are smaller than  $Y$ , we use the simple predecessor algorithm  $pred_X\langle C \rangle$  (line 11). Note also that the loop starts with the largest PF-interval (line 6), since we are interested in the largest predecessor.

---

**Algorithm 2**  $pred_X \langle \mathcal{C}_{\prec} \rangle (X = \langle lb_X, ub_X \rangle, Y = \langle lb_Y, ub_Y \rangle)$

---

```

1:  $\tilde{X}, \hat{X}, \check{X} \leftarrow 3Slices(X, Y)$ 
2:  $\tilde{Y}, \hat{Y}, \check{Y} \leftarrow 3Slices(Y, X)$ 
3:  $[\tilde{X}_{pf}^1, \dots, \tilde{X}_{pf}^{\hat{o}}] \leftarrow decomp(\tilde{X})$ 
4:  $[\tilde{Y}_{pf}^1, \dots, \tilde{Y}_{pf}^{\hat{o}}] \leftarrow decomp(\tilde{Y})$ 
5:  $s \leftarrow \top$ 
6: for  $i = \hat{o}$  down to 1 do
7:    $s \leftarrow \max(\max_{j \in \{1..i\}}(pred_X \langle \mathcal{C} \rangle (\tilde{X}_{pf}^i, \hat{Y}_{pf}^j)), \max_{j \in \{i+1..\hat{o}\}}(pred_X \langle \mathcal{C} \rangle (\tilde{X}_{pf}^i, \check{X}_{pf}^j)))$ 
8:    $s \leftarrow \max(s, pred_X \langle \mathcal{C}_{\prec} \rangle (\tilde{X}_{pf}^i, \tilde{X}_{pf}^i))$ 
9:   if  $s \neq \top$  then
10:    return  $s$ 
11: return  $pred_X \langle \mathcal{C} \rangle (\tilde{X}, \hat{Y} \uplus \check{Y})$ 

```

---

### 3.2 $pred_X$ for PF-Interval

It remains to show how to find the predecessor for two identical PF-intervals (line 8). Algorithm 3 constructs the upper bound by performing a binary search in the F-set and the main idea is depicted in Figure 1 (Right). It partitions the F-set into two halves (line 9) and first checks if there is a solution in the upper half. (line 11 and the right vertical line in the figure). Notice that by Lemma 2, we can simply call  $hs \langle \mathcal{C} \rangle$ . This upper half needs to be compared only with the upper half  $\hat{Y}_{pf}$  of  $Y_{pf}$ , since the lower half  $\check{Y}_{pf}$  violates the ordering constraint. If there is a solution (line 10), the algorithm is called recursively within  $\hat{X}_{pf}$  (line 11). Otherwise, the algorithm tries the lower half  $\check{X}_{pf}$ . Now there are two possible choices ( $\tilde{Y}_{pf}$  and  $\hat{Y}_{pf}$ ) and we do not know which one gives a greater bound. The algorithm tries both (lines 13–14) and returns the largest (line 15). Once the F-set becomes a singleton, it is inserted in the prefix and the algorithm considers the next position (lines 3–8).

**Lemma 3.** *Suppose  $hs \langle \mathcal{C} \rangle (X_{pf}, Y_{pf})$  takes  $O(\alpha)$ . Algorithm 3 takes  $O(\alpha c^2 \log^2 n)$ .*

*Proof.* The running time of Algorithm 3 is affected by the number of unfixed positions (maximum is  $c$ ), the number of possible choices  $r$  in the F-set (i.e.,  $\hat{f} - \check{f} + 1$ ) and the universe size  $n$ . Denote the computation time of the algorithm by  $T_n(r, c)$ .  $T_n(r, c)$  can be expressed by the following recurrence relation:

$$T_n(r, c) = \begin{cases} O(\alpha) + \max(T_n(r/2, c), T_n(r/2, c) + O(\alpha c \log n)) & \text{if } r > 1 \\ T_n(n, c - 1) & \text{if } r = 1 \wedge c > 1 \\ O(1) & \text{otherwise} \end{cases}$$

Solving the recurrence relation gives  $T_n(r, c) = O(\alpha c^2 \log^2 n)$ . □

**Theorem 1.** *Assume  $hs \langle \mathcal{C} \rangle (X_{pf}, Y_{pf})$  takes time  $O(\alpha)$ . Then Algorithm 2 takes  $O(\alpha c^2 \log^2 n)$ .*

*Proof.* Locating the first supported  $X_{pf}$  takes  $O(\alpha c^2)$  time. Once the algorithm locates the  $X_{pf}$ , it constructs a predecessor against every possible  $Y_{pf}$ . There

---

**Algorithm 3**  $pred_X(\mathcal{C}_{\leq})(X_{pf} = pf\langle P, \check{f}, \hat{f}, n, c \rangle, Y_{pf} = pf\langle P, \check{f}, \hat{f}, n, c \rangle)$

---

**Require:**  $X_{pf} == Y_{pf}$

- 1: **if** not  $hs(\mathcal{C}_{\leq})(X_{pf}, Y_{pf})$  **then**
- 2:     **return**  $\perp$
- 3: **if**  $\check{f} = \hat{f}$  **then**
- 4:     **if**  $|P| = c - 1$  **then**
- 5:         **return**  $P \uplus \{\check{f}\}$
- 6:     **else**
- 7:          $P', \check{f}', \hat{f}' \leftarrow P \uplus \{\check{f}\}, \check{f} + 1, n - c - |P|$
- 8:         **return**  $pred_X(\mathcal{C}_{\leq})(pf\langle P', \check{f}', \hat{f}', n, c \rangle, pf\langle P', \check{f}', \hat{f}', n, c \rangle)$
- 9:      $\dot{f} \leftarrow (\check{f} + \hat{f})/2$
- 10: **if**  $hs(\mathcal{C})(pf\langle P, \dot{f} + 1, \hat{f}, n, c \rangle, pf\langle P, \dot{f} + 1, \hat{f}, n, c \rangle)$  **then**
- 11:     **return**  $pred_X(\mathcal{C}_{\leq})(pf\langle P, \dot{f} + 1, \hat{f}, n, c \rangle, pf\langle P, \dot{f} + 1, \hat{f}, n, c \rangle)$
- 12: **else**
- 13:      $s_0 = pred_X(\mathcal{C})(pf\langle P, \check{f}, \hat{f}, n, c \rangle, pf\langle P, \dot{f} + 1, \hat{f}, n, c \rangle)$
- 14:      $s_1 = pred_X(\mathcal{C}_{\leq})(pf\langle P, \check{f}, \hat{f}, n, c \rangle, pf\langle P, \dot{f}, \hat{f}, n, c \rangle)$
- 15:     **return**  $\max(s_0, s_1)$

---

are at most  $O(c)$  possible choices and  $O(c)$  of them require only the simple predecessor algorithm  $pred_X(\mathcal{C})$  that takes  $O(\alpha c \log n)$  [17]. At most one of them must be taken care specially by Algorithm 3 and takes  $O(\alpha c^2 \log^2 n)$ . Hence, the total run time is  $O(\alpha c^2 \log^2 n)$ .  $\square$

The above generic algorithm pushes the length-lex ordering constraint into arbitrary binary symmetric constraints. Specialized algorithms can of course be designed for specific constraints. For instance, there exists a specialized algorithm for binary disjoint&length-lex constraint with  $O(1)$  overhead. Space constraints do not allow us to describe it in detail. The key observation is by disjointness, we include the ordering constraint by considering the first element only.

## 4 Dual Modeling for Length-Lex Set Variables

This section considers *fully interchangeable* set-CSPs in which both the variables and the values are fully interchangeable. In the 0/1 matrix formulation, these symmetries are broken by imposing a lexicographic ordering on both the rows and columns. It is guaranteed that some solutions of each symmetry class remains after this process. Since the length-lex representation provides a total ordering on its sets, it also provides an ideal vehicle to break symmetries and we would like to use a similar technique with length-lex variables. Variable symmetries can be broken by imposing an ordering on the set variables. If the values are also interchangeable, we can consider the dual problem and impose an ordering on the dual variables. Unfortunately, it is unclear whether enforcing the length-lex ordering on both variables and values will still leave some solutions in each symmetry class, since the length-lex ordering is different from the lex ordering (See Example 6 below). The contribution of this section is to show that

	Original	0/1 of Original	Padded	0/1 of Padded
w	{1,3}	{1,0,1,0}	{-4,-3,1,3}	{1,1,0,0,1,0,1,0}
x	{3,4}	{0,0,1,1}	{-4,-3,3,4}	{1,1,0,0,0,0,1,1}
y	{1,2,4}	{1,1,0,1}	{-4,1,2,4}	{1,0,0,0,1,1,0,1}
z	{1,3,4}	{1,0,1,1}	{-4,1,3,4}	{1,0,0,0,1,0,1,1}

**Table 2.** Preserving the length-lex ordering by padding dummy elements

imposing a *double length-lex ordering* on a fully interchangeable set-CSP does not eliminate all solutions in each symmetry class.

**Definition 6 (Set-CSP).** A set-CSP is a pair  $\langle V \times D, C \rangle$ , where  $V$  denotes the set of variables,  $D$  denotes the universe for these variables. An (primal) assignment  $\gamma : V \rightarrow \mathbb{P}(D)$  maps variables to sets.  $C : (V \rightarrow \mathbb{P}(D)) \rightarrow \text{bool}$  is a constraint that specifies which assignments are solutions (i.e.  $C(\gamma) = \text{true}$ ).

**Definition 7 (Fully Interchangeable Set-CSP).** A set CSP is fully interchangeable if and only if when  $\gamma$  is a solution, for any bijective  $\sigma : V \rightarrow V$  and  $\tau : D \rightarrow D$ , and a mapping function  $\phi_f(s) = \{f(e) | e \in s\}$ , assignment  $\gamma' = \phi_\tau \circ \gamma \circ \sigma$  is also a solution.

The key observation is that, when sets are of the same cardinality, their length-lex order is equivalent to the lexicographic order. By padding some dummy elements to make all sets the same size, we can reduce the length-lex ordering technique to the lexicographic order of 0/1 matrices.

*Example 6.* Table 2 illustrates difference between length-lex and lex ordering for a universe  $U(4)$  and four sets  $w, x, y, z$ . The first column shows the sets in length-lex order. These sets are not in lex-order as  $x >_{lex} y$ . The 0/1 characteristic function (second column) is not in anti-lex-order either since  $x <_{lex} y$ . By padding dummy elements (third column), the sets are in both length-lex-order and lex-order and their 0/1 characteristic functions are in anti-lex order.

The formal definition of padding is as follows.

**Definition 8 (Padding).** We abuse the notation of a universe to allow negative value element, such that  $U'(n) = \{-n, \dots, -1, 1, \dots, n\}$ .  $pad_n : \mathbb{P}(U(n)) \rightarrow \mathbb{P}(U'(n))$  maps a set to a  $n$ -set padded by dummy elements. Formally, given  $s \subseteq U(n)$  and  $c = |s|$ ,  $pad_n(s) \equiv \{-n, \dots, -(c+1), s_1, \dots, s_c\}$ .

**Lemma 4.** Suppose  $s, t \subseteq U(n)$ .  $s \preceq t \Leftrightarrow pad_n(s) \leq_{lex} pad_n(t)$ .

*Proof.* Trivial when  $|s| = |t|$ . When  $|s| < |t|$ , denote  $s' = pad_n(s), t' = pad_n(t)$ , observe that  $s'_{1..n-|t|} = t'_{1..n-|t|}$  as they are dummy elements.  $s'_{n-|t|+1}$  is a dummy negative element, whilst  $t'_{n-|t|+1} = t_1$ , Hence  $s' \leq_{lex} t'$ .  $\square$

**Definition 9 (Double Length-Lex Primal/Dual Set-CSP).** Let  $\langle P_M \times N, C \rangle$  be a CSP where  $P_M = \{P_1, \dots, P_m\}$  are (primal) set variables and  $N = \{1, \dots, n\}$ . Its double length-lex primal/dual version is defined as  $\langle P_M \times N \uplus Q_N \times M, C' \rangle$  where  $Q_N = \{Q_1, \dots, Q_n\}$  are the dual set variables,  $M = \{1, \dots, m\}$  and

$$C' \equiv C \wedge (P_1 \preceq \dots \preceq P_m) \wedge (Q_1 \preceq \dots \preceq Q_n) \wedge \bigwedge_{i \in N, j \in M} (j \in P_i \Leftrightarrow i \in Q_j)$$

Name	Abbrev.	SG	EC	SS	BI	tl	cpu
Length-Lex/seq	Length-Lex	o	o	o	o	900	C2D-M 2.53GHz
ROBDD-split/seq [9]	Split/seq	o	x	x		600	P4 2.8GHz
ROBDD-split/minDom [9]	Split/dyn	o				600	P4 2.8GHz
ROBDD-domain/seq [9]	Domain/seq	x	o	o		600	P4 2.8GHz
ROBDD-bound/seq [9]	Bound/seq	x	x	o		600	P4 2.8GHz
Pair-atmost-1/minDom [21]	Pair/dyn	o					Xeon 3.8GHz
Dual-subset-Bound/seq [5][6]	Subset/seq		x	o		240	P4 2GHz
Dual-hybrid/seq [5][6]	Hybrid/seq		x	o		240	P4 2GHz
Cardinal/seq [4]	Card/seq			o		900	P4 2.4GHz
Cardinal/minDom [4]	Card/dyn	o				900	P4 2.4GHz
Dual-set-int/minDom [14]	Set-int/dyn	o		x		7200	Sun Blade 1000
Valprec+dual/minDom [14]	Valprec/dyn	x		o		7200	Sun Blade 1000
0/1-Matrix-lex-sum/seq [8][22]	Lex-sum/seq	x		o	o	3600	PIII 1GHz
0/1-Matrix-lex/seq [8][22]	Lex/seq	x		x	o	3600	PIII 1GHz
Max-Variety/maxDeg [23]	VM/dyn				o		Ultra60 360MHz

**Table 3.** Evaluation Overview

**Theorem 2.** *Given a fully interchangeable CSP, its double length-lex primal/dual version does not eliminate all solutions in each symmetry class.*

*Proof.* (sketch) Consider a solution  $\gamma$ , transform it to  $\gamma' = pad_{\max(n,m)} \circ \gamma$ . Every sets in the range of  $\gamma'$  are of the same cardinality. The case reduces to the double anti-lex ordering in the 0/1 matrix model.  $\square$

## 5 Experimental Evaluation

This section compares the length-lex set domain with other set-variable representations in four standard benchmarks: The Social Golfer (SG), Error Correcting Code (EC), Steiner System(SS) and Balanced Incomplete Block Design (BI) problems. The goal is to compare the performance (time and failures) between different domain representations (see Table 3). We try our best to compare different static modeling techniques under the same search strategy (with few explicitly stated exceptions). For some techniques (like the ROBDD domain), a variety of approaches and search strategies were proposed and it is impossible to list all results. Hence we choose the best overall approaches as stated by the authors. Correspondingly, in Table 3, symbol  $o$  indicates the compared approaches, while symbol  $x$  denotes the fact that the corresponding papers also evaluate this benchmark but the results is not included in our comparison for space reasons. The column  $tl$  indicates the time limit in seconds (timeout instances are denoted as  $-1$ ) and the column  $cpu$  denotes the processor type and speed used. No existing approach, except ours, was evaluated on all benchmarks. On all benchmarks, the search heuristic of our algorithm uses a simple static ordering. Note that dynamic symmetry-breaking techniques are available for some of these benchmarks but such comparison is out of the scope of this paper.

*Social Golfer Problem* Figure 2 gives the social golfer model in COMET which takes 3 parameters `nbgroup`, `nbsize`, and `nbweek`.  $X[w, i]$  is the primal set vari-

```

var<CP>{set{int}} X[Weeks,Groups](cp,Golfers,nbsize);
var<CP>{set{int}} Y[Golfers](cp,WeekGroups,nbweek);
solve<cp> {
  forall(w in Weeks, i in Groups, j in Groups: i < j)
    cp.post(1ldisjointLeq(X[w,i],X[w,j]));
  forall(w in Weeks, wo in Weeks: w < wo, i in Groups, j in Groups)
    cp.post(1latmostIntersection(X[w,i],X[wo,j],1));
  forall(w in Weeks: w < Weeks.getUp())
    cp.post(1latmostIntersectionLeq(X[w,1],X[w+1,1],1));
  cp.post(1lchanneling(all(w in Weeks, g in Groups) X[w,g],Y));
  forall(g in Golfers: g < Golfers.getUp())
    cp.post(1lleq(Y[g],Y[g+1]));
}

```

**Fig. 2.** Model of Social Golfer Problem in COMET

able representing the  $i$ -th group in  $w$ -th week and  $Y[g]$  is the dual set variable denoting the groups assigned to golfer  $g$ . Like most approaches, the search initially fixes some variables due to the symmetries, i.e., the first week and first group of the second week in the primal model and the first `nbsize` players in the dual model. The search first instantiates the first group of every week, then performs a week-wise labeling (except for 6-5-5, where a simple week-wise labeling is used).

The length-lex representation gives very robust results. It quickly solves all instances that other approaches can solve. Compared to ROBDD, length-lex is significantly faster. It dramatically outperforms (in speed and in the explored nodes) Cardinal and Set-int/dyn, a dual-modeling approach using integer dual variables. Pair-at-most-1 uses the subset-bound representation and some decomposition ideas which were inspired by length-lex. In general, length-lex also dominates this approach significantly in time and in explored nodes, except on 10-3-\* problems for which it is roughly similar when machines are scaled (We run our tests on an energy efficient mobile processor, while Pair-at-most-1 is on non-consumer-level Xeon processor with a significantly faster processor speed).

*Error Correcting Code Problem* The error correcting code problem is a challenging optimization problem which requires to explore the entire search tree to prove the optimality. This benchmark was first proposed in [5] in which only a graph is reported, making it impossible to compare on an instance-by-instance basis. The ROBDD approach was able to prove the optimality of 51 instances (compared to 48 in [5]) with a shorter total time. Hence we compare length-lex with the ROBDD approach. The authors claimed there are 11 difficult instances (see Table 5 (Left)) that not all ROBDD-based solvers are able to solve. One of them was able to solve 4. Length-lex solves 6 of the difficult instances and is significantly faster. For the easy instances, length-lex is more than 30 times faster than the ROBDD approach to solve all instances (Table 5 (Right)).

*Steiner System* The Steiner triple system is a special class of Steiner system (with  $t = 2, k = 3$ ) which has drawn more attention. Hence, the results are given

g,s,w	Length-Lex		Split/seq		Split/dyn		Set-int/dyn		Card/dyn	Pair/dyn	
	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Time	Fails
4,2,4	<b>0.01</b>	<b>0</b>					0.19	266			
4,2,5	<b>0.01</b>	<b>0</b>					0.52	884			
4,2,6	<b>0.01</b>	<b>0</b>					0.51	721			
4,2,7	<b>0.01</b>	<b>0</b>					0.15	97			
4,3,4	<b>0.01</b>	<b>10</b>					0.81	3222			
4,3,5	<b>0.40</b>	<b>732</b>	32.1	5165	26	3812			165.63		
4,3,6	<b>0.02</b>	<b>29</b>	23	2132	15.2	1504			94.67		
4,4,4	<b>0.06</b>	<b>111</b>					8.11	15759			
4,4,5	<b>0.05</b>	<b>57</b>					5.63	7510			
5,2,3	<b>0.01</b>	<b>0</b>					1.07	521			
5,2,4	<b>0.01</b>	<b>0</b>					78.46	95063			
5,2,5	<b>0.01</b>	<b>0</b>					1956.71	2554588			
5,2,6	<b>0.01</b>	<b>0</b>									
5,2,7	<b>0.02</b>	<b>1</b>									
5,2,8	<b>0.02</b>	<b>1</b>									
5,2,9	<b>0.02</b>	<b>1</b>					6841.59	3895064			
5,3,3	<b>0.01</b>	<b>1</b>					140.06	491452			
5,3,4	<b>0.01</b>	<b>1</b>									
5,3,5	<b>0.02</b>	<b>5</b>									
5,3,6	<b>0.41</b>	316	1.5	82	1	<b>34</b>			-1		
5,3,7	74.59	46117			<b>13.1</b>	<b>528</b>			-1		
5,4,2	<b>0.01</b>	11	0.1	<b>0</b>	0.1	<b>0</b>			0.83		
5,4,3	<b>0.02</b>	24	0.3	<b>0</b>	0.3	<b>0</b>	98.23	658755	1.89		
5,4,4	<b>0.14</b>	194	0.6	<b>0</b>	0.6	<b>0</b>	4770.94	30802587	3.13		
5,4,5	1.87	1947	2.3	41	<b>1.7</b>	<b>18</b>			28.65		
5,5,3	<b>0.06</b>	<b>93</b>					0.46	1418			
5,5,4	4.72	<b>6876</b>					<b>2.1</b>	13009			
5,5,5	<b>54.27</b>	<b>50623</b>									
5,5,6	<b>29.21</b>	<b>15769</b>									
5,5,7	<b>0.01</b>	1	0.4	<b>0</b>	0.4	<b>0</b>			-1		
6,3,2	<b>0.00</b>	<b>0</b>					1.16	30			
6,3,3	<b>0.01</b>	<b>1</b>									
6,3,4	<b>0.01</b>	<b>1</b>									
6,3,5	<b>0.02</b>	<b>6</b>									
6,3,6	<b>0.04</b>	10	1.4	<b>0</b>	1.3	7			1.2		
6,4,2	<b>0.01</b>	14	0.1	<b>0</b>	0.1	<b>0</b>	0.99	45	1.75		
6,4,3	<b>0.03</b>	42	1.4	<b>0</b>	0.9	<b>0</b>			4.62		
6,5,2	<b>0.05</b>	118					0.1	<b>60</b>		17.2	171664
6,5,3	<b>2.54</b>	<b>3351</b>								29.6	197607
6,5,4	<b>32.60</b>	31270	80.7	<b>0</b>	40.1	<b>0</b>			-1	39.7	197837
6,5,5	<b>28.76</b>	<b>6758</b>								75.5	239966
6,6,3	<b>0.82</b>	<b>661</b>					414.16	1521747			
7,2,2	<b>0.01</b>	<b>0</b>					0.69	21			
7,3,2	<b>0.01</b>	<b>1</b>					58.94	42			
7,4,2	<b>0.01</b>	<b>0</b>	0.4	<b>0</b>	0.4	<b>0</b>	236.73	63	2.82		
7,4,3	<b>0.03</b>	21	8.4	<b>0</b>	1.7	<b>0</b>			6.37		
7,4,4	<b>0.05</b>	26	481.6		5.1	<b>0</b>			12.46	4.4	27877
7,4,5	<b>0.36</b>	152	-1	-1	12.8	<b>0</b>			17.18		
7,5,2	<b>0.31</b>	574					42.98	<b>84</b>			
7,6,2	0.78	1271					<b>0.53</b>	<b>105</b>			
7,7,2	<b>0.28</b>	<b>0</b>					0.7	308			
8,3,5	34.52	45477	7.8	<b>0</b>	3.9	<b>0</b>			<b>1.01</b>		
8,4,4	<b>0.06</b>	<b>18</b>								157.7	738393
8,5,2	<b>0.25</b>	307	1.6	<b>0</b>	1.6	<b>0</b>			-1		
9,4,4	<b>0.21</b>	94	-1	-1	107.4	<b>0</b>			42.45		
10,3,6	<b>5.86</b>	<b>2941</b>								17.3	57364
10,3,9	233.80	<b>45437</b>								<b>52.4</b>	78613
10,3,10	210.80	<b>25246</b>								<b>67.2</b>	78976
10,4,4	<b>0.27</b>	<b>104</b>								4	22043
10,4,5	<b>0.58</b>	<b>149</b>								4.5	22044

Table 4. Social Golfer Problem

l,d,w	Length-lex		Domain	
	Time	Fails	Time	Fails
8,4,4	<b>0.07</b>	<b>110</b>	1.6	224
9,4,3	<b>2.05</b>	<b>4617</b>	11.3	5615
9,4,6	<b>0.40</b>	<b>908</b>	25.4	16554
10,6,5	<b>0.034</b>	<b>158</b>	26.7	16635
9,4,4	-1	-1	-1	-1
9,4,5	-1	-1	-1	-1
10,4,3	<b>359.3</b>	<b>629822</b>	-1	-1
10,4,4	-1	-1	-1	-1
10,4,5	-1	-1	-1	-1
10,4,6	-1	-1	-1	-1
10,4,7	<b>1.99</b>	<b>4415</b>	-1	-1

	Length-Lex		Domain	
	time	fails	time	fails
Mean	<b>0.0060</b>	<b>5.92</b>	0.2	210.7
Total	<b>0.31</b>		11.4	
min	<b>0.0038</b>	1	0.03	<b>0</b>
25 percentile	<b>0.0042</b>	1	0.03	<b>0</b>
median	<b>0.0046</b>	<b>1</b>	0.04	2
75 percentile	<b>0.0053</b>	<b>1</b>	0.06	25
max	<b>0.47</b>	<b>134</b>	4.16	3740

**Table 5.** Error Correcting Code : 11 Difficult Cases(Left), 51 Easy Cases(Right)

n	Length-Lex		Bound/seq		Domain/seq		Lex-sum/seq		Card/seq	Valprec/dyn	
	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Time	Fail
7	0.002	<b>0</b>	< <b>0.1</b>	8	< <b>0.1</b>	8	<b>0</b>	1	0.01	<b>0</b>	12
9	<b>0.009</b>	<b>1</b>	0.2	325	0.1	9	0.1	250	0.05	0.03	153
13	<b>0.05</b>	<b>10</b>	-1	-1	109.2	24723			0.61	1738.24	3935567
15	<b>0.089</b>	<b>0</b>	0.4	56	1.3	<b>0</b>			0.91		
19	<b>0.459</b>	<b>164</b>							7.94		
21	<b>1.04</b>	<b>448</b>							39.07		
25	<b>14.074</b>	<b>5100</b>									
27	<b>23.548</b>	<b>7066</b>									
31	<b>5.289</b>	<b>0</b>	23.3	280	-1	-1			48.52		

**Table 6.** Experimental Results on the Steiner Triple System.

in two tables. Table 6 depicts the results for the Steiner triple system, while Table 7 shows the results for the remaining instances. For the triple system, the search adopted the labeling technique of [4](and [22]), which assigns the smallest available value to the first possible domain (column-wise labeling). Under our set dual modeling approach, it is equivalent to label the dual variables sequentially. Length-lex solves all instances and outperforms other representations significantly. For the remaining instances, the search uses the standard sequential labeling. Once again, length-lex is the most robust representation and is able to solve all instances efficiently.

*Balanced Incomplete Block Design Problem* For this problem, we give a comprehensive list containing instances from [23] and [8] (some small and trivial instances were removed due to space constraints). VM/dyn is a randomized approach with a maximum node limit 10000. Every instance was run 50 times and not all instances finishes within the limit and only the average time and nodes of the successful instances was reported in [23]. Lex/seq gives only a logarithmic scale for most instances and we can only report the time and number of fails as a range, [22] gives the number of fails for some smaller instances as well. Note that this is the first result of a set representation on the BIBD problem and the length-lex representation is robust and effective.

t,k,n	Length-Lex		Bound/seq		Domain/seq		Hybrid/seq		Subset/seq	
	Time	Fails	Time	Fails	Time	Fails	Time	Fails	Time	Fails
2,4,13	<b>0.008</b>	<b>0</b>	0.1	157	0.1	<b>0</b>	0.14	<b>0</b>	0.02	1
2,4,16	<b>0.14</b>	98	421.4	522706	0.6	<b>15</b>				
2,5,21	<b>0.04</b>	<b>0</b>	0.5	413	1.4	<b>0</b>	2.83	<b>0</b>	0.1	<b>0</b>
2,6,16	<b>0.005</b>	<b>1</b>	-1	-1	80.7	15205				
3,4,8	<b>0.006</b>	<b>0</b>	0.1	18	0.1	<b>0</b>	0.08	2	0.03	8
3,4,16	<b>3.43</b>	<b>0</b>	9.7	274	548.7	<b>0</b>	54.69	132	7.11	240
3,6,22	2.84	<b>0</b>	8.3	1608	-1	-1	54.98	42	<b>2.47</b>	92

**Table 7.** Experimental Results on the Steiner System.

## 6 Conclusion

This paper presented the first experimental evaluation of the length-lex domain for set variables. The implementation was based on two novel technical contributions: a generic propagation algorithm which pushes the length-lex ordering constraint into any symmetric binary constraints and an adaptation of the symmetry-breaking technique from 0/1 matrices to the length-lex ordering. The resulting implementation, which consists of 18,000 lines of C++, demonstrates that the length-lex representation for set variables is robust and efficient across the standard benchmarks.

## References

1. Puget, J.F. Pecos a high level constraint programming language. In Proc. of Spicis., 1992.
2. Gervet, C. Interval propagation to reason about sets: Definition and implementation of a practical language. *Constraints* **1**(3), 1997, 191–244
3. Azevedo, F., Barahona, P. Modelling digital circuits problems with set constraints. In CP-2000, 414–428.
4. Azevedo, F. Cardinal: A finite sets constraint solver. *Constraints* **12**(1), 93–129, 2007.
5. Sadler, A., Gervet, C. Hybrid set domains to strengthen constraint propagation and reduce symmetries. In CP-2004, 604–618.
6. Sadler, A., Gervet, C. Enhancing set constraint solvers with lexicographic bounds. *J. Heuristics* **14**(1), 23–67, 2008.
7. Flener, P., Frisch, A., Hnich, B., Kiziltan, Z., Miguel, I., Pearson, J., Walsh, T. Breaking row and column symmetries in matrix models. In CP-2002, 462–476.
8. Frisch, A.M., Hnich, B., Kiziltan, Z., Miguel, I., Walsh, T. Propagation algorithms for lexicographic ordering constraints. *Artificial Intelligence* (170), 2006.
9. Peter Hawkins, V.L., Stuckey, P.J. Solving set constraint satisfaction problems using robdds. *JAIR* **24**, 109–156, 2005.
10. Gervet, C., Van Hentenryck, P. Length-lex ordering for set csps. In AAAI-2006.
11. Hnich, B., Kiziltan, Z., Walsh, T. Combining symmetry breaking with other constraints: Lexicographic ordering with sums. In AMAI-2004.
12. Cheng, B.M.W., Choi, K.M.F., Lee, J.H.M., Wu, J.C.K. Increasing constraint propagation by redundant modeling: an experience report. *Constraints* **4**(2), 167–192, 1999.

v,b,r,k,l	Length-Lex		VM/dyn		Lex/seq		Lex-sum/seq	
	Time	Fails	Time	Nodes	Time	Fails	Time	Fails
6,20,10,3,4	0.009	<b>1</b>	0.033	61	<b>0</b>	43	<b>0</b>	43
6,30,15,3,6	<b>0.018</b>	<b>1</b>	0.14	95	0.1	68	0.1	68
6,40,20,3,8	<b>0.041</b>	<b>3</b>	0.39	128	0.1	108	0.1	108
6,80,40,3,16	<b>0.82</b>	<b>32</b>	3.6	245	<b>0.1-1</b>	100-1000		
7,21,9,3,3	0.008	<b>0</b>	0.045	75	<b>0</b>	42	<b>0</b>	42
7,28,12,3,4	<b>0.010</b>	<b>0</b>	0.12	86	0.1	64	0.1	64
7,35,15,3,5	<b>0.012</b>	<b>0</b>	0.27	109	0.1	88	0.1	88
7,42,18,3,6	<b>0.015</b>	<b>0</b>	0.48	139	0.2	115	0.2	115
7,84,36,3,12	<b>0.040</b>	<b>0</b>	4.2	254	0.1-1	100-1000		
7,91,39,3,13	<b>0.047</b>	<b>0</b>	5.4	280	0.1-1	100-1000		
9,24,8,3,2	<b>0.012</b>	<b>1</b>			0.1	48	0.1	48
9,72,24,3,6	<b>0.17</b>	<b>27</b>	2.7	252	<b>0.1-1</b>	100-1000		
9,84,28,3,7	<b>0.30</b>	<b>43</b>	4.2	257	1 - 10	1000-10000		
9,96,32,3,8	<b>0.50</b>	<b>66</b>	6.3	296	1 - 10	1000-10000		
9,108,36,3,9	<b>0.80</b>	<b>97</b>	14	365	1 - 10	1000-10000		
9,120,40,3,10	<b>1.27</b>	<b>138</b>	14	268	<b>1 - 10</b>	1000-10000		
10,90,27,3,6	<b>1.05</b>	<b>150</b>	5.3	289	<b>1 - 10</b>	<b>100-1000</b>		
10,120,36,3,8	<b>4.77</b>	<b>576</b>	13	377	<b>1 - 10</b>	1000-10000		
11,110,30,3,6	<b>7.66</b>	1192	16	366	<b>1 - 10</b>	<b>100-1000</b>		
12,88,22,3,4	<b>5.65</b>	1173	5.1	296	<b>1 - 10</b>	<b>100-1000</b>		
13,52,12,3,2	<b>0.11</b>	<b>15</b>	2.9	218	<b>0.1-1</b>	100-1000		
13,78,18,3,3	<b>0.47</b>	<b>78</b>	3.5	282	<b>0.1-1</b>	100-1000		
13,104,24,3,4	<b>1.52</b>	<b>207</b>	8.7	344	<b>1 - 10</b>	<b>100-1000</b>		
15,21,7,5,2	<b>24.78</b>	<b>16891</b>			<b>10 - 100</b>	$10^5 - 10^6$		
15,70,14,3,2	<b>0.11</b>	<b>0</b>	5.5	383	<b>0.1-1</b>	100-1000		
16,32,12,6,4	<b>3.84</b>	<b>980</b>			10 - 100	$10^6 - 10^7$		
16,80,15,3,2	<b>0.68</b>	<b>148</b>	4.7	485	1 - 10	<b>100-1000</b>		
19,57,9,3,1	<b>0.13</b>	<b>6</b>	8.2	802	1 - 10	100-1000		
22,22,7,7,2	<b>74.26</b>	<b>21552</b>			<b>10 - 100</b>	$10^5 - 10^6$		

**Table 8.** Experimental Results on Balanced Incomplete Block Design

13. Geelen, P.A. Dual viewpoint heuristics for binary constraint satisfaction problems. In ECAI-1992, 31–35.
14. Law, Y.C., Lee, J.H.M. Symmetry breaking constraints for value symmetries in constraint satisfaction. Constraints **11**, 2006.
15. Puget, J.F. An efficient way of breaking value symmetries. In AAI-2006.
16. Walsh, T. General symmetry breaking constraints. In CP-2006. 650–664.
17. Van Hentenryck, P., Yip, J., Gervet, C., Doooms, G. Bound consistency for binary length-lex set constraints. In AAI-2008, 375–380.
18. Yip, J., Van Hentenryck, P. Length-lex bound consistency for knapsack constraints. In SAC-2009.
19. Malitsky, Y., Sellmann, M., van Hove, W.J. Length-lex bounds consistency for knapsack constraints. In CP-2008, 266–281.
20. Katsirelos, G., Narodytska, N., Walsh, T. Combining symmetry breaking and global constraints (2009) CSCLP-2008.
21. van Hove, W.J., Sabharwal, A. Filtering atleast1 on pairs of set variables. In CPAIOR-2008, 382–386.
22. Kiziltan, Z. Symmetry breaking ordering constraints. Phd Thesis. Uppsala University.
23. Meseguer, P., Torras, C. Exploiting symmetries within constraint satisfaction search. Artificial Intelligence **129**(1-2)133–163, 2001.