Two Encodings of DNNF Theories

Jean Christoph Jung[1] and **Pedro Barahona**[2] and **George Katsirelos**[3] and **Toby Walsh**[4]

Abstract. The paper presents two new compilation schemes of Decomposable Negation Normal Form (DNNF) theories into Conjunctive Normal Form (CNF) and Linear Integer Programming (MIP), respectively. We prove that the encodings have useful properties such as unit propagation on the CNF formula achieves domain consistency on the DNNF theory. The approach is evaluated empirically on random as well as real-world CSP-problems.

1 Introduction

The graphical structures used in knowledge compilation (for an overview see [9]) have found significant applications in several areas, including truth maintenance systems [8], diagnosis [7] and verification. By far the most common use has been the application of BDDs [4] to verification.

Here, our purpose is three-fold: first, we show that these knowledge compilation structures can be used in constraint programming as compact representations of ad-hoc constraints; second, that we can then decompose these constraints into either CNF or MIP and use these decompositions in hybrid solvers; and third, that it is worth investigating compilation of constraints expressed extentionally into more expressive forms, such as DNNF. This may be easier and more fruitful than compilation of arbitrary formulas.

The rest of the paper is structured as follows. First we define several knowledge compilation forms and constraints in sections 1.1 and 1.2, respectively. Then we introduce encodings of these forms into CNF and prove some useful properties in section 2. Finally, we present an initial empirical evaluation of these encodings in section 3 before we conclude.

1.1 Propositional Theories

First we define the languages NNF, DNNF and BDD along the lines of [9].

Definition 1 *Let P be a set of propositional variables. The language NNF is defined as the set of rooted, directed acyclic graphs (DAG) where each leaf node is labeled with true, false or a propositional literal (from P) and each internal node is labeled with $\wedge$ or $\vee$ and has arbitrarily, but finitely many children.*

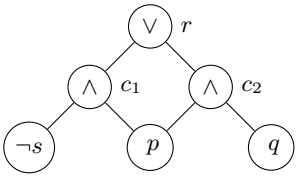
We can restrict this language by requiring some properties, such as *decomposability* and *decision*. Decomposability is the property that for every and-node, any two children do not share any variable. A formula has the property Decision if every node is a decision node, i.e., every node is *true*, *false* or an or-node of the form $(p \wedge \alpha) \vee (\neg p \wedge \beta)$, where p is a propositional variable and α and β are decision nodes. These conditions, fulfilled or not fulfilled, give rise to several subclasses of NNF. In particular, we want to deal with the classes DNNF and certain forms of BDD.

Definition 2 *The language DNNF is the subset of NNF that fulfills the decomposability condition. The language BDD is the subset of NNF where the root of each sentence is a decision node. FBDD is the intersection of the classes DNNF and BDD. OBDD is the subclass of FBDD where on each path from the root to the sinks (true and false) the variables occur in a fixed order.*

These languages are as expressive as NNF, despite the restrictions that are imposed. However, they are exponentially less succinct [9]. This means that there exist infinite families of formulas such that the smallest FBDD(OBDD) is exponential in the size of the formula, whereas there exists a polynomially sized DNNF(FBDD) representation. On the other hand, every FBDD(OBDD) is also a DNNF(FBDD) and therefore the converse does not hold.

Figure 1 shows an example for a DNNF formula. In particular, the children of the and-nodes c_1, c_2 are variable disjoint.

Figure 1. Example for a DNNF formula



1.2 Constraints

We denote variables by uppercase letters $X, X_1, \ldots, X_n$. The *domain* $dom(X)$ of a variable X is the set of allowed values for X. We deal here only with finite domains and denote domain elements with lowercase letters. A *constraint* C on a sequence of variables $X_1, \ldots, X_k$ (the *scope* of C) is defined as a subset of the Cartesian product of the domains of the variables, i.e. $C \subseteq D(X_1) \times \ldots \times D(X_k)$. Every $c \in C$ is called *support* of C. A tuple $(d_1, \ldots, d_k)$ *satisfies* a constraint C if $(d_1, \ldots, d_k) \in C$.

A *constraint satisfaction problem (CSP)* consists of a set of variables, corresponding (finite) domains and a set of constraints. The goal is to find an assignment of values to all the variables that satisfies all constraints. A constraint can be given implicitly, e.g.,

[1] Universidade Nova de Lisboa, email: j.jung@fct.unl.pt
[2] Universidade Nova de Lisboa, email: pb@di.fct.unl.pt
[3] NICTA, Sydney, email: george.katsirelos@nicta.com.au
[4] NICTA and UNSW, Sydney, email: tw@cse.unsw.edu.au

alldifferent, that restricts a set of variables to take pairwise different values, or explicitly as set of allowed tuples.

A constraint is *generalized arc consistent (GAC)* if for each variable X in its scope and every $d \in dom(X)$ there is a satisfying assignment with $X = d$.

2 Encodings

2.1 CNF encoding

Given a DNNF theory Δ, we can define a CNF encoding of Δ. Note that every formula Δ corresponds directly to a DAG. We will use the name Δ for both for the formula and for the DAG. Applying unit propagation on this CNF encoding enforces GAC on the constraint described by Δ.

Intuitively, we create a set of clauses that propagate unsatisfiability through the graph, i.e., if we know something to be *false* this will be propagated in the graph. This can be done very easily upwards: If the children of an or-node are known to be *false*, then the or-node is also false. If one child of an and-node is known *false*, the and-node is also false. We need one more rule to propagate also downwards in the DAG, so that if it is known there is no model for the formula that evaluates any of the ancestors of a node c to *true*, then we infer there exists no model that sets c to true. For every node c of the graph, we create a corresponding propositional variable, which we also refer to as c. We do not introduce any variables for the leaf nodes, but instead we use the attached literal or *true* or *false*. We summarize the rules in Definition 3.

Definition 3 *The CNF-encoding CNF(Δ) is the set of all clauses that are created by the following rules:*

1. *for every or-node $c = c_1 \vee \ldots \vee c_k$:*
   $\neg c_1 \wedge \ldots \wedge \neg c_k \rightarrow \neg c \equiv [c_1, \ldots, c_k, \neg c]$
2. *for every and-node $c = c_1 \wedge \ldots \wedge c_k$:*
   $\neg c_1 \vee \ldots \vee \neg c_k \rightarrow \neg c \equiv \bigwedge_{i=1}^{k} [c_i, \neg c]$
3. *for every node c with parents $p_1, \ldots, p_k$:*
   $\neg p_1 \wedge \ldots \wedge \neg p_k \rightarrow \neg c \equiv [p_1, \ldots, p_k, \neg c]$
4. *for the root r of Δ:*
   $[r]$

We consider again Figure 1. As indicated there, we introduce the variables r, c_1 and c_2 for the root and the and-nodes, respectively. Table 1 shows the clauses that are created by Definition 3. In the following we prove some important properties of this encoding.

Table 1. CNF encoding of the DNNF in Figure 1

case	clauses
1	$[c_1, c_2, \neg r]$
2	$[s, \neg c_1], [\neg p, \neg c_1], [\neg p, \neg c_2], [\neg q, \neg c_2]$
3	$[c_1, c_2, \neg p]$
4	$[r]$

Theorem 1 *The size of the CNF encoding of an DNNF theory Δ, CNF(Δ), is polynomial in the size of Δ.*

Proof. By Definition 3 we create one clause for every node (case 3), for every and-node at most k clauses, where k is the maximal out-degree of an and-node, and one clause for every or-node. Obviously this is in $O(k \cdot |\Delta|)$, because the length of the formula $|\Delta|$ bounds the number of nodes.

Theorem 2 *Δ is satisfiable iff CNF(Δ) is satisfiable.*

Proof. We prove first the direction "⇒". Assume Δ is satisfiable. Then there is a model M of Δ. Corresponding to the model we can define a sub-tree of Δ that represents M in the following way:

$$tree_M(c) = \begin{cases} \{c\} \cup tree_M(c_i) & c = c_1 \vee \ldots \vee c_k \\ & \quad \text{and } M \models \Delta(c_i) \\ \{c\} \cup \bigcup_{i=1}^{k} tree_M(c_i) & c = c_1 \wedge \ldots \wedge c_k \\ \{c\} & c \text{ is a literal} \\ \emptyset & \text{otherwise} \end{cases} \quad (1)$$

We have to consider if the function is well-defined, because the definition refers to the semantic of the formula in the first case: It is not obvious that there exists an i with $M \models c_i$. Surely, if we assume $M \models c$, then we can find such an i. We observe that the function is initially called with the root of Δ and we know that M is a model for Δ. By the definition of (1) we have that $M \models c$ for every c that $tree_M$ is called with.

Now we define an interpretation I and show that I is a model for CNF(Δ).

$$c^I = \begin{cases} 1 & c \in tree_M(root(\Delta)) \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

To show that I models CNF(Δ) we look at every clause of the encoding and show that I is a model for every clause.

Case 1. For every node $c = c_1 \vee \ldots \vee c_k$ clauses of the form $l = [c_1, \ldots, c_k, \neg c]$ are generated. Then we have:

$$\begin{aligned} I \models l \quad & \text{iff} \quad \exists i. I \models c_i \text{ or } I \models \neg c \\ & \text{iff} \quad \exists i. c_i^I = 1 \text{ or } c^I = 0 \\ & \text{iff} \quad \exists i. c_i \in tree_M(root(\Delta)) \text{ or } c \notin tree_M(root(\Delta)) \\ & \text{iff} \quad c \in tree_M(root(\Delta)) \Rightarrow \exists i. c_i \in tree_M(root(\Delta)) \end{aligned}$$

which is obviously fulfilled by the definition of $tree_M$.

Cases 2 and 3. For clauses produced by these cases in Definition 3 we can argue similarly to case 1.

Case 4. For the unit clause $[root(\Delta)]$ it is trivially fulfilled. This closes the first part of the proof.

For the other direction we assume CNF(Δ) is satisfiable and let M be a model. We show that M is also a model for Δ. For this we use the following lemma:

Lemma 1 *Let M be a model of CNF(Δ), c be a node in Δ (thus also the corresponding propositional variable) and t_c the sub-DAG of Δ rooted at c. Then it holds $M \models c$ implies $M \models t_c$.*

Proof by induction on the DAG. The *induction base* is immediate, because c coincides with t_c for leaf nodes. In the *induction step* we have to distinguish two cases. First, let c be first an or-node $c = c_1 \vee \ldots \vee c_k$ and assume $M \models c$. Note that we have a corresponding clause $[c_1, \ldots, c_k, \neg c]$ in the encoding. Now we have:

$$\begin{aligned} M \models \text{CNF}(\Delta) \quad & \Rightarrow \quad M \models [c_1, \ldots, c_k, \neg c] \\ & \overset{M \models c}{\Rightarrow} \quad M \models [c_1, \ldots, c_k] \\ & \Rightarrow \quad \exists i. M \models c_i \\ & \overset{I.H.}{\Rightarrow} \quad M \models t_{c_i} \\ & \Rightarrow \quad M \models t_c \end{aligned}$$

If, on the other hand, c is an and-node we can argue likewise, which finishes the proof of the Lemma. For M is a model for the encoding, by clause (4) of Definition 3 M is also a model for the propositional variable attached to the root of Δ. Applying Lemma 1 yields $M \models \Delta$.

Theorem 3 *Unit propagation on* $\texttt{CNF}(\Delta)$ *achieves GAC on* Δ.

The proof follows very closely the argumentation in [18]: A propositional variable p is arc-inconsistent with Δ iff all models M of Δ evaluate p to 0. Let p be arc-inconsistent and assume $\neg p$ is not implied by the clauses. By the rules (3) of Definition 3 we can find an ancestor p' which is not implied to be false. Continuing this argument gives us a path s from p to the root of Δ. Reasoning in the other direction, i.e., using rules (1) and (2), we find a model for Δ in which p is true. More precisely, if we are at an and-node we go to all the children and if we are at an or-node we choose one. Of course, if we are at an or-node x that is in s we choose the ancestor of x in s. In this way we make sure that p appears in the model. The fact that we can construct a model which evaluates p to 1 gives us the contradiction with our assumption that p is arc-inconsistent. So $\neg p$ is implied by $\texttt{CNF}(\Delta)$.

2.2 MIP encoding

SAT solvers are a very powerful tool to use in Constraint Programming. Another class of performant tools is the class of programs that solve (mixed) integer programs (MIP), i.e., linear equalities and inequalities with integer coefficients. Also MIP is a powerful way to model problems. Here we show a way to model DNNF theories using MIP with the intuition that each solution to the MIP corresponds to a model of the DNNF and vice versa. As for the CNF encoding we introduce a propositional variable for each inner node. For each positive literal p we use a corresponding binary variable x_p and for each negative literal $\neg p$ the term $1 - x_p$.

Definition 4 *The MIP-encoding* $\texttt{MIP}(\Delta)$ *is the set of all constraints that are created by the following rules:*

1. *for every or-node* $c = c_1 \vee \ldots \vee c_k$*:*
   $c \leq \sum_{i=1}^{k} c_i$
2. *for every and-node* $c = c_1 \wedge \ldots \wedge c_k$*:*
   $k \cdot c \leq \sum_{i=1}^{k} c_i$
3. *for the root* r *of* Δ*:*
   $r = 1$

The (in-)equalities are quite intuitive and mimic exactly the properties of a model of the theory: The model evaluates the root to *true* (last equation). If an or-node is *true* in the model, there must be a reason for this, i.e., the or-node has at least one successor that is also *true* (first inequality). If an and-node is *true* in the model then all its successors should be evaluated to *true* (second inequality). With this idea in mind we can prove the following theorem.

Theorem 4 *The constraints of* $\texttt{MIP}(\Delta)$ *admit a solution iff* Δ *is satisfiable.*

Proof. Assume first we have a solution of $\texttt{MIP}(\Delta)$. A solution S assigns a value $S(c)$ to every variable c. Using S we define an interpretation I that models Δ. Note that the range of I contains only the original propositional variables (not the variables added during the encoding).

$$c^I = \begin{cases} 1 & S(c) = 1 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

We prove first an analogon to Lemma 1.

Lemma 2 *Let* S *be a solution of* $\texttt{MIP}(\Delta)$*,* c *be a node in* Δ *(thus also the corresponding propositional variable) and* t_c *the sub-DAG of* Δ *rooted at* c*. Then it holds* $S(c) = 1$ *implies* $I \models t_c$*.*

The proof is by induction on the structure of the DNNF. For the *induction base* assume that c is a leaf node. Obviously $S(c) = 1$ implies $c^I = 1$ which is equivalent to $I \models t_c$. In the *induction step* we distinguish two cases. Let c first be an and-node: $c = c_1 \wedge \ldots \wedge c_k$. Then we have $S(c) = 1$ implies $S(c_i) = 1$ (by the inequality for and-nodes in Definition 4). Applying the hypothesis yields $I \models t_{c_i}$ for all $i \in \{1, \ldots, k\}$, hence $I \models t_c$. Likewise we can argue for or-nodes. This finishes the proof of the lemma.

For sure, the variable corresponding to the root of Δ is 1 in the solution. Lemma 2 yields now $I \models \Delta$.

For the other direction assume that Δ evaluates to 1 under some interpretation I. We extend I to the variables corresponding inner nodes of Δ by using the obvious rules, e.g., $c^I = 1$ iff $c = c_1 \wedge \ldots \wedge c_k$ and $c_i^I = 1$ for all i between 1 and k. Clearly, I can be interpreted as a solution for the MIP encoding of δ, because the applied rules respect the constraints 1 and 2 of Definition 4. Moreover, since $I \models \Delta$ the variable corresponding to the root evaluates to 1, so all conditions are fulfilled.

3 Experiments

We conducted experiments to evaluate both the size of the encodings that we propose as well as the time needed to solve random instances using these encodings, using RSat 2.0 [17]. In order to compare against a realistic baseline, we also used the CSP solver Gecode [11] to solve these problems.

3.1 Problem generation

We generated a set of non-binary random instances, each described by the 5-tuple $\langle n, d, a, p_1, p_2 \rangle$ where n is the number of variables, d is the uniform domain size, a is the uniform constraint arity, p_1 is the density of the constraint graph and p_2 is the looseness of each of the constraints. The probabilities are treated as proportions, which means that of the $\binom{n}{a}$ possible a-ary constraints exactly $p_1 \cdot \binom{n}{a}$ are created. Likewise, for each constraint exactly $p_2 \cdot d^a$ tuples (supports) are created. We tried to generate hard problems, so that satisfiability cannot be shown without backtracking. One way to describe the hardness of problems is in terms of the constrainedness parameter κ [13]. Their experiments show that in many problems the phase transition typically occurs for κ ranging from 0.75 to 1. Equation 4 (which is from [6]) gives the definition of κ in the case of random CSPs generated with both uniform degree and domain size. Using this equation we can calculate density and looseness of a CSP as a function of n, a, d and κ such that the generated problem is in the transition phase with a high probability.

$$\kappa = \frac{-p_1 \cdot \binom{n}{a} \cdot \log_2(p_2)}{n \cdot \log_2(d)} \tag{4}$$

3.2 The compilation

Previous approaches use a compilation of constraints to OBDDs [16], [10]. We want to show that it is worth thinking about a compilation to DNNF, because the size of the encoding can be much smaller. So we compared a CNF encoding of the OBDD encoding of a CSP with the CNF encoding of the DNNF encoding of this problem. We used direct domain encoding, i.e., for every variable X and every value v in the domain of X, we introduce a propositional variable $(X = v)$. We include in both CNF encodings the clauses that encode the domain: $\bigvee_{v \in dom(X)}(X = v)$ and $(X \neq v_1 \vee X \neq v_2)$ for all $v_1, v_2 \in dom(X), v_1 \neq v_2$. We compile each constraint of the CSP separately to a OBDD and DNNF, respectively. In case of OBDD we fix a variable and a value ordering according to which we successively create the OBDD. For example let the current branching variable be $(X = 1)$. Then we split the table into two parts: one where we delete the X-column and take only the rows with $X = 1$ (this will represent the high successor of the node), and one where we take only the columns with $X \neq 1$ (which will correspond to the low successor). We create recursively OBDDs for the two sub-tables and create a node with the mentioned successors. We use caching in the sense that for every table at most one node is created. The obtained OBDD is translated to CNF by the Tseitin transformation [10]. Note that UP yields GAC on the constraint also for this transformation.

In every step of DNNF compilation we choose a variable which we branch on. For example, let X be the next branching variable and $dom(X) = \{v_1, \ldots, v_k\}$. Additionally let C be the current table. Then we create k DAGs $X = v_i \wedge \text{DNNF}(C|(X = v_i))$ for each $i \in \{1, \ldots, k\}$ where $|$ denotes the projection of C to a (variable, value)-pair. The roots of these DAGs are connected by a k-ary or-node. The obtained DNNF is translated into CNF using the rules from Definition 3.

We are free to choose any variable ordering, static or dynamic, in both OBDD compilation and DNNF compilation. Note that value ordering does not have influence on the size because all solutions are encoded not just one. We can conceive the OBDD and DNNF data structure as representation of the search space for one constraint. Hence, for minimizing the data structures we can basically apply arbitrary known variable heuristics that try to minimize the search space. In particular, we used the following heuristics:

- Lexicographical ordering lex.
- minimal domain minDom: The static variable ordering that orders the variables increasingly according to their domain size. Ties are broken by lexicographical ordering.
- most constrained variable mostCon: The same as minDom with the only difference that it is dynamic, i.e., the domain size at the time of branching is taken into account.
- FORCE [1]: A static heuristic that orders variables such that variables that are close to each other in the constraint graph, are also close to each other in the ordering.

3.3 Results

We created 100 instances of problems $\langle 30, 5, 4, p_1, p_2 \rangle$ where we chose p_2 uniformly distributed between 0.05 and 0.6 and calculated p_1 according to Equation (4), where κ was set to 0.95 in order to get close to a phase transition.

For each of these problems, we can create a SAT instance which is the conjunction of the CNF encodings of the OBDD(DNNF) representation of each constraint. We will refer to them as CNF encodings of the OBDD(DNNF) representations of the instances, even though the OBDD(DNNF) does not represent the entire problem. As variable ordering we use in both cases lex. Additionally, we compare with a recently proposed family of SAT mappings, namely k-AC [3], in particular with 0-AC and 3-AC. We chose 3-AC, as the authors say it is usually the best to choose $k = a-1$, and 0-AC, because the mapping is quite similar to ours.

We compare finding the first solution to these problems using Gecode with the built-in extensional constraint against using a SAT solver on the CNF encoding of the OBDD (DNNF) representation of the instances.

We report first on the size of the CNF encodings. In table 2 we compare the number of variables and number of clauses as well as the total size of the two CNF encodings. The size of the CNF is the sum of the sizes of all the clauses. This is generally accepted as a realistic indicator of performance, as unit propagation is in the worst case linear in the size of the CNF. Our second measure for the quality of an encoding is the number of solved instances S_t within t seconds. In particular we report on S_1, S_{10} and S_{100}. As you can see in Table 2, the total size of the DNNF is smallest, but the encoding introduces more variables than OBDD and 3-AC. Despite our expectations based on the size of the CNF encoding, we see that performance using the DNNF encoding of the problems is slower than using the OBDD encoding. We observed that the SAT solver in case of the OBDD representation explores more nodes per time than in case of the DNNF representation, hence the average case performance of unit propagation is better for OBDD. We conjecture that the reason for this discrepancy for the average case behaviour between the two CNF encodings is that the DNNF encoding introduces large clauses which need to be examined more times before they cause any unit propagation and cannot be handled by the specialized routines for binary and ternary clauses that many SAT solvers employ. However, in this series of experiments the 3-AC mapping outperforms all other approaches with respect to solved instances.

Table 2. Size of the CNF encodings.

encoding	# vars	# clauses	Size	S_1	S_{10}	S_{100}
Gecode	-	-	-	40	63	87
DNNF	10696	28660	79249	27	46	72
OBDD	7238	42329	126881	30	54	81
0-AC	11292	66960	178401	21	35	58
3-AC	150	22410	111033	70	92	100

In contrast to randomly generated problems, real-world constraint satisfaction problems usually have an internal structure that can be exploited by the solvers. As a reference for real-world problems we compiled a huge instance of a configuration problem (Renault from CLib [21]) into DNNF and OBDD, respectively. However, we added some random constraints to connect the constraint graph and make it a hard problem. We used the same approach as in [20], i.e., added 10 constraints with random scope and tightness, and looked for a constraint that is hard to solve with Gecode (more than 1 second). Table 3 shows the results with respect to the variable ordering we used. We included also the time that the SAT solver needed to solve the instance (for comparison: Gecode found the first model after 1.6 seconds).

We observe that the heuristics maxCon and minDom perform on DNNF at least as well as minDom on OBDDs with respect to the number of variables. Note that far less clauses are needed and that the size of the DNNF encoding is only half the size of the OBDD encod-

Table 3. Size of the CNF encodings.

encoding	heuristic	# vars	# clauses	Size	time (s)
DNNF	lex	24639	74792	196659	0.22
DNNF	maxCon	13551	40838	112175	0.11
DNNF	minDom	14664	44091	120704	0.14
DNNF	FORCE	17305	52634	131965	0.15
OBDD	minDom	14535	85845	254672	0.30
0-AC	-	118132	1099156	2801392	8.54
(a-1)-AC	-	432	214366845	$2 \cdot 10^9$	-

ing. Comparing with the lexicographical order we can see that it is worth to think about other heuristics to construct the DNNF, because we gain almost factor 2 in number of variables. The FORCE heuristic is slightly outperformed by the others. With respect to the k-AC heuristics we observe that, in contrast to our first experiments, they are by far outperformed. Note that we only calculated the sizes for the $(a-1)$-AC mapping, because it took to long to compute the encoding itself. The reason for the extreme growth of size is that the mapping depends exponentially on the degree of the constraint and the Renault instance contains some tables over 10 or more variables. A DNNF encoding shrinks if the constraint is very tight or very loose.

4 Related Work

BDDs have been used before to represent constraints and enforce GAC on them [14, 5]. In these, generalized arc consistency is maintained by a specialized algorithm. More recently, decompositions to CNF of knowledge compilation structures which enforce GAC by unit propagation have been proposed. In [2], Bacchus proposed a decomposition into CNF of deterministic finite automata. DFAs are roughly equivalent to multi-valued BDD in which long edges are not allowed. Eén and Sörensson [10] explored expressing pseudo boolean constraints as both BDDs and sorting networks and subsequently decomposing them into CNF. Further, [12, 3] show ways to map CSP problems to SAT problems, but [12] is restricted to binary constraints.

The closest work to our own is that of [18], where the authors propose a CNF decomposition of the grammar constraint. In fact, this decomposition is very similar to the one that we propose in section 2. Implicitly, the decomposition of the grammar constraint uses an intermediate form that is an And/Or graph. This graph can be seen as a DNNF in which the gates are arranged in layers and each layer may contain only and-gates or only or-gates, alternating between the two. Our approach is more general in two ways: first, we can decompose any DNNF, not only And/Or graphs; second, the DNNF may be generated in any way, not only as an intermediate result of a grammar constraint.

Other works that propose MIP encodings of graphical structures are [15, 19]. In particular, the encoding of Bayesian networks into MIP in [19] is similar to our MIP encoding.

5 Conclusion

We introduced two new decompositions of DNNF theories, one into CNF and the other into MIP. We evaluated these decompositions empirically on random problems and showed that performance can be comparable to using a CSP solver or an OBDD decomposition. On the other hand, the resulting encoding was smaller with DNNF. Our results suggest, first, that random constraints are not necessarily amenable to compilation to DNNF; and second, that we can improve on our current results by evaluating more sophisticated approaches of compiling constraints into DNNF. On the example of the Renault instance we showed that the compilation into DNNF on a structured problem can outperform the compilation into OBDD as well as recently proposed mappings like k-AC. In future work we plan to explore better heuristics or completely new methods to compile to DNNF.

ACKNOWLEDGEMENTS

We would like to thank the referees for their comments which helped improve this paper.

REFERENCES

[1] Fadi A. Aloul, Igor L. Markov, and Karem A. Sakallah, 'Force: a fast and easy-to-implement variable-ordering heuristic', in *ACM Great Lakes Symposium on VLSI*, pp. 116–119, (2003).
[2] Fahiem Bacchus, 'GAC via unit propagation', in *Proceedings of CP-2007*, pp. 133–147, (2007).
[3] Christian Bessière, Emmanuel Hebrard, and Toby Walsh, 'Local consistencies in SAT', in *SAT*, pp. 299–314, (2003).
[4] Randal E. Bryant, 'Graph-based algorithms for boolean function manipulation', *IEEE Trans. Computers*, **35**(8), 677–691, (1986).
[5] Kenil C. K. Cheng and Roland H. C. Yap, 'Maintaining generalized arc consistency on ad-hoc n-ary boolean constraints', in *ECAI 2006*, pp. 78–82, (2006).
[6] Marco Correia and Pedro Barahona, 'On the integration of singleton consistency and look-ahead heuristics', in *Proceedings of the annual ERCIM workshop on constraint solving and constraint logic programming*, eds., Francois Fages, Sylvain Soliman, and Francesca Rossi, Rocquencourt, France, (June 2007).
[7] Adnan Darwiche, 'Model–based diagnosis using structured system descriptions', *Journal of Artificial Intelligence Research*, **8**, 165–222, (1998).
[8] Adnan Darwiche, 'On the tractable counting of theory models and its application to truth maintenance and belief revision.', *Journal of Applied Non-Classical Logics*, **11**(1-2), 11–34, (2001).
[9] Adnan Darwiche and Pierre Marquis, 'A knowlege compilation map', *Journal of Artificial Intelligence Research*, **17**, 229–264, (2002).
[10] Niklas Eén and Niklas Sörensson, 'Translating pseudo-boolean constraints into SAT.', *Journal on Satisfiability, Boolean Modeling and Computation*, **2**, 1–26, (2006).
[11] Gecode Team. Gecode: Generic constraint development environment, 2006. Available from http://www.gecode.org.
[12] Ian P. Gent, 'Arc consistency in SAT', in *ECAI*, pp. 121–125, (2002).
[13] Ian P. Gent, Ewan MacIntyre, Patrick Prosser, and Toby Walsh, 'The constrainedness of search', in *AAAI/IAAI, Vol. 1*, pp. 246–252, (1996).
[14] P. Hawkins and P.J. Stuckey, 'A hybrid BDD and SAT finite domain constraint solver', in *Proceedings of the Practical Applications of Declarative Programming, PADL 2006*, LNCS, pp. 103–117. Springer, (2006).
[15] S. Joy, J. E. Mitchell, and B. Borchers, 'Solving MAX-SAT and weighted MAX-SAT problems using branch-and-cut', Technical report, Troy, NY 12180, (1998).
[16] Nina Narodytska and Toby Walsh, 'Constraint and variable ordering heuristics for compiling configuration problems', in *IJCAI*, pp. 149–154, (2007).
[17] Knot Pipatsrisawat and Adnan Darwiche, 'Rsat 2.0: SAT solver description', Technical Report D–153, Automated Reasoning Group, Computer Science Department, UCLA, (2007).
[18] Claude-Guy Quimper and Toby Walsh, 'Decomposing global grammar constraints.', in *Proceedings of CP-2007*, pp. 590–604, (2007).
[19] E. Santos Jr., 'On the generation of alternative explanations with implications for belief revision', in *Uncertainty in Artificial Intelligence*, pp. 339–347, (1991).
[20] Kostas Stergiou and Nikos Samaras, 'Binary encodings of non-binary constraint satisfaction problems: Algorithms and experimental results', *J. Artif. Intell. Res. (JAIR)*, **24**, 641–684, (2005).
[21] VeCoS group, IT-university of Copenhagen. Configuration benchmarks library, 2005. Available from http://www.itu.dk/research/cla/externals/clib/.