DNNF-based Belief State Estimation*

Paul Elliott and Brian Williams
(pelliott, williams)@mit.edu
MIT SSL and CSAIL
Cambridge, MA 02139

Abstract

As embedded systems grow increasingly complex, there is a pressing need for diagnosing and monitoring capabilities that estimate the system state robustly. This paper is based on approaches that address the problem of robustness by reasoning over declarative models of the physical plant, represented as a variant of factored Hidden Markov Models, called Probabilistic Concurrent Constraint Automata. Prior work on Mode Estimation of PCCAs is based on a *Best-First Trajectory Enumeration* (BFTE) algorithm. Two algorithms have since made improvements to the BFTE algorithm: 1) the *Best-First Belief State Update* (BFBSU) algorithm has improved the accuracy of BFTE and 2) the MEXEC algorithm has introduced a polynomial-time bounded algorithm using a *smooth deterministic decomposable negation normal form* (sd-DNNF) representation.

This paper introduces a new *DNNF-based Belief State Estimation* (DBSE) algorithm that merges the polynomial time bound of the MEXEC algorithm with the accuracy of the BFBSU algorithm. This paper also presents an encoding of a PCCA as a CNF with probabilistic data, suitable for compilation into an sd-DNNF-based representation. The sd-DNNF representation supports computing k belief states from k previous belief states in the DBSE algorithm.

Introduction

The purpose of estimation is to determine the current, hidden state of the system. An estimator infers the current state by reasoning over a model of the system dynamics, the commands that have been executed and the resulting sensory observations. In model-based programming, the models written by the system engineers can be used to diagnose the system, using a mode estimator. The mode estimator is capable of automatically doing system-wide diagnostic reasoning and is completely reusable for different applications.

This paper introduces a new *DNNF-based Belief State Estimation* (DBSE) algorithm that combines features of two other algorithms, the *Best-First Belief State Update* (BFBSU) algorithm (Martin, Ingham, & Williams 2005; Martin 2005) and the MEXEC algorithm (Barrett 2005). The BFBSU and MEXEC algorithms, in turn, are extensions of the *Best-First Trajectory Enumeration* (BFTE) algorithm (Williams *et al.* 2003).

All 4 algorithms, DBSE, BFBSU, MEXEC, and BFTE, reason over a variant of factored *Hidden Markov Models* (HMMs), called *Probabilistic Concurrent Constraint Automata* (PCCA) (Williams & Ingham 2002). PCCAs factor the HMM by both requiring that components have independent probability distributions and by encoding zero probabilities as a constraint theory.

The BFTE algorithm computes the k most likely trajectories, approximating the most likely belief states as the most likely trajectories. Trajectories are computed by testing candidate consistency against prime implicates in an Optimal Constraint Satisfaction Problem (OCSP) solver (Williams *et al.* 2003). The consistency test used has a worst-case time complexity that is exponential in the size of the model, though in practice it is often polynomial.

The trajectory approximation made by BFTE is a poor approximation when two or more of the leading belief states are close in probability and composition (Martin 2005). In these cases, the two belief states may transition into the same next state, but by different trajectories, as they have different initial conditions. The BFTE approximation treats each trajectory that ends in the same state as a different estimate, which means the same state may appear more than once in the k estimates that the algorithm keeps. The algorithm also underestimates the probability of that state, as it has divided the probability across multiple trajectories.

The BFBSU algorithm improves upon the accuracy of the BFTE algorithm by instead enumerating the k best belief states using the same OCSP solver instead of trajectories. BFBSU also improves upon the accuracy of the BFTE algorithm by estimating the probability of getting an observation for a particular state. BFBSU uses a sparse observation probability table for this purpose. The table approximates the number of possible observations, given a state, so the probability of an observation is one over this value, assuming a uniform distribution.

The micro executive MEXEC algorithm reduces the complexity bound of the BFTE algorithm by replacing the prime implicates used in the OCSP solver with a *smooth deterministic decomposable negation normal form* (sd-DNNF) (Darwiche 2001) representation. The sd-DNNF representation supports the queries needed by the BFTE algorithm with a

*This work was funded in part under grant #902364.

time bound that is polynomial in the size of the sd-DNNF. The MEXEC algorithm, like (Kurien & Nayak 2000), also adds the ability to estimate over an N-step time window, where BFTE was only able to handle a single step. The hard polynomial upper space and time bounds of the MEXEC algorithm are especially important for embedded applications, where sufficient memory is generally allocated for the worst case and response times need to be known.

The new DBSE algorithm presented in this paper merges the polynomial space and time bound of the MEXEC algorithm with the accuracy of BFBSU algorithm by using an sd-DNNF representation to estimate the best k belief states. The DBSE algorithm improves upon the accuracy of the BFBSU algorithm in two ways: 1) the sd-DNNF representation admits polynomial-time model counting, so the observation probabilities are no longer approximate and 2) the algorithm is able to correctly handle transitions between components that are not fully independent, because they depend on an unobserved condition. This algorithm does not include the n-step capability of the MEXEC algorithm.

This paper also provides a complete encoding of the PCCA model as a Conjunctive Normal Form (CNF) model plus a set of OCSP probabilities for the transition variables. This CNF encoding is used with the C2D (Darwiche 2005) compiler to generate an sd-DNNF suitable for use by the DBSE algorithm.

This paper starts out by presenting the parts of the PCCA model. It then introduces an encoding of the PCCA model as a CNF theory suitable for computing the HMM update equations. The CNF encoding can be compiled with the C2D compiler into a sd-DNNF that is suitable for online estimation. The paper then provides the DBSE estimation algorithm, including the running time.

PCCA Model

We model the physical plant being diagnosed as a factored HMM that is compactly encoded as a PCCA (Williams *et al.* 2003). A PCCA represents a set of discrete, partially-observable, and concurrently-operating components that are connected together to form a system. Each automaton has a set of conditional probabilistic transitions, which capture both nominal and faulty behavior.

A PCCA is a composition of Probabilistic Constraint Automata (PCA). A PCA A_a is defined by the triple $A_a = \langle \Pi_a, \mathbb{M}_a, \mathbb{T}_a \rangle$:

1. $\Pi_a = \Pi_a^m \cup \Pi_a^c \cup \Pi_a^o \cup \Pi_a^d$ is a finite set of discrete variables, which completely describe the component. All $x \in \Pi_a$ have a finite domain $\mathbb{D}(x)$. m, c, o, and d correspond to *mode*, *control*, *observation*, and *dependent* variables, respectively. The mode variables are the estimated variables. The dependent variables are the intermediate variables needed to define the behavior of a single component. We denote the non-mode variables Π_a^{cod}. $\Sigma(\Pi)$ is the complete set of full assignments to variables Π, and $\mathbb{C}(\Pi)$ is the set of all possible constraints on variables Π.
2. $\mathbb{M}_a : \left\{\Sigma(\Pi_a^m) \rightarrow \mathbb{C}\left(\Pi_a^{cod}\right)\right\}$, the modal constraints, map each mode to a constraint that must hold true when the component is within that mode.

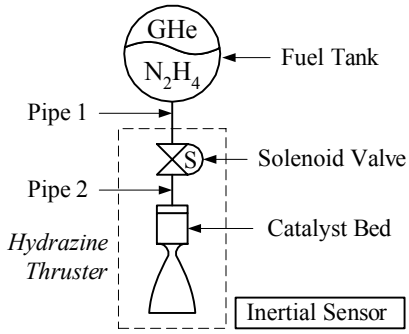

Figure 1: Monopropellant Propulsion System Schematic

3. $\mathbb{T}_a : \left\{\Sigma(\Pi_a^m) \times \mathbb{C}\left(\Pi_a^{cd}\right) \times \Sigma(\Pi_a^m) \rightarrow \Re[0,1]\right\}$ represent probabilistic, conditional transition functions. Consider a single transition $(\sigma_m^t, C^{t,t+1}, \sigma_m^{t+1})$. σ_m^t represents the source mode of the component at time t. $C^{t,t+1}$, the transition guard, represents the conditions under which this transition can occur, and σ_m^{t+1} is the target mode of the component at time $t+1$ after the transition. The transition functions $\mathbb{T}_a$ assign the probability $\mathbf{P}\left(\sigma_m^{t+1} | \sigma_m^t, C^{t,t+1}\right)$ to a transition.

Transitions in different components may be conditioned on the same variables, but semantically, given the truth of the transition guards, the transition taken for each component is decided independently based on its local transition probabilities. When two transitions depend on a variable whose value is not known at the current time, the transition guards are only partially known and so it is necessary to considered the possible values of the variable and how they impact the choice of transitions.

BFBSU assumes the guards are always independent, an assumption that is only correct if all jointly conditioned variables are known every estimation cycle. When a jointly conditioned variable is not fully known, BFBSU allows impossible transition combinations to occur. This DBSE implementation relaxes this condition so that impossible transition combinations are never taken together, even when not fully known.

A PCCA model $\mathcal{P}$ is defined by the triple $\mathcal{P} = \langle \mathcal{A}, \Pi, \mathbb{Q} \rangle$. $\mathcal{A} = \{A_1, \ldots, A_n\}$ is the finite set of PCAs; one PCA for each component. $\Pi = \bigcup_{a=1..n} \Pi_a$ is the set of all variables defined in $\mathcal{P}$. $\mathbb{Q} \in \mathbb{C}(\Pi)$ is a constraint over all the variables that captures the interconnections between components.

Example PCCA

To illustrate a PCCA model, consider a simplified monopropellant thruster. A schematic of the propulsion subsystem is shown in Figure 1. We model this propulsion subsystem as a set of two components: a fuel tank and a solenoid valve. We assume that a properly opened solenoid valve always lead to a nominal inertial sensor measurement.

Fuel Tank: The fuel tank PCA model A_{tank} is shown graphically in Figure 2. A_{tank} is defined by the triple $A_{\text{tank}} = \langle \Pi_{\text{tank}}, \mathbb{M}_{\text{tank}}, \mathbb{T}_{\text{tank}} \rangle$. The variables are $\Pi_{\text{tank}} =$

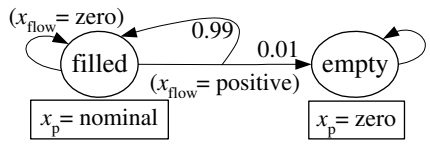

Figure 2: Fuel Tank PCA A_{tank}.

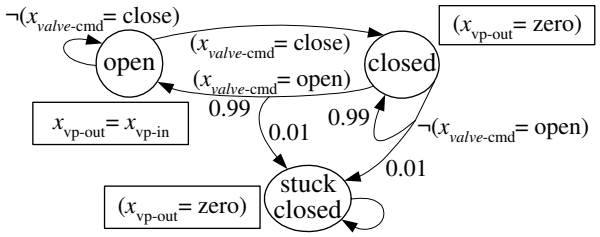

Figure 3: Solenoid Valve PCA A_{valve}.

$\{x_{\text{tank}}, x_{\text{flow}}, x_{\text{p}}\}$, where the fuel tank, represented by x_{tank}, resides in one of two discrete modes, $\mathbb{D}(x_{\text{tank}}) = \{\text{filled}, \text{empty}\}$. x_{flow} describes whether fuel is flowing from the tank. x_{p} describes whether the tank is pressurized, which indicates whether or not the fuel tank contains fuel. The modal constraint $\mathbb{M}_{\text{tank}}$ and transitions $\mathbb{T}_{\text{tank}}$ are

$\mathbb{M}_{\text{tank}}$

x_{tank}	C
filled	$\{(x_{\text{p}} = \text{nominal})\}$
empty	$\{(x_{\text{p}} = \text{zero})\}$

$\mathbb{T}_{\text{tank}}$

x^t_{tank}	x^t_{flow}	x^{t+1}_{tank}	p
filled	zero	filled	1
filled	positive	filled	0.99
filled	positive	empty	0.01
empty	——	empty	1

Solenoid Valve: The solenoid valve PCA model A_{valve} is shown graphically in Figure 3, and is defined in a manner similar to the Fuel Tank.

Combined PCCA Model

Combining these components, the PCCA model $\mathcal{P}$ is defined by the three components: 1) $\mathcal{A} = \{A_{\text{tank}}, A_{\text{valve}}\}$, 2) $\Pi = \Pi_{\text{tank}} \cup \Pi_{\text{valve}}$, and 3) $\mathbb{Q}$ links x_{p} to $x_{\text{vp}_{\text{in}}}$ and $x_{\text{vp}_{\text{out}}}$ to x_{flow}. The components are connected through a single pressure variable. There is flow when the pressure at the output of the valve is not zero.

sd-DNNF Model

We will use the sd-DNNF compiler C2D (Darwiche 2005), which converts a CNF theory into an sd-DNNF. To use this compiler, the PCCA must be encoded as a CNF theory plus transition probabilities. The resulting sd-DNNF encoding of the model presented here is designed to allow for the easy computation of the HMM belief state propagation equations. Given the observations σ_o^{t+1} and the commands σ_c^t, the two HMM belief state propagation equations (Baum & Petrie 1966) are:

$$f\left(\sigma_m^{t+1}\right) = \mathbf{P}\left(\sigma_o^{t+1}|\sigma_m^{t+1}\right) \sum_{\sigma_i^t \in B_k^t} \mathbf{P}\left(\sigma_m^{t+1}|\sigma_i^t, \sigma_c^t, \sigma_d^{t,t+1}\right) g\left(\sigma_i^t\right), \tag{1}$$

$$g\left(\sigma_m^{t+1}\right) = \frac{f\left(\sigma_m^{t+1}\right)}{\sum_{\sigma_i^{t+1} \in B_k^{t+1}} f\left(\sigma_i^{t+1}\right)}. \tag{2}$$

Equation 1, $f\left(\sigma_m^{t+1}\right)$, computes the probability $\mathbf{P}\left(\sigma_m^{t+1}|\sigma_o^{0,t}, \sigma_c^{0,t+1}\right)$ using the distribution $g\left(\sigma_i^t\right)$ at time t, which is approximated by the k best belief states B_k^t. The probability is then normalized by Equation 2, $g\left(\sigma_m^{t+1}\right)$. Normalization is necessary for two reasons: 1) we are ignoring a normalization term $\mathbf{P}\left(\sigma_o^{0,t}, \sigma_c^{0,t+1}\right)$, so the result may not sum to 1, and 2) we are only computing the k highest probabilities for the k best σ_m^{t+1}, so we are losing some solutions.

In Equation 1, the observation probability $\mathbf{P}\left(\sigma_o^{t+1}|\sigma_m^{t+1}\right)$ is

$$\frac{\#\text{Models}(\sigma_o^{t+1} \wedge \sigma_m^{t+1} \wedge B_k^t \wedge \sigma_o^t \wedge \sigma_c^t)}{\#\text{Models}(\sigma_m^{t+1} \wedge B_k^t \wedge \sigma_o^t \wedge \sigma_c^t)},$$

where $\#\text{Models}$ designates counting the number of models of the theory for which its argument is true. In other words, the probability is the number of ways that we could have gotten our observation divided by the number of ways to get any observation. The transition probability $\mathbf{P}\left(\sigma_m^{t+1}|\sigma_i^t, \sigma_c^t, \sigma_d^{t,t+1}\right)$ is computed for each of the k previous belief states σ_i^t, where the probability of transitioning to the the belief state σ_m^{t+1} is encoded on a per-PCA A_a basis in the transition function $\mathbb{T}_a$.

Encoding of a PCCA Model for Compilation

This section shows a novel encoding of the PCCA Model as a CNF theory plus the transition probabilities, suitable for estimating the next belief states using k previous belief states. The PCCA model has three types of explicit constraints: 1) the global constraint $\mathbb{Q}$, which is already a CNF constraint, 2) the modal constraints $\mathbb{M}_a$, and 3) the transition constraints $\mathbb{T}_a$. In addition to encoding the explicit constraints, the resulting sd-DNNF must support computing belief states from k prior belief states, which will require performing some additional operations on the CNF prior to using C2D.

Modal Constraints The modal constraints are encoded as $(x_a = v) \Rightarrow \mathbb{M}_a((x_a = v))$, for each mode assignment $(x_a = v)$.

Transition Encoding The transitions designate a single probabilistic choice per PCA. Choosing a particular transition τ_a means that the PCA is in the source mode σ_m^t of the

transition at time t and is in the target mode σ_m^{t+1} of the transition at time $t+1$. Choosing the transition also means that the guard $C^{t,t+1}$ is consistent over the time interval $[t, t+1]$. The probability of the choice is $\mathbb{T}_a(\tau_a)$.

To encode the transition, we create a transition variable t for each PCA A_a. The domain of a transition variable is the set of all non-zero probability transitions $\tau_a \in \mathbb{T}_a$. To capture the constraints imposed by the transition, we construct the constraint $(t = \tau_a) \Rightarrow \sigma_m^t \wedge \sigma_m^{t+1} \wedge C^{t,t+1}$, for each transition assignment $(t = \tau_a)$.

The transition probability is needed to compute $\mathbf{P}\left(\sigma_m^{t+1} | \sigma_i^t, \sigma_c^t, \sigma_d^{t,t+1}\right)$. This probability cannot be encoded in the CNF and so it is stored externally. There is a one-to-one correspondence between assignments σ_t and probabilities, hence it is easy to associate the probabilities with the sd-DNNF after compilation.

Belief States Encoding DBSE supports tracking the k best belief states, where k is an input parameter to the compilation process. A belief state represents a possible state of the system, and from each of the k possible previous belief states, the DBSE algorithm computes the k best next belief states. To support k independent previous belief states all transitioning to the same next belief state, we encode k sets of the model variables at time t, along with k sets of the constraints $C_1, \ldots, C_k$ containing those model variables. The previous belief states must still share a single coherent next state, so the remaining variables and constraints are unchanged.

The choice of which previous belief state led to a particular state is encoded by creating a variable b with the domain $[1, k]$. Choosing a belief state implies that the previous belief state assignment has been chosen with the probability of the previous belief state. Using this variable b, the choice is encoded as $(b = i) \Rightarrow C_i$, for each belief state variable assignment $(b = i)$ and ith duplicate constraint C_i. The online algorithm is responsible for associating the ith belief state with the variables of the constraint C_i. The online algorithm can directly associate the belief state probability with $(b = i)$.

sd-DNNF Representation

The compiled PCCA representation consists of the two parts: 1) an sd-DNNF representation $\mathcal{D}$ of the CNF encoding, as output from C2D, and 2) a mapping of transition variables to their corresponding modeled probability $\mathbf{P}(\tau_a)$, from the PCCA model.

Estimation Algorithm

The online estimation algorithm needs to compute the transition and observation probabilities for each candidate. The true probability of the candidate is the product of the two, per Equation 1. Traditionally, one computes the first and then the second, as one needs to know what the expected state is in order to compute the observation probability. A key benefit of using sd-DNNFs in this estimation process is that the observation and transition probabilities can be computed in parallel, in a factored fashion, for every candidate, allowing the estimation algorithm to select the best candidates efficiently using an exact metric of their factored product. Specifically, the sd-DNNF theory is factored into hierarchically independent sets of deterministic or-nodes, which can be chosen between arbitrarily. We can compute the observation probability of each choice at the same time as the transition probability of each choice, and then efficiently decide between the choices locally based on the product of the two probabilities.

We briefly explain why the deterministic nodes can be decided independently. We distinguish between two types of or-nodes: deterministic and indistinguishable. A deterministic or-node is one in which different paths ensure different models are chosen over the remaining variables, in our case the estimated variables σ_m^{t+1}, in compliance with the deterministic requirement of the sd-DNNF. An indistinguishable node, by contrast, is a disjunction over indistinguishable models, which have become indistinguishable through projection, and are not allowed to be part of a proper sd-DNNF. When projecting an sd-DNNF, the sd-DNNF is guaranteed to only have indistinguishable nodes at the leaves if the projected variables are all first in the decomposition. In the DBSE algorithm, all variables are summed-out or assigned values except the σ_m^{t+1} variables, so these variables must appear first in the decomposition. To make the sd-DNNF compliant again, we need to make a single walk through the DNNF and sum-out the indistinguishable nodes.

To sum-out an indistinguishable node, we add both model counts and the transition probabilities of all the node's children together into a single pair of model counts and a single transition probability. The indistinguishable node can then be replaced by a new node containing these three values. The and-nodes that are descendants also need to be eliminated by multiplying their children, rather than adding. The and-node, too, can be replaced by a new node containing the product of its children.

Once we have chosen a path for every deterministic node and eliminated every indistinguishable node, we can collapse the entire sd-DNNF into a single and-node with a list of literals as its leaves. Since and-nodes multiply their children and multiplication is associative, the transition and observation probabilities of each or-node will be multiplied into the final answer. Thus, the probabilities can be computed independently, per deterministic node, and the decision for each deterministic node can be made based on the local multiplied pair of probabilities.

Online

The online algorithm, shown in Figure 4, is responsible for taking an sd-DNNF $\mathcal{D}$, up to k belief states B_k^t, the previous σ_o^t and next σ_o^{t+1} observations, and the commands σ_c^t and then computing the k next belief states B_k^{t+1}. The algorithm, therefore, is incremental and only requires keeping track of the previous time step.

The first step of the algorithm assigns the k best belief states from the previous state to the k belief state variables.

B_k^{t+1} **DBSE**$\left(\mathcal{D}, B_k^t, \sigma_o^t, \sigma_c^t, \sigma_o^{t+1}\right)$

1. $\mathcal{D} \leftarrow \mathcal{D}|(\sigma_m^t \in B_k^t) \wedge \sigma_o^t \wedge \sigma_c^t$
2. Assign $\mathbf{P}\left(B_k^t\left(i\right)\right)$ to $(b = i)$ in $\mathcal{D}$
3. Compute $\#\text{Models}\left(\mathcal{D}\right)$, $\#\text{Models}\left(\mathcal{D}|\sigma_o^{t+1}\right)$, and $\mathbf{P}\left(\mathcal{D}\right)$
4. $\mathcal{D} \leftarrow \mathcal{D} \Downarrow_{\Pi^{m\,t+1}}$
5. $B_k^{t+1} \leftarrow$ Enumerate the k best models of $\mathcal{D}$
6. Normalize and return B_k^{t+1}

Figure 4: An algorithm that computes k belief states using a sd-DNNF model.

It also assigns the observations gathered at the previous point in time and the commands that have since been issued. Assignment is performed by conditioning the sd-DNNF on the conjunction of the assignments.

The second step assigns the belief state probabilities to the belief state variables.

In the third step, we walk through the sd-DNNF $\mathcal{D}$ and compute the three parts of the update equation, Equation 1: 1) The denominator of the observation probability, 2) the numerator of the observation probability, and 3) the transition probability. As discussed earlier, these can only be combined into a single probability at deterministic nodes, where we can guarantee the only operation is multiplication. The values are computed for each node in the DNNF.

The fourth step projects the DNNF onto the mode variables that we are estimating. The projection uses the DNNF project algorithm, except we keep track of the number of models that we are projecting, as computed in step 3. We refer to this as summing-out the variables. In this process, all indistinguishable nodes are eliminated, as they are no longer needed. This ensures that every or-node is a deterministic node, choosing between different mode-variable instantiations, so each model of the sd-DNNF appears exactly once.

The fifth step involves enumerating the k best models of $\mathcal{D}$, using the value of Equation 1 computed for each deterministic node. A linear programming algorithm is used to compute the k most probable choices for each node based on the combination of the k best candidates of its children. Nodes will only have an estimate if they have a deterministic node as a child or if they are themselves deterministic nodes. When the algorithm completes, the root node will have the probabilities of k most likely belief states and which children led to those probabilities. It then can walk back through the DNNF and build a set of k belief states based on which choices were part of each estimate.

The final step re-normalizes the belief states so the probabilities sum to 1. This step performs the calculation of Equation 2. We then return the new k best belief states.

Running Time

This section addresses the running time of the algorithm. The first step of the algorithm in Figure 4 requires a single iteration through $\mathcal{D}$, assigning each leaf literal a value of *true* or *false*. The second steps assigns k probabilities to k leaf literals, and thus the time taken is $O(k)$. The third step requires one iteration through $\mathcal{D}$, visiting every edge once. The fourth step, like the third, makes one pass through $\mathcal{D}$, visiting each edge and leaf once. The fifth step requires two iterations over $\mathcal{D}$. The first iteration identifies the best belief states and then the second extracts the best belief states. The first iteration requires $O(k)$ space per edge and node of the sd-DNNF. The merging of the child's estimates with the parent's estimates requires $O(k)$ time per edge for or-node parents and $O(k^2)$ time per edge for and-node parents. The second iteration propagates the selection at the root down each edge for each of the k estimates. The final step requires averaging the k probabilities.

For a DNNF with n nodes, e edges and supporting k belief states, in total the DBSE algorithm has a space complexity of $O(kn+ke)$ and a time complexity of $O(kn+k^2e)$. Since $n \leq e$ for all sd-DNNFs, we can simplify these bounds to $O(ke)$ and $O(k^2e)$ for space and time, respectively.

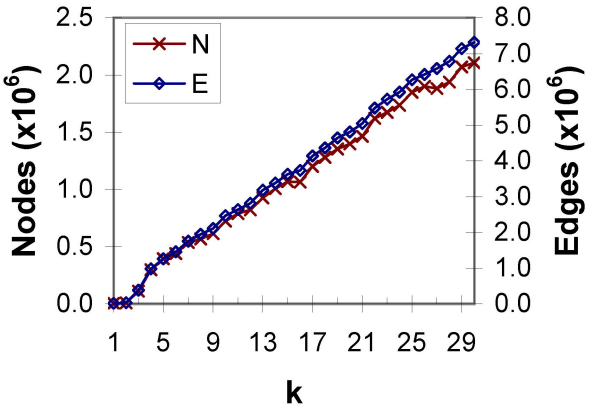

Figure 5: The number of nodes and edges needed to represent the Mars EDL model as a function of the number of belief states k encoded in the CNF formula. The number of nodes is $71\times10^3\times k$ and the number of edges is $244\times10^3\times k$ for $k \geq 4$, with an RMS error of 2.5×10^3 and 4.7×10^3, respectively. For $k = 1, 2$, the number of nodes is $4.4\times10^3\times k$ and the number of edges is $14.8 \times 10^3 \times k$, with an RMS error of 370 and 340, respectively.

Preliminary Results

The encoding presented in this paper has been used to generate a set of sd-DNNFs using a Mars Entry, Descent, and Landing (EDL) model (Ingham 2003), which is roughly the size of a spacecraft subsystem. The model has forty-two variables, of which ten are dependent variables and ten are state variables, with an average domain size of 4.4. For the set, we varied k from 1 to 30. The C2D algorithm employs randomization in generating a decomposition of the CNF theory, and was given no guidance as to which decompositions to consider. Thus, to make the data more regular, the C2D compiler was run 10 times on each CNF formula. The number of nodes and edges of the smallest model was kept

and plotted in Figure 5.

It can be seen in Figure 5 that the sd-DNNF is linear in k. For $k \geq 4$, the number of nodes is $71 \times 10^3 \times k$ and the number of edges is $244 \times 10^3 \times k$. The RMS error for the slope of the line is 2.5×10^3 and 4.7×10^3 for the number of nodes and edges, respectively. The first two data points, however, tell a somewhat different picture. For the first two data points, which have only 2 or 3 copies of the variables and constraints, respectively, the C2D compiler is able to find a much smaller encoding than for the remaining data points. For these first two points the number of nodes is $4.4 \times 10^3 \times k$ and the number of edges is $14.8 \times 10^3 \times k$, with an RMS error of 370 and 340, respectively. For these first two values of k, the C2D's binary variable representation is able to encode exactly the possible assignments to the belief state variable. Thus, the C2D compiler is forced to identify a single decision point at which the model breaks into k independent pieces. For larger k, the decision as to which belief state is active is spread across many decision points, which we believe is what leads to a less efficient decomposition, as represented by the larger slope.

We believe that generating consistently small sd-DNNFs, at a size proportional to the first two data points, is possible by guiding the decomposition. The CNF formula contains $k + 1$ copies of the same variables and $k + 1$ copies of the state constraints and global constraints. It also has k copies of the transition constraints. The C2D compiler is currently expected to find the large number of symmetries and then order the decomposition to take advantage of them.

The linearity of the sd-DNNF with respect to k allows us to express the space and time bounds given in the previous section more precisely in terms of the more precise model size. The new space bound is $O(k^2 e_n)$ and the new time bounds is $O(k^3 e_n)$, where e_n is the size of the model for $k = 1$. For the Mars EDL model, e_n was shown to be approximately 244×10^3 for $k \geq 4$. However, we believe that by providing guidance to the C2D compiler, e_n may be reduced to the $k = 1, 2$ model size of 14.8×10^3 edges.

We now compare these results to the two previous algorithms. The BFBSU algorithm displayed a linear space and time bound as a function of k in (Martin 2005). For the MEXEC algorithm, a linear space and time bound as a function of the size of the sd-DNNF was published in (Barrett 2005). Within these time and space bounds, the MEXEC algorithm can extract any one of the best, equally-likely leading candidates. The MEXEC algorithm is not able to extract the next most likely candidate(s). The sd-DNNF of the MEXEC algorithm is assumed to be linear as a function of n, the size of the time-step window that the estimation algorithm uses.

For comparison with the MEXEC algorithm, let $k = n$. The DBSE algorithm requires k times more space and k^2 more time than the MEXEC algorithm. This extra computational overhead is necessary to store the information needed to extract the k best estimates, rather than just the set of estimates that all share the same likelihood. The number of estimates that share the same likelihood varies significantly from estimation cycle to estimation cycle, depending on how much uncertainty there is as to the current state. With respect to BFBSU, the DBSE algorithm requires k times more space than BFBSU, which is presently necessary to store k copies of the compiled constraints. However, unlike BFBSU, which has experimentally shown an average case bound, this space bound is a known upper bound. Similarly, the k^2 factor more than the BFBSU algorithm for the time bound is an known upper-bound, as opposed to BFBSU's time bound. The algorithm also computes a more accurate estimate than BFBSU.

Conclusion

This paper has presented the DBSE algorithm for estimating the k best belief states using an sd-DNNF representation. The algorithm has a space requirement that is quadratic in k times the size of the $(k = 1)$ sd-DNNF. It has a running time that is cubic in k times the size of the $(k = 1)$ sd-DNNF. While keeping the polynomial time bounds of the MEXEC algorithm, this algorithm improves the accuracy of both the BFBSU and the MEXEC algorithms.

References

Barrett, A. 2005. Model compilation for real-time planning and diagnosis with feedback. In Kaelbling, L. P., and Saffiotti, A., eds., *IJCAI*, 1195–1200. Professional Book Center.

Baum, L., and Petrie, T. 1966. Statistical inference for probabilistic functions of finite-state Markov chains. *Annals of Mathematical Statistics* 37:1554–1563.

Darwiche, A. 2001. Decomposable negation normal form. *J. ACM* 48(4):608–647.

Darwiche, A. 2005. C2D v2.20. http://reasoning.cs.ucla.edu/c2d.

Ingham, M. 2003. *Timed Model-based Programming: Executable Specifications for Robust Mission-Critical Sequences*. PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics.

Kurien, J., and Nayak, P. P. 2000. Back to the future for consistency-based trajectory tracking. In *AAAI/IAAI*, 370–377.

Martin, O.; Ingham, M.; and Williams, B. 2005. Diagnosis as Approximate Belief State Enumeration for Probabilistic Concurrent Constraint Automata. In *Proceedings of the AAAI*.

Martin, O. 2005. Accurate belief state update for probabilistic constraint automata. Master's thesis, Massachusetts Institute of Technology, MIT MERS.

Williams, B. C., and Ingham, M. D. 2002. Model-based Programming: Controlling Embedded Systems by Reasoning About Hidden State. In *Eighth Int. Conf. on Principles and Practice of Constraint Programming*.

Williams, B. C.; Ingham, M.; Chung, S. H.; and Elliott, P. H. 2003. Model-based Programming of Intelligent Embedded Systems and Robotic Space Explorers. In *Proceedings of the IEEE*, volume 9, 212–237.