Approximate Compilation for Embedded Model-based Reasoning

Barry O'Sullivan[*] and **Gregory M. Provan**[†]
Department of Computer Science, University College Cork, Ireland
{b.osullivan|g.provan}@cs.ucc.ie

Abstract

The use of embedded technology has become widespread. Many complex engineered systems comprise embedded features to perform self-diagnosis or self-reconfiguration. These features require fast response times in order to be useful in domains where embedded systems are typically deployed. Researchers often advocate the use of compilation-based approaches to store the set of environments (resp. solutions) to a diagnosis (resp. reconfiguration) problem, in some compact representation. However, the size of a compiled representation may be exponential in the treewidth of the problem. In this paper we propose a novel method for compiling the most preferred environments in order to reduce the large space requirements of our compiled representation. We show that approximate compilation is an effective means of generating the highest-valued environments, while obtaining a representation whose size can be tailored to any embedded application. The method also provides a graceful way to tradeoff space requirements with the completeness of our coverage of the environment space.

Introduction

Model-based reasoning, and its application to diagnosing or (re)configuring complex systems, is a computationally intensive task; it is an NP-hard problem in general. For product configuration, several researchers have explored the use of compilation methods (Amilhastre, Fargier, & Marquis 2002; Subbarayan 2005). In general, knowledge compilation separates the computational task into an instance-dependent and an instance-independent part (Cadoli & Donini 1997; Darwiche & Marquis 2002b). Compiling the instance-independent part into a structure Θ, corresponding to a compact representation of its solution space, can lead to a speedup in online inference. This is because the computational task is linear in the size of the compiled form.

If the knowledge-base we wish to compile is represented as a constraint satisfaction problem (CSP), the space required to compile it is exponential in the tree-width of the CSP's constraint graph (Bodlaender 1997). For real-world problems the size of the compiled form can often be too large for practical inference.

[*] Also 4C and CTVR. Supported by SFI grant 03/CE3/I405.

[†] Supported by SFI grant 04/IN3/I524.

In this paper we propose a technique to compile a sound but incomplete representation of a knowledge-base that can smoothly trade off the space required to store all (partial) solutions, which we refer to as feasible environments, for the completeness of the coverage of the compiled representation. This approach is suitable for embedded real-time applications, where speed of response is critical, but space is highly constrained. In order to effectively make such trade-offs we compile only the highest-valued environments. The valuation used can reflect the likelihood of an environment being selected, its cost or some historical notion of its preference. For example, in diagnosis the valuation can come from the prior failure probabilities of the system components, which provide relative likelihoods of faults occurring in subsets of components.

We define two measures to determine the "quality" of an approximate compilation. First, η measures the relative fraction of important environments that are generated by the approximate compilation Θ_φ, relative to the full compilation Θ. Second, λ measures the proportion of the space the approximate compilation Θ_φ requires, relative to that of Θ.

We model probabilistic information, e.g., about the relative likelihood of customers choosing various options over past interactions in configuration, using a probabilistic CSP. We then empirically study the kinds of tradeoffs that are possible with probabilistic CSPs. We show that one can maintain the majority of most-likely solutions by compiling a relatively small percentage of the total number of solutions, given a naive method of compiling all solutions. We also examine the impact of approximate compilation on using prime implicates and DNNF as the compilation targets. We show that approximate compilation is an effective means of generating the highest-valued solutions that fit within a prespecified amount of memory.

Preliminaries

We are interested in the approximate compilation of NP-Hard problems such as the constraint satisfaction problem.

Definition 1 (Constraint Satisfaction Problem) *A constraint satisfaction problem (CSP) is a 3-tuple $P \hat{=} \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ where $\mathcal{X}$ is a finite set of variables $\mathcal{X} \hat{=} \{x_1, \ldots, x_n\}$, $\mathcal{D}$ is a set of finite domains $\mathcal{D} \hat{=} \{D(x_1), \ldots, D(x_n)\}$ where the the domain $D(x_i)$*

is the finite set of values that variable x_i can take, and a set of constraints $\mathcal{C} \doteq \{c_1, \ldots, c_m\}$. Each constraint c_i is defined by the ordered set $var(c_i)$ of the variables it involves, and a set $sol(c_i)$ of allowed combinations of values. An assignment of values to the variables in $var(c_i)$ satisfies *c_i if it belongs to $sol(c_i)$. A* feasible solution *to a constraint satisfaction problem is an assignment of a value from its domain to each variable such that every constraint in $\mathcal{C}$ is satisfied. We denote the* set of feasible solutions *to P as $sols(P)$.*

We will refer to a (partial) solution as an environment. An m-ary environment has m variables instantiated.

Definition 2 (Feasible Environment) *A feasible environment E of P is a subset of a feasible solution of P, i.e. $E \subseteq S \in sols(P)$. $\mathcal{H}$ is the set of all feasible environments of P, i.e. $\mathcal{H} = \bigcup_{S \in sols(P)} \{E | E \subseteq S\}$.*

We associate valuations with environments in order to reason about those that are most preferred. A valuation denotes the importance, preference or probability of an environment. We represent the valuation of an environment E as $\upsilon(E)$.

In applying compilation to the problem we proceed as follows. Given a problem P, intensionally defining the set of feasible environments, we replace P by an equivalent, but computationally more efficient, compiled representation Θ. Thus given an entailment problem for determining consequences α of $\mathcal{C}$, i.e., $\mathcal{C} \cup \mathcal{K} \models \alpha$, we can compile $\mathcal{C}$ into C' and express this as $\mathcal{C}' \models \alpha \vee \bigvee_{\xi \in \mathcal{K}} \neg\xi$, where $\mathcal{K}$ is the set of varying constraints. In this paper we take $\mathcal{K}$ to be a set of unary constraints that we call assumptions.

Existing approximate compilation techniques typically weaken the problem representation. For example, papers such as (Selman & Kautz 1996) and (del Val 1995) present studies of approximating propositional and first-order formulae using Horn lowest upper bound (LUB) representations, as well as their generalisations. In contrast, we are interested in using the valuation function υ to compile a subset of *most preferred* (feasible) environments. This is similar to the penalty logic framework introduced in (Darwiche & Marquis 2002a), except that in this case we compile only a *subset* of the most preferred environments based on $\mathcal{C} \cup \mathcal{H}$, using a threshold, φ. In other words, we compile all environments such that their valuation is at least as good as a given bound. For example, in a weighted CSP context, where better solutions have lower valuations, we would compile all environments whose valuation $\upsilon(E) \leq \varphi$.

This approach is a general one, and can be applied to several compilation methods. For example, with regard to the prime implicates (or labels) computed by an ATMS (de Kleer 1986) or consequence generation (Darwiche 2002), we ensure that no label (consequence) will have valuation worse than a bound φ. Whether the approach will actually result in good coverage/space tradeoffs depends on the valuation and the compilation target, as we discuss later.

Threshold-Based Prime Implicates

This section describes our notion of threshold-based compilation in terms of prime implicates. Analogous definitions can be specified for other forms of compilation. Here we represent our CSP P in terms of an equivalent propositional logic theory Δ.

Definition 3 (Prime Implicate) *An implicate of Δ is a clause β such that $\Delta \models \beta$. A prime implicate of Δ is an implicate of Δ that is minimal with respect to $\models$.*

Computing the set of prime implicates of Δ, $PI(\Delta)$, generates an equivalent theory Δ', which is important in that entailment of a clause α can be determined in linear time in the size of $\Delta' \cup \alpha$. This is because we can check if a query β is entailed by Δ since $\Delta \models \beta$ iff $\exists\gamma \in PI(\Delta)$ that is a sub-clause of β.

We extend this notion of computing (minimal) sub-clauses of queries in a compilation to that of computing a sound but incomplete set of sub-clauses, which we call threshold implicates.

Definition 4 (Threshold Implicate) *A threshold implicate γ of a theory Δ is a clause γ such that $\Delta \models \gamma$ and $\upsilon(\gamma) \leq \varphi$ for some threshold valuation φ. A threshold prime implicate of Δ is a threshold implicate of Δ which is minimal with respect to $\models$.*

If $PI(\Delta)$ is the full set of prime implicates of Δ, and $PI_\varphi(\Delta)$ is the threshold prime implicates of Δ, then $PI_\varphi(\Delta) \subseteq PI(\Delta)$ and $PI(\Delta) \models PI_\varphi(\Delta)$ (but the converse does not necessarily hold).

In other words, the threshold prime implicates are a subset of the prime implicates, restricted only by removing prime implicates that exceed the threshold valuation φ.

An Example

We adopt the T-shirt configuration example of (Subbarayan 2005). This example addresses configuring 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 configuration requirements: (1) if we choose the MIB print then the color black has to be chosen as well, and (2) if we choose the small size then the STW print (including a big picture of a whale) cannot be selected as the large whale does not fit on the small shirt. We can represent this configuration problem as the following CSP:

- variables $\mathcal{X} = \{x_1, x_2, x_3\}$ denote colour, size and print.
- domains $D(x_1) = \{black, white, red, blue\}$, $D(x_2) = \{small, medium, large\}$, and $D(x_3) = \{MIB, STW\}$.
- constraints $\mathcal{C} = \{c_1, c_2\}$, where $c_1 \equiv (x_3 = MIB) \Rightarrow (x_1 = black)$; and $c_2 \equiv (x_3 = STW) \Rightarrow (x_2 = small)$.

There are $|D(x_1)| \times |D(x_2)| \times |D(x_3)|$ = 24 possible consistent assignments, of which 11 are feasible configurations.

Now assume that we have collected customer data on the probabilities $\upsilon(x_i)$ of a customer choosing x_i (colour, size and print). If each choice is independent of the other, $\upsilon(x_i)$ takes the following distributions, given the domains defined above: $\upsilon(x_1) = (.5, .1, .2, .2)$; $\upsilon(x_2) = (.1, .3, .6)$; $\upsilon(x_3) = (.7, .3)$. For example, $\upsilon(x_3) = (.7, .3)$

	small	medium	large
Black,MIB	.035	.105	.210
Black, STW	-	.045	.090
White,STW	-	.009	.018
Red,STW	-	.018	.036
Blue,STW	-	.018	.036

Figure 1: Weighted solutions to the T-shirt example. Inconsistent solutions are shown with a –.

may mean that, based on past sales history 70% of customers selected the MIB print, and 30% selected the STW print.

There are many ways that we can compile this problem so that simple table-lookup can identify consistent configurations for a customer. For example, we could compile all complete feasible solutions, or we could compile all feasible partial solutions. If we compile all full solutions (i.e., ternary environments with values for x_1, x_2, x_3), Figure 1 shows the the likelihoods of the full set of possible consistent solutions, which is our target compilation Θ. The cumulative valuation of these solutions is $\upsilon(\Theta) = 0.62$.

Our approach focuses on compiling only a subset of the most-likely solutions. We introduce a threshold φ, such that we will not generate any solution s with valuation $\upsilon(s) < \varphi$.[1] For example, if $\varphi = 0.1$, then we will compile only 2 solutions ({Black, MIB, medium}, {Black, MIB, large}), which together have total valuation of 0.315, out of the valuation of all solutions of 0.62.

We use two parameters to measure the "quality" of an approximate compilation, with respect to the target compilation Θ. The first measure, λ, is the proportion of memory relative to that of Θ, which is $\frac{2}{11} = 0.18$ in this case. The second measure, η, is the proportion of the cumulative valuation relative to that of Θ, which is $\frac{0.315}{0.62} = 0.508$.

If instead we compile all feasible partial solutions, then a full compilation generates 30 environments: 9 unary, 10 binary and 11 ternary environments. If we introduce a threshold $\varphi = 0.1$, then we will compile only 16 of these environments, with compilation parameters $\lambda = \frac{16}{30} = 0.53$ and $\eta = \frac{4.685}{5.26} = 0.89$.

We use the threshold valuation φ to induce an approximate compilation with particular (λ, η) parameters. In other words, by varying φ we can compile a representation that requires some fraction of the memory of the full compilation, trading off coverage of solutions in the process.

Valuation-Based Compilation Analysis

In this paper we focus on a valuation widely used in areas of cost-based abduction, such as diagnosis (Console, Portinale, & Dupré 1996). In this valuation we assign a probability p to each assumption: $Pr : \mathcal{K} \rightarrow [0, 1]$. The valuation of an environment $E \in \mathcal{H}$ is given by $Pr(E) = \prod_{H \in E} \Pr(H)$, where we assume that all assumptions are independent, such that we can compute the joint probability $Pr(E)$ by the products of the probabilities for $H \in E$.

The semantics of this valuation are that we can interpret a probability as a degree of likelihood of choosing an assignment. Starting from a maximum valuation of 1 (which represents a product that violates none of the restrictions of $\mathcal{H}$) all valuations less than 1 correspond to solutions which are increasingly less-preferred (i.e., less likely to be chosen). Hence, our objective is to compute *maximum*-probability solutions. In the case of probabilistic CSPs, the valuation will guarantee the we compile the *maximum*-probability solutions. If the valuation corresponds to a utility measure, then our derived approximate compilation will include the *maximum*-utility solutions.

We now examine the valuations for compilations, and in particular the tradeoffs of relative size of the compilation version the total relative value of the compilation.

We pose an optimisation task, that of computing the least-cost feasible environments η^* up to a threshold φ. In other words, we compile all feasible environments $E \in \eta^*$ such that $\upsilon(E) \leq \varphi$. The objective of our approximate compilation is to provide coverage for a fixed percentage of possible queries. We use the following notation for specifying the relative value of a partial compilation:

Definition 5 (Environment Set Valuation) *The valuation associated with a environment set $\mathcal{H}$ (or equivalently, with a complete compilation Θ of $\mathcal{H}$), is given by the sum over all valuations: $\upsilon(\Theta) = \sum_{E \in \bigcup_{(h \in \mathcal{H})} 2^h} \upsilon(E)$.*

Definition 6 (Partial Environment Set Valuation) *The valuation of a partial compilation Θ_φ with valuation threshold φ is: $\upsilon(\Theta_\varphi) = \sum_{E \in \bigcup_{(h \in \mathcal{H})} 2^h} \{\upsilon(E) | \upsilon(E) \leq \varphi\}$.*

We can use these notions to define a key parameter for our experimental analysis, the valuation coverage ratio.

Definition 7 (Valuation Coverage Ratio) *We define the* valuation coverage ratio η *of a partial compilation Θ_φ, with valuation threshold φ, as the fraction of the complete system valuation provided by Θ: $\eta = \frac{\upsilon(\Theta_\varphi)}{\upsilon(\Theta)}$.*

The second key parameter in which we are interested is the relative memory of a partial compilation, which we can formalize as follows. Let $|\Theta|$ be a measure for the size of the original compiled CSP, and $|\Theta_\varphi|$ be a measure for the size of the CSP compiled based on valuation threshold φ. For simplicity, we assume that all feasible environments (solutions) take up equal memory, and define a ratio based only on the relative number of solutions.

Definition 8 (Memory Reduction Ratio) *The memory reduction of a partial compilation Θ_φ, with respect to compiling the full CSP into Θ, is given by $\lambda = \frac{|\Theta_\varphi|}{|\Theta|}$.*

Applicability of Different Valuations

One of the key issues for valuation-based compilation is the compilation methods for which it is applicable. To describe that, we introduce a notion of valuation monotonicity.

Definition 9 (Valuation Monotonicity) *Given two environments α and β such that $\alpha \subseteq \beta$, a valuation υ is monotonic if $\upsilon(\alpha) \geq \upsilon(\beta)$.*

[1] In probabilistic valuations the preferred valuations have high probabilities.

A wide range of valuations are monotonic, including minimum-cardinality, probability, and Order-of-Magnitude probability (Spohn 1988).

A threshold-bounded compilation is guaranteed to be more space-efficient than a full compilation, which occurs when $\lambda \leq 1$, when the following conditions hold:

Lemma 1 *Given a compilation method that can explicitly represent feasible environments and a monotonic valuation* υ, $\lambda \leq 1$.

There are several important compilation approaches that can explicitly represent (minimal) feasible environments, including prime implicates, DNNF, or simply enumerating feasible environments.

Example 1 Consider the case of DNNF for the following boolean formulae $\Delta : A \wedge X \Rightarrow \bar{Y},\ A \wedge \bar{X} \Rightarrow Y,\ B \wedge Y \Rightarrow \bar{Z},\ B \wedge \bar{Y} \Rightarrow Z$. We can compile these formulae holding X, Y, Z to be the fixed part and A, B to be the variable part, to obtain the DNNF structure (Darwiche 2002) shown in Figure 2. In this structure, the nodes denote either $\wedge/\vee$ symbols or literals. The structure in Figure 2(a) encodes four possible solutions: $\{\}, \bar{A}, \bar{B}, \bar{A} \wedge \bar{B}$. If we introduce a threshold on solution-cardinality of 1 (i.e., we want solutions with no more than 1 assumable), then we will prune the solution $\bar{A} \wedge \bar{B}$; the pruned DNNF is shown in Figure 2(b). Note that, since the size of this graphical representation of DNNF has size determined by the number of edges, we have reduced the size of the DNNF from 14 to 12 edges, while reducing the number of solutions encoded from 4 to 3.

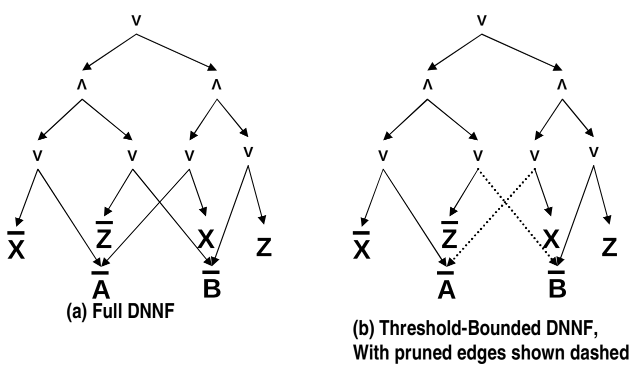


Figure 2: DNNF compilations for a simple boolean formula. (a) shows the full DNNF representation, while (b) shows the representation with a cardinality-2 solution pruned by removing two (dashed) edges from the structure.

Analysis of Different Valuations

This section analyses the impact of two parameters on the size and effectiveness of a partial compilation: (1) *valuation differential*, the difference in degree of preference between any two different valuations; and (2) *valuation distribution*, the relative proportion of different preferences. For example, in configuration, a customer may assign a preference ordering in which some assumptions are more preferred than others; in that case the distribution specifies the *relative proportions* of highly preferred to weakly preferred assumptions, and the differential indicates the difference in *degree* of preference among the assumption valuations.

We study the impact of valuation differential on partial compilation tradeoffs using a parameterised valuation function $\upsilon(c) = \gamma\epsilon^{\kappa(c)}$, for $\epsilon \leq 1, \kappa(c) \in \mathbb{N}^{+}$, and constant γ. If we fix ϵ and vary κ, this function, based on the Order-of-Magnitude probability proposed in (Spohn 1988), approximates a probability distribution. This valuation allows us to vary the difference in valuation between c_i and c_j that have different κ-rankings, e.g., if $\kappa(c_i) = 1 + \kappa(c_j)$. Hence, if we set $\epsilon = 0.1$ then c_i is 10 times more likely than c_j; if we set $\epsilon = 0.01$, c_i is 100 times more likely than c_j.

Figure 3 shows the impact of the value of ϵ on the possible types of tradeoff curves. At one extreme, the value $\epsilon = 1$ produces an equi-loss situation where there is no value to compilation. The benefit of compilation improves as ϵ grows smaller, i.e., as the gap between valuations increases.

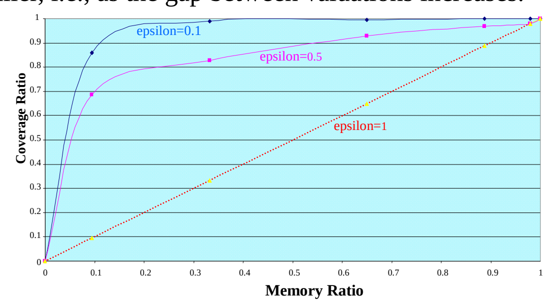
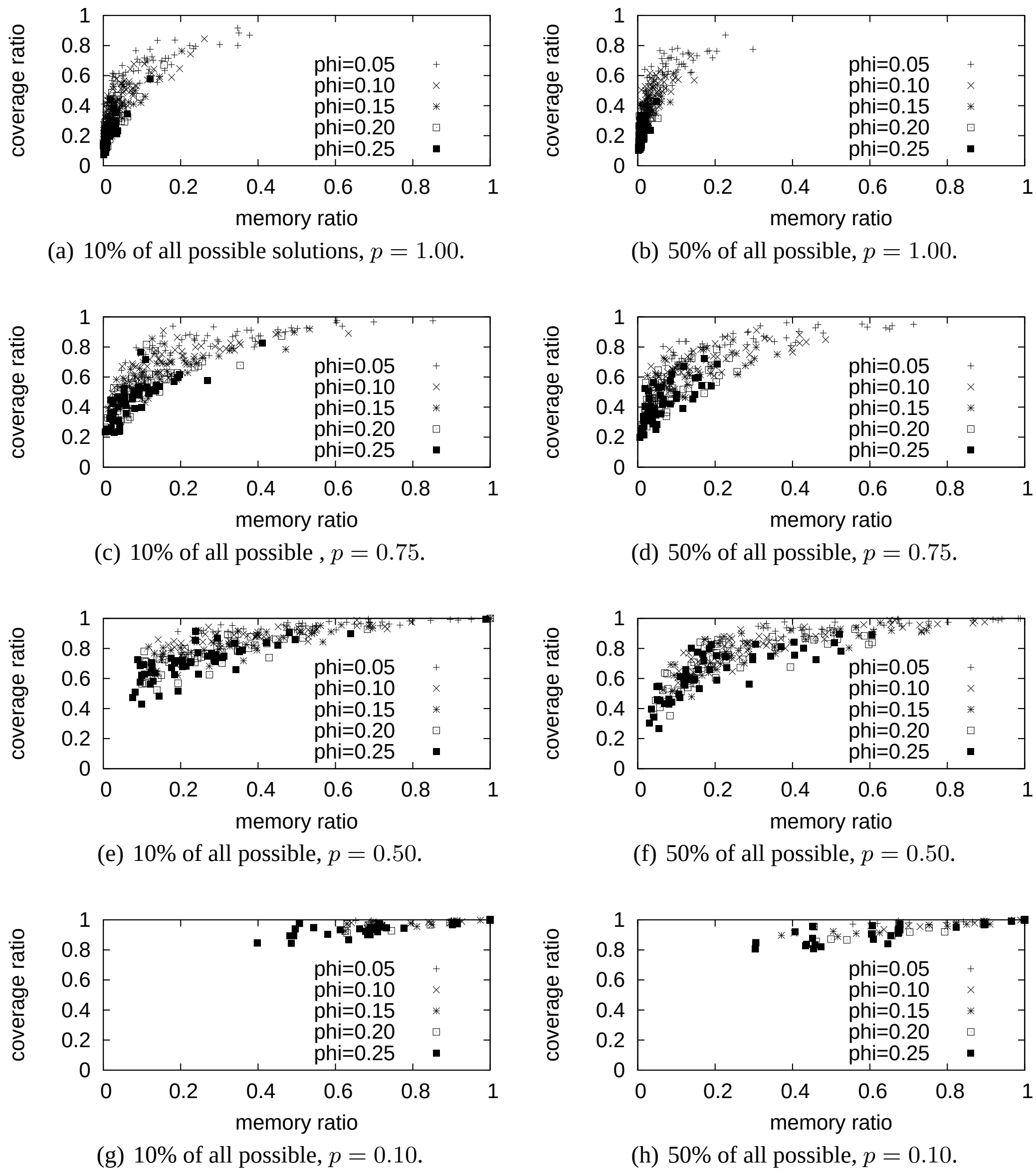


Figure 3: Curves depicting the influence of the valuation differential. In using a parameterised valuation function $\upsilon(c) = \gamma\epsilon^{\kappa(c)}$, we can vary the parameter ϵ to generate different tradeoff curves.

The valuation distribution also affects the relative efficiency of a partial compilation, i.e., having a high query coverage with large reduction in memory. If a valuation is skewed, in the sense that some environments are highly preferred and others are not at all preferred, then we can compute a space-efficient partial compilation. If most environments are relatively equally preferred, then little is gained by partial compilations.

Empirical Evaluation

The objective of our empirical evaluation was to demonstrate that the savings that can be achieved by compiling only the most preferred environments is consistent with the formal analysis presented above. We implemented our approach as an "approximate compiler" in the lazy functional programming language Haskell. We considered the task of compiling a set of solutions of predetermined cardinality in which each one was generated as a set of uniform randomly generated assignments to a set of variables. Our experiments are based on solution sets involving 10 variables, with binary domains, i.e. each variables takes either a 0 or 1 value. For the purposes of the evaluation we generated solution sets containing 10, 20, 30, 40 and 50% of all possible assignments one can generate for 10 variables over a binary domain of values. Note that this should not be seen as an unreasonable number of variables. In order to properly evaluate how well our method performs we compiled the set



(a) 10% of all possible solutions, $p = 1.00$.

(b) 50% of all possible, $p = 1.00$.

(c) 10% of all possible , $p = 0.75$.

(d) 50% of all possible, $p = 0.75$.

(e) 10% of all possible, $p = 0.50$.

(f) 50% of all possible, $p = 0.50$.

(g) 10% of all possible, $p = 0.10$.

(h) 50% of all possible, $p = 0.10$.

Figure 4: Partial compilation results. The parameter p is the proportion of all possible assignments to the variables having a non-1 probability of being a **fault**. We plot coverage ratio (η) on the y-axis and memory ratio (λ) on the x-axis.

of unique partial feasible environments associated with each solution, which is exponential in the number of variables. Of course, in a real context, we would only be interested in the environments better than some threshold and so, for suitably low values of our threshold, we can often avoid the worst-case complexity of the method considerably.

We interpret the randomly generated solution set as the set of solutions to a probabilistic constraint satisfaction problem. We assigned probabilities to the domain values of each variable based on a uniform randomly generated probability. We interpret each non-1 probability as representing the probability of the corresponding assignment being a **fault**. We varied the proportion, p, of all possible domain values that had a non-1 probability associated with them. Therefore, in a strict sense, our probabilities should be interpreted as likelihoods, however the distinction is not important here. For experimental purposes we selected p from $[0.10, 0.25, 0.50, 0.75, 1.00]$; however, for space reasons, we omit the results for $p = 0.25$.

For each combination of parameters we generated 50 solution sets and compiled them. For each compilation step we compute (1) η (as the ratio of the sum of the valuations of the environments compiled divided by the total valuation of all unique environments in the original solution set), and (2) λ (as the ratio of the number of environment in compiled divided by the total number of unique environments in the original solution set).

Our measure of relative memory (λ) is *very conservative*. No effort is made to compute a compact representation of the set of environments we compile. If a target representation, such as DNNF, was used the space savings would be considerably greater.

Figure 4 shows the results of our evaluation. Each row of plots in this figure corresponds to different settings of p, decreasing from top to bottom. The rows correspond to the extremes of the range of solution set cardinalities we used, i.e. 10% and 50% of all possible solutions. For each plot we show a scatter of the points corresponding to different thresholds, φ. For experimental purposes $\varphi \in \{0.05, 0.10, 0.15, 0.20, 0.25\}$.

The results confirm the analytical predictions presented earlier. We almost always observe a saving in space when we compile the most preferred environments. A saving corresponds to a point being placed above a line with unit slope. The only points where we do not observe savings are when we compile all environments, giving a relative memory of 1.0. However, this is exactly what is to be expected: if we compile everything, we need all the original space. As p ranges from 0.1 to 1.0, we move from a setting where we observe relatively fewer savings (Figure 4(g) and 4(h)) to regions with dramatic savings. For example, when $p = 0.50$ we can often cover more than 80% of the environments with only 20% of the space (Figure 4(e) and 4(f)).

The number of solutions being compiled in our experiments does not seem to have a dramatic effect on the savings we have observed. However, studying this for very large numbers of variables is part of our future work. We plan to use importance-based sampling to increase the size of problem that we can study. However, the limit on the number of variables here comes primarily from the restrictions placed upon us by our desire to make perfect measurements on the savings achievable through partial compilation.

Related Work

The literature contains a considerable body of work on compilation, and on its application to a variety of model-based applications, such as configuration and diagnosis. The standard compiled representations include prime implicates (de Kleer 1986), DNNF (Darwiche 2002), Ordered Binary Decision Diagrams (OBDDs) (Bryant & Meinel 2001), cluster-trees (Pargamin 2003), and finite-state automata (Amilhastre, Fargier, & Marquis 2002). All approaches compile sound and complete representations, even if they approximate the original problem, e.g., by approximating propositional and first-order formulae using Horn lowest upper bound representations and their generalisations (Selman & Kautz 1996; del Val 1995).

Our approach is novel with respect to the literature on compilation since we sacrifice completeness of our compiled form for savings in space. This is a very useful property in domains such as real-time embedded systems, where we typically only require access to the most preferred, most likely or most cost-effective environments of our problem.

Conclusion

We have proposed a novel method for compiling the most preferred environments in problems as a means to reduce the large space requirements of compiled representations. We have provided both a theoretical analysis and an empirical evaluation of the approach. The method provides a graceful way to trade off space requirements with completeness of our coverage of the environment space to fit the requirements of embedded systems, up to including all solutions/environments.

Our future work will involve the development of a trace-based algorithm for generating approximate compilations based on a preference threshold. We will also study the savings that can be achieved using various target representations such as DNNF when compiling large real world problems. In addition, we intend to examine whether we can modify OBDDs to generate approximate compilations with guaranteed lower space requirements.

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.

Bodlaender, H. L. 1997. Treewidth: Algorithmoc techniques and results. In *MFCS*, 19–36.

Bryant, R., and Meinel, C. 2001. Ordered binary decision diagrams. In Hassoun, S., and Sasao, T., eds., *Logic Synthesis and Verification*. Kluwer.

Cadoli, M., and Donini, F. M. 1997. A survey on knowledge compilation. *AI Commun.* 10(3-4):137–150.

Console, L.; Portinale, L.; and Dupré, D. T. 1996. Using compiled knowledge to guide and focus abductive diagnosis. *IEEE Trans. Knowl. Data Eng.* 8(5):690–706.

Darwiche, A., and Marquis, P. 2002a. Compilation of propositional weighted bases. In *NMR*, 6–14.

Darwiche, A., and Marquis, P. 2002b. A knowledge compilation map. *J. Artif. Intell. Res. (JAIR)* 17:229–264.

Darwiche, A. 2002. A compiler for deterministic, decomposable negation normal form. In *AAAI/IAAI*, 627–634.

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

del Val, A. 1995. An analysis of approximate knowledge compilation. In *IJCAI (1)*, 830–836.

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

Selman, B., and Kautz, H. A. 1996. Knowledge compilation and theory approximation. *J. ACM* 43(2):193–224.

Spohn, W. 1988. Ordinal conditional functions: A dynamic theory of epistemic states. In *Causation in Decision, Belief Change, and Statistics*. 105–134.

Subbarayan, S. 2005. Integrating CSP Decomposition Techniques and BDDs for Compiling Configuration Problems. In *CP-AI-OR*. Prague: Springer LNCS 3524.