Set Constraints Representation using ROBDDs

Mathias Möhl

Seminar Constraint Programming WS 0405
Department of Computer Science
Saarland University
Saarbrücken, Germany

Abstract. The efficiency of constraint solvers depends heavily on the representation that is used for the constraint variables. Concerning set constraints this point becomes central because naive exact representations have exponential size and are therefore in most cases only approximated. Since this approximation always comes along with weaker propagation it is desirable to have a representation that is as precise as possible but has still a handable size. Reduced ordered binary decision diagrams ROBDDs can be used as such a representation which is exact in all cases and claims to be tractable in practical applications. In cases where this representation is still to large ROBDDs can also be used to obtain a representation with guaranteed linear space bound but weaker propagation as a trade-off. Another advantage of the ROBDD representation is its modularity which offers an easy and efficient way to specify new constraints reusing existing ones. This extended abstract basically summarizes the work by V. Lagoon and P. Stuckey published in [7] and [6].

1 Introduction

Set constraints are constraints whose variables do not have integers or real numbers as values but sets of them. In this report we restrict our self to sets ranging over a finite set of integers. Many practical problems can be encoded as a set constraint satisfaction problem. Grouping people according to certain conditions could be such an example: Each person is identified by an integer and the groups are represented by variables ranging over sets of them. Typical constraints restrict the cardinality of these sets, enforce the exclusion or inclusion of some elements or specify subset relations among different variables.

While integer constraint solver usually represent a variable by the set of all possible values that the variable can take, this leads immediately to an exponential size for set constraints: A set variable over integers $1, ..., n$ could have any subset of $\{1, ..., n\}$ as value and the set of possible values would thus be the power set of $\{1, ..., n\}$ which has 2^n elements. The way out of this trouble that most current approaches take, is to replace the exact representation by an approximated one. But the guaranteed linear space of these approximated representations does not come for free: Propagation on approximated representations can never be as strong as a propagation that benefits from all details of an exact

representation. Since weaker propagation usually involves more search, the size of the search tree may dominate the space consumption of the constraint solver and eliminate the efficiency gain.

As an alternative, *reduced ordered binary decision diagrams (ROBDDs)* can be used as an exact representation for set variables. ROBDDs as a restriction of BDDs (C.Y.Lee 1959 [8], S.B.Akers 1978 [1]) are first mentioned by S.Fortune [5]. R.E.Bryant [3] describes ROBBDs as directed acyclic graphs which represent boolean formulas in a very efficient way and discusses several application areas for ROBBDs. He also mentions that they can be used to represent sets. Since variable domains are sets, one can use ROBBDs to represent them. Because this representation is exact one can build strong, namely *domain consistent*, propagators based on it. As each exact representation the ROBDDs may have exponential size in worst case, never the less there are evidences that the size of the ROBDDs is tractable in practical applications and that the small search tree causes an over all efficiency gain compared to approximating bounds representations.

A further advantage of the ROBDD is based on the fact that ROBDDs are nothing else than efficient structures to represent boolean formulae. The fact that a boolean formula can be composed out of smaller ones can be lifted to constraints such that the development of new constraints is reducible to their specification as conjunction, disjunction, negation or even existential quantification of existing constraints. This does not only offer an easy interface to specify new constraints which hides all implementation details, but also leads to efficient implementations of the constraints specified in that way. One important aspect for this efficiency is for example, that intermediate variables for existential quantification are compiled away.

The constraint solver implemented by Lagoon and Stuckey shows that all these benefits of the ROBDD representation for set constraints are not only theoretical advantages, but also better results in practice. With the set solver based on ROBDD representation the computation is not only much faster than in concurrent approaches, but also problem instances are tractable on which other solvers run out of space.

The remainder of this extended abstract is structured as follows. In chapter (2) we will explain the basic concepts and structures involved in set constraint solving, in particular set variables, domains, set constraints and propagators. In chapter (3) we will define ROBDDs and show that variable domains and constraints can by represented by them. In chapter (4) we describe different types of propagators that work on ROBDD representations and in chapter (5) we show other benefits that come with the ROBDD representation. In chapter (6) we finally present some experimental results which reflect all advantages of the ROBDD representation that were explained in the previous chapters.

2 Set Constraints

This chapter will give a short overview over important concepts related to the idea of set constraints. After making precise what constraint satisfaction problems, set variables and set constraints are, we explain how propagators work with these structures and which types of propagators are used for set constraints.

2.1 Constraint Satisfaction Problems for Set Constraints

In [2] Apt defines a *constraint satisfaction problem (CSP)* as a tuple $(C, v_1 \in D_1, ..., v_n \in D_n)$ where C is a set of constraints and $v_1, ..., v_n$ are variables which could take any value of their respective domains $D_1, ..., D_n$. A constraint $c \in C$ is just a subset of $D_{i(1)} \times ... \times D_{i(m)}$, where $i(1), ..., i(m)$ is a subsequence of $1, ..., n$. A constraint c is interpreted such that the variables $v_{i(1)}, ..., v_{i(m)}$ satisfy c iff $(v_{i(1)}, ..., v_{i(m)})$ is an element of c. The variables which are constraint by c, namely $\{ v_j \mid j \in \{i(1), ..., i(m)\} \}$, are denoted by $V(c)$ in the following.

Finding a solution for a CSP is basically the process of narrowing the domains $D_1, ..., D_n$ such that all constraints $c \in C$ are satisfied. Each variable assignment in such a final domain is called a solution of the CSP. The process of narrowing the domains splits up in two parts: *Propagation* is a technique that uses one *propagator* for each constraint c to remove variable constellations that do not satisfy c. Since propagation is normally not complete, *distribution* is used to split up the CSP in smaller ones and to search for solutions in the smaller CSPs.

CSPs whose variables do not represent integer or real numbers but sets of integers are called set constraint satisfaction problems. In real constraint programming interfaces the user does not have to specify the constraints directly as subsets of $D_1 \times ... \times D_n$ but is rather provided some operators which generate these sets if they are applied to some constraint variables and constants. Typical such operators for set variables u, v, w, the constant set d and constant integer k are $k \in v$ (element), $k \notin v$ (not element), $v = w$ (equality), $v \subseteq w$ (subset), $u = v \cup w$ (union), $u = v \cap w$ (intersection), $u = v - w$ (subtraction), $v = \overline{w}$ (complement), $v \neq w$ (disequality), $v \neq d$(disequality constant), $|v| = k$ (cardinality equals), $|v| \leq k$ (cardinality upper bound) and $|v| \geq k$ (cardinality lower bound). A simple constraint satisfaction problem formulated with some of these operators is the following:

(1) $Var\ v \in \{1, 2, 3, 4\}$
(2) $3 \notin v$
(3) $|v| = 2$
(4) $v \neq \{1, 4\}$
(5) $v \neq \{2, 4\}$

In line (1) a set variable v ranging over $\{1, 2, 3, 4\}$ is introduced, in lines (2) to (5) constraints on this variable are imposed using the operators mentioned above. The only solution to this example is $v = \{1, 2\}$.

A *set constraint solver* is a program that finds (one or all) solutions of such a set CSP. Set constraint solver use different methods to internally represent the domains of set variables of the CSP they try to solve. The main challenge in finding a suitable representation is the fact that exact representations have exponential size in worst case. While integer variable domains are usually represented by the set of all elements in the domain, this is intractable for set variables: The set of the initial domain of a set variable ranging over $1, ..., n$ is the power set of $\{1, ..., n\}$ and thus has size 2^n. Most systems therefore use approximated representations which do not store all possible values but only an upper and a lower bound with respect to a certain order, usually set inclusion. These bounds are then interpreted such that the variable may take any value that is not smaller than the lower and not bigger than the upper bound. Such an approximation adds all elements to a domain D, which are in the convex closure of D abbreviated with $conv(D)$.

The bounds representation can be made more precise by adding an upper and a lower bound for the cardinality of the set, another approach shown in [9] improves the approximation by adding bounds with respect to the lexicographic order as additional information. The lexicographic bounds are usefull in particular if the set variables are constraint to represent singleton sets.

2.2 Propagators

Constraint solver use *propagators* to narrow the domains of the variables. For each imposed constained one propagator is generated. A propagator for a constraint c of a CSP $(C, v_1 \in D_1, ..., v_n \in D_n)$ computes new domains $D'_{i(1)}, ..., D'_{i(m)}$ for the variables in $V(c)$ such that the new domains contain only elements that satisfy certain consistency requirements. The minimal requirement for a propagator is that the new domain of a variable contains no elements that were not in it's old domain. This property is even fullfilled by the propagator which does not change the domains at all; it only ensures, that the propagator does not make the problem even harder. To make sure that the propagators really narrow the domains as much as possible, most propagators fullfill stronger requirements, they are *bounds consistent* or even better *domain consistent*.

2.3 Bounds and Domain Consistency

A propagator for a CSP C and a constaint c is domain consistent iff for each variable $v \in V(c)$ a value is only in the narrowed domain of v if there exists a solution of the constraint c where v has this value. This means that any further narrowing would lead to the loss of a solution. Systems which use bounds representations for the domains of the variables can not have domain consistent

propagators, because the approximated domains do not contain enough information to ensure this property. Bounds consistency is intuitively the best consistency that a propagator working on variable domains in bounds representations can obtain. A domain consistent propagator narrows $D_{i(1)}, ..., D_{i(m)}$ down to $conv(D'_{i(1)}), ..., conv(D'_{i(m)})$ iff the corresponding domain consistent propagator narrows $conv(D_{i(1)}), ..., conv(D_{i(m)})$ down to $D'_{i(1)}, ..., D'_{i(m)}$.

3 ROBDDs

3.1 Formalization of ROBDDs

A ROBDD is formally just a representation for a boolean formula F or the corresponding boolean function . ROBDDs are acyclic directed graphs with a root node and (at most) two terminal nodes. The terminal nodes are labeled with 1 and 0 and all other nodes are labeled with a variable from F. Each non-terminal node has two outgoing edges, a 0-edge and a 1-edge. The value of F is calculated in the ROBDD by starting at the root node and taking the 0-egde out of a node if the corresponding variable has value 0 an taking the 1-edge otherwise. The label of the terminal node, where this traversal ends, represents the value of the boolean expression. ROBDDs should be small and unique in the sense that for each boolean formula there exists exactly one corresponding ROBDD and vice versa. To reach this goal, ROBDDs are restricted by the following conditions:

1. The variables are ordered by some ordering $\prec$ such that if a node w is reachable from a node v then $v \prec w$.
2. There are no nodes with the same variable label and 1- and 0-branches.
3. Each non-terminal node has different 1- and 0-branches

Condition 1 breaks symmetries, condition 2 avoids the existance of dublicated subtrees and condition 3 disallows for redundant tests.

3.2 ROBDD Representation of Set Variable Domains

Since ROBDDs are just an efficient way to represent boolean formulas, one can represent set variable domains with ROBDDs if one can encode the possible values of a set varible as the solutions of a boolean formula. This can be done as shown in the following.

Each set variable v with domain $D \subseteq \{1, ..., n\}$ is associated with boolean variables $Vars(v) := v_1, ..., v_n$ and a boolean formula F containing (at most) these variables. Each solution of the formula F represents one possible element of D, namely the set $\{\, i \in \{1, ..., n\} \mid v_i = 1 \text{ in this solution} \,\}$. If v would range over $\{1, 2, 3\}$ and D would be $\{\{1, 2\}, \{1, 3\}, \emptyset\}$, then this could be represented as the boolean formula $(v_1 \wedge v_2 \wedge \neg v_3) \vee (v_1 \wedge \neg v_2 \wedge v_3) \vee (\neg v_1 \wedge \neg v_2 \wedge \neg v_3)$, where each of the three parts in brackets represents one of the three elements of D. The ROBDD corresponding to this formula is shown in Figure 1. 0-edges are represented as dotted lines in the figure , 1-edges are solid. Since each solution

of F corresponds to a path from the root node of the ROBDD to the 1-node, for each element of D there exists such a path. The ROBDD corresponding to a variable v with domain D will be denoted as $R(v)$ or $R(D)$ in the following.

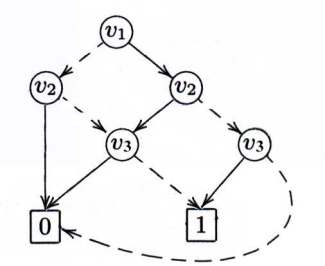

Fig. 1. A ROBDD representing the set $\{\{1,2\},\{1,3\},\emptyset\}$

3.3 ROBDD Representation of Constraints

ROBDDs can not only be used as a representation for the variable domains, but also as a representation for the constraints. Since constraints are just subsets of the domains they can be represented in the same way. The only difference between constraints and variable domains is that domains range over one variable while constraints (in most cases) range over several variables. The ROBDDs representing constraints thus have nodes labeled with the boolean variables of all set variables that they constrain. Each path from the root node to the 1-node represents a variable assignement that satisfies the constraint, each path from the root to the 0-node corresponds to a variable assignment which does not satisfy it.

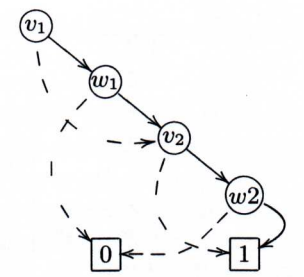

Fig. 2. A ROBDD representing the constraint $v \subseteq w$ for variables v, w ranging over $\{1,2\}$.

A ROBDD for the constraint $v \subseteq w$ where v and w range over $\{1, 2\}$, is shown in Figure 2. If we want to check, weather $v \subseteq w$ holds for $v = \{1, 2\}$ and $w = \{1\}$ we follow the corresponding path from the root and land at the 0 node. The constraint is thus not satisfied for this assignment. A ROBDD representing a constraint c will be denoted R(c), overloading this operator for both variables and constraints.

4 Propagators for ROBDD Representations

4.1 A Domain Propagator for ROBDD Representations

A propagator that works on variable domains and constraints in ROBDD representation has to somehow compute with these structures. There are efficient algorithms to construct new ROBDDs out of existing ones that can be used for this purpose. In particular one can easily build ROBDDs that represent the conjunction ($r_1 \wedge r_2$), disjunction ($r_1 \vee r_2$) or negation ($\neg r_1$) of ROBDDs r_1,r_2 and r, where the semantic of these operations is exacly the same as with their corresponding boolean formulas. Further more one can also existentially quantify over a boolean variables of a ROBDD ($\exists_v.r$). The semantic of $r_1 = \exists_v.r_2$ is that the variable v does not occur in r_1 anymore, and each assignment of the remaining variables of r_2 which can be extended to a solution of r_2 by an assignment of v is a solution of r_1. The existential quantification over a set of variables will be denoted as $\exists_V.r$ where V is a set of variables. This is well defined as $\exists_{\{v_1,...,v_n\}}.r = \exists_{v_1}. \ ... \ \exists_{v_n}.r$ since the order of existential quantifications does not matter.

For a ROBDD $R(c)$ of a constraint c one can use the above mentioned operations to compute $r = R(c) \wedge (\wedge_{v \in V(c))} R(v)$. The ROBDD r now contains the boolean variables corresponding to all set variables that c constraints. The solutions of r are exactly the variable assignments that are licensed by the repsective domains of the variables and also satisfy the constraint c, or in other words: The solutions of r are exactly the solutions of the constraint c under the current variable domains. If we now choose one set variable v and existentially quantify over all variables of r but $Vars(v)$, we get a ROBDD r' that contains only the boolean variables $Vars(v)$. A variable assignement for $Vars(v)$ is then a solution of r' iff there exists a solution of the constraint where v has this value. Thus r' represents a narrowed domain of v which satisfies all requirements of domain consistency. A domain propagator for a constraint c and a variable v is hence defined by the following formula: $dom(c)(v) = \exists_{Vars(V(c)\setminus\{v\})}.R(c) \wedge (\wedge_{v \in V(c))} R(v)$

4.2 Space Consumption of ROBDDs

While a strong (domain consistent) propagation can be guaranteed for set constraints in ROBDD representation, the size of them is claimed to be small in practical applications, but there is no guarantee for that. Since ROBDDs for variable domains are no approximations but have a one to one correspondence

to their domains, they may have exponential size (in the number of integers over which they range), since there are 2^n different possible domains ranging over integers $1, ..., n$.

The big difference to naive representations with exponential size is that ROBDD's for the initial domains have a small (even constant) size. Initial domains, which contain all values of $2^{\{1,...,n\}}$ have a corresponding ROBDD which consist of just one node, the 1-node. The exponential size of ROBDDs of variable domains thus may only appear after complex propagation operations.

Since constraints are also represented as ROBDDs, one also needs to answer the question, whether the size of these ROBDDs lies within desirable bounds. The good news here is that ROBDD representation for most basic constraints have linear or even constant size, only constraints involving the cardinality of a set have quadratic size. The sizes of ROBDDs for the basic constraints are shown in detail in the following table:

Constraint c	size of $R(c)$
$k \in v$	$O(1)$
$k \notin v$	$O(1)$
$v = w$	$O(n)$
$v \subseteq w$	$O(n)$
$u = v \cup w$	$O(n)$
$u = v \cap w$	$O(n)$
$u = v - w$	$O(n)$
$v = \overline{w}$	$O(n)$
$v \neq w$	$O(n)$
$\|v\| = k$	$O(k(n-k))$
$\|v\| \leq k$	$O(k(n-k))$
$\|v\| \geq= k$	$O(k(n-k))$

Summarizing the space issue one can say that as in every exact representation there is only an exponential space bound for ROBDDs, but at least the problem specifications (consisting of initial variable domains and constraints) are at most quadratic if only basic constraints are used.

4.3 ROBDDs and Bounds Representation

If one aims to develop a system with better than exponential bounds on the sizes of the domain representations, there is no other way than to approximate the real domains in the representation: Since there are exponentially many (2^n) possible domains ranging over $\{1, ..., n\}$, a representation must have exponential size if no two domains are represented by the same structure (and hence if it is exact). Approximated representations map different domains to the same representation, bounds representation for example maps all domains which have the same convex closure to the same structural representation.

The idea of bounds representation can also be realized with ROBDD representation by representing domains $D_1, ..., D_k$ with the same convex closure all with the same ROBDD. The ROBDD that is chosen to represent them all is the smallest ROBDD of $R(D_1), ..., R(D_k)$ and that is always the ROBDD of the domain D_j with $D_j = conv(D_j)$. Such domains are called convex and the corresponding ROBDDs have linear size. Further more the ROBDD of the convex closure of a domain can be calculated in linear time.

Systems which use bounds representation can't use the same propagators as systems with the exact ROBDD representation. Surely domain consistent propagators can take the convex closures of ROBDDs as input, but this has two consequences: First the result is no more domain consistent and second the resulting ROBDDs are not necessarily ROBDDs of convex domains. The second consequence does not harm since one can easily (in linear time) compute the corresponding convex closure after running the propagator to get back to the approximated representation. The first consequence comes with every approximated representation, but happily the propagator is at least bounds consistent as one can easily verify looking at the definition of bounds consistency, which involves a domain consistent propagator in the same way as our construction.

4.4 Split Domain Representation

Yet another representation off set variable domains which is based on ROBDDs is called split domain representation. The idea behind it is basically to combine the small size of the bounds representation with the benefits of exact representations. A domain is no more represented by just one exact ROBDD r but by a pair (r_1, r_2) of two ROBDDs. r_1 is exactly the approximated ROBDD of the bounds representation and r_2 is the remainder which makes the representation exact again in the sense that the exact ROBDD r can be obtained as $r = r_1 \wedge r_2$. r_2 again may have exponential size in worst case, the motivation for this splitting is that the size of (r_1, r_2) is smaller than r in many cases. A very encouraging result that supports this hope is that the size of (r_1, r_2) is never bigger that the size of r, i.e. splitting never makes the situation worse. There is also evidence that the split representation is not that sensitive to a bad variable ordering: While reordering the variables may cause a large blow-up of the size of r, this is not the case for (r_1, r_2).

Domain propagators working on split domain representations are a little bit more complex, since they have to transform two ROBDDs into two new ones. However in practical applications, this does not seem to lead to an efficiency loss.

5 Modularity as Further Benefits

Another advantage of the ROBDD representation results from the fact that constraints are not only implicitly defined by the implementation of a corresponding propagator but are given as explicit (ROBDD-) structures. Operations on these

structures allow to build new constraints reusing existing ones. As mentioned before, conjunction, disjunction, negation and existential quantification are operations that can efficiently be computed on ROBDDs. Hence also constraints represented by ROBDDs can be defined as the conjunction, disjunction, negation or existential quantification of other constraints. This induces an interface to specify new constraints that hides all the implementation details to the user. The user only specifies the semantic of a new constraint as a logic formula, the system constructs a domain consistent propagator that implements this semantic.

While the conjunction of two constraints c_1 and c_2 can also be realized in other set constraint systems by just running the propagators for both c_1 and c_2, there is no obvious way to model disjunction and negation of constraints, if c_1 and c_2 are only implicitly defined by their propagators. Further more the conjunction modelled as parallel application of several domain consistent constraint propagators is no more domain consistent in general. Existential quantification of constraints can also be modelled by systems without ROBDD representation, but again the ROBDD realization leads to better efficiency: While the standard way to existentially quantify over a variable v is to introduce v as an intermediate variable that is no further constrained outside the constraint, ROBDD existential quantification makes use of this locality of v and just compiles it away. Hence v does not have to be represented explicitly in the memory and no distribution over v is necessary during search.

An easy constraint which shows most aspects of the above mentioned modularity is the constraint $|v \cap w| \leq 1$. It can be definded by the formula $\exists_u . u = v \cap w \wedge |u| \leq 1$. The ROBDD induced by this formula has variables $Vars(v) \cup Vars(w)$, the variable u is compiled away and is no more visible in the resulting ROBDD. The size of the ROBDD for the constraint $|v \cap w| \leq 1$ is quadratic, since the constraints for $v \cap w$ and $|u| \leq 1$ have linear size, their conjunction at most multiplies their sizes and the existential quantification never increases the size. The same constraint realised in the usual way by introducing a variable u and posting two constraints $u = v \cap w$ and $|u| \leq 1$ has a weaker propagation and the search for a solution is more complex, since a concrete assignment for u must be determined for each solution.

Also global constraints with strong propagation can be build using this technique. Global constaints are large conjunctions of similar basic constraints. The constraint $disjoint(v_1, ..., v_n)$ is an example for such a global constraint, its semantic is that the sets $v_1, ..., v_n$ are disjoint. This can be expressed as the fact that $|v_i \cap v_j| = 0$ for each pair $v_i, v_j \in \{v_1, ..., v_n\}$. The constraint $|v_i \cap v_j| = 0$ can be defined analogue to the constraint $|v_i \cap v_j| \leq 1$ described in the previous paragraph, the conjunction $\wedge_{i,j \in \{1,...,n\}, i \neq j} |v_i \cap v_j| \leq 0$ defines the global constraint. For global constraints the fact that the constraint propagator for the conjunction of constraints is stronger than setting up constraint propagators for each part of the conjunction is extremely important. Global constraints can always be simulated by a series of elementary constraints, but due to the much

stronger propagation one should always prefer global constraints where they are available.

6 Experimental Results

A set domain solver using ROBDD representation has been implemented by Lagoon and Stuckey[6]. This solver can be configured such that is uses either the normal exact ROBDD domain representation, the ROBDD bounds representation or the split domain representation. The advantages and disadvantages of these representations and the corresponding propagations can thus be easily compared. Additional effects caused by different implementations or different programming languages are minimized in that way. The solver was also compared to the ECL^iPS^e solver as an example for a set constraint solver using set bounds representation. However the interpretation of this comparison it a little bit difficult, since differences in the speed of the solving process are not only caused by the different representations that the solvers use but also by their implementation. There are set bounds solver which are faster than ECL^iPS^e, so outperforming the ECL^iPS^e system does not directly mean to outperform all solvers based on bounds representation.

6.1 The Social Golfers Problem

The social golfers problem is a common set constraint benchmark. The task is to group $N = g \times s$ players in g groups of size s for each of w weeks such that no two players are in the same group more than once. The problem is easy to encode using a global constraint $partition(v_1, ..., v_n)$ which says that the sets $v_1, ..., v_n$ are a partition of $\{1, ..., n\}$. In the solver by Lagoon and Stuckey this constraint can easily be defined, in ECL^iPS^e it has to be simulated using several basic constraints. The time needed to solve different problem instances are shown in Figure 3. Also the number of fails during search and the memory consumption are shown there. The lather is not available for ECL^iPS^e and the sizes shown for the solver by Lagoon and Stuckey are only crude estimates, since the work of a garbage collector influences the shown values and may lead to an over estimation.

Comparing the bounds representation and the exact domain representation one can see that the guaranteed space bounds of bounds representation are clearly outperformed by the strong propagation of the domain consistent variant. The memory needed to solve a problem instance in bounds representation is in most cases much higher than in the exact representation. This a little surprising fact is caused by the much weaker propagation and the huge search space that is caused by it. The search space seems to be the dominating factor in the memory consumption of the whole process. Traversing the search space also needs so much time, that the domain consistent solver is clearly faster in most cases. The few cases (e.g. the 5-8-3 instance) were the bounds solver is faster can be explained in the way that in these cases a solution was by accident found very

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					

Fig. 3. Results of the social golfers benchmark

fast in the search space. Large instances that are still feasible for the domain consistent solver lead to a run out of memory for the bounds solver.

The ECL^iPS^e system behaves in most cases even worse than the bounds solver using ROBDD implementation. The reason (in addition to the fact that they are implemented in different programming languages) for this is that the bounds solver can benefit from the global partition constraint. The stronger propagation that it offers result in faster solving and a smaller search tree.

The improvements that should be obtained by the split domain representation seem to have almost no effect in this benchmark. The split domain propagator behaves almost equal to the domain propagator that uses one ROBDD for exact representation. There are even instances (like 5-4-3) where the standard domain consistent solver is faster than the one using split domain representation. However on average the split domain solver is a little bit faster.

The advantage of the strong propagation of both domain consistent solvers is best visible for the largest instance (7-5-5). The bounds consistent solvers search in a more and more growing search tree without finding a solution until they run out of space. The domain consistent propagators narrow down the initial domains already in the beginning such that the system recognizes that there is no solution without doing any search at all.

7 Conclusion

We have given an overview over set constraints and in particular we made the difference between domain propagation and bounds propagation precise. We then defined ROBDDs and demonstrated how to represent set variable domains and constraints using these structures. We also showed how to build a domain constistent propagator for this representation and gave two alternative variable domain representations resulting in ROBDDs with smaller size. We then mentioned further benefits emerging from the explicit representation of constraints as ROBDDs. Finaly we presented some experimental results which give evidence that the theoretical benefits also pay off in practice.

Summarizing one can say that ROBDDs are an efficient structure to represent the domains of set variables. Their worst case exponential size is in real applications more than compensated by the small search trees that result from the strong propagation. Further (minor) improvements can be made by using the split domain representation. In addition to the efficiency resulting from ROBDD representations, with the existence of boolean operations on ROBDDs there comes a canonical interface to specify a lot of efficient constraints, for example many global constraints. This interface abstracts away all implementation details while guaranteeing a maximal degree of consistency for a strong propagation.

A topic for future work could be the investigation of new domain approximations between exact and bounds representation. The goal should be to have as much propagation power as possible with a representation of still handable size. The realization of some more complex set constraints in ROBDD representation, for example selection constraints [4], is another open issue.

References

1. Sheldon B. Akers. Binary Decision Diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, June 1978.
2. Krzysztof R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.
3. Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318, 1992.
4. Denys Duchier. Configuration of labeled trees under lexicalized constraints and principles. To appear in the Journal of Language and Computation, December 2000.
5. Steven Fortune, John E. Hopcroft, and Erik Meineche Schmidt. The complexity of equivalence and containment for free single variable program schemes. In Giorgio Ausiello and Corrado Böhm, editors, *ICALP*, volume 62 of *Lecture Notes in Computer Science*, pages 227–240. Springer, 1978.
6. Peter Hawkins, Vitaly Lagoon, and Peter J. Stuckey. Set bounds and (split) set domain propagation using robdds. In Geoffrey I. Webb and Xinghuo Yu, editors, *Australian Conference on Artificial Intelligence*, volume 3339 of *Lecture Notes in Computer Science*, pages 706–717. Springer, 2004.
7. Vitaly Lagoon and Peter J. Stuckey. Set domain propagation using robdds. In Wallace [10], pages 347–361.
8. C.Y. Lee. Representation of Switching Circuits by Binary-Decision Programs. *Bell Systems Technical Journal*, 38:985–999, July 1959.
9. Andrew Sadler and Carmen Gervet. Hybrid set domains to strengthen constraint propagation and reduce symmetries. In Wallace [10], pages 604–618.
10. Mark Wallace, editor. *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*. Springer, 2004.