

Encoding CSPs with Zero-suppressed Decision Diagrams

Esben Rune Hansen
IT University of Copenhagen
esben@itu.dk

Abstract

A well established technology in the knowledge compilation community is the encoding of constraint satisfaction problems (CSPs) by binary decision diagrams (BDDs). A technology that among other things has found its use interactive configuration and other kinds of decision support.

The main contribution of this paper is the observation that Zero-suppressed binary decision diagrams (ZBDDs) that originally was intended for set-manipulation are very well suited to represent CSPs efficiently. As ZBDDs are already efficiently supported by the popular decision diagram package Cudd [17], no extra implementation is needed in order to use ZBDDs instead of BDDs.

Using the real-world CSP instances from CLib [5], we empirically demonstrate that using ZBDDs instead of BDDs often result in significantly smaller representations. We also show that this trend holds as well during the compilation of the decision diagrams, which can make the difference between whether or not a CSP can be compiled, given the amount of RAM available.

1 Introduction

Representing CSPs by BDDs was proposed in [8] and a tool called CLAB, that constructs a BDD encoding of a CSP specified by the user, is publicly available at [11]. The main use for this technology is found in decision support.

A main issue in encoding a CSP into a BDD is that a CSP is containing integer variables whereas a BDD only contains binary variables. In order to represent a CSP by a BDD the CSP has to be transformed into a constraint problem on binary variables, that is a SAT problem. Two transformations are used [21]: *log-encoding* and *direct encoding*. In log-encoding each integer value corresponds to the bit-string of the value. In direct encoding each integer value k corresponds to a bit-string where all but the k th bit is zero. As an example 5 is 101 in log-encoding and 00000100 given that the domain of the variable is $0, \dots, 7$.

In this paper will compare the structure of Binary

Decision Diagrams (BDDs) with the structure of Zero-suppressed Binary Decision Diagrams (ZBDDs) [10], when they are used to represent CSPs. We will both consider direct encoded and log-encoded CSPs.

2 Related Work

Various techniques has been proposed to reduce the size of decision diagrams and/or the time used to compile them. Choosing a good variable ordering is crucial to minimize the size of the decision diagram and various heuristics to optimize the variable ordering has been proposed [3]. The problem of finding an optimal variable ordering is orthogonal to the problem considered in this paper.

Various alternative decision diagrams has been proposed in order to reduce the size of the representation. Among others, Constraint Decision Diagrams [4], Difference Decision Diagrams [16] and AND/OR Decision Diagrams [13]. Since Zero-suppression can be used on any binary decision diagram the results in this paper can be combined with any of the representations above.

Another way to reduce the size of a decision diagram is by decomposition as proposed in [18] [19] or as proposed in [9]. These decomposition techniques apply to binary decision diagrams and therefore also to ZBDDs. Hence we might suspect that by a combining decomposition and ZBDDs could achieve even further savings than the ones achieved by decomposed BDDs.

Compiling CSPs into DFAs for the use of decision support was proposed in [20] and improved in [1]. A very similar technique is to compile the CSP into an MDD [12]. The aim of this paper is not to compare MDD/DFAs with the binary representations. Both techniques are used. Usually binary representations are used when complex operations like conjunctions and disjunctions has to be performed as for instance during the compilation of a CSP into a decision diagram, whereas the MDD/DFA approach usually are used when fast traversal of the decision diagram is important. During the compilation of a CSP, the MDD/DFAs approach are significantly slower than the BDD approach [14][15].

3 Preliminaries

We Consider a CSP: $CSP = (X, \mathcal{D}, C)$, where $X = \{x_1, \dots, x_N\}$ is the set of variables, C the set of constraints and $\mathcal{D} = \{D_1, \dots, D_N\}$ is the multi-set of variable domains, such that the domain of a variable x_i is D_i . For simplicity we assume for every $1 \leq i \leq N$ that $D_i = \{0, \dots, |D_i| - 1\}$. A CSP where all are $\{0, 1\}$ is denoted a *binary CSP* (aka SAT-problem).

A *single assignment* is a pair (x_i, a) where $x_i \in X$ and $a \in D_i$. The assignment (x_i, a) is said to have support, iff there exists a solution to CSP where x_i is assigned a . If a single assignment (x_i, a) , where $a \in D_i$, has support, a is said to be in the valid domain for x_i . A *partial assignment* ρ is a set of single assignments to distinct variables, and a *complete assignment* is an assignment that assigns all variables in X .

Definition 1 (Ordered Binary Decision Diagram). *An ordered decision diagram on n binary variables $B = \{x_1^{bin}, \dots, x_n^{bin}\}$ is a layered directed acyclic graph $G(V, E)$ with $n + 1$ layers (some of which may be empty) and exactly one root. We use $var(u)$ to denote the layer in which the node u resides. In addition the following properties must be satisfied:*

- *There are exactly two nodes in layer $n+1$. These nodes have no outgoing edges and are denoted the 1-terminal and the 0-terminal*
- *All nodes in layer 1 to n have exactly two outgoing edges, denoted the low and high edge respectively. We use $low(u)$ and $high(u)$ to denote the end-point of the low and high edge of u respectively.*
- *For any edge $(u, v) \in E$ it is the case that $var(u) < var(v)$*

We use E_{low} and E_{high} to denote the set of low and high edges respectively. An edge (u, v) such that $var(u) + 1 < var(v)$ is called a *long edge* and is said to skip layer $var(u) + 1$ to $var(v) - 1$.

Definition 2 (ROBDD [2]). *An OBDD is called reduced iff it holds that:*

1. $var(u) \neq var(v) \vee low(u) \neq low(v) \vee high(u) \neq high(v)$ for any two distinct nodes u, v (merging)
2. $high(u) \neq low(u)$ for all nodes u . (skipping)

If a variable x_i^{bin} is skipped by a path from root to 1-terminal in a ROBDD it means that x_i^{bin} can have any value in the domain that is x_i^{bin} can have all values in the domain of x_i^{bin} .

Definition 3 (ZBDD [10]). *An OBDD is called Zero-Suppressed iff it holds that:*

1. $var(u) \neq var(v) \vee low(u) \neq low(v) \vee high(u) \neq high(v)$ for any two distinct nodes u, v (merging)
2. $high(u) \neq false$ for all nodes u . (skipping)

If a variable x_i^{bin} skipped by a path from root to 1-terminal in a ZBDD it means that $x_i^{bin} = false$.

Note that ROBDDs and ZBDDs only differ in the reduction rule that are used to skip variable layers. The reduction rule used to merge isomorphic sub-tree are the same for both decision diagrams.

Definition 4 (Solution – ROBDD). *A complete assignment ρ_{bin} to X_{bin} is a solution to a binary CSP encoded by a OBDD $G(V, E)$ iff there exists a path P from the root in G to the 1-terminal such that for every assignment $(x_i^{bin}, b) \in \rho$, where $b \in \{low, high\}$, there exists an edge (u, v) in P such that one of the following holds:*

- $var(u) < i < var(v)$
- $var(u) = i$ and $(u, v) \in E_b$

Definition 5 (Solution – ZBDD). *A complete assignment ρ_{bin} to X_{bin} is a solution to a binary CSP encoded by a ZBDD $G(V, E)$ iff there exists a path P from the root in G to the 1-terminal such that, for any $(x_i^{bin}, true) \in \rho_{bin}$ and for no $(x_i^{bin}, false) \in \rho_{bin}$, there exists an edge (u, v) in P for which:*

- $var(u) = i$
- $(u, v) \in E_{true}$

In the rest of this paper the term *binary decision diagram* (BDD) will denote the decision diagram introduced in Definition 2 and the term *zero-suppressed binary decision diagram* (ZBDD) will denote the decision diagram introduced in Definition 3. By (Z)BDD we denote a decision diagram that is either a BDD or a ZBDD.

3.1 Encoding a CSP by binary decision diagrams

In order to represent an integer problem by a binary decision diagram one has to fix a bijective mapping between assignments to a set of Boolean variables and assignments to the integer variables of the problem. When this mapping have been found the constraints on the integer variables are mapped to corresponding binary constraints, and solutions on the binary constraints is mapped to integer solutions. Two ways have been used: *log-encoding* and *direct-encoding*.

In *Log-encoding* every assignment (x_i, a) is mapped to the binary representation of a , thereby using $\lceil \log |D_i| \rceil$ binary variables to model each integer variable x_i . If a domain of some variable x_i is not a power of 2 some assignment to

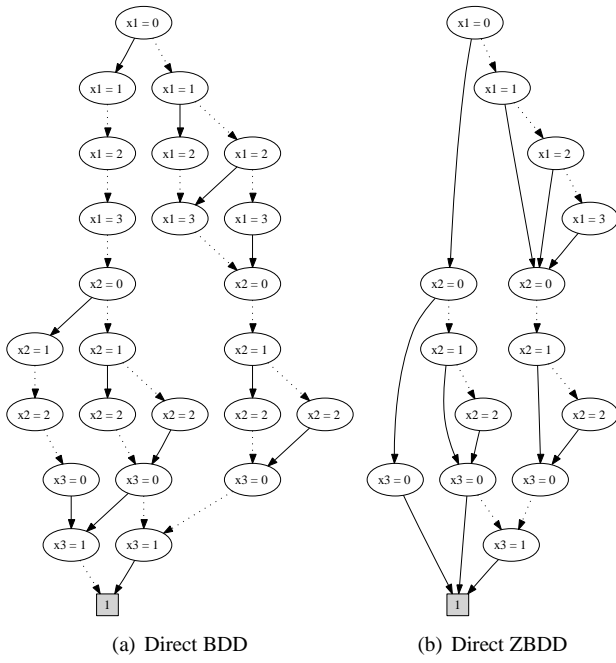


Figure 2. Direct Encoding of the CSP from Example 1. The dotted edges are low-edges and the solid edges are high-edges. The nodes in the figure are labeled with the CSP assignments to which they correspond. Hence a node labeled $x_k = a$ in the figure corresponds to the Boolean variable $x_{x_k=a}^{bin}$. For simplicity, edges going to the 0-terminal are not included in the figure.

Solution to an MDD A complete assignment ρ of values on a set integer variables $X = \{x_1, \dots, x_N\}$ is a solution to an MDD $G(V, E)$ iff there exists a path P from the root in G to the 1-terminal such that for every assignment $(x_i, a) \in \rho$, where $a \in D_i$, there exists an edge (u, v) in P such that one of the following holds:

- $var(u) < i < var(v)$
- $var(u) = i$ and $(u, v) \in E_a$

Encoding a CSP by an MDD Encoding a CSP by an MDD is done by making a one-to-one correspondence between the layers in the MDD and the variables in the CSP, that is each node in layer k corresponds to the CSP variable x_k . Further every node at layer $i \leq N$ has $|D_i|$ outgoing edges where each edge is labeled by a distinct value in D_i . Figure 4(a) shows an MDD that encodes the CSP from Example 1.

4 Direct (Z)BDDs

In this section we will consider the difference in size between a direct-BDD and a direct-ZBDD. We will show that a direct-BDD never is smaller than a direct-ZBDD and indicate why a direct-ZBDD usually is orders of magnitude smaller than a direct-BDD. We will also indicate that a direct-BDD is larger than a log-encoded (Z)BDD by the following observation:

Observation 1. *The direct-BDD encoding of a CSP with a single variable can be exponentially larger than the corresponding log-BDD or log-ZBDD.*

Proof. Consider a CSP with a single variable with a domain D_1 that can only be assigned to one value. In this case a direct-BDD will contain $|D_1|$ nodes where as both the log-BDD and the log-ZBDD will contain $\lceil \log |D_1| \rceil$ nodes. \square

In practical instances the domains are small, which keeps the size of the direct-BDD to be far from exponential in the size of the corresponding log-BDD. In most cases the direct-BDD is 2-4 times larger than the log-BDD, as we will see in the experimental section.

Lemma 1. *A direct-BDD encoding of a CSP will have no long edges except for edges with end-point in the 0-terminal*

Proof. If a path ends in the 1-terminal Equation (1) from Section 3.1 has to be satisfied. In order to ensure that Equation (1) is satisfied in a direct-BDD, we have to check all variables, hence all paths from the root to the 1-terminal in the direct-BDD contain no long edges.

We still have to proof that paths which ends at the 0-terminal contains no long edges except for the edges with end-point in the 0-terminal. If a path p starts at the root and ends in the 0-terminal there will be exactly one node u in p that both has:

- an outgoing edge (u, v) , that is contained in some path from the root to the 1-terminal
- an outgoing edge (u, v') , that is only contained in path ending at the 0-terminal

Since u is contained in a path from the root to the 1-terminal the sub path of p from the root to u contains no long edges, as we showed in the first half of this proof. Further, since all paths that contains (u, v') ends at the 0-terminal, (u, v') will have its end-point at the 0-terminal because of the node elimination rule. \square

Lemma 2. *The number of nodes in a direct-ZBDD encoding of a CSP can never be larger than the number of nodes in a corresponding direct-BDD.*

Proof. Follows from Lemma 1 and the fact that all edges in a ZBDD that are not part of any path from the root to the 1-terminal will have endpoints in the zero-terminal. Hence, in representation of direct encoded CSPs, all nodes that will be eliminated by the BDD reduction rule will also be eliminated by the ZBDD reduction rule. \square

The nodes of direct-BDD can be divided into three different types:

- I Nodes that represent a decision based on whether the value a has been assigned to the variable x_i in the CSP
- II Nodes that represent that the assignment of the value a to x_i violates the CSP
- III Nodes that represent that since one value has already been assigned to x_i by an edge above the current node no other values can be assigned to x_i

If a ZBDD is used instead of a BDD, the ZBDD is contain exactly the nodes of type I. All other nodes will be eliminated.

Example 2. The direct-BDD in Figure 2(a) consist of 23 nodes: 14 nodes of type I, 7 nodes of type II and 2 nodes of type III. Therefore the ZBDD in Figure 2(b) consist of 14 nodes. These nodes have the same labels as the 14 nodes of type I in the direct-BDD in Figure 2(a).

The fraction of the nodes in a direct-BDD that is of type II depends on the tightness of the CSP. The fraction of the nodes of type III depends on the size of the domains.

Most real life CSP instances have somewhat larger domains and are much tighter than the CSP we have considered in Example 2. In the experimental section the instances encoded by a direct-BDD are 3-14 times larger than when encoded by a direct-ZBDD.

Direct-ZBDDs do not perform well on very loose constraints. This can be stated by the following observation:

Observation 2. The direct-ZBDD encoding of a CSP with a single variable can be exponentially larger than the corresponding log-BDD or log-ZBDD.

Proof. Consider a CSP with a single variable x_1 that can be assigned to any value in D_1 . A direct-ZBDD will use $|D_1|$ nodes to encode this. A log-BDD and a log-ZBDD will at most use $\lceil \log |D_1| \rceil$ nodes. \square

Example 3. Figure 3 highlights how the tightness of a constraint influence the size of the decision diagrams for the different binary representations of the CSP.

Each x - coordinate corresponds to a CSP on a single variable with the domain $0, \dots, 38$. For any k the CSP

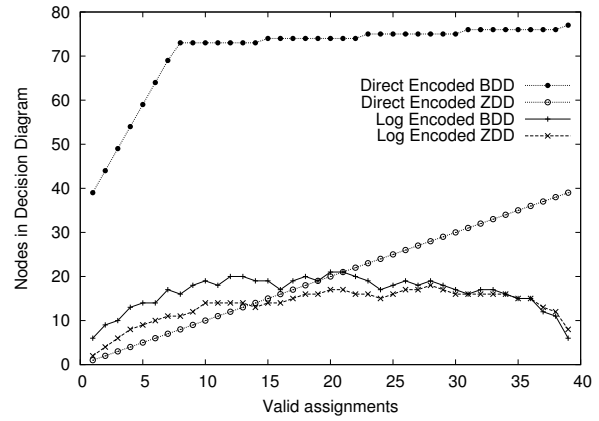


Figure 3.

depicted at the x -coordinate k has the solutions:

$$\bigcup_{i=1}^k \{5 \cdot i \% 39\}$$

That is $\{5\}$ if $k = 1$, $\{5, 10\}$ if $k = 2, \dots$, $\{5, 10, 15, 20, 25, 30, 35, 1, 6\}$ if $k = 9$ and so on, hence every CSP depicted at the x -coordinate k has exactly k solutions.

The solution sets in the example are chosen to minimize the consecutive integers in the solution and hence minimize the effect of consecutiveness in the solution set from the current consideration.

Example 3 highlights the following trends in the sizes of the encoding of a variable block:

- The direct-BDDs are significantly larger than the other representations
- The direct-ZBDDs are the smallest representation for tight constraints but are larger than log-encoded decision diagrams for loose constraints.
- The two log-encodings have similar curves but for tight constraints the log-ZBDD is smaller than the log-BDD.
- The number of nodes in Direct ZBDDs equals the number of valid assignments. We note that is exactly the same number of edges that would be used by an MDD to encode the same constraint.

5 The similarities between MDDs and direct-ZBDDs

Example 3 seemed to suggest some similarities between MDDs and direct-ZBDD. The similarity can be summarized in the following observation

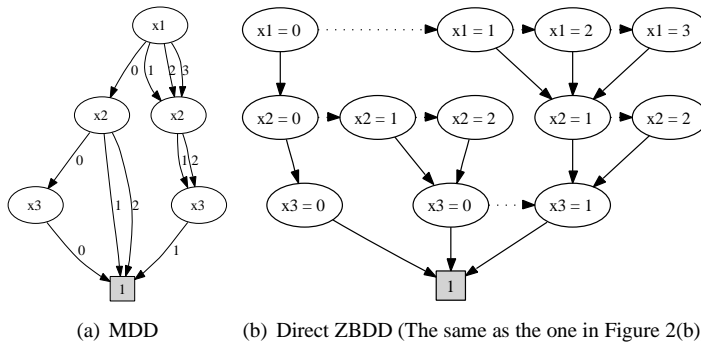


Figure 4. MDD and direct-ZBDD encoding of the CSP from Example 1

Observation 3. Consider the encoding of the same CSP by:

- An MDD where node-elimination rule has not been performed
- A direct-ZBDD where the merging rule has only been performed on the first node in each variable block

Then there is a one-to-one correspondence between the edges in MDD and the high-edges in the direct-ZBDD.

This fact can be realized by considering Figure 4. In Figure 4(b) we have divided the nodes of the ZBDD into 3 layers corresponding to the variable block in which they reside. Observe for the two first layer the number of nodes in the ZBDD and the number of edges that leaves the layer in the ZBDD and the number of edges that leaves the layer in the MDD is the same. For the last layer this is not the case because of two reasons:

- In the MDD one of the nodes has removed by node elimination decreasing the number of MDD edges leaving the third layer by 2.
- In in ZBDD two nodes labeled $x_3 = 1$ is merged into one decreasing the number of nodes in the third layer and the number of nodes leaving the third layer by 1.

Hence converting a ZBDD into and MDD and vice versa is a very simple operation.

6 Experiments

In this section will make an empirical comparison of the size of representing CSPs using log-BDDs, log-ZBDDs, direct-BDDs and direct-ZBDDs. We will both consider the sizes of the compiled representations and consider the size of the intermediate representations during compilation. The experimental section are considering the following instances:

Renault, Big-PC, 1-6+22-32, Pc2, and Bike2 are product configuration benchmarks from the CLib [5] library.

12×12 queens is the problem of placing 12 queens on a 12×12 chess board such that no queen can capture any other queen.

One Pair is a CSP on four variables each with the domain $\{1, \dots, 40\}$ and with the constraint that there has to exist exactly one pair of variables that has the same value-assignment.

5×27 queens is the problem of placing 5 queens on a 5×27 chess board such that no queen can capture any other queen.

Big-PC(53) and Big-PC(53)* are both derived from the constraint problem Big-PC by only including the 53rd constraint out of the 58 constraint that Big-PC consist of. In Big-PC(53) all the variables from Big-PC are included in the decision diagram. In Big-PC(53)* only the variables that occur in the 53rd constraint are included in the decision diagram.

6.1 The size of the compiled decision diagrams

Figure 6 shows the size of the representation of the constraint problems mentioned above using different decision diagrams. The sizes of the MDDs are only included to highlight the similarities between direct-ZBDDs and MDDs. Comparing multi-valued decision diagrams with their binary counterparts is outside the scope of this paper. For each instance the smallest binary decision diagram representation is marked with bold. We have also included the tightness of the instances which is defined as

$$\frac{\prod_{i=0}^n |D_i|}{|sol(CSP)|}$$

where $|sol(CSP)|$ is the number of solutions to the CSP.

For all compiled instances the direct-BDD is the largest representation. Whether the size of the direct-ZBDD is smaller than the log-ZBDD seems to depend on the tightness of the instance. Further the difference in size between log-ZBDD and the log-BDD also seem to depend on the tightness. For most of the configuration problems, which all are tight problems, both the log-ZBDDs and the direct-ZBDDs are around half the size of the log-BDD. In the tightest problem *Big-PC* the size of the direct-ZBDD is 25% of the size of log-BDD. We also note that the tight combinatorial problem *12×12 queens* achieve the same reductions in size by using ZBDDs instead of BDDs.

The two problems combinatorial problems *One Pair* and *5 × 27 queens* are chosen to test the ZBDD approach on very loose problems. On these instances the direct-ZBDD is larger than the log-ZBDD and the log-ZBDD is only marginally smaller than the log-BDD.

For the problem *Big-PC(53)* the log-BDD is the smallest representation, and the Direct-ZBDD is almost four times as large as the log-BDD. This constraint is the subject of Section 6.3.

6.2 The size during the compilation

In Figure 5 we have plotted the size of the multi-rooted decision diagram during the compilation of the Renault instance which is the largest configuration benchmark in CLib. The compilation has been done by compiling each individual constraint into a decision diagram and building the final decision diagram by repeatably replacing the two largest decision diagrams by the conjunction of the two.

Figure 5(a) shows that direct-ZBDDs and log-ZBDDs uses less nodes than the log-BDD encoding except from in the beginning where the sizes of all representations are small. Further the maximal number of nodes used during the compilation is significantly smaller. This number is quite important as this is what makes the difference between whether an instance can be compiled or not given the amount of available amount of memory available. Figure 5(b) shows that the direct-BDD performs far worse than the direct-ZBDD.

6.3 The representation of variables that do not occur in the constraint

Direct-ZBDDs perform remarkably well in most cases, especially on configuration instances. In the compilation of the individual constraints the size of the direct-ZBDD encoding is larger than the log-encoding. This is due to the fact that most of the individual constraints are on small scopes, and all variables that are not in the scope of the constraint can take any value. By Observation 2, constraint problems with variables that can take any value can be en-

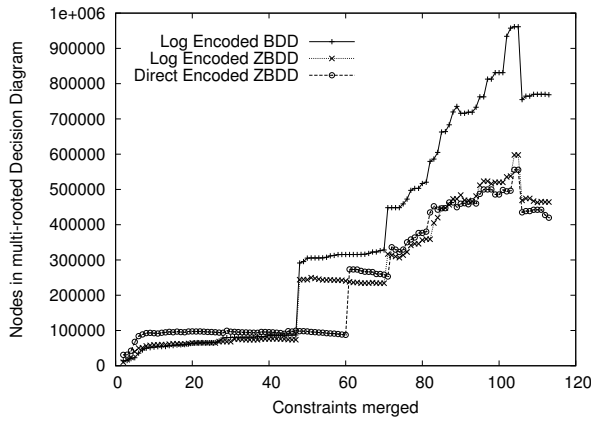
coded much more efficient by log-(Z)BDDs than by direct-ZBDDs. However, the representation of the individual constraints are usually very small and dominated by the size of decision diagrams that is created by conjoining the constraints during the compilation.

This is not always the case: The configuration problem *Big-PC* consists of 58 individual constraints where all but one can be represented by small decision diagrams. The one exception is the 53rd constraint:

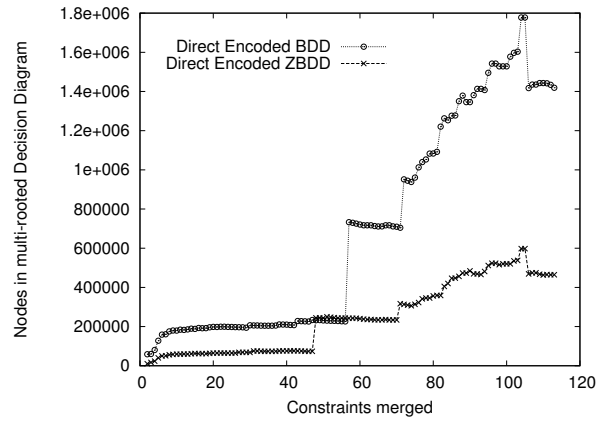
$$(x_1 = 3) \implies (x_{18} = x_{47}) \wedge (x_{23} \leq x_{46}) \wedge (x_{24} \geq x_{46})$$

where the variable ordering is x_1, \dots, x_{124} . Since the value x_{18}, x_{23} and x_{24} has to be compared against x_{46} and x_{47} , the assignment to x_{18}, x_{23} and x_{24} has to be stored by branching on the values assigned to the variables. This creates a 3511 sub-paths in the decision diagram from x_{25} to x_{45} where all combination of assignments in the domain of the variables are valid. The problem can be encoded very efficiently by MDDs since it will use a single long edge for each sub-path that skips the CSP variables x_{24}, \dots, x_{45} . Both log-ZBDDs and direct-ZBDDs will need a node for each binary variable that are used to encode these integer variables, in each of the 3511 sub-path. However, since log-encoding uses less binary variables to encode integer domains than direct encoding, the number of variables used to encode these variables are 228 when direct encoding is used and 65 when log-encoding is used. A log-BDD will be able to skip the layers encoding variables with domains that are a power of 2, and is therefore doing marginally better than log-ZBDD. To sum up log-BDDs are using 203638 nodes, log-ZBDDs use 238215 nodes, and direct-ZBDDs use 800508 nodes to encode these paths. Note that these sizes are close to the actual sizes of the encodings of *Big-PC(53)*.

A way to avoid the trivial and expensive encoding of the variables x_{25}, \dots, x_{45} from the decision diagram on *Big-PC(53)* is to encode a decision diagram where only the variables that occurs in the encoded constraint is included in the decision diagram. The result of such an encoding is the instance *Big-PC(53)**. Note that this reduces the size of the direct-ZBDD by a factor 100, making direct-ZBDDs the most efficient representation of the constraint. The size of the other binary decision diagrams also decreases dramatically. In order to implement this a approach using a ZBDD package one will need to implement some changes to the ZBDD package. This is needed since the ZBDD package will, for each decision diagram that it contains, need information about which binary variables that are included the decision diagram. However such a change might result in a significant speedup during the compilation for all kinds of binary decision diagrams. Especially when direct-ZBDDs are used.



(a) Compilation by log-BDD, log-ZBDD and direct-ZBDD



(b) Compilation by direct-BDD and direct-ZBDD

Figure 5. The compilation of the Renault instance.

Name	Log-encoding		Direct Encoding		MDD Encoding		Tightness
	BDD Nodes	ZBDD Nodes	BDD Nodes	ZBDD Nodes	Nodes	Edges	
Product Configuration							
Renault	768560	464376	1392861	419651	329134	426212	$7.02 \cdot 10^{47}$
Big-PC	356696	158309	1291598	88780	132595	100192	$1.12 \cdot 10^{80}$
1-6+22-32	20935	7792	61942	10668	8094	11193	$8.90 \cdot 10^{30}$
Pc2	13332	7411	43326	6131	3906	6136	$1.89 \cdot 10^{29}$
Bike2	2113	2083	11862	1668	938	1726	$4.98 \cdot 10^{14}$
Combinatorial Problems							
12×12queens	141753	65450	435170	45833	33549	47418	$6.27 \cdot 10^8$
One Pair	156845	149254	2384454	185360	37882	318720	10.52
5×27queens	562762	500047	4497110	1107635	215448	3677650	3.19
Single Constraints							
Big-PC(53)	213911	278403	1609820	812834	3797	36294	1.33
Big-PC(53)*	9985	36294	72837	7401	3797	36294	1.32

Figure 6. The size of some configuration benchmarks using different decision diagrams.

7 Conclusion

We have shown on numerous real-world instances that the representation that log-encoded ZBDDs are smaller than log-encoded BDDs and that direct encoded ZBDDs are much smaller than direct encoded BDDs, and that this especially is the case for tight constraints. We have further shown that a direct encoded ZBDDs has some structural similarities with a MDDs and that direct-ZBDDs in most tested real-world instances performs better than the log-BDD.

For compilation we have shown an example where both kinds ZBDDs perform far better than the BDD. We have further observed that in some case compiling by direct-ZBDDs causes a large intermediate representation. We have

illustrated this by a real-world example and proposed a technique that will resolve this problem and improve the efficiency of compilation in general.

References

- [1] J. Amilhastre, H. Fargier, and P. Marquis. Consistency restoration and explanations in dynamic CSPs-application to configuration. *Artificial Intelligence*, 2002.
- [2] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 1986.
- [3] R. E. Bryant. On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Transactions On Computers*, 40(2):205–213, 1991.

- [4] K. C. K. Cheng and R. H. C. Yap. Constrained decision diagrams. In M. M. Veloso and S. Kambhampati, editors, *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 366–371. AAAI Press / The MIT Press, 2005.
- [5] CLib. Configuration benchmarks library. Available online at: <http://www.itu.dk/research/cla/externals/clib/>, 2007.
- [6] T. Hadzic, E. R. Hansen, and B. OSullivan. On automata, mdds and bdds in constraint satisfaction. In *ECAI 2008 workshop on Inference Methods Based on Graphical Structures of Knowledge*, 2008.
- [7] T. Hadzic, R. M. Jensen, and H. R. Andersen. Calculating valid domains for bdd-based interactive configuration, 2007. [arXiv.org:0704.1394](http://arxiv.org/0704.1394) [cs.AI].
- [8] T. Hadzic, S. Subbarayan, R. M. Jensen, H. R. Andersen, J. Møller, and H. Hulgaard. Fast backtrack-free product configuration using a precompiled solution space representation. In *PETO Conference*, pages 131–138. DTU-tryk, June 2004.
- [9] E. R. Hansen and P. Tiedemann. Compressing configuration data for memory limited devices. In *AAAI-07*, page 15, 2007.
- [10] S. ichi Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In *DAC '93: Proceedings of the 30th international conference on Design automation*, pages 272–277, New York, NY, USA, 1993. ACM.
- [11] R. M. Jensen. CLab: A C++ library for fast backtrack-free interactive product configuration. <http://www.itu.dk/people/rmj/clab/>, 2007.
- [12] T. Kam, T. Villa, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Multi-valued decision diagrams: Theory and applications. *International Journal on Multiple-Valued Logic*, 4:9–62, 1998.
- [13] R. Mateescu and R. Dechter. Compiling constraint networks into AND/OR multi-valued decision diagrams (AOMDDs). In F. Benhamou, editor, *CP*, volume 4204 of *Lecture Notes in Computer Science*, pages 329–343. Springer, 2006.
- [14] D. Miller and R. Drechsler. Implementing a multiple-valued decision diagram package. In *ISMVL '98: Proceedings of the The 28th International Symposium on Multiple-Valued Logic*, page 52, Washington, DC, USA, 1998. IEEE Computer Society.
- [15] D. M. Miller and R. Drechsler. On the construction of multiple-valued decision diagrams. *ismvl*, 00:245, 2002.
- [16] J. B. Møller, H. Hulgaard, and H. R. Andersen. Symbolic model checking of timed guarded commands using difference decision diagrams. *Journal of Logic and Algebraic Programming*, 52-53(1-2):53–77, 2002.
- [17] F. Somenzi. CUDD: Colorado university decision diagram package. <ftp://vlsi.colorado.edu/pub/>, 1996.
- [18] S. Subbarayan. Integrating CSP decomposition techniques and BDDs for compiling configuration problems. In R. Barták and M. Milano, editors, *Proceedings of the Second International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR 2005*, pages 351–365, 2005.
- [19] S. Subbarayan. An empirical comparison of csp decomposition methods. In *Proceedings of the Doctoral Program, CP 2007*, 2007.
- [20] N. R. Vempaty. Solving constraint satisfaction problems using finite state automata. In *AAAI*, pages 453–458, 1992.
- [21] T. Walsh. Sat v csp. In *CP '02: Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming*, pages 441–456, London, UK, 2000. Springer-Verlag.