Set Domain Propagation Using ROBDDs

Vitaly Lagoon and Peter J. Stuckey

Department of Computer Science and Software Engineering
The University of Melbourne, Vic. 3010, Australia
{lagoon,pjs}@cs.mu.oz.au

Abstract. Propagation based solvers typically represent variables by a current *domain* of possible values that may be part of a solution. Finite set variables have been considered impractical to represent as a domain of possible values since, for example, a set variable ranging over subsets of $\{1, \ldots, N\}$ has 2^N possible values. Hence finite set variables are presently represented by a lower bound set and upper bound set, illustrating all values definitely in (and by negation) all values definitely out. Propagators for finite set variables implement *set bounds* propagation where these sets are further constrained. In this paper we show that it is possible to represent the domains of finite set variables using reduced ordered binary decision diagrams (ROBDDs) and furthermore we can build efficient *domain* propagators for set constraints using ROBDDs. We show that set domain propagation is not only feasible, but can solve some problems significantly faster than using set bounds propagation because of the stronger propagation.

1 Introduction

Many constraint satisfaction problems are naturally expressed using set variables and set constraints, where the set variables range over subsets of a given a finite set of integers.

One widely-adopted approach to solving CSPs combines backtracking tree search with constraint propagation. *Constraint propagation*, based on local consistency algorithms, removes infeasible values from the domains of variables to reduce the search space. One of the most successful consistency techniques is *domain consistency*[1] [13] which ensures that for each constraint c, every value in the domain of each variable can be extended to an assignment satisfying c. For simplicity, we refer to constraint propagation based on domain consistency as *domain propagation*. *Propagators* are functions to perform constraint propagation.

Unfortunately domain propagation where the variables range over sets is usually considered infeasible. For example, if a variable must take as a value a subset of $\{1, \ldots, N\}$ there are 2^N possible values. Simply representing the domain of possible values seems to be impractical. For this reason constraint

[1] Equivalently, hyper-arc [9] or generalized arc consistency [2].

propagation over set variables is always, as far as we are aware, restricted to use *set bounds propagation* [6].

In set bounds propagation, a set variable v has its domain represented by two sets L the least possible set (that is all the values that must be in the set v) and U the greatest possible set (that is the complement of all values that cannot be in set v). Set bounds propagators update the least and greatest possible sets of variables v according to the constraints in which they are involved.

Interestingly set bounds propagation as defined in [6] is equivalent to modelling the problem using Boolean constraints and applying domain propagation to these constraints. That is the set variable v is represented by N Boolean variables $v_1, \ldots, v_N$ which represent the propositions $v_i \Leftrightarrow i \in v$. Refined versions of set bounds propagation, that add a representation of the cardinality of a variable to its domain, such as [1, 10], give stronger propagation.

In this paper we investigate this Boolean modelling approach further. We show it is possible to represent the domains of finite set variables using reduced ordered binary decision diagrams (ROBDDs). We shall show that domains can be represented by very small ROBDDs, even if they represent a large number of possible sets. Furthermore we represent set constraints using ROBDDs, and use this to build efficient *domain* propagators for set constraints. ROBDD representations of simple set constraints are surprisingly compact.

In summary the contributions of this paper are:

- We show how to efficiently representing domains of (finite) set variables using ROBDDs.
- We show that we can build efficient constraint propagators for primitive set constraints using ROBDDs
- We illustrate the wealth of modelling possibilities that arise using ROBDDs to build propagators.
- We give experiments illustrating that set domain propagation is advantageous for (finite) set constraint problems.

2 Preliminaries

In this paper we consider finite set constraint solving (over sets of integers) with constraint propagation and tree search.

Let $\mathcal{V}$ denote the set of all set variables. Each variable is associated with a finite set of possible values (which are themselves sets), defined by the domain of the CSP. A *domain* D is a complete mapping from a fixed (countable) set of variables $\mathcal{V}$ to finite sets of finite sets of integers. The *intersection* of two domains D_1 and D_2, denoted $D_1 \sqcap D_2$, is defined by the domain $D_3(v) = D_1(v) \cap D_2(v)$ for all v. A domain D_1 is *stronger* than a domain D_2, written $D_1 \sqsubseteq D_2$, if $D_1(v) \subseteq D_2(v)$ for all variables v. A domain D_1 is equal to a domain D_2, denoted $D_1 = D_2$, if $D_1(v) = D_2(v)$ for all variables v.

We also use *range* notation whenever possible: $[L \mathrel{..} U]$ denotes the set of sets of integers $\{d \mid L \subseteq d \subseteq U\}$ when L and U are sets of integers.

We shall be interested in the notion of an *initial domain*, which we denote D_{init}. The initial domain gives the initial values possible for each variable. In effect an initial domain allows us to restrict attention to domains D such that $D \sqsubseteq D_{init}$. For our purposes we shall restrict attention to initial domains D_{init} where $D_{init}(x) = [\emptyset \mathrel{..} \{1, \ldots, N\}]$ for all variables x.

A *valuation* θ is a mapping variables to sets of integer values, written $\{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\}$ where $d_i \in D(x_i)$. We extend the valuation θ to map expressions or constraints involving the variables in the natural way.

Let *vars* be the function that returns the set of variables appearing in an expression, constraint or valuation. In an abuse of notation, we define a valuation θ to be an element of a domain D, written $\theta \in D$, if $\theta(v_i) \in D(v_i)$ for all $v_i \in vars(\theta)$.

Constraints A constraint places restriction on the allowable values for a set of variables and is usually written in well understood mathematical syntax. For our purposes we shall restrict ourselves to the following *primitive set constraints*: (element) $k \in v$, (not element) $k \notin v$, (equality) $v = w$, (constant) $v = d$, (subset) $v \subseteq w$, (union) $u = v \cup w$, (intersection) $u = v \cap w$, (subtraction) $u = v - w$, (complement) $v = \overline{w}$, (disequality) $v \neq w$, (cardinality equals) $|v| = k$, (cardinality lower) $|v| \geq k$, and (cardinality upper) $|v| \leq k$, where u, v, w are set variables, d is a constant finite set and k is a constant integer. A *set constraint* is a (possibly existentially quantified) conjunction of primitive set constraints. Later we shall introduce more complex forms of set constraint.

We define $solns(c) = \{\theta \mid vars(\theta) = vars(c) \wedge \models_\theta c\}$, that is the set of θ that make the constraint c hold true. We call $solns(c)$ the *solutions* of c.

Propagators and Propagation Solvers In the context of propagation-based constraint solving, a constraint specifies a propagator, which gives the basic unit of propagation. A *propagator* f is a monotonically decreasing function from domains to domains, i.e. $D_1 \sqsubseteq D_2$ implies that $f(D_1) \sqsubseteq f(D_2)$, and $f(D) \sqsubseteq D$. A propagator f is *correct* for constraint c iff for all domains D

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

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

Example 1. For the constraint $c \equiv v \subseteq w$ the function $f(D)(v) = \{d_1 \in D(v) \mid d_1 \subseteq \bigcup D(w)\}$, $f(D)(w) = D(w)$, is a correct propagator for c. Let $D_1(v) = \{\{1\}, \{1, 3, 5\}, \{6\}, \{3, 4\}\}$ and $D_1(w) = \{\{1\}, \{3, 4, 5\}, \{2\}\}$, then $f(D_1) = D_2$ where $D_2(v) = \{\{1\}, \{1, 3, 5\}, \{3, 4\}\}$ and $D_2(w) = D_1(w)$.

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

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

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

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

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

Define the *domain propagator* for a constraint c as

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

Example 2. For the constraint $c \equiv v \subseteq w$ and let $D_3 = dom(c)(D_1)$ where $D_1(v) = \{\{1\}, \{1,3,5\}, \{6\}, \{3,4\}\}$ and $D_1(w) = \{\{1\}, \{3,4,5\}, \{2\}\}$, then $D_3(v) = \{\{1\}, \{3,4\}\}$ and $D_3(w) = \{\{1\}, \{3,4,5\}\}$. Now D_3 is domain consistent with respect to c.

Set Bounds Consistency Domain consistency has been considered prohibitive to compute for constraints involving set variables. For that reason, set bounds propagation [6, 10] is typically used where a domain maps a set variable to a lower bound set of integers and an upper bound set of integers.

We shall enforce this by treating the domain $D(x)$ of a set variable x as if it were the range $[\cap D(x) \mathinner{..} \cup D(x)]$. In practice if only set bounds propagators are used, all domains will be of this form in any case. Define $ran(D)$ as the domain D' where $D'(x) = [\cap D(x) \mathinner{..} \cup D(x)]$. The set bounds propagator returns the smallest set range which includes the result returned by the domain propagator.

Define the *set bounds propagator* for a constraint c as

$$sb(c)(D)(v) = \begin{array}{l} D(v) \cap \\ [\cap dom(c)(ran(D))(v) \mathinner{..} \cup dom(c)(ran(D))(v)] \end{array} \; v \in vars(c)$$
$$sb(c)(D)(v) = D(v) \text{ otherwise}$$

Example 3. For the constraint $c \equiv v \subseteq w$ and let $D_4 = sb(c)(D_1)$, where D_1 is defined in Example 2. Now $ran(D_1)(v) = [\emptyset \mathinner{..} \{1,3,4,5,6\}]$ and $ran(D_1)(w) = [\emptyset \mathinner{..} \{1,2,3,4,5\}]$, and hence $D_4(v) = \{\{1\}, \{1,3,5\}, \{3,4\}\}$ and $D_4(w) = \{\{1\}, \{3,4,5\}, \{2\}\}$.

Boolean Formulae and Binary Decision Diagrams We make use of the following Boolean operations: $\wedge$ (conjunction), $\vee$ (disjunction), $\neg$ (negation), $\rightarrow$ (implication), $\leftrightarrow$ (bi-implication) and $\exists$ (existential quantification). We denote by $\exists_V F$ the formula $\exists x_1 \cdots \exists x_n F$ where $V = \{x_1, \ldots, x_n\}$.

The *Reduced Ordered Binary Decision Diagram* (ROBDD) [3] is a canonical representation of Boolean function. An ROBDD is a directed acyclic graph. There are two terminal nodes are labeled with the Boolean constants 0 (false) and 1 (true). Each non-terminal node is labelled by a variable and has two branches, a 1-branch and a 0-branch, corresponding respectively to the two possible values of the variable. Thus, the value of a Boolean function for a given assignment of variables can be computed as a terminal reached from the root by following the corresponding 1- or 0-branches. An ROBDD obeys the following restrictions:

- Variables are totally ordered by some ordering $\prec$ and each child node (with label w) of a node labelled v is such that $v \prec w$.
- There are no duplicate subgraphs (nodes with the same variable label and 1- and 0-branches).
- There are no redundant tests (each non-terminal node has different 1- and 0-branches).

Example 4. The ROBDD for the formula $(v_1 \rightarrow w_1) \wedge (v_2 \rightarrow w_2)$ with the ordering of variables $v_1 \prec v_2 \prec w_1 \prec w_2$ is shown in Figure 2(a). The ROBDD is read as follows. Each non-leaf node has a decision variable. If the variable is false (0) we take the left 0-branch (dashed), if the variable is true (1) we take the right 1-branch (solid). One can verify that for example a valuation that make the formula true, e.g. $\{v_1 \mapsto 1, v_2 \mapsto 0, w_1 \mapsto 1, w_2 \mapsto 1\}$, give a path to the node 1, while valuations that make the formula false, e.g. $\{v_1 \mapsto 0, v_2 \mapsto 1, w_1 \mapsto 1, w_2 \mapsto 0\}$, give a path to the node 0.

The ROBDD-based representations of Boolean functions are *canonical* meaning that for a fixed order of variables two Boolean functions are equivalent if and only if their respective ROBDDs are identical.

There are efficient algorithms for many Boolean operations applied to ROBDDs. Let $|r|$ be the number of nodes in ROBDD r then the complexity of the basic operations for constructing new ROBDDs is $O(|r_1||r_2|)$ for $r_1 \wedge r_2$, $r_1 \vee r_2$ and $r_1 \leftrightarrow r_2$, $O(|r|)$ for $\neg r$, and $O(|r|^2)$ for $\exists x.r$. Note however that we can test whether two ROBDDs are identical, whether $r_1 \leftrightarrow r_2$ is equivalent to true (1), in $O(1)$.

Although in theory the number of nodes in an ROBDDs can be exponential in the number of variables in the represented Boolean function, in practice ROBDDs are often very compact and computationally efficient. This is due to the fact that ROBDDs exploit to a very high-degree the symmetry of models of a Boolean formula.

3 Basic Set Domain Propagation

3.1 Modelling Set Domains using ROBDDs

The key step in building set domain propagation using ROBDDs is to realise that we can represent a finite set domain using an ROBDD.

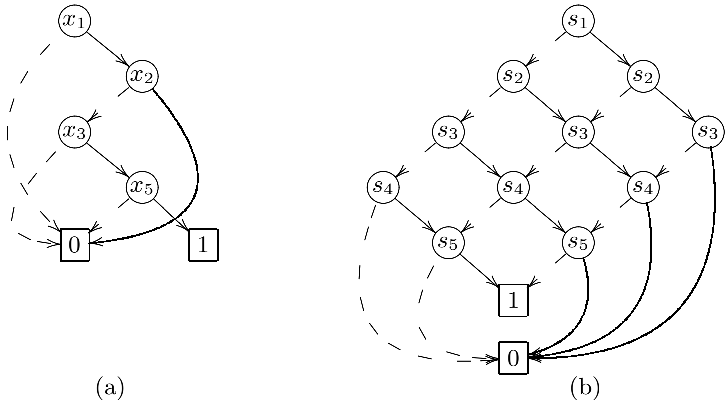


Fig. 1. ROBDDs for (a) $[\{1,3,5\} \mathrel{..} \{1,3,\ldots N\}]$ and (b) $(s \subseteq \{1,2,3,4,5\}\ ,\ |s| = 2)$.

Let x be subset of $\{1,\ldots,N\}$, then we can represent all values of x using a vector of Boolean variables $V(x) = \langle x_1,\ldots,x_N\rangle$, where $x = \{i \mid x_i = 1\}$. We will order the variables $x_1 \prec x_2 \cdots \prec x_N$. An ROBDD allows us to represent (some) subsets of $2^{\{1,\ldots,N\}}$ efficiently.

The initial domain for x, $D_{init}(x) = [\emptyset \mathrel{..} \{1,\ldots,N\}]$, is represented by the $O(1)$ sized ROBDD 1, independent of N.

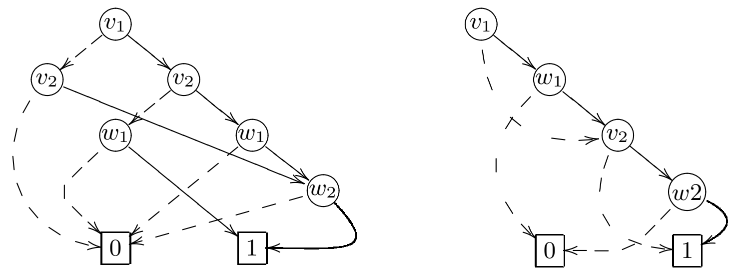
If $D(x) = [\{1,3,5\} \mathrel{..} \{1,3,\ldots,N\}]$, we can represent these 2^{N-4} sets by the ROBDD shown in Figure 1(a), again independent of N. Indeed any domain $D'(x)$ of the form $[L \mathrel{..} U]$ can be represented in $O(|L| + N - |U|)$.

Even some sets of subsets that may appear difficult to represent are also representable efficiently. Consider representing all the subsets of $\{1,2,3,4,5\}$ of cardinality 2. Then the ROBDD representing this domain is shown in Figure 1(b).

3.2 Modelling Primitive Set Constraints using ROBDDs

We will convert each primitive set constraint c to an ROBDD $B(c)$ on the Boolean variable representations $V(x)$ of its set variables x. We must be careful in ordering the variables in each ROBDD in order to build small representations of the formula.

Example 5. Consider the two representations of the ROBDD for the constraint representing $v \subseteq w$ where $D_{init}(v) = D_{init}(w) = [\emptyset \mathrel{..} \{1,2\}]$ shown in Figure 2. The first representation with the ordering $v_1 \prec v_2 \prec w_1 \prec w_2$ is much larger than the second with ordering $v_1 \prec w_1 \prec v_2 \prec w_2$. In fact if we generalize for initial domains of size N the ordering $v_1 \prec \cdots \prec v_N \prec w_1 \prec \cdots \prec w_N$ leads to an exponential representation, while the order $v_1 \prec w_1 \prec v_2 \prec w_2 \prec \cdots \prec v_N \prec w_N$ is linear.



(a) Order: $v_1 \prec v_2 \prec w_1 \prec w_2$ (b) Order: $v_1 \prec w_1 \prec v_2 \prec w_2$

Fig. 2. Two ROBDDs for $v \subseteq w$

By ordering the Boolean variables pointwise we can guarantee linear representations $B(c)$ for each primitive constraint c except those for cardinality. The table of expressions and sizes is shown in Figure 3. The cardinality primitive constraints use a subsidiary Boolean formula defined by the function $card(v, l, u, n)$ which is the cardinality of v restricted to the elements from $n + 1, \ldots, N$ is between l and u. It is defined recursively as follows:

$$card(v, l, u, n) = \begin{cases} 1 & \text{if } l \leq 0 \wedge N - n \leq u \\ 0 & \text{if } N - n < l \vee u < 0 \\ (\neg v_{n+1} \wedge card(v, l, u, n + 1)) \;\; \vee & \text{otherwise} \\ (v_{n+1} \wedge card(v, l - 1, u - 1, n + 1) & \end{cases}$$

This definition also shows why the representation is quadratic since the right hand sides directly represent ROBDD nodes. For any formula $card(v, l, u, n)$ it involves at most a quadratic number of subformulæ.

3.3 Basic Set-Domain Solver

We build a domain propagator $dom(c)$ for constraint c using ROBDDs as follows. Let $vars(c) = \{v_1, \ldots, v_n\}$ then we define

$$dom(c)(D)(v_i) = \overline{\exists}_{V(v_i)}.B(c) \wedge \bigwedge_{j=1}^{n} D(v_j)$$

This formula directly defines the domain consistency operator. Because each of $B(c)$ and $D(v_j)$ is an ROBDD this is directly implementable using ROBDD operations.

The basic propagation mechanism improves slightly on the mathematical definition by (a) only revisiting propagators when the domain of one of their variables changes (b) ensuring that the last propagator is not re-added by its own changes since it is idempotent. We can also execute unary constraint propagators just once. The code for solv is shown below where for simplicity the first argument is a set of constraints, rather than their domain propagators.

c	Boolean expression $B(c)$	size of ROBDD
$k \in v$	v_k	$O(1)$
$k \notin v$	$\neg v_k$	$O(1)$
$v = w$	$\bigwedge_{1 \leq i \leq N} (v_i \leftrightarrow w_i)$	$O(N)$
$v = d$	$\bigwedge_{i \in d} v_i \wedge \bigwedge_{1 \leq i \leq N, i \notin d} \neg v_i$	$O(N)$
$v \subseteq w$	$\bigwedge_{1 \leq i \leq N} (v_i \rightarrow w_i)$	$O(N)$
$u = v \cup w$	$\bigwedge_{1 \leq i \leq N} (u_i \leftrightarrow (v_i \vee w_i))$	$O(N)$
$u = v \cap w$	$\bigwedge_{1 \leq i \leq N} (u_i \leftrightarrow (v_i \wedge w_i))$	$O(N)$
$u = v - w$	$\bigwedge_{1 \leq i \leq N} (u_i \leftrightarrow (v_i \wedge \neg w_i))$	$O(N)$
$v = \overline{w}$	$\bigwedge_{1 \leq i \leq N} \neg (v_i \leftrightarrow w_i)$	$O(N)$
$v \neq w$	$\bigvee_{1 \leq i \leq N} \neg (v_i \leftrightarrow w_i)$	$O(N)$
$\|v\| = k$	$card(v, k, k, 0))$	$O(k(N-k))$
$\|v\| \geq k$	$card(v, k, N, 0)$	$O(k(N-k))$
$\|v\| \leq k$	$card(v, 0, k, 0)$	$O(k(N-k))$

Fig. 3. Boolean representation of set constraints and size of ROBDD.

$\mathsf{solv}(C, D)$
    $Q := C$
    while $(\exists c \in Q)$
        $Q := Q - \{c\}$
        $D' := dom(c)(D)$
        $V := \{v \in \mathcal{V} \mid D(v) \neq D'(v)\}$
        $Q := (Q \cup \{c' \in C \mid vars(c') \cap V \neq \emptyset\}) - \{c\}$
        $D := D'$
    endwhile
    return D

Note that detecting the set V of variables changed by the $dom(c)$ is cheap because the equivalence check $D(v) = D'(v)$ for ROBDDs is constant time (the two pointers must be identical).

4 Towards a More Efficient Set Domain Solver

4.1 Removing Intermediate Variables

For many real problems the actual problem constraints will not be primitive set constraints, but rather more complicated set constraints that can be split into primitive set constraints. This means that we introduce new variables and multiple propagators for a single problem constraint.

Example 6. Consider the following constraint $|v \cap w| \leq 1$ which is found in many problems, requiring that v and w have at most one element in common. This can be represented as primitive set constraints by introducing an intermediate variable u as $\exists u. u = v \cap w \wedge |u| \leq 1$. For set bounds propagation this breaking down of the constraint is implicitly performed.

Now the splitting into primitive constraints does not affect the propagation strength of the resulting set of propagators. But it does add a new variable, and complicate the propagation process by making more constraints.

Using an ROBDD representation of the more complex constraint we can avoid splitting the constraint. Instead we construct a Boolean representation directly using ROBDD operations. Essentially we can take the ROBDDs for the individual primitive constraints and project away the intermediate variable.

Example 7. We can build a propagator $dom(c)$ directly for the constraint $c \equiv |v \cap w| \leq 1$, by constructing the ROBDD $\exists_{V(u)}.B(u = v \cap w) \wedge B(|u| \leq 1)$. The resulting ROBDD will have size at worst $O(N^2)$ since the individual parts are linear sized, and the conjunction at most multiplies the sizes and (in this case) existential quantification strictly reduces the size of the ROBDD.

This contrasts with set bounds propagation where we must carefully construct a new propagator for each kind of primitive constraint we wish to support. Typically only the given primitive constraints are provided and intermediate variables are silently introduced. This approach is used for instance in set constraint solvers provided by ECL^{i}PS^e [5].

4.2 Building Global Constraints

Just as we can create more complex ROBDDs for constraints that are typically broken up into primitive sets constraints, we can also build domain propagators for complex conjunctions of constraints using ROBDD operations, so called *global constraints*.

Example 8. Consider the following constraint $\texttt{partition}(v_1, \ldots, v_n)$ which requires that the sets $v_1, \ldots, v_n$ form a partition of $\{1, \ldots, N\}$. The meaning in terms of primitive set constraints is

$$\begin{array}{c} \bigwedge_{i=1}^n \bigwedge_{j=i+1}^n (\exists u_{ij}. u_{ij} = v_i \cap v_j \wedge u_{ij} = \emptyset) \quad \wedge \\ \exists w_1 \ldots w_{n+1}. w_{n+1} = \emptyset \wedge \bigwedge_{i=1}^n (w_i = v_i \cup w_{i+1}) \wedge w_1 = \{1, \ldots, N\} \end{array}$$

Now $dom(\texttt{partition}(v_1, \ldots, v_n))$ is much stronger in propagation than the set of propagators for the primitive constraints in its definition. For example consider $\texttt{partition}(v_1, \ldots, v_3)$ and domain D where $D(v_1) = D(v_2) = \{\{1\}, \{2\}\}$ and $D(v_3) = \{\{2\}, \{3\}\}$ and $N = 3$. While the individual propagators will not modify the domain, the global propagator will revise the domain of v_3 to $\{\{3\}\}$.

We can build a single ROBDD representing this formula, and hence we have built a domain constructor for the entire constraint.

$$\begin{array}{c} \bigwedge_{i=1}^n \bigwedge_{j=i+1}^n (\exists_{V(u_{ij})}.B(u_{ij} = v_i \cap v_j) \wedge B(u_{ij} = \emptyset)) \quad \wedge \\ \exists_{V(w_1) \ldots V(w_{n+1})}.B(w_{n+1} = \emptyset) \wedge \bigwedge_{i=1}^n B(w_i = v_i \cup w_{i+1}) \wedge B(w_1 = \{1, \ldots, N\}) \end{array}$$

The size of this entire ROBDD is $O(nN)$.

Clearly as we build larger and larger global constraint propagators we will build larger and larger ROBDD representations. There is a tradeoff in terms of strength of propagation versus time to propagate using this ROBDD.

Example 9. Consider the constraint $\texttt{atmost}([v_1, \ldots, v_n], k)$ introduced in [11], which requires that each of the n sets of cardinality k should intersect pairwise in at most one element. The meaning in terms of primitive set constraints is

$$\bigwedge_{i=1}^{n} |v_i| = k \quad \wedge \quad \bigwedge_{i=1}^{n} \bigwedge_{j=i+1}^{n} (\exists u_{ij}.u_{ij} = v_i \cap v_j \wedge |u_{ij}| \leq 1)$$

While we can create an ROBDD that represents the entire global constraint the resulting ROBDD is exponential in size. Hence it is not practical to use as a global propagator.

4.3 Breaking Symmetries with Order Constraints

In many problems involving set constraints there will be large numbers of symmetries which can substantially increase the size of the search space. One approach to removing symmetries is to add order constraints $v < w$ among sets. $v < w$ means that the list of Booleans $V(v)$ is lexicographically smaller than the list of Booleans $V(w)$. These order constraints can be straightforwardly represented using ROBDDs of size $O(N)$. We represent $v < w$ by the formula $\mathit{ltset}(v, w, 1)$ where

$$\mathit{ltset}(v, w, n) = \begin{cases} 0 & n > N \\ (\neg v_n \wedge w_n) \vee ((v_n \leftrightarrow w_n) \wedge \mathit{ltset}(v, w, n+1)) & \text{otherwise} \end{cases}$$

5 Experiments

We have implemented a set-domain solver based on the method proposed in this paper. Our implementation written in Mercury [12] integrates a fairly naïve propagation-based mechanism with an ROBDD library. We have chosen the CUDD package [4] as a platform for ROBDD manipulations.

We conducted a series of experiments using our solver and compared the experimental results with the data collected by running the ECL^{i}PS^e [5] `ic_sets` set bounds propagation solver on the same problems. All experiments described below are conducted on a machine with Pentium-M 1.5 GHz processor running Linux v2.4.

5.1 Ternary Steiner Systems

A ternary Steiner system of order n is a set of $n(n-1)/6$ triplets of distinct elements taking their values between 1 and n, such that all the pairs included in two different triplets are different. A Steiner triple system of order n exists if and only if $n \equiv 1, 3 \mod 6$, as shown by Kirkman [7]. A ternary Steiner system of

Table 1. ROBDD set domain solver for finding Steiner systems. The simple model with separate constraints and intermediate variables.

Problem	Model size $\times 10^3$	"element in set" labeling						"element not in set" labeling					
		sequential			first-fail			sequential			first-fail		
		time sec.	fails	size $\times 10^3$	time sec.	fails	size $\times 10^3$	time sec.	fails	size $\times 10^3$	time sec.	fails	size $\times 10^3$
steiner(7)	3	0.1	1	18	0.0	1	18	0.0	0	16	0.0	0	16
steiner(9)	11	4.6	365	324	4.7	370	325	1.3	100	180	1.4	100	180
steiner(13)	47	—	—	—	—	—	—	—	—	—	—	—	—
steiner(15)	140	—	—	—	—	—	—	23.9	0	1,058	24.1	0	1,058

order n is naturally modelled using set variables $s_1 \dots s_m$, where $m = n(n-1)/6$ and constrained by:

$$\bigwedge_{i=1}^{m} (s_i \subseteq \{1, \dots, n\} \wedge |s_i| = 3) \quad \wedge \quad \bigwedge_{i=1}^{m-1} \bigwedge_{j=i+1}^{m} (|s_i \cap s_j| \leq 1 \wedge (s_i < s_j))$$

where we have added ordering constraints $s_i < s_j$ to remove symmetries.

We conducted a series of experiments finding the first solution for Steiner systems of orders 7,9,13 and 15. We use four labeling strategies given by cross-combinations of two ways of choosing a variable for labeling and two modes of assigning values. The choice of variable is either sequential or "first-fail" i.e., a variable with a smallest domain is chosen at each labeling step. Once a variable is chosen we explore the two possibilities for each possible element of set from least to greatest. Each element is either a member of the set represented by the variable, or it is not. The alternative (element in set) or (element not in set) we try first induces the two modes of value assignment. This order is significant in practice, as we show in our experiments.

Table 1 summarizes the experimental data produced by our BDD-based solver for the model where each primitive set constraint is represented separately and intermediate variables are introduced (see Example 6). For each problem we provide the size of the corresponding model. The sizes listed in this and the following tables are measured in the number of (thousands of) ROBDD nodes. For each run we count the run-time, the number of fails i.e., the number of times the solver backtracks from its choice of value, and the maximum number of live ROBDD nodes reached during the run. The "—" entries correspond to runs which could not complete in ten minutes.

A better ROBDD model of the constraints for the Steiner problem merges the constraints on each pair s_i and s_j, constructing a single ROBDD for each $\psi_{ij} = (|s_i \cap s_j| \leq 1) \wedge (s_i < s_j)$, and avoiding the intermediate variable. The results using this model are shown in Table 2. While the resulting model can be larger, the search space is reduced since $dom(\psi_{ij})$ propagates more strongly than the set $\{dom(|s_i \cap s_j| \leq 1), dom(s_i < s_j)\}$. and the time and peak size usage is substantially decreased. Hence it is clearly beneficial to eliminate intermediate variables and merge constraints.

Table 2. ROBDD set domain solver for finding Steiner systems. The improved model with merged constraints and no intermediate variables.

Problem	Model size $\times 10^3$	"element in set" labeling						"element not in set" labeling					
		sequential			first-fail			sequential			first-fail		
		time sec.	fails	size $\times 10^3$	time sec.	fails	size $\times 10^3$	time sec.	fails	size $\times 10^3$	time sec.	fails	size $\times 10^3$
steiner(7)	3	0.0	0	4	0.0	0	4	0.0	0	4	0.0	0	4
steiner(9)	11	0.0	2	17	0.1	798	23	0.1	9	20	0.1	9	20
steiner(13)	76	—	—	—	336.3	21,264	398	160.5	24,723	295	80.2	6,195	288
steiner(15)	92	2.8	0	172	—	—	—	2.2	0	160	2.4	0	160

Table 3. ECL^{i}PS^e set bounds solver for finding Steiner systems.

Problem	"element in set"				"element not in set"			
	sequential		first-fail		sequential		first-fail	
	time	fails	time	fails	time	fails	time	fails
steiner(7)	0.0	9	0.0	9	0.0	10	0.0	10
steiner(9)	1.0	542	1.2	542	4.9	1,394	4.8	1,394
steiner(13)	—	—	—	—	—	—	—	—
steiner(15)	2.4	6	33.9	7,139	2.8	65	3.4	65

We compared the set domain propagation solver against the ECL^{i}PS^e set bounds propagation solver `ic_sets` using the same model and search strategies. We had to implement a version of $v < w$ for sets using reified constraints. Table 3 shows the corresponding results to Table 2.

Clearly the set domain solver involves far less search than the set bounds solver. It is also able to find solutions to the hardest example `steiner(13)`. The set domain solver is faster than set bounds solving except for `steiner(15)` with sequential labeling, and when with first-fail labeling it gets stuck in a very unproductive part of the search space. One should note that with first-fail labeling the search spaces explored are quite different, since the variable ordering changes.

In order to better gauge the relationship between the two approaches without being biased by search strategy we ran experiments to find all distinct Steiner systems of orders 7 and 9 modulo the lexicographic ordering. Table 4 summarizes the results of the experiments. We list the best results for the both solvers. The results are in both cases are obtained using the "first-fail" and "element not in set" labeling strategy. The run of the ECL^{i}PS^e solver for $n = 9$ was aborted after one hour. Due to a large number of solutions neither of the two solvers can enumerate all solutions for $n = 13$ and $n = 15$. Clearly the reduced search space for the set domain solver is of substantial benefit in this case.

Table 4. Finding all solutions of Steiner ternary systems

Problem	solutions	ECL^{i}PS^e		ROBDD-based solver	
		time	fails	time	fails
steiner(7)	30	3.9	3015	0.1	38
steiner(9)	840	—	—	22.5	8934

5.2 The Social Golfers Problem

The problem consists in arranging $N = g \times s$ players into g groups of s players each week, playing for w weeks, so that no two players play in the same group twice. The problem is modeled using an $w \times g$ matrix of set variables v_{ij} where $1 \le i \le w$ and $1 \le j \le g$ are the week and the group indices respectively. The experiments described in this section are based on the following model.

$$\bigwedge_{i=1}^{w} \left(\texttt{partition}^{<}(v_{i1}, \ldots, v_{ig}) \ \wedge \ \bigwedge_{j=1}^{g} |v_{ij}| = s\right) \ \wedge$$
$$\left(\bigwedge_{i,j\in\{1\ldots w\},\ i\neq j} \ \bigwedge_{k,l\in\{1\ldots g\}} |v_{ik} \cap v_{jl}| \le 1\right) \wedge \left(\bigwedge_{i=1}^{w-1} \bigwedge_{j=i+1}^{w} v_{i1} \le v_{j1}\right)$$

The global constraint $\texttt{partition}^{<}$ is a variant of the partition constraint defined in Section 4.2 imposing a lexicographic order on its arguments i.e., $v_1 < \cdots < v_n$. The corresponding propagator is based on a single ROBDD.

Table 5 summarizes the results of our experiments with social golfers problem. Each instance of the problem shown in the table under "Problem" column is in the form $w - g - s$, where w is a number of weeks in the schedule, g is a number of groups playing each week, and s is a size of each group. We used the same instances of the problem as in [8]. For the ROBDD-based solver we provide model sizes, run times, number of fails and peak numbers of live nodes reached during the computation. For the ECL^{i}PS^e solver we provide the run times and numbers of fails. The experiments in the both solvers were carried out using "element in set" labeling strategy. Entries "—" indicate that the run was aborted after ten minutes.

As we can see from the table the ROBDD-based solver betters the ECL^{i}PS^e solver in all but one case (4-9-4). Note also that the ROBDD-based solver using "first-fail" labeling is able to cope with all twenty instances of the problem, while the ECL^{i}PS^e solver runs out of time in eight cases. The entries for 4-9-4 and 4-6-5 demonstrate that the ROBDD-based approach is sometimes more sensitive to domain sizes than the set-bounds propagation. Set-domain representations may grow rapidly as the number of potential set members grows. The more detailed set-domain information requires more computational resources for its propagation. As a result, in cases like 4-9-4 even a computation not involving backtracking can be less efficient than a search-intensive solution based on a simpler propagation.

There are three instances in the table: 5-4-3, 6-4-3 and 7-5-5 for which the problem has no solution. Reporting that no solution exists for these cases requires a solver to explore the entire search space. Note that neither of the three

Table 5. Social golfers

Problem	Model size $\times 10^3$	ROBDD-based solver						ECL^{i}PS^e			
		sequential			first-fail			sequential		first-fail	
		time sec.	fails	size $\times 10^3$	time sec.	fails	size $\times 10^3$	time sec.	fails	time sec.	fails
2-5-4	10	0.1	0	36	0.1	0	36	5.3	10,468	5.8	10,468
2-6-4	21	0.2	0	115	0.2	0	108	35.5	64,308	38.7	64,308
2-7-4	43	0.6	0	333	0.6	0	297	70.3	114,818	75.4	114,818
2-8-5	119	3.3	0	1,216	3.1	0	1,136	—	—	—	—
3-5-4	21	0.5	0	185	0.5	0	187	9.3	14,092	11.2	14,092
3-6-4	40	2.8	0	1,125	2.3	0	921	59.2	83,815	74.5	83,815
3-7-4	79	17.6	0	2,120	3.5	0	1,454	113.6	146,419	155.7	146,419
4-5-4	35	1.1	0	447	1.3	0	479	10.5	14,369	15.0	14,369
4-6-5	84	292.9	0	6,703	171.5	0	3,675	—	—	—	—
4-7-4	44	—	—	—	21.8	0	2,132	135.8	149,767	221.5	149,767
4-9-4	125	—	—	—	338.4	0	10,470	22.7	19,065	38.1	19,065
5-4-3	19	50.6	5,165	435	44.4	3,812	443	—	—	—	—
5-5-4	52	7.0	41	1,247	4.4	18	1,226	267.3	199,632	317.0	199,632
5-7-4	87	—	—	—	54.7	0	1,978	—	—	—	—
5-8-3	96	20.5	0	1,528	6.6	0	1,456	4.1	2,229	4.4	2,229
6-4-3	27	40.0	2,132	339	29.6	1,504	320	—	—	—	—
6-5-3	54	3.2	82	344	2.0	34	323	—	—	—	—
6-6-3	48	2.8	0	702	2.5	7	571	2.7	1,462	2.9	1,462
7-5-3	72	—	—	—	28.4	528	838	—	—	—	—
7-5-5	80	static fail						—	—	—	—

instances can be solved by the ECL^{i}PS^e solver, while the ROBDD-based solver always completes in under a minute. The case of 7-5-5 is especially interesting emphasizing the strength of set-domain propagation. In this case our solver reports inconsistency of the initial configuration before the actual labeling is started. Thus, the problem is solved quickly (under 0.5 sec) and there is no search involved.

Finally, we compared our experimental results with those collected by Kiziltan [8] for the same instances of the social golfers problem. The experiments reported by Kiziltan are based on four different Boolean models incorporating intricate techniques of symmetry breaking. Comparing the results for the "first-fail" strategy of the ROBDD-based solver with the best results of Kiziltan we could see that our solver improves on the number of fails in all but two cases and in run time in six cases out of the twenty.[2] Given the current naïve implementation of our system we find these results very encouraging.

[2] The timing comparisons are not very meaningful having been carried out on different machines.

6 Conclusions and Future Work

We have shown that set domain propagation is practical and advantageous for solving set problems. Set domain propagation using ROBDDs opens up a huge area of exploration: determining which constraints can be efficiently represented using ROBDDs, and determining the best model for individual problems. Clearly not all constraints (such as `atmost`) can be handled solely using ROBDD propagation, which leads us to ask how we can create more efficient global propagators for such constraints. Some avenues of investigation are using size bounded ROBDDs to approximate the global constraints, or building more complicated partial propagation approaches using ROBDD building blocks. Clearly we should incorporate set bounds propagation into our solver, this is straightforward by separating the fixed Boolean variables (L and $\bar{U}$) from the non-fixed Boolean variables in the domain representation. Together with higher priority for set bounds propagators this should increase the efficiency of set domain propagation. Finally, it would be interesting to compare the set domains solver versus a set bounds solver with better cardinality reasoning such as Cardinal [1].[3]

References

1. F. Azevedo and P. Barahona. Modelling digital circuits problems with set constraints. In *Proceedings of the 1st International Conference on Computational Logic*, volume 1861 of *LNCS*, pages 414–428. Springer-Verlag, 2000.
2. C. Bessiére and J.-C. Régin. Arc consistency for general constraint networks: preliminary results. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 398–404, 1997.
3. R. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.
4. http://vlsi.colorado.edu/~fabio/CUDD/.
5. http://www.icparc.ic.ac.uk/eclipse/.
6. C. Gervet. Interval propagation to reason about sets: Definition and implenentation of a practical language. *Constraints*, 1(3):191–244, 1997.
7. T. P. Kirkman. On a problem in combinatorics. *Cambridge and Dublin Math. Journal*, pages 191–204, 1847.
8. Z. Kiziltan. *Symmetry Breaking Ordering Constraints*. PhD thesis, Uppsala University, 2004.
9. K. Marriott and P. J. Stuckey. *Programming with Constraints: an Introduction*. The MIT Press, 1998.
10. T. Müller. *Constraint Propagation in Mozart*. PhD thesis, Universität des Saarlandes, Naturwissenschaftlich-Technische Fakultät I, Fachrichtung Informatik, 2001.
11. A. Sadler and C. Gervet. Global reasoning on sets. In *FORMUL'01 workshop on modelling and problem formulation*, 2001.
12. Z. Somogyi, F. Henderson, and T. Conway. The execution algorithm of Mercury, an efficient purely declarative logic programming language. *JLP*, 29(1–3):17–64, 1996.
13. P. Van Hentenryck, V. Saraswat, and Y. Deville. Design, implementation, and evaluation of the constraint language cc(FD). *JLP*, 37(1–3):139–164, 1998.

[3] The code is not publicly available.