

On Automata, MDDs and BDDs in Constraint Satisfaction

Tarik Hadzic and Esben Rune Hansen and Barry O’Sullivan

Abstract. In this paper we analyze relationships between the variants of *deterministic finite-state automata* (DFAs), *multi-valued decision diagrams* (MDDs) and *binary decision diagrams* (BDDs) as currently used for compiling *constraint satisfaction problems* (CSPs). We highlight the limitations and benefits of using Boolean encodings and BDDs in comparison to their multi-valued counterparts: MDDs and DFAs. In particular, we show the close relationship between these structures when Boolean encoding of a CSP is using *clustered* variable ordering. We also note that differences between the variants of DFAs and MDDs used in the CSP literature are minor, and appear only due to removal of *redundant* nodes in MDDs. We experimentally compare these structures over a set of real-world and random instances.

1 Introduction

An ever increasing list of different compact representations for constraint satisfaction problems [Bry86, Vem92, Weg00, MMD07] makes it increasingly hard to identify the right one for a given task. While recently there was significant work on describing relationships between a number of these structures [DM02], there is still a lot to clarify.

In particular, a class of ordered acyclic directed graphs including *deterministic finite-state automata* (DFAs) [Vem92, AFM02], *multi-valued decision diagrams* (MDDs) [Weg00], and *binary decision diagrams* (BDDs) [MAH02] has often been used for decision support [MAH02, AFM02] and enhancing inference in CSP search [Pes04, CY06, AHHT07].

While DFAs and MDDs represent integer variables directly by labeling each edge with a CSP value, BDD representations allow only binary labeling of edges, therefore requiring a Boolean encoding of the CSP. This is usually done by using either *log-encoding* or *direct-encoding* [Wal00].

In this paper we highlight the limitations and benefits of using BDDs in comparison to their multi-valued counterparts: MDDs and DFAs. In particular, we show the close relationship between these structures when Boolean CSP variables are ordered in finite-domain *clusters*. We also note that differences between the variants of DFAs and MDDs used in the CSP literature are minor, and appear only due to removal of *redundant* nodes in MDDs. Our theoretical observations are supplemented by experimental comparison of these structures over a set of real-world and artificially constructed instances.

2 Background

In this section we briefly review the three representations as used in constraint satisfaction community. A *constraint satisfaction problem* $\mathcal{C}(X, D, C)$ is defined over variables $X = \{x_1, \dots, x_n\}$ with finite domains D_1, \dots, D_n . Each constraint $c \in C$ is defined over a subset of variables $\text{scope}(c) \subseteq X$, and defines the set of assignments to the variables in the $\text{scope}(c)$ allowed by the constraint. The solution space of $\text{Sol}(\mathcal{C})$ denotes the set of all solutions to the CSP \mathcal{C} . For illustration purposes we will consider a *T-Shirt configuration problem* in Example 1.

Example 1 (An Example CSP) Consider specifying a T-shirt by choosing the color (black, white, red, or blue), the size (small, medium, or large) and the print ("Men In Black" - MIB or "Save The Whales" - STW). There are two constraints that we have to observe: if we choose the MIB print then the color black has to be chosen as well, and if we choose the small size then the STW print cannot be selected. The CSP model of the T-shirt example consists of variables $X = \{x_1, x_2, x_3\}$ representing color, size and print. Variable domains are $D_1 = \{0, 1, 2, 3\}$ (standing for black, white, red, blue), $D_2 = \{0, 1, 2\}$ (small, medium, large), and $D_3 = \{0, 1\}$ (MIB, STW). The two constraints translate to $C = \{c_1, c_2\}$, where c_1 is $(x_3 = 0) \Rightarrow (x_1 = 0)$ and c_2 is $(x_3 = 1) \Rightarrow (x_2 \neq 0)$. \diamond

Both *finite-state automata* and various forms of *decision diagrams* have been used to represent CSP solutions, either for enhancing inference during search [Pes04, CY06] or for decision support [MAH02, AFM02].

Deterministic Finite Automaton (DFA) A *finite-state automaton* A is defined by the input alphabet Σ , set of states S , initial state s_0 and state-transition function $\sigma : S \times \Sigma \rightarrow S$. Some of states in S are *accepting states*. A sequence of symbols is accepted by A if it corresponds to a sequence of transitions from the initial state to an accepting state. Sequences of symbols accepted by A , define a language accepted by the automaton. We denote such language as $L(A)$. *Deterministic finite automaton* (DFA) additionally requires that for each state there is *at most one* transition for each input symbol.

For our purpose, DFA is represented by a rooted directed graph $G(V, E)$, where every state $s \in S$ corresponds to a node $u \in V$ (root $r \in V$ corresponds to initial state s_0). Every state-transition $(s, a) \rightarrow s'$ is represented by a labeled edge (u, a, u') .

While there could be many ways in which a DFA could encode solutions to a given CSP, considerations in this paper are limited only to a special construction used in [Vem92] and [AFM02]. The alphabet is limited to CSP domain values D_1, \dots, D_n . DFA is constructed such that for every satisfying assignment $\{(x_1, a_1), \dots, (x_n, a_n)\}$, there is a path starting from the root, ending in an accepting node, which consist of edges e_1, \dots, e_n where every edge e_i is labeled with $a_i \in D_i$. Hence, every word in the $L(A)$ is a sequence of n symbols (a_1, \dots, a_n) , that directly corresponds to a solution in Sol . Hence: $L(A) = Sol(C)$

Decision Diagrams A *decision diagram* is a rooted directed acyclic graph $G(V, E)$ where every node u is labeled with a variable x_i and every edge e , originating from a node labeled x_i , is labeled with a value $a \in D_i$. No node may have more than one outgoing edge with the same label. The decision diagram contains a special *terminal* node 1, that has no outgoing nodes. The terminal node has to be reachable by every other node in V .

A decision diagram represents a function $f : D_1 \times \dots \times D_n \rightarrow \{0, 1\}$, defined over variables $\{x_1, \dots, x_n\}$. A value of the function $f(a)$, $a = (a_1, \dots, a_n)$, for an assignment $\{(x_1, a_1), \dots, (x_n, a_n)\}$ is defined by traversing G from the root, and at every node u labeled with variable x_i , following an edge labeled with a_i . If there is no such edge $f(a) = 0$. Otherwise, if traversal ends in terminal 1, $f(a) = 1$.

If all domains D_i are binary, i.e. $D_1 = \dots = D_n = \{0, 1\}$, then we have a *binary decision diagram* (BDD), otherwise we have a *multi-valued decision diagram* (MDD). Note that the terminology for decision diagrams is still not universally agreed upon, and both MDDs and BDDs are used in slightly different variations across the literature.. In this paper we consider only *ordered decision diagrams*, that is decision diagrams where the variables labeling nodes in a path from the root to the terminal are in a fixed order. For a natural ordering x_1, \dots, x_n , for every edge $e(u, a, u')$, where u is labeled with x_i and u' labeled with x_j , it holds $i < j$. In the rest of the paper, when referring to an MDD or a BDD, we always assume fixed variable ordering.

While a decision diagram could encode the set of CSP solutions in several ways, we consider only an immediate approach, where we enforce that the set of satisfying assignments to a function f represented by the MDD $Sol(f) = \{a = (a_1, \dots, a_n) \mid f(a) = 1\}$ is identical to the solution set of a CSP, i.e. $Sol(f) = Sol(C)$.

3 Isomorphism and Redundancy of MDDs

It can be seen from the previous section that MDDs and DFAs are highly related structures. We will now precisely describe their relationship by discussing two well known reduction operations for decision diagrams: *merging isomorphic nodes* and *eliminating redundant nodes* [Bry86]. Note that every node u , labeled with a variable x_i , is associated with a function $f^u : D_i \times \dots \times D_n \rightarrow \{0, 1\}$ in a similar way that the root of a decision diagram is associated with the function f .

Definition 1 (Isomorphic Nodes) Nodes $u_1, u_2 \in V$ in an MDD $G(V, E)$ are isomorphic iff they represent the same function $f^{u_1} = f^{u_2}$.

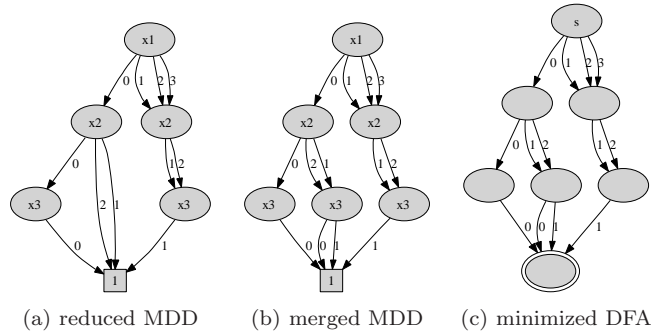


Figure 1. Encoding of the T-shirt CSP.

Isomorphic nodes can be efficiently detected by traversing a decision diagram in a bottom-up fashion and searching for nodes u, u' labeled with the same variable x_i and having identical children nodes for every outgoing edge. This is a crucial operation in obtaining compact representations. When all isomorphic nodes are merged we obtain a *merged MDD*. In fact, it can be easily seen that this structure is identical to *minimized DFAs* (MDFAs), that were used in [Vem92, AFM02].

Definition 2 (Redundant Nodes) A node u , labeled with x_i is redundant iff f^u does not essentially depend on x_i , i.e. f^u restricted by $x_i = a$ leads to the same function for all $a \in D_i$.

Node u , labeled with x_i is redundant if for all $a \in D_i$, there is a an outgoing edge (u, a, u') ending in the same child u' . Then, all incoming edges to u are redirected to u' , and u is removed. The new edges skip variable x_i , indicating that all assignments to x_i are allowed. We refer to these as *long edges*. If in addition to merging all isomorphic nodes we *remove redundant nodes* we get a *reduced MDD*. In fact, terms MDDs and BDDs most often refer to their reduced versions. In this paper we will refer to unreduced merged MDDs as *merged MDDs*.

While reduced MDDs are potentially smaller than merged MDDs (MDFAs), due to long edges, they are also more complicated to use, since long edges must be considered separately for many MDD queries (such as optimization [HA06]). It is not immediately clear which version is more suitable in general. All three representations: reduced MDDs, merged MDDs and MDFAs, for the T-Shirt example, are shown in Figure 1.

4 Compiling CSPs into BDDs

The most famous member of the decision diagram family are reduced *binary decision diagrams* (BDDs) [Bry86]. This is largely due to availability of many highly optimized BDD-manipulation packages [LNne, Som96] that lead to enormous computational advances in many areas of computer science, in particular model checking and verification. Reduced BDDs are therefore a legitimate choice for compiling CSPs.

However, in order to compile into BDDs, the CSP variables X first have to be *encoded by Boolean variables* X_b . For each finite domain variable x_i we select k_i Boolean

variables $X_b^i = \{x_j^i \mid j = 1, \dots, k_i\}$ and decide how to map each $a \in D_i$ into a *different* bit vector $enc_i(a) = (a_1, \dots, a_{k_i}) \in \{0, 1\}^{k_i}$. This completely specifies our encoding function $enc = (enc_1, \dots, enc_n)$. There is a 1-1 relationship between finite domain CSP solutions, and solution set of the corresponding BDD.

There are several standard encodings [Wal00]. The *log encoding* is the most commonly used [MAH02]. In this scheme each x_i is encoded with $k_i = \lceil \log |D_i| \rceil$ Boolean variables, each representing a digit in binary notation. In this case, $x_i = a \Leftrightarrow x_j^i = a_j$ where $a = \sum_{j=1}^{k_i} 2^{j-1} a_j$. The *direct encoding* (or *1-hot encoding*) is also common. Each finite domain variable x_i is encoded with $k_i = |D_i|$ Boolean variables such that $x_i = a \Leftrightarrow x_j^i = a_j$ where $x_j^i = 1$ for $j = a$ and $x_j^i = 0$ for $j \neq a$.

Ordering Boolean Variables While the set of Boolean variables is completely specified as the union of all encoding variables, $X_b = \bigcup_{i=1}^n X_b^i$ we still have to decide how to order the variables. The current practice [HSJ⁺04] is to respect the variable ordering among finite domain variables $x_1 < \dots < x_n$, by *clustering* Boolean variables X_b^i into finite-domain blocks. That is,

$$x_{j_1}^{i_1} < x_{j_2}^{i_2} \Leftrightarrow i_1 < i_2 \vee (i_1 = i_2 \wedge j_1 < j_2).$$

We refer to this as a *clustered variable ordering*. We will use a mapping $cvar(x_j^i) = i$ to denote the CSP variable x_i of an encoding variable x_j^i and, with a slight abuse of notation, we will apply $cvar$ also to BDD nodes u labeled with x_j^i .

Example 2 (BDDs for Binary-encoded CSPs.)

Recall that in the T-shirt example $D_1 = \{0, 1, 2, 3\}$, $D_2 = \{0, 1, 2\}$, $D_3 = \{0, 1\}$. The log encoding variables are $x_1^1 < x_2^1 < x_3^1 < x_1^2 < x_2^2 < x_3^2$, inducing a variable set $X_b = \{1, 2, 3, 4, 5\}$. The log-BDD with clustered variable ordering is shown in Figure 2(a). The direct encoding variables are $x_1^1 < x_2^1 < x_3^1 < x_4^1 < x_1^2 < x_2^2 < x_3^2 < x_4^2 < x_1^3 < x_2^3$, inducing a variable set $X_b = \{1, \dots, 9\}$. The direct-BDD with clustered variable ordering is shown in Figure 2(b). \diamond

5 Impact of Clustered Variable Ordering

We will show now that regardless of Boolean encoding, there is a strong correlation between a BDD and an MDD for the CSP \mathcal{C} when *clustered* variable ordering is used. While we focus on clustered orderings only, note however that other orderings are also possible, such as interleaved ordering of log-encoding that was particularly well suited for linear arithmetic constraints [BB03].

Let B be a reduced BDD and M a reduced MDD for a given CSP \mathcal{C} . We will show that a subset of nodes in the MDD induces a *merged MDD* that is *partially reduced*, i.e. that might contain some redundant nodes.

For a BDD (V, E) we denote with

$$L_i = \{u \in V \mid cvar(u) = i\}, \quad i = 1, \dots, n+1$$

the *layer* of the nodes in the BDD that encode the CSP variable x_i (we take $L_{n+1} = \{1\}$). We denote as

$$In_i = \{u \in L_i \mid \exists (u', u) \in E cvar(u') < cvar(u)\}$$

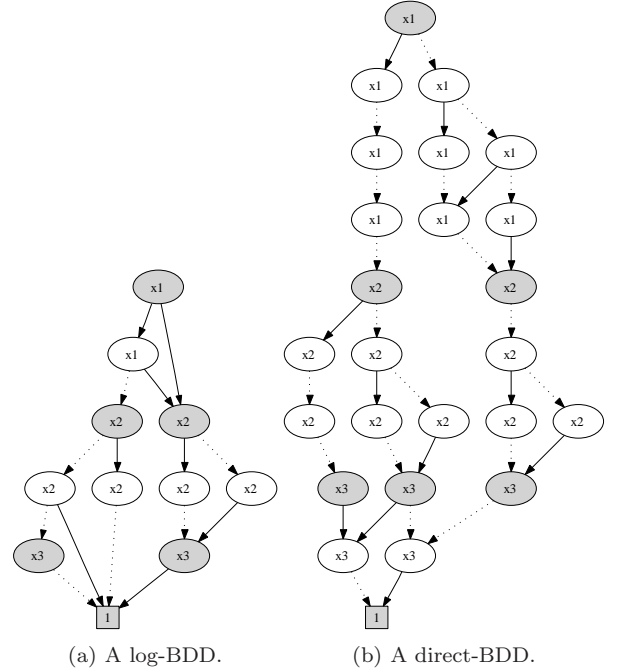


Figure 2. BDDs for the T-shirt example. Edges labeled with 0 and 1 are drawn as dashed and full lines, respectively. For convenience the nodes are marked with the integer-variable they encode (not with their own node label).

those nodes that are connected with an edge to a node in one of the previous layers (taking $In_1 = \{r\}$). Let $V_i = \{u \in V \mid cvar(u) = i\}$ denote the set of nodes in the i -th layer.

For a $u \in In_i$ and $u' \in In_j$, $j > i$, we write $u \xrightarrow{a} u'$ if starting from u and *traversing* the L_i layer with encoding of value a , $enc(a) = (a_1, \dots, a_{k_i})$, we end up in u' . If u is not labeled with the first encoding variable, i.e. it is labeled with x_j^i where $j > 1$, we take that all the first $j-1$ bit combinations (a_1, \dots, a_{j-1}) are allowed.

Definition 3 (Induced MDD) Given a reduced BDD $B(V, E)$, we define an induced MDD $M_B(V_B, E_B)$ by $V_B = \bigcup_{i=1}^{n+1} In_i$ and $E_B = \{(u, a, u') \mid u \xrightarrow{a} u', u, u' \in V_B\}$.

The above definition is sound since, for any $u \in In_i$, traversal $u \xrightarrow{a} u'$ ends by definition in $u' \in In_j$, for some $j > i$. Since we used clustered variable ordering, M_B respects the same ordering of CSP variables as reduced MDD M . It can be seen from the construction of M_B that the following holds:

Proposition 1 For a given CSP \mathcal{C} , represented by reduced BDD B using clustered variable ordering, the MDD M_B induced from B is merged and represents the same solution space $Sol(\mathcal{C})$.

Also note that induced MDD is *partially reduced*, i.e. it might contain some redundant nodes. All long edges from the reduced BDD, skipping all encoding variables X_b^i would be translated into long edges over variable x_i in the induced MDD. However, some redundant nodes would be introduced. Namely, long edges over the i -th layer of M_B are possible only

if encoding of x_i with $\{x_i^j \mid j = 1, \dots, k_i\}$ allows all assignments to x_i^j variables whenever all assignments to the CSP variable x_i are allowed. This happens iff $|D_i| = 2^{k_i-1}$. This is the case for example for a log-encoding of a variable x_3 in the T-shirt example, with domain size 1. On the other hand, if $|D_i| < 2^{k_i-1}$ there would be no long edges over $V_i(M_B)$ because such edges would inevitably allow some Boolean assignments that do not translate to integer value in D_i . For direct encoding, no long edges are possible.

For example, consider the emphasized nodes in a direct-BDD (Figure 2(b)) and note that these vertices correspond to the vertices of *merged MDD* in Figure 1(b) (i.e. no long edges). Similarly, emphasized vertices in the log-BDD (Figure 2(a)) correspond to the *reduced MDD* in Figure 1(a) (long edges were created since $|D_3| = 1$).

Impact on Size The above observations imply that the difference in size between BDDs with clustered encodings and MDDs depends mostly on the nature of domains D_i and encoding function *enc*, rather than combinatorial structure of solution space. It is illustrative to consider transforming a merged MDD (MDFA) into a reduced BDD. For each node u , labeled with variable x_i , we replace each outgoing edge $e(u, a, u')$ (representing $x_i = a$) with a path of $|X_b^i|$ nodes, encoding the Boolean assignment $(x_1^i = a_1, \dots, x_{k_i}^i = a_{k_i})$. After these transformations, some introduced nodes are eliminated once the merging of isomorphic nodes and elimination of redundant nodes is performed.

For a merged MDD $M(V, E)$ where d denotes the size of the largest CSP domain, there is at most $\lceil \log(d) \rceil \cdot |E|$ edges in the corresponding log-BDD, and at most $d \cdot |E|$ edges in the corresponding direct-BDD before merging and reduction. The worst-case size of $d \cdot |E|$ can be achieved for direct-BDDs when between any two nodes u, u' in a merged MDD there is at most one edge (u, a, u') . Then, d edges would be added, and no nodes would be merged or reduced. On the other hand, applying reduction can achieve significant savings against MDDs when there are many edges between the same pairs of nodes. In a log-BDD, a single edge between u, u' representing assignment to the first bit $x_1^i = a_1$, and skipping all remaining bits $x_j^i, j = 2, \dots, k_i$, corresponds to 2^{k_i-1} MDD edges between u and u' . In conclusion, while either representation could be smaller, we could expect large differences in size between the structures only for sufficiently large domains.

6 Experiments

We compared the size of reduced MDDs, MDFAs, BDDs with log encoding (log-BDDs) and BDDs with direct encoding (direct-BDDs) over a set of real-world and random instances:

Product configuration The instances *esvs*, *fs*, *pc2*, *1-6+22-32*, *Big-PC* and *Renault* are real-world product configuration instances from Clib [CLI07].

All different The instances *alldiff(k)* encode the constraint $\forall_{i \neq j} : x_i \neq x_j$ on four variables x_1, \dots, x_4 with the domain $1, \dots, k$.

Less than The instances *lessthan(k)* encode the constraint $\sum_{i=1}^5 x_i < 3k$ on the five variables x_1, \dots, x_5 with the domain $0, \dots, k$.

The results are shown in Figure 3. We estimate the size of representation by counting the number of edges. For reduced MDDs and MDFAs (merged MDDs) we report the number of nodes and edges. For BDDs we report the nodes only, as the number of edges is two times larger than the number of nodes.

Product configuration In all the product configuration instances the MDFA/MDD representation uses roughly four times less edges than log-BDDs. Furthermore, direct-BDDs are two to five times larger than the log-encoded BDDs. This seems to suggest that MDFA/MDDs are the preferred data structure in product configuration.

All different In the *alldiff* instances the space requirements of the log-encoded BDD and the MDD instance is approximately the same. Note that the size of the direct-BDD in *alldiff(80)* is about 20 times larger than the size of the log-BDD.

Less than In *lessthan* instances, log-BDDs are much smaller than MDDs. In the instance *lessthan(2500)* the MDD is approximately 60 times larger than the log-BDD. In representing the instance *lessthan(250)* the direct-BDD was using roughly 30 times more nodes than log-BDD.

The reduction in size achieved by node elimination by the MDD compared to the MDFA is insignificant for all instances that have been considered except for some of the very small configuration instances. This seems to suggest that it is reasonable to use the (simpler) MDFAs even though they might use more space than reduced MDDs.

7 Conclusions

We have compared a class of highly related CSP representations: MDFAs, MDDs and BDDs. We have observed that MDFAs as used in constraint programming community, correspond closely to MDDs. We have shown that, if clustered variable ordering is used, the BDDs share the basic structure with MDDs/MDFAs and differences in size can become significant only for large variable domains.

One implication of above observations is that the size of the clustered BDDs compared to the MDDs cannot be asymptotically larger if domain sizes are constant, as it is usually assumed in product configuration. However if the domain sizes are variable the BDD can be exponentially larger than the MDD for direct-encoding and the MDD can be exponentially larger than BDD using log-encoding.

We have experimentally compared the considered structures over a set of real-world and artificially constructed instances. In this comparison we have shown that though reduced MDDs are usually smaller than MDFAs the difference in size is insignificant. We have further observed that though log-encoded BDDs are usually larger than MDDs/MDFAs for configuration instances the reverse is the case for linear inequalities. Finally we have observed that the direct-encoding performs worse than log-encoding, especially on variables with large domains.

References

- [AFM02] J. Amilhastre, H. Fargier, and P. Marquis. Consistency restoration and explanations in dynamic CSPs

Instance	MDD		MDFA		Log-BDD	Direct-BDD
	Nodes	Edges	Nodes	Edges	Nodes	Nodes
ESVS	83	185	96	222	306	1732
FS	632	1149	766	2017	3044	15797
PC2	3906	6136	3906	6136	13332	43326
1-6+22-32	8094	11193	8094	11193	20935	62022
Big-PC	100192	132595	100271	132889	356696	1700488
Renault	329134	426212	329134	426212	768560	1419061
aldiff(10)	386	2560	386	2560	2224	5337
alldiff(40)	11521	93680	11521	93680	52949	494345
alldiff(80)	88641	758560	88641	758560	407249	7353525
lessthan(10)	64	593	65	604	354	1099
lessthan(250)	1504	314753	1505	315004	17326	597379
lessthan(2500)	15004	31272503	15005	31275004	265842	Out of Memory

Figure 3. A list comparing of the size of MDDs, MDFA, log-BDDs and direct-BDDs, when they encode the same CSP.

- [AHHT07] Henrik Reif Andersen, Tarik Hadzic, John N. Hooker, and Peter Tiedemman. A Constraint Store Based on Multivalued Decision Diagrams. In C. Bessiere, editor, *Principles and Practice of Constraint Programming (CP 2007)*, Lecture Notes in Computer Science. Springer, 2007.
- [BB03] Constantinos Bartzis and Tevfik Bultan. Construction of efficient BDDs for bounded arithmetic constraints. In Hubert Garavel and John Hatcliff, editors, *TACAS*, volume 2619 of *Lecture Notes in Computer Science*, pages 394–408. Springer, 2003.
- [Bry86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 1986.
- [CLi07] CLib. Configuration benchmarks library. Available online at: <http://www.itu.dk/research/cla/externals/clib/>, 2007.
- [CY06] Kenil C. K. Cheng and Roland H. C. Yap. Maintaining generalized arc consistency on ad-hoc n-ary boolean constraints. In Gerhard Brewka, Silvia Coradeschi, Anna Perini, and Paolo Traverso, editors, *ECAI*, pages 78–82. IOS Press, 2006.
- [DM02] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.
- [HA06] Tarik Hadzic and Henrik Reif Andersen. A BDD-based Polytime Algorithm for Cost-Bounded Interactive Configuration. In *Proceedings of AAAI’06*, 2006.
- [HSJ⁺04] 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.
- [LNne] J. Lind-Nielsen. BuDDy - A Binary Decision Diagram Package. <http://sourceforge.net/projects/buddy>, online.
- [MAH02] J. Møller, H. R. Andersen, and H. Hulgaard. Product configuration over the internet. In *INFORMS Conference on Information Systems and Technology*, 2002.
- [MMD07] Robert Mateescu, Radu Marinescu, and Rina Dechter. AND/OR Multi-Valued Decision Diagrams for Constraint Optimization. In Frédéric Benhamou, editor, *CP*, Lecture Notes in Computer Science. Springer, 2007.
- [Pes04] Gilles Pesant. A regular language membership constraint for finite sequences of variables. In *Proceedings of Principles and Practice of Constraint Program-*
- [Som96] F. Somenzi. CUDD: Colorado university decision diagram package. <ftp://vlsi.colorado.edu/pub/>, 1996.
- [Vem92] Nageshwara Rao Vempaty. Solving constraint satisfaction problems using finite state automata. In *AAAI*, pages 453–458, 1992.
- [Wal00] Toby Walsh. SAT v CSP. In Rina Dechter, editor, *CP*, Lecture Notes in Computer Science, pages 441–456, 2000.
- [Weg00] Ingo Wegener. *Branching Programs and Binary Decision Diagrams*. Society for Industrial and Applied Mathematics (SIAM), 2000.