Incremental Algorithms for Approximate Compilation

AAAI-08 ID: 605

Abstract

Compilation is an important approach to a range of inference problems, since it enables linear-time inference in the size S of the compiled representation. However, the main drawback is that S can be exponentially larger than the size of the original function. To address this issue, we propose an incremental, approximate compilation technique that guarantees a sound and space-bounded compilation for weighted boolean functions, at the expense of query completeness. In particular, our approach selectively compiles all solutions exceeding a particular threshold, given a range of weighting functions, without having to perform inference over the full solution-space. We describe incremental, approximate algorithms for the prime implicant and DNNF compilation languages, and provide empirical evidence that these algorithms enable space reductions of several orders-of-magnitude over the full compilation, while losing relatively little query completeness.

Introduction

A broad range of compilation approaches have been proposed in the literature, such as prime implicants (de Kleer 1986), DNNF (Darwiche 2001), Ordered Binary Decision Diagrams (OBDDs) (Bryant 1992), cluster-trees (Pargamin 2003), and finite-state automata (Amilhastre, Fargier, & Marquis 2002). The benefit of these approaches is that they enable inference that is linear in the size of the compiled representation; the drawback is that the size of the compiled representation can be exponentially larger than that of the original function. For example, the number of prime implicants of a set m of arbitrary clauses is $O(3^m)$ (Chandra & Markowsky 1978), and the size-complexity of compiled representations (e.g., DNNF, OBDD) for problems in propositional logic, Bayesian networks and constraint satisfaction is exponential in the treewidth of their interaction graph (Darwiche 2001; Jensen, Lauritzen, & Olesen 1990).

In this article we propose a sound but incomplete incremental approximation technique that enables us to trade off the size of the compilation for coverage of the most-preferred solutions. We assume that we have a valuation function that identifies the most likely solutions (or satisfying assignments) of f. Such functions are well-known in the literature for a range of applications; for example, in diagnosis we may choose the probabilistically most-likely diagnoses, and in configuration we may choose a cardinality-minimal configuration.

Given a preference relation ϕ over f, we approximate $\zeta(f)$ using a valuation function to compile the most likely solutions (or satisfying assignments) of f by pruning solutions below a threshold ϱ. Hence, given a threshold valuation ϱ, our approach guarantees the preservation of all solutions exceeding ϱ, given a range of preference relations ϕ.

We empirically validate this approach for two important compilation targets, prime implicants and DNNF. For each target language, we propose incremental algorithms that generate approximate (sound but incomplete) representations that trade off solution coverage for space and inference efficiency, without having to compute the full compilation (which may be of size exponential in certain parameters of f). We provide empirical evidence that incremental compilation achieves space reductions of several orders-of-magnitude over the full compilation, while losing relatively little query completeness.

Related Work

This section reviews prior work in related areas. It is important to note that we focus on the *size* of the compilation, which few previous papers have addressed. Most previous studies of compilation have focused on a variety of other questions, such as the existence of solutions, or of relevance and necessity of hypotheses (Eiter & Gottlob 1995). Further, to our knowledge, this is the first proposal of *incremental, approximate* compilation algorithms based on solution preference relations.

The complexity of compilation has been addressed in many papers, including (Cadoli *et al.* 2002; 1996; Liberatore 2001). Cadoli et al. (2002) introduce a hierarchy of compilation complexity classes. Ferrara et al. (2007) have proven that for temporal logic model checking, preprocessing cannot reduce complexity, given polynomial space bounds (in the size of the input) on the output.

In the case of compiling using BDDs, the asymptotic worst-case complexity results indicate that many important BDD operations are NP-complete, NP-hard or tend to require unaffordable amounts of memory (Hachtel & Somenzi 2000). The size of a BDD is determined both by the function being represented and the variable ordering. There are functions that have only exponentially sized BDDs (Bryant 1991), although this is not true for the average boolean function. Using variable ordering, one cannot guarantee that a BDD will not be of size exponential in the number of variables, since the problem of finding the best variable ordering is NP-hard (Bollig & Wegener 1996). Further, the approximation problem is also hard, as for any constant $c > 1$ it is NP-hard to compute a variable ordering resulting in an OBDD with a size that is at most c times larger than optimal (Sieling 2002).

In the case of other compilation targets, like DNNF, the size of the DNNF generated for problems in propositional logic, Bayesian networks and constraint satisfaction is exponential in the *treewidth of their interaction graph* (Darwiche 2001; Jensen, Lauritzen, & Olesen 1990).[1]

Notation and Preliminaries

Propositional Logic

We assume a standard propositional logic in this paper. We use a finite alphabet of propositional symbols, $\Sigma = \{z_1, z_2, ..., z_n\}$, using the usual boolean connectives $\wedge, \vee, \neg$, and $\Rightarrow$ for constructing well-formed formulae from Σ. A literal is a propositional symbol or its negation. A clause $z_1 \vee z_2 \vee \cdots \vee z_k \vee \neg z_{k+1} \vee \cdots \vee \neg z_n$ is a disjunction of literals. A clause is Horn (definite Horn) if $k \leq 1$ ($k = 1$). A function (or formula) f is a conjunction of clauses; in this article we assume that a formula is defined over n symbols, unless stated otherwise. The size of a formula f is $|f|$.

A (partial) interpretation γ for Σ is a mapping from (a subset of) Σ to $\{\mathsf{true}, \mathsf{false}\}$, where interpretations can be extended to boolean formulae in the usual recursive way. Γ is the set of all interpretations. A *model* of a formula f is an interpretation γ that maps f to true(written $\gamma \models \mathsf{true}$). $\mathcal{B}$ is the set of all boolean formulae over Σ. The function $atoms : \mathcal{B} \rightarrow 2^{\Sigma}$ maps a formula f to the set of propositional symbols occurring in f.

Prime implicants An *implicant* I of a formula f is a conjunction of literals such that $I \Rightarrow f$. An implicant I is a *prime implicant* if, for every conjunct J obtained by removing one or more literals from I, $J \not\Rightarrow f$. In other words, a prime implicant is a minimal implicant of f. The disjunction Δ of all prime implicants of a formula f is equivalent to f, i.e. Δ preserves the models of f. The size of Δ is the sum of the size of all prime implicants.

Decomposable Negation Normal Form A formula f is in *Negation Normal Form* (NNF) if its literals are joined using only the operators $\vee$ and $\wedge$. The *Decomposable Negation Normal Form* (DNNF) is a subclass of NNF satisfying the decomposability property, which states that the elements of a conjunction do not share atoms. In other words, for every conjunction $\alpha = \alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n$ in a DNNF, it holds that $atoms(\alpha_i) \cap atoms(\alpha_j) = \emptyset$ for $i \neq j$. A DNNF can be represented by a rooted directed acyclic graph, where each leaf node is associated with a literal or truth value, and each intermediate node corresponds to either $\vee$ or $\wedge$. Given this representation, the size of a DNNF is the number of edges in the graph.

Compilability

We address abduction problems whose instances can be expressed as the pair $\langle f, \sigma \rangle$, where f is the fixed part (instance-independent) and σ is the varying part (instance-dependent). Given a problem P, an instance $\langle f, \sigma \rangle$ of P with $f \in \mathcal{B}$ and $\sigma \in \Sigma^*$ and a query function $Q_P : \mathcal{B} \times \Sigma^* \rightarrow \{\mathsf{yes}, \mathsf{no}\}$, we are interested in finding the value $Q_P(f, \sigma)$.

Definition 1 (Compilation)**.** *Given a problem P, an instance $\langle f, \sigma \rangle$ of P and a query function Q_P, we define a* compilation function of f, *$\zeta(f)$, such that there exists a query function $Q'_P : \zeta(\mathcal{B}) \times \Sigma^* \rightarrow \{\mathsf{yes}, \mathsf{no}\}$ and $\forall \langle f, \sigma \rangle \in \mathcal{B} \times \Sigma^*$ it holds that $Q'_P(\zeta(f), \sigma) = Q_P(f, \sigma)$.*[2]

It is clear from the definition that a compilation $\zeta(f)$ preserves the models of f. In order to find the value $Q(f, \sigma)$, we process the fixed part f off-line, thus obtaining $\zeta(f)$, and then we find the value $Q'_P(\zeta(f), \sigma)$. Compilation is worthwhile if answering the query $Q'_P(\zeta(f), \sigma)$ is easier than answering $Q(f, \sigma)$. Note that we do not restrict the time needed to compute the function ζ.

This definition of compilation captures all of the compilation approaches that have been proposed in the literature, such as prime implicants (de Kleer 1986), DNNF (Darwiche 2001), Ordered Binary Decision Diagrams (OBDDs) (Bryant 1992), cluster-trees (Pargamin 2003), and finite-state automata (Amilhastre, Fargier, & Marquis 2002). These approaches all make space/time tradeoffs, i.e., they typically generate compilations, from which a variety of classes of inference can be done in time linear in the size of the compiled representation, although the compilation may require significant space. However, these typical compilation approaches are space intensive.

Preference-Based Compilation

This section introduces the notion of a preference function ϕ over f, and of compiling f with respect to ϕ.

Preference-Based Ranking

Assume that we have a full compilation $\zeta(f)$ such that we can specify the space of solutions, Λ. We can rank-order the solutions of Λ according to a given preference relation ϕ into a set of equivalence classes, in which each equivalence class is characterised by the same ϕ value.

We now consider the case where we use a preference criterion ϕ to guide the choice of compilation target, i.e., we aim to compile the *most-preferred* solutions.

Definition 2 (Preference Function)**.** *A preference function ϕ defined over the solutions Λ of f defines a partial ordering $\succ$*

[1]It turns out that many real-world problems, e.g., the ISCAS benchmark circuits (Brglez, Bryan, & Kozminski 1989), do not have treewidths bounded by some relatively small integer, in order to ensure compact DNNF compilations.

[2]Note that this definition differs from the original definition of (Cadoli *et al.* 2002), where a compilation is assumed to be of size polynomial in $|f|$ and answering the query Q'_P is assumed to require a time polynomial in $|\sigma| + |\zeta(f)|$.

over Λ. We say that solution $\lambda_1 \in \Lambda$ is preferred to solution $\lambda_2 \in \Lambda$, written $\lambda_1 \succ \lambda_2$, if $\phi(\lambda_1) > \phi(\lambda_2)$.

In this article, we assign a preference function (valuation) to variables, and then use this valuation to compute the most preferred solutions.

We now restrict ourselves to the domain of propositional abduction. A propositional abduction problem (PAP) can be defined using a triple $\langle H, \mu, f\rangle$ where H and μ are sets of variables, while f is a propositional formula. H is typically referred to as the hypotheses, and μ as the manifestations. Some important preference-criteria that are widely used in the literature for PAP compilation include:

Subset-inclusion $\boldsymbol{\phi_{\subseteq}}$: $2^H \rightarrow \lambda$; $\lambda_1 \succ \lambda_2$ if $\phi_{\subseteq}(\lambda_1) \subseteq \phi_{\subseteq}(\lambda_2)$.

Cardinality $\boldsymbol{\vartheta}$: $2^H \rightarrow |\lambda|$; $\lambda_1 \succ \lambda_2$ if $\vartheta(\lambda_1) < \vartheta(\lambda_2)$.

Probability $\boldsymbol{Pr}$: $2^H \rightarrow [0,1]$; $\lambda_1 \succ \lambda_2$ if $Pr(\lambda_1) > Pr(\lambda_2)$.

The most preferred solution in a compilation can be computed based on the specific preference relation, using the composition function of the preference relation. One key criterion is that the compilation preserves the preference relations, which is guaranteed for the well-known compilation languages (since they preserve the models of f).

Definition 3 (Preference Preservation)**.** *A compilation $\zeta(f)$ preserves a preference relation ϕ if, given $\langle f, \sigma\rangle$, for any pair of solutions λ_1, λ_2 such that $\lambda_1, \lambda_2 \in \Lambda_f$ and $\lambda_1, \lambda_2 \in \Lambda_{\zeta(f)}$, $\lambda_1 \succ \lambda_2$ is valid in $\zeta(f)$ iff $\lambda_1 \succ \lambda_2$ is valid in f.*

Preferred Approximate Compilation

Since we are interested in compiling only the most-preferred solutions, we define *preferred approximate compilations* as compilations including only a subset of most-preferred solutions:

Definition 4 (Preferred Approximate Compilation)**.** *Given a compilation $\zeta(f)$ with space of solutions $\Lambda_{\zeta(f)}$, $\zeta(f)$ denotes a* preferred approximate compilation of f *if: (1) $\Lambda_{\zeta(f)} \subseteq \Lambda_f$; and (2) $\forall \lambda_1 \in \Lambda_{\zeta(f)}, \forall \lambda_2 \in \Lambda_f \setminus \Lambda_{\zeta(f)}$, it holds that $\lambda_1 \succ \lambda_2$.*

To focus on the most-preferred solutions, we assign a valuation threshold ϱ. We aim to compile all solutions with valuations at least as preferred as ϱ, denoted Λ_ϱ; solutions with valuations less-preferred than ϱ are denoted $\Lambda_{\bar{\varrho}}$.

Definition 5 (ϱ-sound compilation)**.** *Given a preference function ϕ over f and a threshold ϱ, the preferred approximate compilation $\zeta(f)$ denotes a ϱ-*sound compilation of f *if: (1) $\zeta(f)$ preserves the preference relation ϕ; and (2) $\zeta(f)$ contains every solution $\lambda \in \Lambda_\varrho$, i.e. $\Lambda_{\zeta(f)} \supseteq \Lambda_\varrho$.*

In the following, we describe algorithms that incrementally compile based on a threshold ϱ. The algorithms generate partial solutions, i.e., partial interpretations consistent with f, incrementally extending these partial solutions to representations corresponding to complete solutions only if the partial solution at each step is more preferred than ϱ. We call this approach partial-solution extension (PSE). We prove the following result for the set ϕ^* of preference functions $\boldsymbol{\phi_{\subseteq}}$, $\boldsymbol{\vartheta}$, $\boldsymbol{Pr}$.

Theorem 1. *Given a preference function $\phi \in \phi^*$ over f and a threshold ϱ, partial-solution extension (PSE) is guaranteed to generate a ϱ-*sound compilation of f.

Proof: First, it is trivial to show that, given two partial solutions λ_1 and λ_2 which agree on truth-assignments except that λ_2 makes one more assignment than λ_1, $\lambda_1 \succ \lambda_2$.[3] Using this, we can now show that we never exclude a solution $\lambda \in \Lambda_\varrho$. Assume that partial solution λ_1 is less preferred than ϱ; by pruning λ_1, there can be no more-preferred extension λ' to λ_1, since for any extension λ'' of λ_1, $\lambda_1 \succ \lambda''$. Hence, we never include any solutions which are extensions of λ_1. Using an inductive argument, for any partial solution we never omit any solution $\lambda \in \Lambda_\varrho$ through PSE. □

In the next section, we will show PSE algorithms for prime implicants and DNNF; we omit proofs of the correctness of these algorithms due to space limitations, but note that they are special cases of the above theorem. Our notion of preference-based compilation is different to the use of cost functions for DNNF minimal-diagnosis extraction (Darwiche 1998). The cost-function approach aims to compute the most-preferred diagnosis given a complete DNNF and an observation; in our case we are incrementally compiling all solutions which are more preferred than a given threshold ϱ. One outcome of this difference is that, whereas using probabilistic cost-functions can prune a complete DNNF such that valid solutions may be lost (Darwiche 1998), the threshold-based incremental compilation guarantees that no solutions $\lambda \in \Lambda_\varrho$ will fail to be included in the approximate compilation, as shown in Theorem 1.

Incremental Prime Implicant Generation

In this section we present an algorithm that incrementally generates ϱ-sound compilations of a formula f, using prime implicants as target representation. The algorithm is based on the PRIME algorithm, proposed by Shiny and Pujari (Shiny & Pujari 2002); we modified the original algorithm in order to generate approximate compilations. The algorithm employs a matrix representation of a formula f, i.e., a binary matrix M whose columns correspond to clauses and rows correspond to literals. The generic input of the algorithm is the submatrix $M(S,T)$, which is obtained from M considering only a subset S of the clauses and ignoring a subset T of the literals. In order to compute the prime implicants of $M(S,T)$, the matrix is split into two submatrices, and the prime implicants of the submatrices are computed recursively and merged together, yielding the set of prime implicants of the original matrix.

The algorithm is presented in figure 1. Lines 2–7 represent the recursion base cases. The rest of the algorithm contains the recursive calls and the merging of prime implicants. This is carried out according to the theorems presented by Shiny and Pujari. Note that not all prime implicants are added to the set P: the procedure ADD-IMPLICANT ensures that only those prime implicants with valuation exceeding the threshold ϱ are included in P, and discards partial solutions less preferred than ϱ.

[3]We call λ_2 an extension of λ_1.

```
PRIME-APPROX($M(S,T), \varrho$)
1  if $T = \emptyset$
2     then return $\emptyset$
3  if $M(S,T)$ has only one column
4     then return SINGLE-COLUMN($M$)
5  if $M(S,T)$ has only one row
6     then return SINGLE-ROW($M$)
7  $r \leftarrow$ most frequent literal in $T$
8  $S_r \leftarrow$ clauses of $S$ containing $r$
9  $P_1 \leftarrow$ PRIME-APPROX($M(S - S_r, T \cup \{r\}), \varrho$)
10 $P_2 \leftarrow$ PRIME-APPROX($M(S_r, T \cup \{r\}), \varrho$)
11 for all $p \in P_1$
12      do if $E(p, S_r) = S_r$
13            then ADD-IMPLICANT($P, p, \varrho$)
14            else ADD-IMPLICANT($P, p \cup \{r\}, \varrho$)
15 for all $q \in P_2$
16      do if $E(q, S - S_r) = S - S_r$
17            then ADD-IMPLICANT($P, q, \varrho$)
18                 $P_2 \leftarrow P_2 - \{q\}$
19            else for all $p \in P_1, q \in P_2$
20                    do ADD-IMPLICANT($P, p \cup q, \varrho$)
21 Remove all subsumed paths in $P$
22 return $P$
```

```
ADD-IMPLICANT($P, p, \varrho$)
1  if $\phi(p) \geq \varrho$
2     then $P \leftarrow P + \{p\}$
```

Figure 1: PRIME algorithm adapted to approximate compilation.

Incremental DNNF Generation

We present here a PSE algorithm to generate ϱ-sound DNNF compilations of a formula f, based on the algorithm proposed by Darwiche (2001).

The DNNF algorithm, shown in figure 2, takes as inputs a node T in the decomposition tree (i.e. a binary tree whose leaves correspond to the clauses in the formula), and an instantiation of variables α and a threshold ϱ. The DNNF representation is computed recursively, with each recursive call carried out only if the instantiation exceeds the threshold ϱ; therefore, the algorithm does not compute portions of the DNNF containing solutions less preferred than ϱ.

The algorithm can be made more efficient by associating a cache structure to the nodes of the decomposition tree. This improved version of the algorithm is reported in figure 3. Note that in this case instantiations with different valuations can be associated with the same cache entry; in general, this might lead to a non ϱ-sound approach, because the algorithm might return the same DNNF structure for instantiations with different valuations.

In order to preserve ϱ-soundness, we order instantiations by their valuation. Thus, we calculate and store in the cache the DNNF relative to the most preferred instantiation α; we then return the same DNNF for all subsequent (less preferred) instantiations that match with α. Therefore, the resulting DNNF is always a superset of the DNNF that would result using the non-caching algorithm.

Another refinement consists in pruning the DNNF with

```
DNNF($T, \alpha, \varrho$)
1  if $T$ is a leaf node with clause $\varphi$
2     then $\gamma \leftarrow$ CONDITION($\varphi, \alpha$)
3     else $A \leftarrow atoms(T_l) \cap atoms(T_r) - atoms(\alpha)$
4          for $\beta \in$ {instantiations of $A$ such that $\phi(\alpha \wedge \beta) \geq \varrho$}
5              do $\gamma \leftarrow \gamma \vee$ (DNNF($T_l, \alpha \wedge \beta, \varrho$)$\wedge$
                      DNNF($T_r, \alpha \wedge \beta, \varrho$) $\wedge \beta$)
6  return $\gamma$
```

Figure 2: DNNF algorithm adapted to approximate compilation. CONDITION replaces literals in φ with true(false) if they are consistent (inconsistent) with α. T_l and T_r respectively denote the left and right children of T.

```
DNNF2($T, \alpha, \varrho$)
1  $\psi \leftarrow subinst(\alpha, atoms(T))$
2  if $cache_T(\psi) \neq NIL$
3     then return DNNF-PRUNE($cache_T(\psi), \varrho / \phi(\alpha)$)
4  if $T$ is a leaf node with clause $\varphi$
5     then $\gamma \leftarrow$ CONDITION($\varphi, \alpha$)
6     else $A \leftarrow atoms(T_l) \cap atoms(T_r) - atoms(\alpha)$
7          $I \leftarrow$ {instantiations of $A$ ordered by their valuation}
8          for $\beta \in I$ such that $\phi(\alpha \wedge \beta) \geq \varrho$
9              do $\gamma \leftarrow \gamma \vee$ (DNNF2($T_l, \alpha \wedge \beta, \varrho$)$\wedge$
                      DNNF2($T_r, \alpha \wedge \beta, \varrho$) $\wedge \beta$)
10 $cache_T(\psi) \leftarrow \gamma$
11 return $\gamma$
```

Figure 3: Improved DNNF algorithm using caching and pruning. $subinst(\alpha, atoms(T))$ is the subset of α pertaining to $atoms(T)$.

the procedure DNNF-PRUNE. This procedure takes a DNNF γ and a threshold ϱ as inputs, and removes solutions less preferred than ϱ. Nodes of γ are removed according to their valuations, where the valuation of an *and*-node is calculated using the composition function of the preference relation, while the valuation of an *or*-node is equal to the valuation of its most-preferred child. In case the root of γ is an *or*-node, the procedure removes children less preferred than the threshold. In case the root of γ is an *and*-node, the procedure removes children that never occur in above-the-threshold combinations, where each combination t contains all the leaf-node children of the root, as well as one child of each *or*-node-child of the root, and $\phi(t)$ is calculated using the composition function of the preference relation.

Note that the algorithm DNNF-PRUNE removes only a subset of solutions less preferred than ϱ. Therefore, DNNFs computed with algorithm DNNF2 include more solutions than those computed with algorithm DNNF; however, DNNF2 is significantly more efficient than the non-caching, non-pruning algorithm.

Empirical Results

We have implemented the above algorithms and carried out experiments that compare the prime implicant and DNNF approaches, measuring the size and coverage trade-off of

```
DNNF-PRUNE($\gamma, \varrho$)
1  if $\gamma$ is not a leaf
2     then for each child $c$ of $\gamma$
3           do DNNF-PRUNE($c, \varrho$)
4  if $\gamma$ is an or–node
5     then for each children $c$ of $\gamma$
6           do if $\phi(c) < \varrho$
7                then remove $c$
8  elseif $\gamma$ is an and–node
9     then $N \leftarrow \emptyset$
10         for each child $c$ of $\gamma$
11             do if $c$ is an or-node
12                  then $N \leftarrow N + \{$children of $c\}$
13                  else $N \leftarrow N + \{c\}$
14         for each node $j$ in $N$
15             do $remove[j] \leftarrow$ TRUE
16         $S \leftarrow \{$ all combinations of nodes in $N$ $\}$
17         for each combination $t$ in $S$ such that $\phi(t) \geq \varrho$
18             do for each node $j$ in $t$
19                  do $remove[j] \leftarrow$ FALSE
20         for each node $j$ in $N$ such that $remove[j] =$ TRUE
21             do remove $j$
22 return $\gamma$
```

Figure 4: Pruning a DNNF to remove solutions below ϱ.

compilations. All our experiments were run with a set of formulae representing a suite of digital circuits. The digital circuits were generated randomly by a circuit generator program (Provan & Wang 2007), such that the circuits have properties similar to those of the ISCAS circuit benchmarks (Brglez, Bryan, & Kozminski 1989). Each circuit is represented by a formula f defined over a set of boolean variables $V = H \cup K$, where a variable $h_i \in H$ denotes the health of gate i, and variables in K denote input/output signals. To each variable $h_i \in H$ we assign a probability valuation. We defined the probability that gate i is functioning, $Pr(h_i = 1) = p$, by randomly sampling the value for p from a uniform distribution over ranges $R = [i, 1]$, with $0 \leq i < 1$. The failure probability, $Pr(h_i = 0)$, is the complement $1 - p$.

Figure 5 shows how the number of solutions varies with different thresholds and weight ranges. Depending on the range R we obtain distributions that decrease more or less gradually to 0. The number of preferred prime implicants decreases as $i \rightarrow 1$ for all threshold values $\varrho > 0$. We can control the number of solutions (and thus the size and coverage of the approximate compilation) by selecting appropriate (i, ϱ) combinations. In the following experiments we used a range $R = [0.99, 1]$; however, our experimental data indicate that we obtain similar results using different ranges.

Figure 6 shows the size of DNNF compilations for input formulae with 28 to 64 clauses, using various valuation thresholds. The size of compilations grows exponentially in the size of input; however, approximate compilations can save a significant amount of space when compared to full compilations. We obtain similar results with prime implicant compilations.

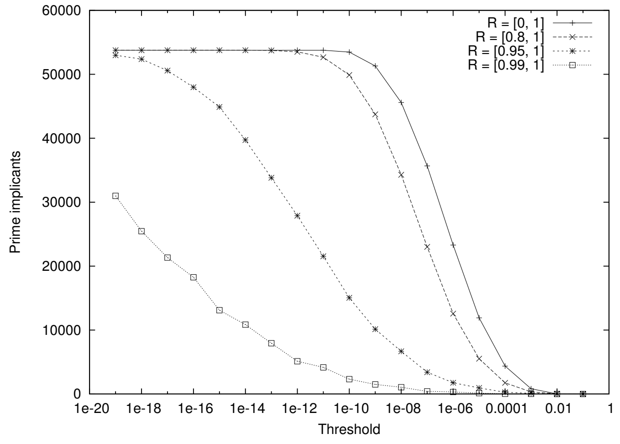

Figure 5: Number of prime impicants of a formula f encoding a 16-gate circuit; each graph refers to a different probability range.

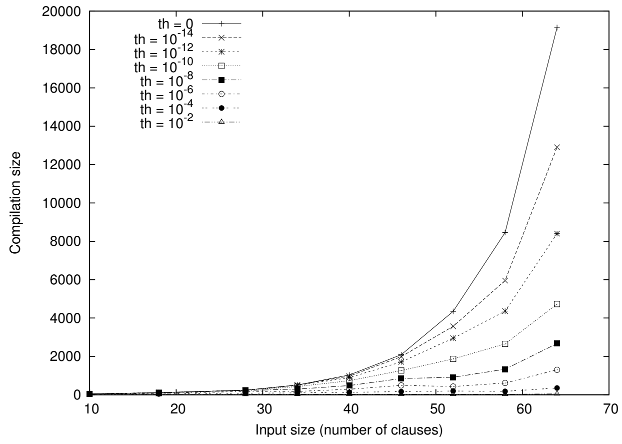

Figure 6: Size of DNNF compilations using various threshold values. Compilations were obtained using the caching and pruning algorithm.

Figure 7 compares the size of prime implicant and DNNF compilations for input formulae with 10 to 74 clauses, using threshold $\varrho = 10^{-10}$. We can see that DNNF compilations are more succint than prime implicants, achieving in some cases a memory saving of 50%.

Figure 8 shows the Cumulative Distribution Functions (CDFs) of solution distributions for circuits with 10 to 18 gates, encoded in formulae with 58 to 108 clauses. These graphs refer to prime implicant compilations; we obtain similar results using DNNF. Here we see that, using approximate compilations, we obtain significant solution coverage yet require just a fraction of the memory of the full compilation. In particular, we obtain up to 5 orders-of-magnitude space savings, while maintaining $> 90\%$ query-coverage; moreover, for all circuits very high coverage ratio ($> 99\%$) is possible with 3-4 orders-of-magnitude space savings. Note that the space savings increase with circuit (or formula) size, meaning that this approach scales well with the size of f.

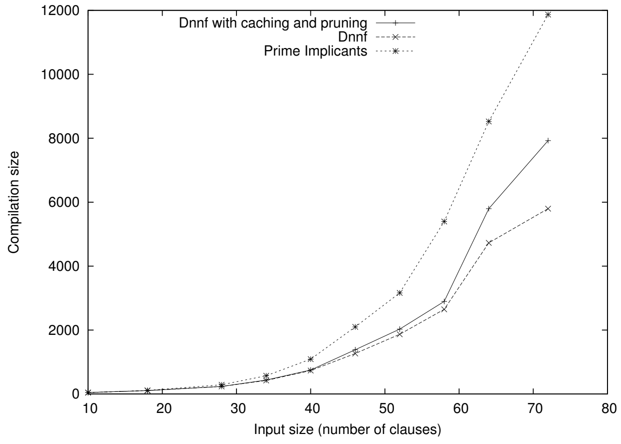

Figure 7: Comparing the size of prime implicant and DNNF compilations using threshold $\varrho = 10^{-10}$.

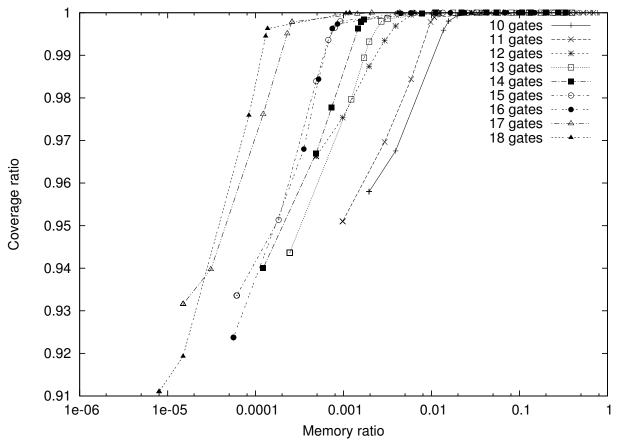

Figure 8: CDFs of solution distributions for prime implicant compilations.

Summary and Conclusions

We have proposed a Partial-Solution Extension approach for incrementally compiling the most-preferred solutions of boolean formulae. The approach focuses on the size and solution coverage of approximate compilations, as well as on the memory required to compute them. We have provided a theoretical analysis and presented PSE algorithms for prime implicants and DNNF target languages. All the algorithms are ϱ-sound, i.e. they compute approximate compilations that include all solutions more preferred than the valuation threshold ϱ. Experimental results have been reported that empirically demonstrate the space efficiency of approximate compilations, showing that we can achieve orders-of-magnitude space savings while covering the majority of solutions.

References

Amilhastre, J.; Fargier, H.; and Marquis, P. 2002. Consistency restoration and explanations in dynamic CSPs Application to configuration. *Artif. Intell.* 135(1-2):199–234.

Bollig, B., and Wegener, I. 1996. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers* 45(9):993–1002.

Brglez, F.; Bryan, D.; and Kozminski, K. 1989. Combinational profiles of sequential benchmark circuits. In *IEEE International Symposium on Circuits and Systems*, volume 3, 1929–1934.

Bryant, R. E. 1991. On the complexity of vlsi implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Trans. Comput.* 40(2):205–213.

Bryant, R. 1992. Symbolic boolean manipulation with ordered binary-decision diagrams. volume 24. 293–318.

Cadoli, M.; Donini, F.; Liberatore, P.; and Schaerf, M. 1996. Feasibility and unfeasibility of off-line processing. *Proceedings of the Fourth Israeli Symposium on Theory of Computing and Systems (ISTCS96)* 100–109.

Cadoli, M.; Donini, F.; Liberatore, P.; and Schaerf, M. 2002. Preprocessing of Intractable Problems. *Information and Computation* 176(2):89–120.

Chandra, A., and Markowsky, G. 1978. On the number of prime implicants. *Discrete Mathematics* 24:7–11.

Darwiche, A. 1998. Model-Based Diagnosis using Structured System Descriptions. *Journal of Artificial Intelligence Research* 8:165–222.

Darwiche, A. 2001. Decomposable negation normal form. *Journal of the ACM (JACM)* 48(4):608–647.

de Kleer, J. 1986. An assumption-based TMS. *Artif. Intell.* 28(2):127–162.

Eiter, T., and Gottlob, G. 1995. The complexity of logic-based abduction. *Journal of the ACM (JACM)* 42(1):3–42.

Ferrara, A.; Liberatore, P.; and Schaerf, M. 2007. Model Checking and Preprocessing. In *Proc. AI*IA 2007: Artificial Intelligence and Human-Oriented Computing*. Springer.

Hachtel, G. D., and Somenzi, F. 2000. *Logic Synthesis and Verification Algorithms*. Norwell, MA, USA: Kluwer Academic Publishers.

Jensen, F.; Lauritzen, S.; and Olesen, K. 1990. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly* 4:269–282.

Liberatore, P. 2001. Monotonic reductions, representative equivalence, and compilation of intractable problems. *Journal of the ACM (JACM)* 48(6):1091–1125.

Pargamin, B. 2003. Extending cluster tree compilation with non-boolean variables in product configuration. In *Proceedings of the IJCAI-03 Workshop on Configuration*.

Provan, G., and Wang, J. 2007. Evaluating the adequacy of automated benchmark model generators for model-based diagnostic inference. In *Proceedings of IJCAI-07*.

Shiny, A., and Pujari, A. 2002. Computation of Prime Implicants using Matrix and Paths. *Journal of Logic and Computation* 8(2):135–145.

Sieling, D. 2002. The nonapproximability of obdd minimization. *Inf. Comput.* 172(2):103–138.