Set Bounds and (Split) Set Domain Propagation Using ROBDDs

Peter Hawkins, Vitaly Lagoon, and Peter J. Stuckey

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

Abstract. Most propagation-based set constraint solvers approximate the set of possible sets that a variable can take by upper and lower bounds, and perform so-called set bounds propagation. However Lagoon and Stuckey have shown that using reduced ordered binary decision diagrams (ROBDDs) one can create a practical set domain propagator that keeps all information (possibly exponential in size) about the set of possible set values for a set variable. In this paper we first show that we can use the same ROBDD approach to build an efficient bounds propagator. The main advantage of this approach to set bounds propagation is that we need not laboriously determine set bounds propagations rules for each new constraint, they can be computed automatically. In addition we can eliminate intermediate variables, and build stronger set bounds propagators than with the usual approach. We then show how we can combine this with the set domain propagation approach of Lagoon and Stuckey to create a more efficient set domain propagation solver.

1 Introduction

It is often convenient to model a constraint satisfaction problem (CSP) using finite set variables and set relationships between them. A common approach to solving finite domain CSPs is using a combination of a global backtracking search and a local constraint propagation algorithm. The local propagation algorithm attempts to enforce consistency on the values in the domains of the constraint variables by removing values from the domains of variables that cannot form part of a complete solution to the system of constraints. Various levels of consistency can be defined, with varying complexities and levels of performance.

The obvious representation of the true domain of a set variable as a set of sets is too unwieldy to solve many practical problems. For example, a set variable which can take on the value of any subset of $\{1, \ldots, N\}$ has 2^N elements in its domain, which rapidly becomes unmanageable. Instead, most set constraint solvers operate on an approximation to the true domain of a set variable in order to avoid the combinatorial explosion associated with the set of sets representation. One such approximation [4, 8] is to represent the domain of a set variable by upper and lower bounds under the subset partial ordering relation. A *set bounds* propagator attempts to enforce consistency on these upper and lower bounds. Various refinements to the basic set bounds approximation have been proposed, such as the addition of upper and lower bounds on the cardinality of a set variable [1].

However, Lagoon and Stuckey [6] demonstrated that it is possible to use reduced ordered binary decision diagrams (ROBDDs) as a compact representation of both set domains and of set constraints, thus permitting *set domain* propagation. A domain propagator ensures that every value in the domain of a set variable can be extended to a complete assignment of all of the variables in a constraint. The use of the ROBDD representation comes with several additional benefits. The ability to easily conjoin and existentially quantify ROBDDs allows the removal of intermediate variables, thus strengthening propagation, and also makes the construction of propagators for global constraints straightforward.

Given the natural way in which ROBDDs can be used to model set constraint problems, it is therefore worthwhile utilising ROBDDs to construct other types of set solver. In this paper we extend the work of Lagoon and Stuckey [6] by using ROBDDs to build a set bounds solver. A major benefit of the ROBDD-based approach is that it frees us from the need to laboriously construct set bounds propagators for each new constraint by hand. The other advantages of the ROBDD-based representation identified above still apply, and the resulting solver performs very favourably when compared with existing set bounds solvers.

Another possibility that we have investigated is an improved set domain propagator which splits up the domain representation into fixed parts (which represent the bounds of the domain) and non-fixed parts. This helps to limit the size of the ROBDDs involved in constraint propagation and leads to improved performance in many cases.

The contributions of this paper are:

– We show how to represent the set bounds of (finite) set variables using ROBDDs. We then show how to construct efficient set bounds propagators using ROBDDs, which retain all of the modelling benefits of the ROBDD-based set domain propagators.
– We present an improved approach for ROBDD-based set domain propagators which splits the ROBDD representing a variable domain into fixed and non-fixed parts, leading to a substantial performance improvement in many cases.
– We demonstrate experimentally that the new bounds and domain solvers perform better in many cases than existing set solvers.

The remainder of this paper is structured as follows. In Section 2 we define the concepts necessary when discussing propagation-based constraint solvers. Section 3 reviews ROBDDs and their use in the domain propagation approach of Lagoon and Stuckey [6]. Section 4 investigates several new varieties of propagator, which are evaluated experimentally in Section 5. In Section 6 we conclude.

2 Propagation-based Constraint Solving

In this section we define the concepts and notation necessary when discussing propagation-based constraint solvers. Most of these definitions are identical to those presented by Lagoon and Stuckey [6], although we repeat them here for self-containedness.

Let $\mathcal{L}$ denote the powerset lattice $\langle \mathcal{P}(U), \subseteq \rangle$, where the *universe* U is a finite subset of $\mathbb{Z}$. A subset $K \subseteq \mathcal{L}$ is said to be *convex* if and only if for any $a, b \in K$ and any $c \in \mathcal{L}$ the relation $a \subseteq c \subseteq b$ implies $c \in K$. A collection of sets $C \subseteq \mathcal{L}$ is said to be an *interval* if there are sets $a, b \in \mathcal{L}$ such that $C = \{x \in \mathcal{L} \mid a \subseteq x \subseteq b\}$. We then refer to C by the shorthand $C = [a, b]$. Clearly an interval is convex.

For any finite collection of sets $x = \{a_1, a_2, \ldots, a_n\}$, we define the convex closure operation $conv : \mathcal{L} \to \mathcal{L}$ by $conv(x) = [\cap_{a \in x} a, \cup_{a \in x} a]$.

Let $\mathcal{V}$ denote the set of all set variables. Each set variable has an associated finite collection of possible values from $\mathcal{L}$ (which are themselves sets).

A *domain* D is a complete mapping from the fixed finite set of variables $\mathcal{V}$ to finite collections of finite sets of integers. We often refer to the *domain of a variable* v, in which case we mean the value of $D(v)$. A domain D_1 is said to be *stronger* than a domain D_2, written $D_1 \sqsubseteq D_2$, if $D_1(v) \subseteq D_2(v)$ for all $v \in \mathcal{V}$. A domain D_1 is equal to a domain D_2, written $D_1 = D_2$, if $D_1(v) = D_2(v)$ for all variables $v \in \mathcal{V}$. We extend the concept of convex closure to domains by defining $ran(D)$ to be the domain such that $ran(D)(x) = conv(D(x))$ for all $x \in \mathcal{V}$.

A *valuation* θ is a set of mappings from the set of variables $\mathcal{V}$ to sets of integer values, written $\{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\}$. A valuation can be extended to apply to constraints involving the variables in the obvious 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 say a valuation is 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, Propagators and Propagation Solvers A constraint is a restriction placed on the allowable values for a set of variables. We define the following *primitive set constraints*: (membership) $k \in v$, (non-membership) $k \notin v$, (constant) $u = d$, (equality) $u = v$, (subset) $u \subseteq w$, (union) $u = v \cup w$, (intersection) $u = v \cap w$, (difference) $u = v \setminus w$, (complement) $u = \overline{v}$, (cardinality) $|v| = k$, (lower cardinality bound) $|v| \geq k$, (upper cardinality bound) $|v| \leq k$, where u, v, w are set variables, k is an integer, and d is a ground set value. We can also construct more complicated constraints which are (possibly existentially quantified) conjunctions of primitive set constraints.

We define the *solutions* of a constraint c to be the set of valuations θ that make that constraint true, ie. $solns(c) = \{\theta \mid (vars(\theta) = vars(c)) \ \wedge \ (\models \theta(c))\}$

We associate a *propagator* with every constraint. A propagator f is a monotonically decreasing function from domains to domains, so $D_1 \sqsubseteq D_2$ implies that $f(D_1) \sqsubseteq f(D_2)$, and $f(D) \sqsubseteq D$. A propagator f is *correct* for a constraint c if and only if 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 any constraints.

A *propagation solver* $solv(F, D)$ for a set of propagators F and a domain D repeatedly applies the propagators in F starting from the domain D until a fixpoint is reached. In general $solv(F, D)$ is the weakest domain $D' \sqsubseteq D$ which is a fixpoint (ie. $f(D') = D'$) for all $f \in F$.

Domain and Bounds Consistency A domain D is *domain consistent* for a constraint c if D is the strongest domain containing all solutions $\theta \in D$ of c. In other words D is domain consistent if there does not exist $D' \sqsubseteq D$ such that $\theta \in D$ and $\theta \in solns(c)$ implies $\theta \in D'$.

A set of propagators F maintain domain consistency for a domain D if $solv(F, D)$ is domain consistent for all constraints c.

We define the *domain propagator* for a constraint c as

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

Since domain consistency is frequently difficult to achieve for set constraints, instead the weaker notion of bounds consistency is often used. We say that a domain D is *bounds consistent* for a constraint c if for every variable $v \in vars(c)$ the upper bound of $D(v)$ is the union of the values of v in all solutions of c in D, and the lower bound of $D(v)$ is the intersection of the values of v in all solutions of c in D.

A set of propagators F maintain set bounds consistency for a constraint c if $solv(F, D)$ is bounds consistent for all domains D.

We define the *set bounds propagator* for a constraint c as

$$sb(c)(D)(v) = \begin{cases} conv(dom(c)(ran(D))(v)) & \text{if } v \in vars(c) \\ D(v) & \text{otherwise} \end{cases}$$

3 ROBDDs and Set Domain Propagators

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 $V = \{x_1, \ldots, x_n\}$, and by $\bar{\exists}_V F$ we mean $\exists_{V'} F$ where $V' = vars(F) \setminus V$.

Binary Decision Trees (BDTs) are a well-known method of representing Boolean functions on Boolean variables using complete binary trees. Every internal node $n(v, f, t)$ in a BDT r is labelled with a Boolean variable v, and has two outgoing arcs — the 'false' arc (to BDT f) and the 'true' arc (to BDT t). Leaf nodes are either 0 (false) or 1 (true). Each node represents a single test of the labelled variable; when traversing the tree the appropriate arc is followed depending on the value of the variable. Define the size $|r|$ as the number of internal nodes in an ROBDD r, and $VAR(r)$ as the set of variables v appearing in some internal node in r. A Binary Decision Diagram (BDD) is a variant of a Binary Decision Tree that relaxes the tree requirement, instead representing a Boolean function as a directed acyclic graph with a single root node by allowing nodes to have multiple parents. This permits a compact representation of many (but not all) Boolean functions.

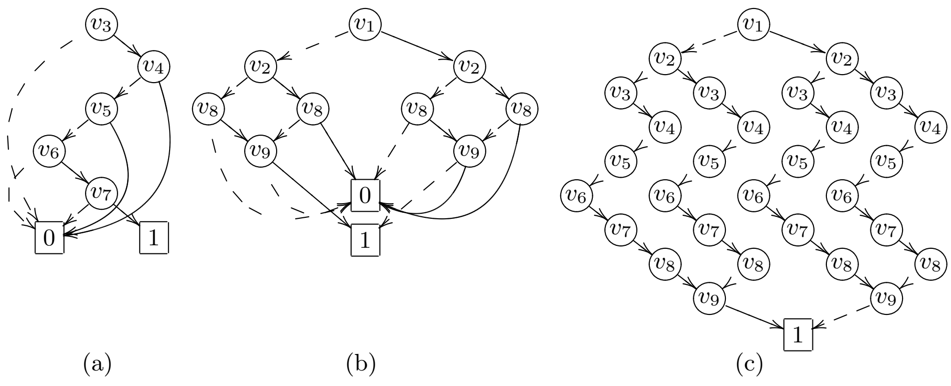

Fig. 1. ROBDDs for (a) $L\overline{U} = v_3 \wedge \neg v_4 \wedge \neg v_5 \wedge v_6 \wedge v_7$ (b) $R = \neg(v_1 \leftrightarrow v_9) \wedge \neg(v_2 \leftrightarrow v_8)$ and (c) $L\overline{U} \wedge R$ (omitting the node 0 and arcs to it). Solid arcs are “then” arcs, dashed arcs are “else” arcs.

Two canonicity properties allow many operations on a BDD to be performed very efficiently [3]. A BDD is said to be *reduced* if it contains no identical nodes (that is, nodes with the same variable label and identical then and else arcs) and has no redundant tests (no node has both then and else arcs leading to the same node). A BDD is said to be *ordered* if there is a total ordering $\prec$ of the variables, such that if there is an arc from a node labelled v_1 to a node labelled v_2 then $v_1 \prec v_2$. A *reduced ordered* BDD (ROBDD) has the nice property that the function representation is canonical up to variable reordering. This permits efficient implementations of many Boolean operations.

We shall be interested in a special form of ROBDDs. A *stick ROBDD* is an ROBDD where for every internal node $n(v, f, t)$ exactly one of f or t is the constant 0 node.

Example 1. Figure 1(a) gives an example of a stick ROBDD representing the formula $v_3 \wedge \neg v_4 \wedge \neg v_5 \wedge v_6 \wedge v_7$. Figure 1(b) gives an example of a more complex ROBDD representing the formula $\neg(v_1 \leftrightarrow v_9) \wedge \neg(v_2 \leftrightarrow v_8)$. One can verify that the valuation $\{v_1 \mapsto 1, v_2 \mapsto 0, v_8 \mapsto 1, v_9 \mapsto 0\}$ makes the formula true by following the path right, left, right, left from the root.

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.

If x is a set variable and $A \in D(x)$ is a subset of $\{1, \ldots, N\}$, then we can represent A as a valuation θ over the Boolean variables $V(x) = \{x_1, \ldots, x_N\}$, where $\theta_A = \{x_i \mapsto 1 \mid i \in A\} \cup \{x_i \mapsto 0 \mid i \notin A\}$. The domain of x can be represented as a Boolean formula ϕ which has as solutions $\{\theta_A \mid A \in D(x)\}$. We will order the variables $x_1 \prec x_2 \cdots \prec x_N$.

An ROBDD allows us to represent (some) subsets of $\mathcal{P}(\{1, \ldots, N\})$ efficiently. For example the subset $S = \{\{3, 6, 7, 8, 9\}, \{2, 3, 6, 7, 9\}, \{1, 3, 6, 7, 8\}, \{1, 2, 3, 6, 7\}\}$, where $N = 9$, is represented by the ROBDD in Figure 1(c). In particular, an interval can be compactly represented as a stick ROBDD of a conjunction of positive and negative literals (corresponding to the lower bound and the complement of the upper bound respectively). For example the subset $conv(S) = [\{3, 6, 7\}, \{1, 2, 3, 6, 7, 8, 9\}]$ is represented by the stick ROBDD in Figure 1(a).

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. By ordering the variables in each ROBDD carefully we can build small representations of the formulae. The *pointwise* order of Boolean variables is defined as follows. Given set variables $u \prec v \prec w$ ranging over sets from $\{1, \dots, N\}$ we order the Boolean variables as $u_1 \prec v_1 \prec w_1 \prec u_2 \prec v_2 \prec w_2 \prec \cdots u_n \prec v_n \prec w_n$.

By ordering the Boolean variables pointwise we can guarantee linear sized representations for $B(c)$ for each primitive constraint c except those for cardinality. The size of the representations of $B(k \in v)$ and $B(k \notin v)$ are $O(1)$, while $B(v = w)$, $B(v = d)$, $B(v \subseteq w)$, $B(u = v \cup w)$, $B(u = v \cap w)$, $B(u = v \setminus w)$, $B(v = \bar{w})$ and $B(v \neq w)$ are all $O(N)$, and $B(|v| = k)$, $B(|v| \leq k)$ and $B(|v| \geq k)$ are all $O(k \times (N - k))$. For more details see [6].

3.3 ROBDD-based Set Domain Propagation

Lagoon and Stuckey [6] demonstrated how to construct a set domain propagator $dom(c)$ for a constraint c using ROBDDs. If $vars(c) = \{v_1, \dots, v_n\}$, then we have the following description of a domain propagator:

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

Since $B(c)$ and $D(v_j)$ are ROBDDs, we can directly implement this formula using ROBDD operations. In practice it is more efficient to perform the existential quantification as early as possible to limit the size of the intermediate ROBDDs. Many ROBDD packages provide an efficient combined conjunction and existential quantification operation, which we can utilise here. This leads to the following implementation:

$$\begin{aligned}
\phi_i^0 &= B(c) \\
\phi_i^j &= \begin{cases} \exists_{V(v_j)}(D(v_j) \wedge \phi_i^{j-1}) & 1 \leq i, j \leq n,\ i \neq j \\ \phi_i^{i-1} & i = j \end{cases} \\
dom(c)(D)(v_i) &= D(v_i) \wedge \phi_i^n
\end{aligned} \tag{2}$$

The worst case complexity is still $O(|B(c)| \times \Pi_{j=1}^{n} |D(v_j)|)$. Note that some of the computation can be shared between propagation of c for different variables since $\phi_i^j = \phi_{i'}^j$ when $j < i$ and $j < i'$.

4 Set Bounds and Split Domain Propagation

4.1 Set Bounds Propagation Using ROBDDs

ROBDDs are a very natural representation for sets and set constraints, so it seems logical to try implementing a set bounds propagator using ROBDD techniques. Since set bounds are an approximation to a set domain, we can construct an ROBDD representing the bounds on a set variable by approximating the ROBDD representing a domain. Only a trivial change is needed to the set domain propagator to turn it into a set bounds propagator.

The bounds on a set domain can be easily identified from the corresponding ROBDD representation of the domain. In an ROBDD-based domain propagator, the bounds on each set variable correspond to the *fixed* variables of the ROBDDs representing the set domains. A BDD variable v is said to be *fixed* if either for every node $n(v, t, e)$ t is the constant 0 node, or for every node $n(v, t, e)$ e is the constant 0 node. Such variables can be identified in a linear time scan over the domain ROBDD. For convenience, if ϕ is an ROBDD, we write $[\![\phi]\!]$ to denote the ROBDD representing the conjunction of the fixed variables of ϕ. Note that if ϕ represents a set of sets S, then $[\![\phi]\!]$ represents $conv(S)$. For example, if ϕ is the ROBDD depicted in Figure 1(c), then $[\![\phi]\!]$ is the ROBDD of Figure 1(a).

We can use this operation to convert our domain propagator into a bounds propagator by discarding all of the non-fixed variables from the ROBDDs representing the variable domains after each propagation step. Assume that $D(v)$ is always a stick ROBDD, which will be the case if we have only been performing set bounds propagation. If c is a constraint, and $vars(c) = \{v_1, \ldots, v_n\}$, we can construct a set bounds propagator $sb(c)$ for c thus:

$$\begin{aligned} \phi_0 &= B(c) \\ \phi_j &= \exists_{VAR(D(v_j))}(D(v_j) \wedge \phi_{j-1}) \\ sb(c)(D)(v_i) &= \overline{\exists}_{V(v_i)}(D(v_i) \wedge [\![\phi_n]\!]) \end{aligned} \tag{3}$$

Despite the apparent complexity of this propagator, it is significantly faster than a domain propagator for two reasons. Firstly, since the domains $D(v)$ are sticks, the resulting conjunctions ϕ_j are always decreasing (technically non-increasing) in size, hence the corresponding propagation is much faster. Overall the complexity can be made $O(|B(c)|)$.

As an added bonus, we can use the updated set bounds to simplify the ROBDD representing the propagator. Since fixed variables will never interact further with propagation they can be projected out of $B(c)$, so we can replace $B(c)$ by $\exists_{VAR([\![\phi_n]\!])}\phi_n$. In practice it turns out to be more efficient to replace $B(c)$ by ϕ_n since this has already been calculated.

This set bounds solver retains many of the benefits of the ROBDD-based approach, such as the ability to remove intermediate variables and the ease of construction of global constraints, in some cases permitting a performance improvement over more traditional set bounds solvers.

4.2 Split Domain Propagation

One of the unfortunate characteristics of ROBDDs is that the size of the BDD representing a function can be very sensitive to the variable ordering that is chosen. If the fixed variables of a set domain do not appear at the start of the variable ordering, then the ROBDD for the domain in effect can contain several copies of the stick representing those variables. For example, Figure 1(c) effectively contains several copies of the stick in Figure 1(a). Since many of the ROBDD operations we perform take quadratic time, this larger than necessary representation costs us in performance.

In the context of groundness analysis of logic programs Bagnara [2] demonstrated that better performance can be obtained from an ROBDD-based program analyzer by splitting an ROBDD up into its fixed and non-fixed parts. We can apply the same technique here.

We split the ROBDD representing a domain $D(v)$ into a pair of ROBDDs $(L\overline{U}, R)$. $L\overline{U}$ is a stick ROBDD representing the Lower and Upper set bounds on $D(v)$, and R is a Remainder ROBDD representing the information on the unfixed part of the domain. Logically $D = L\overline{U} \wedge R$. We will write $L\overline{U}(D(v))$ and $R(D(v))$ to denote the $L\overline{U}$ and R parts of $D(v)$ respectively.

The following result provides an upper bound of the size of the split domain representation (proof omitted for space reasons):

Proposition 1. *Let D be an ROBDD, $L\overline{U} = [\![D]\!]$, and $R = \exists_{VAR(L\overline{U})} D$. Then $D \leftrightarrow L\overline{U} \wedge R$ and $|L\overline{U}| + |R| \leq |D|$.* □

Note that $|D|$ can be $O(|L\overline{U}| \times |R|)$. For example, considering the ROBDDs in Figure 1 where $L\overline{U}$ is shown in (a), R is in (b) and $D = L\overline{U} \wedge R$ in (c) we have that $|L\overline{U}| = 5$ and $|R| = 9$ but $|D| = 9 + 4 \times 5 = 29$.

We construct the split propagator as follows: First we eliminate any fixed variables (as in bounds propagation) and then apply the domain propagation on the "remainders" R. We return a pair $(L\overline{U}, R)$ of the new fixed variables, and new remainder.

$$\begin{aligned}
\phi_0 &= B(c) \\
\phi_j &= \exists_{VAR(L\overline{U}(D(v_j)))}(L\overline{U}(D(v_j)) \wedge \phi_{j-1}) \\
\delta_i &= \overline{\exists}_{V(v_i)}\,(\phi_n \wedge \bigwedge_{j=1}^{n} R(D(v_j))) \\
\beta_i &= L\overline{U}(D(v_i)) \wedge [\![\delta_i]\!] \\
dom(c)(D)(v_i) &= (\beta_i, \exists_{VAR(\beta_i)}\delta_i)
\end{aligned} \tag{4}$$

For efficiency each δ_i should be calculated in an analogous manner to ϕ_i^n of Equation (2).

There are several advantages to the split domain representation. Proposition 1 tells us that the split domain representation is effectively no larger the simple domain representation. In many cases, the split representation can be substantially smaller, thus speeding up propagation. Secondly, we can use techniques from the bounds solver implementation to simplify the ROBDDs representing the constraints as variables are fixed during propagation. Thirdly, it becomes possible to mix the use of set bounds which operate only on the $L\overline{U}$ component of the domain with set domain propagators that need complete domain information.

5 Experimental Results

We have extended the set domain solver of Lagoon and Stuckey [6] to incorporate a set bounds solver and a split set domain solver. The implementation is written in Mercury [10] interfaced with the C language ROBDD library CUDD [9].

A series of experiments were conducted to compare the performance of the various solvers on a 933Mhz Pentium III with 1 Gb of RAM running Debian GNU/Linux 3.0. For the purposes of comparison benchmarks were also implemented using the `ic_sets` library of ECL^{i}PS^e v5.6 [5]. Since our solvers do not yet incorporate an integer constraint solver, we are limited to set benchmarks that utilise only set variables.

5.1 Steiner Systems

A commonly used benchmark for set constraint solvers is the calculation of small Steiner systems. A Steiner system $S(t, k, N)$ is a set X of cardinality N and a collection C of subsets of X of cardinality k (called 'blocks'), such that any t elements of X are in exactly one block. Steiner systems are an extensively studied branch of combinatorial mathematics. If $t = 2$ and $k = 3$ we have the so-called Steiner Triple Systems, which are often used as benchmarks [4, 6]. Any Steiner system must have exactly $m = \binom{N}{t}/\binom{k}{t}$ blocks (Theorem 19.2 of [7]).

We use the same modelling of the problem as Lagoon and Stuckey [6], extended for the case of more general Steiner Systems. We model each block as a set variable $s_1, \ldots, s_m$, with the constraints:

$$\bigwedge_{i=1}^{m} (|s_i| = k) \tag{5}$$

$$\wedge \bigwedge_{i=1}^{m-1} \bigwedge_{j=i+1}^{m} (\exists u_{ij}.u_{ij} = s_i \cap s_j \wedge |u_{ij}| \leq (t-1)) \wedge (s_i < s_j) \tag{6}$$

A necessary condition for the existence of a Steiner system is that $\binom{N-i}{t-i}/\binom{k-i}{t-i}$ is an integer for all $i \in \{0, 1, \ldots, t-1\}$; we say a set of parameters (t, k, N) is admissible if it satisfies this condition [7]. In order to choose test cases, we ran each solver on every admissible set of (t, k, N) values for $N < 32$. Results are given for every test case that at least one solver was able to solve within a time limit of 10 minutes. The labelling method used in all cases was sequential 'element-not-in-set' in order to enable accurate comparison of propagation performance.

Table 1. Performance results on Steiner Systems. The time and number of fails needed to find a solution or prove unsatisfiability are reported. $\times$ denotes abnormal termination on a test-case, either due to global/trail stack overflow in the case of ECL^{i}PS^e, or due to allocating too many BDD variables for the ROBDD solvers. '—' denotes failure to complete a testcase within 10 minutes. In all cases the domain solver and the split solver have the same number of fails

Problem	Separate Constraints							Merged Constraints				
	ECL^{i}PS^e		Bounds		Domain		Split	Bounds		Domain		Split
	Time /s	Fails	Time /s	Fails	Time /s	Fails	Time /s	Time /s	Fails	Time /s	Fails	Time /s
S(2,3,7)	0.6	10	**0.1**	10	**0.1**	**0**	0.2	**0.1**	8	**0.1**	**0**	**0.1**
S(3,4,8)	1.0	21	0.2	21	0.9	**0**	0.9	**0.1**	18	0.2	**0**	0.2
S(2,3,9)	16.7	1,394	5.9	1,394	2.1	100	3.1	0.3	325	**0.2**	**9**	**0.2**
S(2,3,13)	—	—	—	—	—	—	—	—	—	**253.1**	**24,723**	350.6
S(2,4,13)	4.0	313	1.9	313	3.6	32	3.1	**0.2**	157	0.3	**0**	0.3
S(2,3,15)	7.7	65	7.9	65	43.3	**0**	49.7	**0.7**	56	2.8	**0**	3.5
S(2,4,16)	—	—	—	—	—	—	—	—	—	1.3	**15**	**1.2**
S(2,6,16)	—	—	—	—	—	—	—	—	—	**162.4**	**15,205**	166.3
S(3,4,16)	162.0	289	$\times$	$\times$	$\times$	$\times$	$\times$	**24.5**	**274**	—	—	—
S(2,5,21)	6.9	421	6.7	421	227.6	**0**	120.3	**0.9**	413	2.9	**0**	2.9
S(3,6,22)	115.4	1,619	$\times$	$\times$	$\times$	$\times$	$\times$	**19.5**	**1,608**	—	—	—
S(2,3,31)	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	**62.1**	**280**	—	—	—

To compare the raw performance of the bounds propagators we performed experiments using a model of the problem with primitive constraints and intermediate variables u_{ij} directly as shown in Equations 5 and 6. The same model was used in both ECL^{i}PS^e and ROBDD-based solvers, permitting comparison of propagation performance, irrespective of modelling advantages. The results are shown in "Separate Constraints" section of Table 1. In all cases the ROBDD bounds solver performs better than ECL^{i}PS^e, with the exception of two cases which the ROBDD-based solvers cannot solve due to a need for an excessive number of BDD variables to model the intermediate variables (no propagation occurs in these cases).

Of course, the ROBDD representation permits us to merge primitive constraints and remove intermediate variables, allowing us to model the problem as m unary constraints and $\binom{m}{2}$ binary constraints (containing no intermediate variables u_{ij}) corresponding to Equations 5 and 6 respectively. Results for this improved model are shown in the "Merged Constraints" section of Table 1. Lagoon and Stuckey [6] demonstrated this leads to substantial performance improvements in the case of a domain solver; we find the same effect evident here for all of the ROBDD-based solvers.

With the revised model of the problem the ROBDD bounds solver outstrips the ECL^{i}PS^e solver by a significant margin for all test cases. Conversely the split domain solver appears not to produce any appreciable reduction in the

Table 2. First-solution performance results on the Social Golfers problem. Time and number of failures are given for all solvers, and the ROBDD peak live node count for ROBDD-based solvers. A first-fail "element-in-set" labelling strategy is used in all cases. "—" denotes failure to complete a test case within 10 minutes. The cases 5-4-3, 6-4-3, and 7-5-5 have no solutions

Problem	ECL^{i}PS^e		Bounds			Domain			Split		
w-*g*-*s*	time /s	fails	time /s	fails	size $\times 10^3$	time /s	fails	size $\times 10^3$	time /s	fails	size $\times 10^3$
2-5-4	16.2	10,468	**0.2**	30	**41**	**0.2**	**0**	44	**0.2**	**0**	43
2-6-4	107.6	64,308	1.5	2,036	**117**	0.4	**0**	129	**0.3**	**0**	124
2-7-4	210.8	114,818	5.1	4,447	**212**	1.0	**0**	346	**0.9**	**0**	325
2-8-5	—	—	—	—	—	5.3	**0**	1,367	**4.4**	**0**	**1,342**
3-5-4	30.8	14,092	**0.3**	44	**82**	0.8	**0**	199	0.6	**0**	189
3-6-4	200.0	83,815	5.5	2,357	**212**	3.4	**0**	952	**2.9**	**0**	919
3-7-4	404.8	146,419	16.7	5,140	**366**	5.5	**0**	1,504	**4.8**	**0**	1,419
4-5-4	39.5	14,369	**0.6**	47	**137**	2.0	**0**	487	1.6	**0**	461
4-6-5	—	—	**106.2**	19,376	**425**	187.9	**0**	4,159	131.1	**0**	2,880
4-7-4	560.2	149,767	31.7	5,149	**500**	24.7	**0**	2,139	**17.6**	**0**	2,004
4-9-4	95.4	19,065	**6.0**	139	**1,545**	395.7	**0**	13,137	256.9	**0**	5,615
5-4-3	—	—	454.8	103,972	**137**	**52.9**	**3,812**	543	63.1	**3,812**	613
5-5-4	—	—	10.6	2,388	**242**	5.8	**18**	1,333	**4.0**	**18**	1,128
5-7-4	—	—	54.8	5,494	**616**	67.5	**0**	3,054	**48.2**	**0**	2,476
5-8-3	12.0	2,229	**2.1**	19	**761**	10.3	**0**	2,046	10.9	**0**	2,192
6-4-3	—	—	569.8	90,428	**118**	**32.7**	**1,504**	440	36.9	**1,504**	612
6-5-3	—	—	3.9	495	**159**	2.7	**34**	441	**2.1**	**34**	414
6-6-3	8.0	1,462	—	—	—	3.6	**7**	**699**	**2.9**	**7**	709
7-5-3	—	—	—	—	—	37.8	**528**	**1,058**	**31.2**	**528**	1,082
7-5-5	—	—	—	—	—	static fail					

BDD sizes over the original domain solver, and so the extra calculation required to maintain the split domain leads to poorer performance.

5.2 Social Golfers

Another common set benchmark is the "Social Golfers" problem, which consists of arranging $N = g \times s$ golfers into g groups of s players for each of w weeks, such that no two players play together more than once. Again, we use the same model as Lagoon and Stuckey [6], using a $w \times g$ matrix of set variables v_{ij} where $1 \leq i \leq w$ and $1 \leq j \leq g$. It makes use of a global partitioning constraint not available in ECL^{i}PS^e but easy to build using ROBDDs.

Experimental results are shown in Table 2. These demonstrate the split domain solver is almost always faster than the standard domain solver, and requires substantially less space. Note that the BDD size results are subject to the operation of a garbage collector and hence are only a crude estimate of the relative sizes. This is particularly true in the presence of backtracking since Mercury has a garbage collected trail stack.

As we would expect, in most cases the bounds solver performs worse than the domain solvers due to weaker propagation, but can perform substantially better (for example 4-9-4 and 5-8-3) where because of the difference in search it explores a more productive part of the search space first (first-fail labelling acts differently for different propagators). Note the significant improvement of the ROBDD bounds solver over the ECL^{i}PS^e solver because of stronger treatment of the global constraint.

6 Conclusion

We have demonstrated two novel ROBDD-based set solvers, a set bounds solver and an improved set domain solver based on split set domains. We have shown experimentally that in many cases the bounds solver has better performance than existing set bounds solvers, due to the removal of intermediate variables and the straightforward construction of global constraints. We have also demonstrated that the split domain solver can perform substantially better than the original domain solver due to a reduction in the size of the ROBDDs.

Avenues for further investigation include investigating the performance of a hybrid bounds/domain solver using the split domain representation, and investigating other domain approximations in between the bounds and full domain approaches.

References

[1] F. Azevedo. *Constraint Solving over Multi-valued Logics.* PhD thesis, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, 2002.

[2] R. Bagnara. A reactive implementation of Pos using ROBDDs. In *Procs. of PLILP*, volume 1140 of *LNCS*, pages 107–121. Springer, 1996.

[3] R. E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318, 1992. ISSN 0360-0300.

[4] C. Gervet. Interval propagation to reason about sets: Definition and implementation of a practical language. *Constraints*, 1(3):191–246, 1997.

[5] IC-PARC. The ECLiPSe constraint logic programming system. [Online, accessed 31 May 2004], 2003. `http://www.icparc.ic.ac.uk/eclipse/`.

[6] V. Lagoon and P. Stuckey. Set domain propagation using ROBDDs. In M. Wallace, editor, *Proceedings of the International Conference on Principle and Practice of Constraint Programming*, number 3258 in LNCS, pages 347–361. Springer-Verlag, 2004.

[7] J. H. van Lint and R. M. Wilson. *A Course in Combinatorics.* Cambridge University Press, 2nd edition, 2001.

[8] J.-F. Puget. PECOS: a high level constraint programming language. In *Proceedings of SPICIS'92*, Singapore, 1992.

[9] F. Somenzi. CUDD: Colorado University Decision Diagram package. [Online, accessed 31 May 2004], Feb. 2004. `http://vlsi.colorado.edu/~fabio/CUDD/`.

[10] Z. Somogyi, F. Henderson, and T. Conway. The execution algorithm of Mercury, an efficient purely declarative logic programming language. *Journal of Logic Programming*, 29(1–3):17–64, 1996.