# Using Structured Knowledge Representation for Context-Sensitive Probabilistic Modeling

Nikita A. Sakhanenko [a,b,*] George F. Luger [b]

[a] *Institute for Systems Biology, Seattle, WA 98103 USA*

[b] *Computer Science Department, University of New Mexico, Albuquerque, NM 87131 USA*

**Abstract**

We propose a context-sensitive probabilistic modeling system (COSMOS) that reasons about a complex, dynamic environment through a series of applications of smaller, knowledge-focused models representing contextually relevant information. COSMOS uses a failure-driven architecture to determine whether a context is supported, and consequently whether the current model remains applicable. The individual models are specified through sets of structured, hierarchically organized probabilistic logic statements using *transfer functions* that are then mapped into a representation supporting stochastic inferencing. We demonstrate COSMOS using data from a mechanical pump system.

*Key words:* Structured knowledge representation, Context-sensitive reasoning, Probabilistic modeling

## 1 Introduction

We propose and demonstrate a system (COSMOS) for context-sensitive probabilistic modeling that accounts for a changing environment while attempting to diagnose a dynamical system. Similar to a divide-and-conquer approach, COSMOS decomposes a diagnostic problem into a set of stable, invariant contexts where the computational complexity of the problem is radically reduced.

* Corresponding author.

*Email addresses:* nsakhanenko@systemsbiology.org (Nikita A. Sakhanenko), luger@cs.unm.edu (George F. Luger).

A first-order, Turing-complete, stochastic modeling language Generalized Loopy Logic [26] supports COSMOS. Logic-based systems, in particular those utilizing first-order logic, are very powerful in representing relations between different entities. Probabilistic systems, especially probabilistic graphical models, are successful in capturing the uncertainty in data and performing stochastic reasoning. Combining logic-based and probabilistic reasoning allows for more efficient modeling, applicable to complex and changing tasks, and is an important component of COSMOS.

Dynamic systems are often assumed to be stationary and invariant over an entire training data set. When reasoning about a system whose dynamics change according to states of the external environment with little a priori knowledge, then almost every possible aspect of the world must be explicitly represented in the training data and captured by the learning algorithm in order to avoid overlooking hidden relationships. One implication of this is that when a knowledge base changes, the learned general model is discarded as no longer true and a new model must be constructed from scratch.

COSMOS addresses this issue by contextualization, where a small, knowledge-focused model is used to represent the currently active context. The idea of considering stable, invariant regions (contexts) in a data stream is similar to the divide-and-conquer approach: rather than using a large model to represent the entire set of data, we use smaller models to represent single contexts. This reduces model complexity and requires a smaller amount of training data. One of the main reasons for using context-sensitive modeling is that large models are often more susceptible to overfitting data [17]. Restricting the complexity of the model can address this overfitting dilemma (Occam's razor).

To detect whether we are in a context transition, COSMOS uses a failure-driven architecture where *model failure* is the key concept. Model failure, when the model fits new data poorly, indicates a context transition, during which we store the model representing the previous context in a library of models. As a result, depending on context changes, we can swap a currently active model with a model from the library that represents the new context. In order to reduce the search for model substitution, we structure this library according to domain knowledge.

Even though interfaces are an explicit part of most design models, such as electrical schemas, standard probabilistic graphical models typically do not include explicitly defined interfaces. Since real-time diagnostic applications can include a very large number of interrelated entities, initial specification of a probabilistic graphical model is a very challenging task that is simply left to a domain expert. Structured knowledge representation is needed in order to specify the initial models and map these into probabilistic graphical models. We use *transfer functions* for this task.

In COSMOS, the domain expert provides a set of prior models corresponding to sets of identified contexts. Since it is difficult to specify probabilistic graphical models for various contexts in a real-time application with multiple interconnected relations, we use transfer-function diagrams that can be seen as deterministic templates providing prior information about contextual models. Transfer-function diagrams provide a clear, hierarchically organized, structured design and are well suited for mapping into our stochastic logic implementation.

We demonstrate these examples using a pump system, schematically depicted in figure 1. A water pump draws liquid from a reservoir through a pipe (`pipe1`)

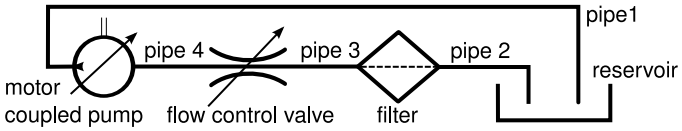



Fig. 1. The diagram representing the simplified pump system.

and ejects the liquid into another pipe (`pipe4`). The pump is driven by an electrical motor. The liquid, that can contain contaminants, is cleared by a filter and deposited back into the reservoir. The flow control modulates the liquid flow. In order to diagnose the system, we install a number of sensors that detect current pressure, flow, and the emission state of the liquid at different locations. There are a number of important diagnostic tasks, including detecting when the filter gets clogged, which can lead to cavitation of the system. COSMOS and the transfer-function representation of these parameters is presented in Section 3. This section also describes other components of COSMOS including the model failure and recovery mechanism. A set of experiments testing COSMOS on data from the pump system are presented in Section 4. Next, we review related research.

## 2 Related work

Logic-based representation for stochastic modeling has been proposed by a number of researchers. Poole [28] develops an approximate inference algorithm for a Turing complete probabilistic logic language where uncertainty is expressed through sets of mutually exclusive predicates annotated with probabilities. To apply Poole's logic the user has to maintain the correct normalization of the probabilistic content, which can be difficult even when expressing simple Bayesian networks (BNs).

Haddawy [8] defined a first-order probabilistic logic used to specify a class of Bayesian networks as a knowledge base. Haddawy proposed a provably correct BN generation algorithm that was later adapted by Ngo and Haddawy [19,20]

to focus the knowledge base on the relevant information. In particular, they used logic expressions as contextual constraints for indexing the probabilistic information to reduce the size of the modeling network.

Friedman et al. [4] and, later, Getoor et al. [5] proposed probabilistic relational models (PRMs) that specify a probability model on classes of objects and use maximum likelihood parameter estimation for parameter learning, while structure learning is done using heuristic search of the best scores in a hypothesis space (which is similar to the notion of an appropriate neighborhood [10]).

Bayesian logic programs (BLPs) offer another knowledge-based model construction appoach and were proposed by Kersting and DeRaedt [11]. This framework generates Bayesian networks specific for given queries using a set of first-order Prolog-like rules with uncertainty parameters. BLPs utilize probabilistic learning from interpretations of inductive logic programming [12].

Richardson and Domingos [29] propose Markov logic networks (MLNs), a probabilistic approach based on general first-order logic. This approach converts logic sentences into a conjunctive normal form (CNF) which is then mapped onto Markov random fields for inference. MLNs utilize a *complete* mapping from first-order predicate calculus with function symbols to probability distributions.

In this paper, we choose a different direction than Richardson and Domingos [29] because of our requirement for domain-dependent and query-dependent model construction. Even though mapping from the CNF sentences of MLNs to Markov fields is straightforward, the practical advantages over Horn-clause-based representations are not obvious. We suggest that Horn clauses provide expressive power by preserving the generality and in the same time supporting embedding various heuristics. We use a stochastic language, called Generalized Loopy Logic (GLL), described in detail in the next section, that combines Horn clauses with BNs similarly to BLPs [11], however GLL maps its sentences into Markov random fields as in MLNs. Our GLL approach can also be applied efficiently to dynamic problems that have strict time and memory constraints.

One of the first attempts to use contextual information in probabilistic modeling was proposed by Haddawy and Ngo [8,19]. Their logic-based stochastic modeling approach utilized explicit contextual information as a way to reduce the size of a model. Here a context is defined as a logic sentence associated with general definitions from a knowledge base. Only relevant definitions, indicated by a matching context, are selected by a knowledge-based model construction algorithm.

Sanscartier and Neufeld [30] proposed another approach that uses context to refine a probabilistic model. They use context-specific independence to make a causal Bayesian network smaller and more accurate. Sanscartier and Neufeld

note that a causal link between two variables can be established only when certain conditional independences are missing for all values of a variable in a distribution. Model reduction is then possible by detecting conditional independences that hold for a subset of the values.

Exploiting the notion of context defined through conditional independencies to improve the performance of a model was also investigated by Turney [36]. In the area of supervised machine learning, Turney studies how features from a multidimensional feature space can be partitioned into different categories using context.

Silver and Poirier [31] applied context to adapt multiple-task neural networks for learning. They replaced multiple outputs of a neural network with a single one while adding a set of inputs that identify an example context. These contextual inputs are task identifiers that associate training examples with particular tasks. Silver and Poirier [31] argue that the contextual inputs represent more specific domain information that supports indexing over a continuous domain of tasks.

Our research is motivated by [8,19], though their contextual mechanism is too simple and discrete for our task: it cannot reflect all the complexity of the internal structures of data. In this paper, we provide a complete specification of context as truth assignments to a specific set of known variables. The choice of variables is similar to the approach by Pearl and Halpern [24,9] that uses exogenous variables (variables that are not in the model) to identify a background of the actual cause of an event. This is different from the notion of a situation index in the situation calculus [1] capturing variables that we may possibly be unaware of to describe a situation.

The notion of context has long been an important part of human understanding. In order to interpret a text, the relevant social environment must be taken into account as it influences the author of the discourse. Van Dijk [37] argues that the relevant features of communicative situations influence language only through participants' subjective views of situations. These views are represented and constantly updated in mental models of the speakers, so-called *context models*. Van Dijk demonstrates that context models control and explain many aspects of interactions that cannot be accounted for otherwise.

In developmental psychology, Gopnik et al. [6] emphasize that a new model based on Bayesian networks and utilizing the principles of dynamic programming can support research on learning in children. Granott et al. [7] attempt to give another psychological perspective on human learning, that of *bridging*, which is an attractor that draws development of a system toward more advanced and more stable levels. Granott et al. [7] argue that bridging is a transition mechanism that people use while learning. The failure-driven approach

presented in this paper is closely related to these approaches for describing human developmental learning.

Although probabilistic logic-based systems provide a useful means of utilizing represented knowledge, the task of representing large knowledge-based models with complex interdependencies in first-order rules can be a daunting task. Srinivas [33] acknowledges that constructing Bayesian networks for complex models by hand is very difficult. In COSMOS we utilize *transfer function* mappings that extend the functional schematics described by Srinivas [33]. The mapping of intermediate representations into Generalized Loopy Logic supports dynamic system modeling, including recursive and time-dependent models, as well as other advantages of GLL over standard probabilistic graphical models.

Koller et al. [13] propose object-oriented Bayesian networks (OOBNs) that allow complex domains to be described in terms of inter-related objects. The structural information encoded by an OOBN and the encapsulation of variables within an object allows the reuse of model fragments in different contexts. Similar object-oriented approaches focus on the modularization of the knowledge representation [14,22,15]. Similar to our use of transfer functions, they show how large networks, normally impractical to construct as a whole, can be woven together from smaller, more coherent and manageable components.

Although, there are many other researchers working on the problems of contextual sensitivity and failure-driven modeling, we have referenced those that have been most influential in developing COSMOS. Other research includes the lifelong learning framework of Thrun [35] and teleo-reactive programs of Nilsson [21].

## 3 The COSMOS system

In this section we introduce COSMOS, a context-sensitive probabilistic modeling system. This system utilizes generalized loopy logic (Section 3.1) as a logic-base probabilistic inferencing engine. COSMOS (Sections 3.3 and 3.4) addresses a modeling problem by partitioning the task into a collection of contexts represented by small, knowledge-focused stochastic models and by choosing appropriate models depending on the current situation. Transfer functions, a hierarchical, structurally organized representation, is used to specify templates for the stochastic models (Section 3.2).

Generalized Loopy Logic (GLL) is an extension of a modeling language developed by Pless [26]. GLL is a logic-based, first-order, Turing-complete stochastic language that combines deterministic and probabilistic reasoning approaches to improve expressive and reasoning power. While the expressive power of traditional Bayesian networks is constrained to finite domains as in the propositional logic, the Generalized Loopy Logic language captures general classes of events and relationships. To represent potentially infinite classes of stochastic relationships such as a Markov process this first-order language combines Horn-clause logic with Bayesian networks. Consequently, knowledge is represented as a set of rules describing the conditional dependences among random variables with stochastic distributions attached to facts and rules.

Specifically, a general GLL sentence is of the form

$$\texttt{head|body}_1\texttt{, ..., body}_k\texttt{ = [p}_1\texttt{, ..., p}_l\texttt{]},$$

where $\texttt{body}_1\texttt{, ..., body}_k$ are the variables of the system on which a variable `head` is conditionally dependent, $l = arity(\texttt{head}) \times \prod_{i=1}^{k} arity(\texttt{body}_i)$ with $arity(x)$ denoting a number of states of a variable $x$. The probabilities are indexed over the states of `head` and $\texttt{body}_1\texttt{, ..., body}_k$. For instance, if $x$ is a predicate valued over $\{low, avg, hi\}$ and $y$ is a boolean predicate, then $P(x|y)$ is defined by the sentence

```
x|y = [[0.5, 0.1, 0.4],[0.3, 0.6, 0.1]].
```

In GLL terms can be full predicates with structure and contain PROLOG style variables. For instance, the sentence `b(N) = [0.5,0.5]` defines that `b` is universally equally probable to take on either of two values. The domain of terms is specified using set notation: `b <- {hi, low}` indicates that `b` is either `hi` or `low`.

The GLL program presented next defines a hidden Markov model (HMM):

```
state <- {true, false}
emit <- {hi, low}
state(N+1)|state(N)=[[0.9,0.1],[0.01,0.99]]
emit(N)|state(N)=Emit
emit(0) = hi
emit(1) = low
emit(2) = hi
```

There are two states, `true` and `false`. The system can start with either one and at each time step either stay in the same state or transition to the other

state. Note that if the system is in the state `true`, then there is a 90% chance that the system will stay in that state at the next time step; however, if the system is in the state `false`, there is only a 1% chance the system will stay in that state. In both states the system can output either `hi` or `low` values.

Further, we can describe the probability of a system that produces an output as a *learnable distribution* (`Emit`). This means that the program specifications are conditioned by the data seen at a particular time. Note also how the recursive rule of the GLL example captures the Markov process across the states `N` of the HMM.

The learnable distribution `Emit` indicates that the conditional probability governing the system's output is to be fitted. The data for learning is obtained from GLL rules and facts (observations). The last three sentences in the program presented earlier are the GLL facts. Note that in each fact the variable `N` is bound. The Generalized Loopy Logic language uses the message-passing inference algorithm known as *loopy belief propagation* [23] (hence the name "Loopy").

GLL can also use other iterative inferencing schemes such as *generalized belief propagation* and *Markov chain Monte-Carlo* [16]. In order to perform inference, GLL converts its first-order program to a Markov random field. Figure 2 demonstrates how the GLL program just presented is converted into a bipartite Markov field. During mapping into a Markov field, each ground instance of a GLL term corresponds to a variable node in the Markov field (ellipse), and each GLL rule with a probability distribution attached to it corresponds to a cluster node (rectangle). If more than one rule unifies with the rule head, then the variable node is connected to more than one cluster node, which results in a product distribution. One of the important features of GLL is its support of dynamic modeling. In GLL, dynamic models can be specified by using recursion and controlling the depth of unfolding of recursive rules when mapping into a Markov random field.

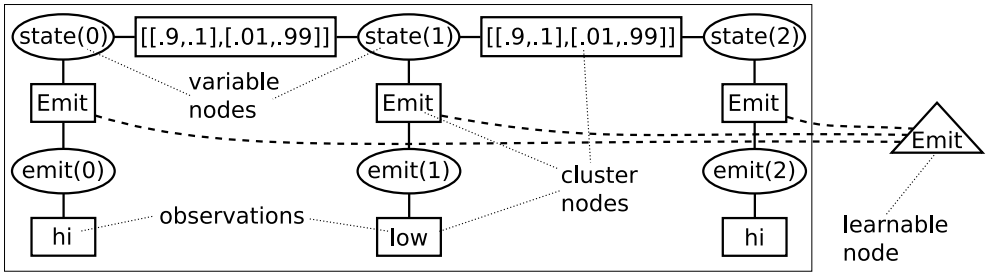



Fig. 2. A Markov random field produced by unrolling the GLL program specifying a hidden Markov model.

During loopy belief propagation nodes of a Markov field exchange messages that are initially set randomly. On update, a message from a cluster node $C$ to

a variable node $V$ is the product of the conditional probability table at $C$ and all the messages to $C$ except the message from $V$. In the other direction, the message from a variable node $V$ to a cluster node $C$ is the normalized product of all the messages to $V$ except the message from $C$. Iterating this process until convergence has been found to be effective for stochastic inference [18] and has been proved to converge to optimal values for acyclic directed graphs [23].

A major feature of GLL is its natural support for parameter learning by assigning of learnable distributions to rules of the GLL program. These parameters are estimated using a variant of the Expectation Maximization (EM) algorithm [3] implemented through the message passing of loopy belief propagation. The EM algorithm estimates learning parameters iteratively, alternating between an expectation (E) step, computing the current estimate of the parameters, and a maximization (M) step, re-estimating these parameters to maximize their likelihood.

GLL utilizes the EM algorithm by adding learnable nodes to the Markov random field representation, the triangular node of figure 2. Each instance of the cluster node to be fitted is connected to the learnable node. By applying loopy belief propagation on the cluster and variable nodes of a Markov field, GLL computes the messages to the learnable nodes. Iterating the propagation algorithm until convergence produces an approximation of the expected values and therefore is equivalent to the E step of the EM algorithm. Averaging over all the cluster nodes connected to the learnable node yields a maximum likelihood estimate of the parameters of the learnable node, which is equivalent to the M step of EM. Therefore, inferencing over the variable and cluster nodes followed by updating the learnable nodes and iterating this process is equivalent to the full EM algorithm.

The representation offered by GLL, predicate logic, is flat; in building models we use structured hierarchical interface to easily specify models for COSMOS. In the next section, we demonstrate the transfer-function diagrams that provide the intermediate representation and map model's components into GLL.

### *3.2 Structured representations: transfer-function diagrams*

To perform diagnostic tasks in dynamic environments such as the pump system, knowledge about the environment supplied by the domain expert must be transformed into a stochastic model. Since knowledge engineering directly in terms of probabilistic models is challenging, an intermediate design model capturing the relationships of the system is needed. *Transfer-function diagrams* define explicit interfaces to the probabilistic models and are naturally mapped into the components of the generalized loopy logic (GLL) language

used to describe stochastic models.

Trasfer-function diagrams provide an intuitive way of engineering design, making the task of system modeling easier. This representation keeps the design process tractable by using an object-oriented approach. The modularity of the input representation supports clarity of design and reusability of its components. The transfer-function diagrams extend the method and set of algorithms originally proposed by Srinivas for mapping device schematics into probabilistic models for diagnosis and repair [33].

A transfer-function diagram is a set of interconnected components. A component receives a set of inputs (**I**) and emits a set of outputs (**O**) constrained by a set of internal state variables (**S**). Note that all the variables are assumed to be discrete. For each output of the component there is a function computing the output that takes a subset of inputs and a subset of internal states as its arguments: $\forall O \in \mathbf{O}$, $\exists F$, $\exists\{I_1, \ldots, I_l\} \subseteq \mathbf{I}$, $\exists\{S_1, \ldots, S_m\} \subseteq \mathbf{S}$ such that

$$F : I_1 \times \ldots \times I_l \times S_1 \times \ldots \times S_m \rightarrow O.$$

Figure 3 illustrates a component (a pipe) from the transfer-function diagrams representing components of the pump system. The boxes inside of the component correspond to functions, whereas ellipses correspond to component's internal states. We offer three justifications for using transfer function dia-

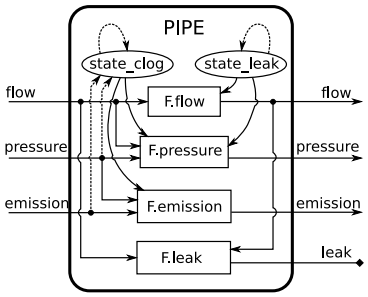



Fig. 3. A component from the transfer function diagram representing a pipe of the pump system.

grams to address the representational issues in building stochastic models.

*Object-oriented abstraction.* Using an object-oriented methodology allows transfer-function diagrams to have components with multiple outputs, an extension to the original functional schematics [33]. Each component is treated as an object with multiple attributes such as the amount of contaminant in the pipe and the pressure for the component (figure 3). Each attribute is modeled by an appropriate function, e.g., the emission condition in the pipe is represented by a function with arguments emission, pressure, and the current internal state of the pipe, which, in turn, can be modeled by another transfer-function di-

agram. This object-oriented representation accepts multiple functions while preserving representational clarity.

There are several other advantages of the object-oriented representation of transfer-function diagrams. By specifying components of the diagram via objects we allow model fragments to be reused. Moreover, we can replicate the inheritance mechanism of object-oriented programming by combining some attributes of different objects into other objects. After mapping into a probabilistic model, the stochastic parameters of the inherited attributes can be learned simultaneously by taking advantage of GLL's parameter learning mechanism (section 3.1).

*Variable typing.* We distinguish two types of model variables: *operating* and *indicating*. Operating variables participate in the operation of the system (e.g., engine speed, flow rate, etc.) whereas indicating variables do not directly affect the functioning of the system (e.g., vibration near the motor). Typically, AI diagnostic representations [33] focus only on the operational behavior of the system and do not use indicating variables. By using indicating variables, transfer-function diagrams support assigning probability distributions that describe the hidden operating states of the components that are essential for diagnosis.

Indicator parameters are also categorized into two types: direct (sensory) and indirect (functional). A physical sensor monitoring some aspects of the environment, such as vibration near the motor in the pump system, is directly represented by a sensory indicator parameter. In oreder to monitor some internal states of a subsystem for which no sensors are available, we use an indirect (functional) indicator parameter modeled as a function of inputs and outputs. The output `leak` (figure 3), which is not used as an input to any other component, is a functional indicator parameter computed as a ratio between the input and the output of the pipe. Note that a functional indicator parameter can be viewed as a *virtual sensor* that indicates whether a specific function within the subsystem is consistent with the data. This is simpler than factoring this information directly into a stochastic model.

*Support of temporal relations.* As seen in the pump example earlier, a knowledge engineer must represent temporal relations between components as well as within components to diagnose such situations as cavitation in the system.

In order to explicitly represent the temporal dynamics of a system, every component's state is made explicit and its temporal change is then captured by a functional dependency on the values from the previous time steps. Temporal relationships are depicted with dotted arrows, e.g., in figure 3 three dotted arrows point to `state_clog` which means that the current state of the pipe being clogged depends on the state's value and two inputs (`pressure` and

emission) from the previous time step.

Once the transfer-function diagram is complete, its components are mapped to GLL statements [26]. Since a GLL program represents classes of probabilistic models, switching to GLL statements provides the modeling system with additional power for capturing dynamic processes. The recursive rules of GLL, for instance, lend themselves nicely to representing potentially infinite structures where some variables have a self-dependency over time. In the next section, we describe this mapping to GLL rules in more detail.

Once a domain expert has specified every system component within a transfer-function diagram, the diagram is converted into a GLL program for further inferencing according to the following mapping:

*1.* Each function of every component in the diagram is mapped into a GLL sentence as shown in figure 4. Inputs and outputs of a function correspond

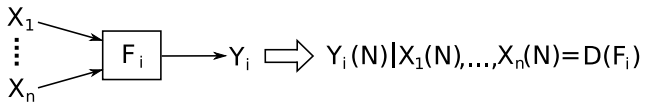



Fig. 4. Mapping of a transfer function into a GLL rule.

to random variables in GLL. Note that `N` stands for the current time step of the system, thus function $\mathtt{F}_i$ has an instant effect, that is, its input and output values are isochronous. Additionally, $\mathtt{D}(\mathtt{F}_i)$ stands for the probability distribution corresponding to function $\mathtt{F}_i$, provided by an expert.

*2.* Each state of every component in the diagram is mapped into a GLL rule according to figure 5. Note that temporal influences on the state are easily

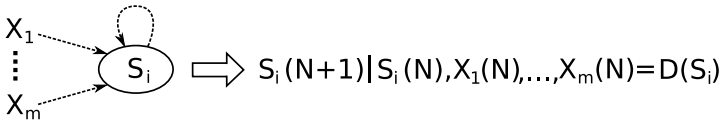



Fig. 5. Mapping of a component's state into a GLL rule.

described by a recursive GLL rule. We use $\mathtt{D}(\mathtt{S}_i)$ to denote the probability distribution corresponding to the function representing the temporal change of state $S_i$.

*3.* Connections between components are included in the corresponding GLL program according to figure 6. When the output $\mathtt{O}_i$ of component $\mathtt{C}_i$ is taken as

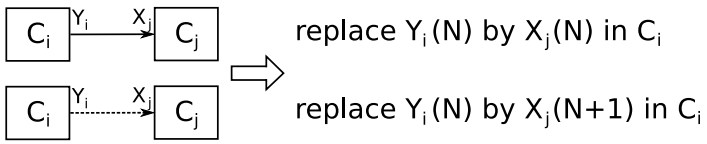



Fig. 6. Mapping connections between components into a GLL program.

an input $\mathtt{I}_j$ to component $\mathtt{C}_j$ at the same time step, we replace $\mathtt{O}_i(\mathtt{N})$ with $\mathtt{I}_j(\mathtt{N})$ in the rule representing the function of $\mathtt{C}_i$ producing $\mathtt{O}_i$. Otherwise, when the output is taken as an input at the next time step, we replace corresponding $\mathtt{O}_i(\mathtt{N})$ with $\mathtt{I}_j(\mathtt{N}+1)$. Recursive rules of GLL support capturing the time change in the system in a natural way.

During the mapping of the transfer-function diagram into a GLL program, the deterministic function, specified by a domain expert as the matrix ($\mathtt{D}(\mathtt{F}_i)$ of figure 4), is transformed into a probability distribution table with zeros and ones. Moreover, the noise and the rate of change can be simulated by adding a probabilistic bias to the deterministic function during this mapping. It is possible in GLL to omit specification of a probability distribution of a sentence by marking it as *learnable*. The GLL system uses an EM-based learning mechanism to infer this distribution from data. By using functions taken from experts' knowledge as an initial approximation of the system (prior knowledge) and then utilizing the learning capabilities of GLL, the model of the system is further refined to closer represent the domain. Transfer-function diagrams mapped into probabilistic graphical models are the initial deterministic patterns of contextual models that are further refined and evolved during context-sensitive probabilistic modeling.

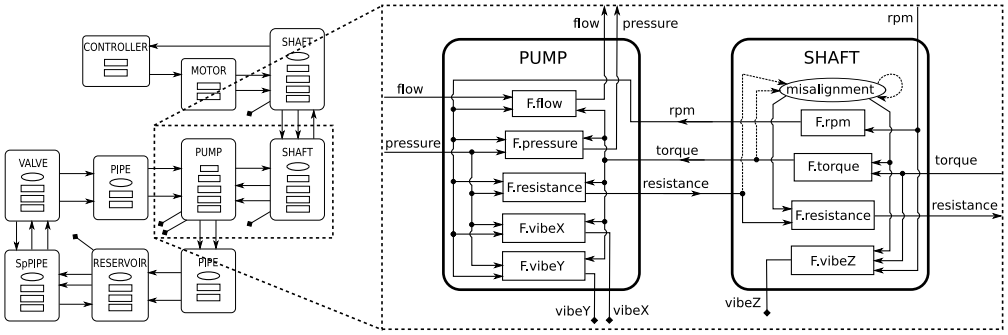



Fig. 7. The general transfer function diagram of the pump system.

In the pump system example, the transfer function diagram (figure 7) is mapped into a GLL program:

```
voltage<-{low,avg,hi}
torque<-{low,hi}
..snip..
voltage(N+1)|resistance(N)=[[0.099,0.9,0.001],[0.99,0.009,0.001]]
resistance(N)|torque(N),in-pressure(N)=Learn
..snip..
```

### *3.3 Failure-Driven Approach to Context-Sensitive Modeling*

To summarize, COSMOS is a probabilistic modeling system which addresses a context-sensitive modeling problem for real-time tasks. The idea of our system is to use a set of small relevant models instead of a single large network to address the computational complexity issue while maintaining or improving the modeling accuracy. By splitting the domain into contexts we are able to construct small models with reduced complexity that capture only the relevant relationships that are currently present in the data. Whenever a knowledge base changes, the learned model is not discarded as in batch processing, but stored for future use. The set of small models captures the different operational contexts of an environment, where each context represents local stationary behavior in the data. Combining these models together by invoking an appropriate model depending on changes in the context provides a means for handling global non-stationary behavior in the data. Consequently, our system attempts to maintain context awareness by incrementally monitoring data changes, dynamically switching currently active models as the context changes.

#### *3.3.1 The notion of context in probabilistic modeling*

We next define the notion of context. Similar to a linguistic context, in probabilistic reasoning a context helps to identify relevance and situational appropriateness of a *model*. We use probabilistic graphical models [23] as suitable representations for a model encoding selected features of knowledge and beliefs about a domain. We assume a *universal set of variables* $\mathbf{V}$ and the model is defined on its subset $\mathbf{U} \subseteq \mathbf{V}$. We follow Halpern and Pearl [9] by distinguishing *endogenous* variables and *exogenous* variables. Endogenous variables of a model $\mathcal{M}$ are the variables on which the model is defined: $En(\mathcal{M}) \equiv \mathbf{U}$. All the variables that are not in $\mathcal{M}$ are called exogenous variables: $Ex(\mathcal{M}) \equiv \mathbf{V} - En(\mathcal{M})$.

A logical conjunction of truth assignments to some exogenous variables of a model $\mathcal{M}$ is called a *context* $\mathcal{C}$ of $\mathcal{M}$. To extend this definition we can use variable assertions instead of truth assignments. We create contexts to capture some stable invariant subset of behaviors of the specified set of exogenous variables of a model: assuming the model fits a data set well, its context logically holds under the available data.

The idea of context-sensitive probabilistic modeling is to partition a continuous data stream into contexts (regions of contextual invariance) and to build a collection of small models, each of which represents a specific context. Every time we encounter a new context (or periodically return to the previous con-

text), the correponding model is envoked to reason under specific contextual conditions.

We next describe a multiple function estimation problem in an environment of changing situations. This is a multidimensional optimization problem: find an optimal collection of probabilistic models that represent a system in particular situations (contexts) accurately and efficiently. The collection of contexts may not be known a priori, therefore we have to find an optimal set of contexts to improve function estimation. Two properties of the set of contexts are accounted for when we search for optimal contexts: (a) context stability and (b) a rate of change of contexts. These properties reflect the belief that a context represents invariant behavior in an environment.

Let $\mathbf{D}$ represent a set of observed data. It is natural to assume that the data set is ordered, $\mathbf{D} = \{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_m\}$, where each $\mathbf{d}_i$, $i \leq m$, is a vector of observations recorded at the $i$th time step for each observable variable of the system. Therefore, we can define a function $s()$, which returns the successive data vector: for each $i < m$, $s(\mathbf{d}_i) = \mathbf{d}_{i+1}$.

The optimal collection of contexts is found by minimizing the error corresponding to how much each context from the set agrees with associated data as well as how many context transitions are present. Consider a collection of contexts $\mathbf{H} = \{\mathcal{C}_1, ..., \mathcal{C}_k\}$. Each context $\mathcal{C}_i$ represents observed invariant behavior in some data set $\mathbf{D}_i$. Therefore, our set of contexts $\mathbf{H}$ corresponds to some $\rho(\mathbf{D})$, a partition of $\mathbf{D}$ into $k$ mutually exclusive subsets $\mathbf{D}_1, \ldots, \mathbf{D}_k$. Note that each $\mathbf{D}_i$ might consist of a set of disjoint data regions instead of a single continous region (see figure 8). The data partition $\rho(\mathbf{D})$ corresponding to our set of contexts $\mathbf{H}$ has an error score associated with it

$$error(\rho(\mathbf{D})) = \sum_{j=1}^{k} [error'_j(\rho(\mathbf{D})) + error''_j(\rho(\mathbf{D}))],$$

where

$$error'_j(\rho(\mathbf{D})) = Pr_{\mathbf{x} \in \mathbf{D}_j}[\mathcal{C}_j(\mathbf{x}) = false],$$
$$error''_j(\rho(\mathbf{D})) = Pr_{\mathbf{x} \in \mathbf{D}_j}[\mathcal{C}_j(s(\mathbf{x})) = false \mid \mathcal{C}_j(\mathbf{x}) = true].$$

Here $\mathcal{C}_j(\mathbf{x})$ is an instantiation of context $\mathcal{C}_j$ on a data vector $\mathbf{x}$.[1] Informally, score $error'_j(\rho(\mathbf{D}))$ indicates the error rate we expect when applying $\mathcal{C}_j$ to instances drawn according to the probability distribution $\mathbf{D}_j$. It captures how much context $\mathcal{C}_j$ disagrees with data set $\mathbf{D}_j$ from data partition $\rho(\mathbf{D})$. Given a successful application of $\mathcal{C}_j$ to an instance, score $error''_j(\rho(\mathbf{D}))$ indicates the

[1] The notation $Pr_{\mathbf{x} \in \mathbf{D}_j}$ indicates that the probability is taken over the instance distribution $D_j$.

expected error rate when applying $\mathcal{C}_j$ to the next instance. Note that when $error'_j(\rho(\mathbf{D}))$ is minimal, $error''_j(\rho(\mathbf{D}))$ denotes the amount of instability (context changes) in the system's behavior described by context $\mathcal{C}_j$ and sampled with data $\mathbf{D}_j$. Figure 8 illustrates two data partitions corresponding to a set of two contexts. Note that $error''_j(\rho(\mathbf{D}))$ is higher for the first partition, see figure 8 (a), than for the second one, since there are less context transitions in (b). On the other hand, $error'_j(\rho(\mathbf{D}))$ is higher for the second partition, since there are more data that disagree with contexts. Minimizing these er-

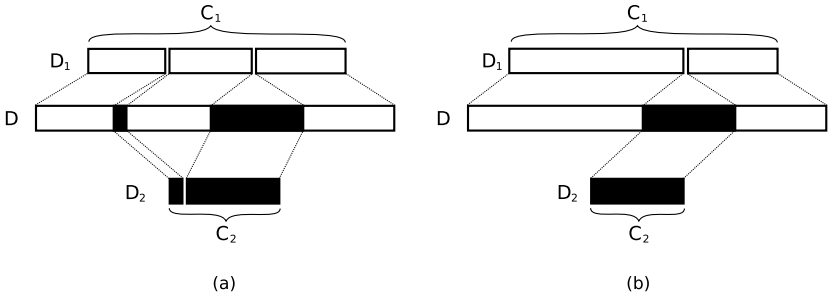



Fig. 8. Illustration of context partitioning.

rors, $error(\rho(\mathbf{D}))$, over all data partitions yields an error score $Score_1(\mathbf{H})$ for a given collection of contexts $\mathbf{H}$. Consequently, the optimal collection of contexts that represents the stable invariant behavior with the smallest number of context changes is found by minimizing the corresponding error score $Score_1(\mathbf{H})$ along every possible set of contexts. One can see that the problem of finding an optimal collection of contexts is similar to the problem of clustering the data according to some stable contiguous patterns.

Recall that each element $\mathcal{C}$ from $\mathbf{H}$ is a context of some model $\mathcal{M}$: there is a connection between $\mathcal{M}$ and $\mathcal{C}$. We view context $\mathcal{C}$ as a condition that *constrains* (structurally and parametrically) the set of all possible models associated with the context. In other words, $\mathcal{C}$ constrains $\Theta$, the parameters of $\mathcal{M}$, and $G$, the structure of $\mathcal{M}$.

Reducing the number of context transitions is important since each time the context changes we need to replace a model corresponding to the previous context with the one that corresponds best to the new context, and this model substitution could be computationally expensive. On the other hand, each model must represent a context (data) most accurately, and in the same time should be as small as possible to reduce the cost of inference. Therefore, while minimizing $Score_1(\mathbf{H})$ we maximize the probability for each $\mathcal{C} \in \mathbf{H}$:

$$\max_{\mathcal{C}\in\mathbf{H}}[Pr(G_c \mid \mathbf{D})] \propto \max_{\mathcal{C}\in\mathbf{H}}[Pr(G_c)Pr(\mathbf{D} \mid G_c)].$$

The prior $Pr(G_c)$ reflects our belief before seeing any data that the structure $G_c$ imposed by the context $\mathcal{C}$ is correct. Simultaneously, we minimize the structural complexity of a model to ensure that the structure of models is

parsimonious:

$$\min_{C \in \mathbf{H}}[size(G_c) + \max_{V \in G_c}[degree(V)]].$$

Here $size(G_c)$ stands for a number of edges in $G_c$, and $degree(V)$ denotes the number of other vertices connected to $V$ by edges (minimizing fan-in/fan-out). Note that if the complexity of a model is not decreased by constraints of the corresponding context, then the entire optimization problem described above can be reduced to a traditional structure search and the parameter estimation for a single model.

Modeling systems that address the context-sensitive probabilistic modeling problem are very important for carrying out complex tasks imposing additional constraints on the running time and memory of the modeling systems. We next describe our *failure-driven* probabilistic modeling that incorporates ideas from developmental learning [25] to model data from dynamic environments.

### *3.3.2 Model failure: an indicator of context change*

Our approach to the probabilistic modeling of changing contexts is based on ideas from developmental learning [25]. We argue that probabilistic inference systems can benefit greatly by emulating these mechanisms. In our psychologically inspired framework for learning, models become more tractable. Most new evidence is rapidly "assimilated" into *existing* small contextual models by updating their parameters, similar to an individual incorporating new events and objects into an *existing* way of thinking. Data sets that exhibit large scale deviations from previously learned models bring the more expensive "accommodation" mechanism into play that *reorganizes* the model to accommodate new data, similar to an individual *reorganizing* his/her existing mental structures to incorporate new information about novel aspects of an environment.

Context switching mechanisms, which are among the main components of our system, employ these two forms of learning within a *failure-driven architecture* (see figure 9). When new data are available, the system checks whether the

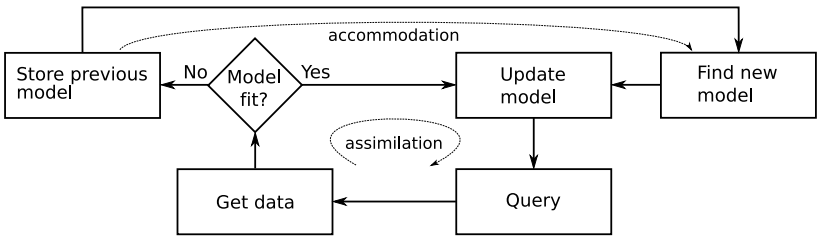



Fig. 9. The flow chart of the failure-driven architecture.

current model fits the dataset well. If it does, the data are incorporated into the model by updating its probability distribution (this is learning by *assimilation*: the model is consistent with new data and it is fine-tuned by assimilating the

dataset). Otherwise, if the model fails to fit the data, the system saves the current model and searches for a new version that will account for the new data (this is learning by *accommodation*: the model is inconsistent with new data, hence in order to account for the dataset the model has to be reorganized).

Failure detection is the core of the architecture (see the conditional "*Model fit?*" in figure 9). Assuming a continuous stream of data, the notion of failure represents the situation when new data are inconsistent with the current model. Essentially, model failure can be identified by estimating the likelihood of the data given the current model: the model fails when this likelihood is below a certain threshold.

The problem of identifying model failure is a special case of a statistical problem of detecting the distribution change from a stream of observations [27]. There are a number of approaches to this problem [2,32,34]. We provide a method that fits naturally into the iterative framework of our context-sensitive probabilistic modeling system.

The idea of the algorithm is to monitor a selected subset of model parameters (triggers) and declare model failure when the true parameters of the model are considerably different than these estimated from new data. The algorithm iteratively checks for failure in the specified data window and, if no break-down is detected, slides the data window further along the data stream. Note that our failure detection is controlled by the size of the data window, the size of the window shift, and the threshold indicating model failure. In general the problem of finding the appropriate window and threshold parameters can be seen as an error minimization problem. Ultimately, we would like to find the parameters that would minimize false positive and false negative errors.

Minimizing false positive error is relatively easy: we partition the training dataset into two subsets, use the first subset to train the model, and employ the second subset to determine window parameters such that the failure detector finds no failure on the second subset. Note that if we have a third training dataset on which the algorithm is expected to find failure, we can perform a similar routine minimizing false negative error to further constrain the parameter set.

Preliminary testing of the algorithm across various window and overlap sizes shows that failure detection becomes more sensitive to data noise when the window size is small, which results in a higher likelihood of a false positive error. Larger windows lead to less sensitive failure detection along with a detection lag, when model failure is only identified after the break-down has occured. Additionally, larger windows demand more computational power.

In general, without an appropriate data set, minimizing a false negative error is a challenging problem. The problem becomes even more difficult when the

difference between two very similar distributions must trigger model failure. A possible way of selecting the window/threshold parameters without a training set for failure detection is to employ data variance. Intuitively, we would like to know the size of a representative subset of the training data and a data window, the variance of which is close to the true variance of the training data. A steep change in variance of such a data window would be a good indicator that the data came from a new distribution. Consider a window with size $K$ and draw $N$ subsets of data by randomly sliding the window along the training dataset. Computing an average variance over $N$ data subsets for a large enough $N$ produces an estimate of our confidence that a window of $K$ elements drawn from the training dataset captures the underlying dependencies observed in the entire training dataset.

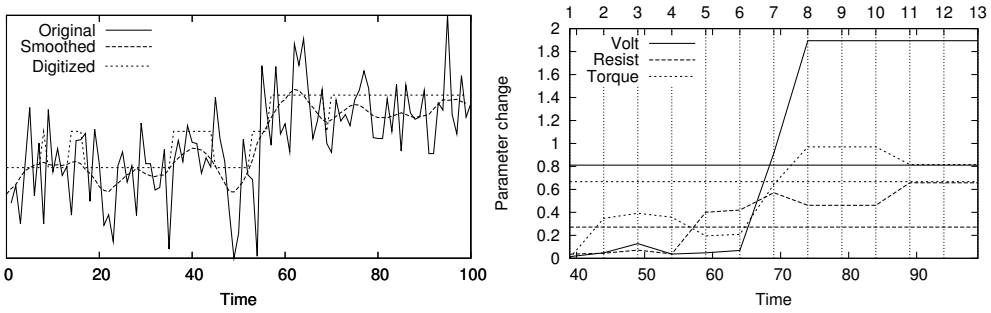
Testing using the data from the pump system shows that at some moment error bars of the variance monotonically decrease as the window size increases: the more data we take, the less changes in the data variation we get. We can automatically select the window as soon as the error bars drop below a certain level as the window size increases.

Once the window size is set, the failure threshold is found by computing an average difference (the Frobenius norm [2]) between a true model parameter and its estimate computed from the window of the training data. Essentially, we can execute the failure detection algorithm using the window of training data and employ the computed difference as the failure threshold.

To illustrate the failure detection method we use temporal data obtained from a variety of sensors installed on the mechanical pump system of Section 1. These sensory data consists of a time series of three parameters: pressure coming into a pump (*InPr*), pressure generated by the pump (*OutPr*), and voltage at the motor driving the pump (*Volt*). In order to estimate the behavior of the pump system depending on how clogged the filter is, we control the valve regulating the amount of fluid coming into the pump (as opposed to literally contaminating the system). During the experiment the pump system starts normal operation with the valve fully open. As the time passes a certain point (around the 53d time step), we partially close the valve to limit the flow of the fluid coming into the pump. A series of 100 data steps is recorded during the experiment. Each signal is then smoothed using a sliding window approach and digitized. Figure 10(a) illustrates the time series of one of the paprameters (*OutPr*) of the pump system.

We selected 35 first time steps to train a DBN, each time slice of which contains 2 hidden variables (resistance at the pump, *Resist*, and torque of the motor, *Torque*) and 3 observable variables (*InPr*, *OutPr*, *Volt*). The appropriate size

[2] A Frobenius norm of a matrix $A = (a_{ij})_{kl}$ is defined as $\|A\|_F = \sum_{i=1}^{k} \sum_{j=1}^{l} |a_{ij}|^2$.




(a) (b)

Fig. 10. (a) The time series of the pressure generated by the pump (*OutPr*) and its smoothed and digitized versions. (b) Failure detection using a 17 point wide sliding window and 12 point overlaps performed on the model trained on data from the pump system. Each horizonatal line corresponds to a failure threshold: once a corresponding change of a distribution (a difference between true and learned model parameters) goes above this threshold, the failure detector signals a model break-down. The grid corresponds to window shifts.

of training data (35 time steps) was identified by leave-one-out cross-validation across models trained on data sets of various sizes.

Figure 10(b) illustrates the performance of the failure detector on the sensory pump data for the trained model (plotted for three model parameters: motor voltage (*Volt*), pump resistance (*Resist*), and motor torque (*Torque*)). The window size (17 data points) and shift (5 data points) as well as the thresholds linked with each model parameter were automatically identified using the method described above.

Recall that the real model break-down happens around time step 54, when the valve of the pump system is partially closed. By monitoring parameter *Resist* the failure can be identified soon – at step 59 (after 4 window shifts), where as by monitoring parameters *Volt* and *Torque* the failure is idetified much later (at around step 69).

In order to avoid incurring costs of detecting failure across a large model, we specify a small subset of *trigger* parameters, whose changes have been identified as most important by the domain experts and are often indicative of model failure. Instead of checking for failure in the entire model, only the set of trigger parameters is monitored. Full-fledged failure detection is enforced once a change in a trigger parameter is discovered. Since different parameters give different detecting performance, it can be useful to employ a combination of these. A two-layerd failure detection can also be used, where a parameter that is prone to data noise but useful in detecting early failure, like *Resist* in figure 10(b), can trigger an alert mode, in which case a more stable parameter, such as *Voltage*, is analyzed to confirm the detected model break-down. Further details on failure detection can be found elsewhere [2,32,34].

COSMOS employs context to reduce the number of stochastic relationships necessary to represent dynamic change. COSMOS ultimately reduces the complexity of probabilistic models by switching between tuned-up, knowledge-focussed models as necessary to represent changing contexts, using the failure-driven iterative approach described in the previous section.

The multi-layer architecture of COSMOS consists of an ensemble of related probabilistic models, a representation of contextual states, and a mechanism for switching between active models based on detected context changes. It also uses a deterministic domain knowledge to manage the adaptation of individual models to context changes. COSMOS incorporates an incremental failure-driven mechanism for learning and repair based on abductive causal reasoning and EM parametric learning that together generate new models better adapted to handle behavior in changing environments.

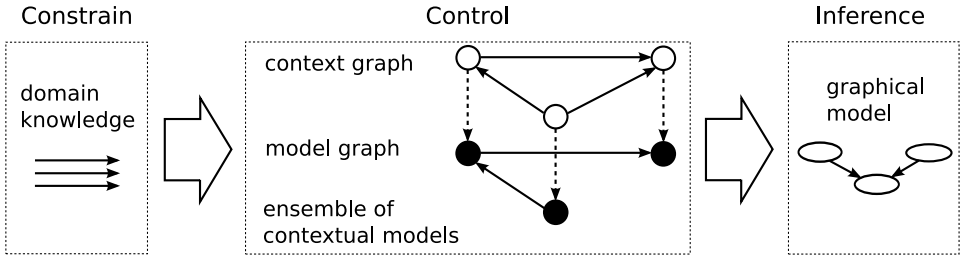



Fig. 11. A diagram of the relationship between domain knowledge, contexts, and models in the COSMOS hyper-model.

A key component of COSMOS is the ensemble of contextual models consisting of a context graph linked with a model graph (figure 11). The context graph represents what we want to learn (the concrete situation), where the model graph is its approximation, and where some contexts or transitions have not yet been encountered (figure 11). The ensemble of contextual models is represented as a finite state machine where contexts correspond to states and transitions between contexts correspond to transitions between states. Context transitions, the edges in the graph, are managed by a domain knowledge base (the COSMOS' top layer). Note that each context, a node in the context graph, is associated with a graphical model comprising COSMOS' lowest layer. The system incrementally populates the ensemple of models by iterating between an assimilation step for model tuning and an accommodation step for model repair and reorganization (see figure 9).

*Initialization.* The incremental process starts with an initial ensemble of expected operational contexts with transitions and corresponding models specified by the domain expert. Transfer-function diagrams provide an interface for model specification. The models are then trained on given datasets for

individual contexts using loopy belief propagation for inferencing. Finally, the expert selects an initially active model from the library to begin.

*Assimilation.* Once initialized, the system considers newly encountered data and checks whether the currently active model fits the data. Using our failure detection algorithm based on the learning mechanism of GLL, a steep change between actual and estimated parameters indicates model failure. COSMOS moves into the accommodation step (next) when model failure is detected, otherwise the system inferences across the model and outputs whatever diagnostic information the user requests.

*Accommodation.* Once model failure is detected, the system identifies every model parameter (endogenous variable) that is unstable. A domain knowledge base specified by Horn-clause rules is used to detect exogenous variables that can cause model changes. We backtrack along the rules to "explain" the changes in the endogenous variables. As a result, this abduction mechanism produces a model that corresponds to the new context. In general, abduction can produce multiple possible explanations (assertions to exogenous variables) of the model failure, identifying a neighborhood of possible context transitions. COSMOS traverses the models linked to the contexts from the neighborhood to find the one that best fits the data. Once the model is found, the system shifts back to the assimilation step. If no appropriate model is found, COSMOS expands the library by building a new contextual model. We believe this structure search can be significantly reduced by using contexts constrained by domain knowledge.

The context-sensitive probabilistic modeling system is an attempt to maintain situation awareness over time. Partitioning the world on context makes the cost of detecting context changes (estimating the target variables) much lower due to the small size of corresponding models. On the other hand, context stability reduces the amount of potentially expensive context estimations (those that require searching for an appropriate model in the ensemble).

## 4 Evaluation of context change and model sensitivity

In this section we consider data from the pump system presented in Section 1. The model whose structure is given in figure 12 is trained on the data corresponding to the normal operational context of the pump system. We describe our results through a set of experiments that test the detection of context changes and demonstrate switching to an appropriate model.

*Experiment 1.* During the pump system operation, the valve is partially closed to simulate a clogged filter. Figure 13(a) shows the difference between the true

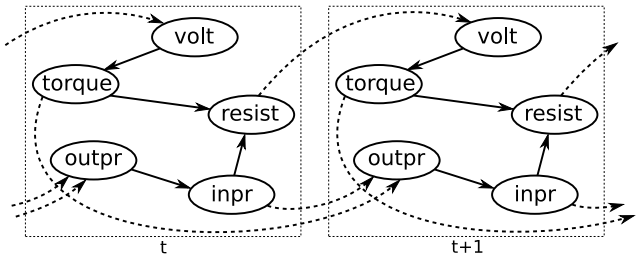
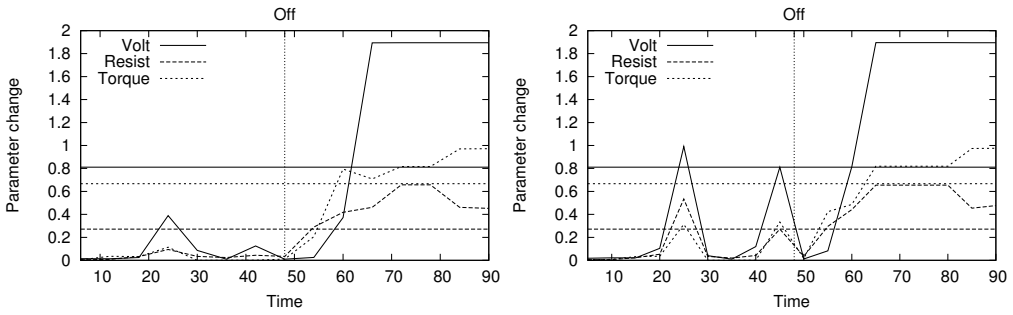



Fig. 12. A DBN corresponding to the GLL program given in section 3.2. Solid arrows correspond to isochronal dependencies, while dashed arrows represent temporal dependencies.




(a) (b)

Fig. 13. Failure detection on three model parameters (*Volt*, *Resist*, *Torque*) in the data stream when the pump system initially operates normally, but then changes behavior at the 48th time step, when the flow valve is partially closed. Figure (a) shows predictions performed using a sliding window of 17 points with 12 point overlaps, whereas figure (b) illustrates these predictions when the sliding window is 15 points wide with 10 point overlaps. The vertical line *Off* shows an actual system break down, while horizontal lines correspond to failure thresholds for the corresponding model parameters (same font type). Figure (a) shows that using *Resist* the context change is detected at the 55th step, using *Torque* the change is detected at the 60th step, and using *Volt* the change is detected at the 65th step. Figure (b) shows that the context change is falsely detected at the 25th time step.

and the estimated parameters *Resist*, *Torque*, and *Voltage* computed over a sliding window. It can be seen that COSMOS identifies the context transition between the 55th and the 65th time step depending on which parameter is used. However, the real context transition occurs at the 48th time step, when the valve is partially closed. In an attempt to capture the context transition earlier, we reduced the size of the window. Figure 13(b) shows that COSMOS prematurely detects a context change at the 25th step.

*Experiment 2.* In this experiment, the pump system starts operating normally, then the valve is partially closed, simulating a highly clogged state of the filter. The valve is then opened and the pump system returns to its normal state of operation. We try to fit several different models representing various operational contexts. Figure 14(a) illustrates the difference between the true

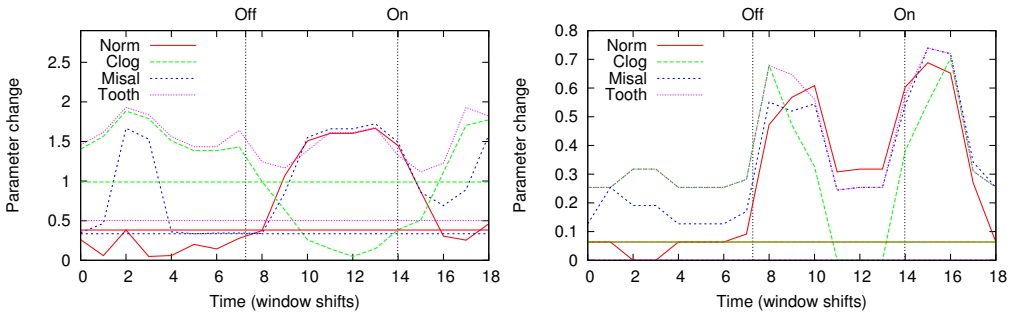



Fig. 14. An illustration of an experiment when the pump system starts operating normally and the valve is partially closed (at the *Off* time step), which is then reopened (at the *On* time step). The detection of context changes is performed using a sliding window of 21 points with 14 point overlap. Vertical lines *Off* and *On* show moments of time when the valve was closed and then reopened. Horizontal lines correspond to failure detection thresholds for the corresponding model parameters. Figure (a) shows the difference between the true *Voltage* parameter and the parameter estimated on the sliding window for various models: a model (*Norm*) trained on data from regular conditions, a model (*Clog*) corresponding to a partially closed flow valve, a model (*Misal*) representing the data when the pump is misaligned, and a model (*Tooth*) capturing the situation when a gear tooth is chipped. Figure (b) shows the same information for the *In Pressure* parameter.

*Voltage* parameter and its estimation using a 21 point wide window. After each estimation the window is shifted 14 points further constructing the time series of parameter changes. Plotting these time series for different models, we can see that the model *Norm* represents the initial normal operation of the pump system better than any other model. When the context changes to *high clogging*, the model *Clog* best represents the changed context. Figure 14(b) shows the same analysis using the parameter *In Pressure*. Note that two "hills" after each context change correspond to the periods of disequilibrium in the physical pump system: when the valve is partially closed, the motor overcompensates for the change at first, but then returns to the optimal state.

*Experiment 3.* The pump system starts operating with one gear having a chipped tooth. Figure 15(a) illustrates the difference between the true and the estimated parameter *Voltage* computed on a sliding window for various models. Figure 15(a) shows that the model *Tooth* represents the current context better than other models, though the behavior of the model *Misal* is close to that of *Tooth*. Figure 15(b) shows the same analysis for the parameter *In Pressure*.

*Experiment 4.* We next evaluate the performance of COSMOS, where we expect smaller models to be more accurate than larger models trained on larger data sets. We trained two contextual models of COSMOS on the training data sets corresponding to two contexts (context 1 and 2). COSMOS models are then compared to a structurally similar model trained on a data set

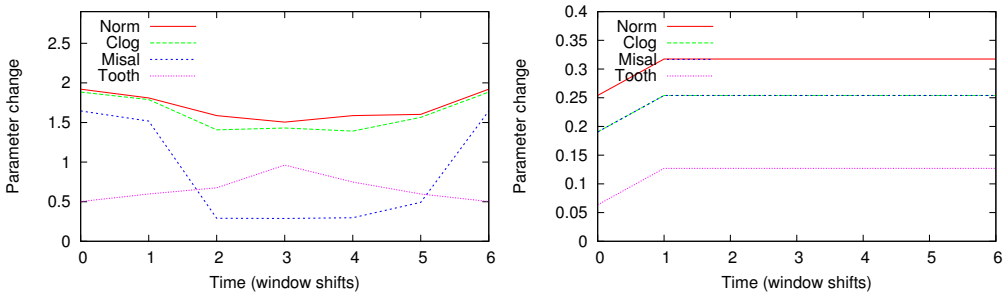



Fig. 15. An illustration of the experiment when the pump system starts operating with a gear having a broken tooth. The figure shows the same information as in figure 14. Figure (a) shows the analysis for the *Voltage* parameter whereas figure (b) the *In Pressure* parameter.

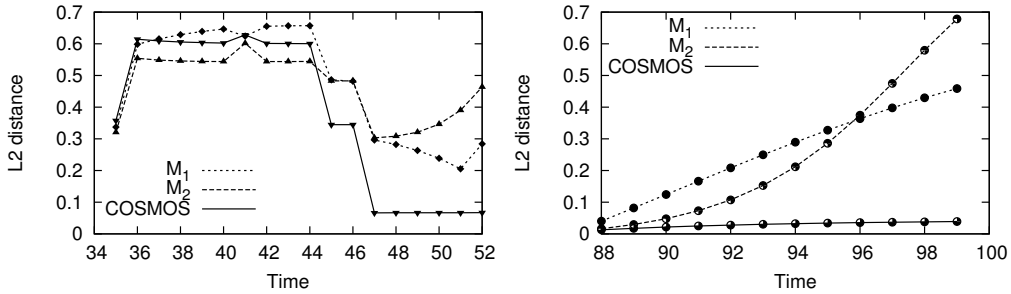



Fig. 16. A plot of L2 distance between model predictions and data demonstrating the extrapolation performance (a) in context 1 and (b) in context 2. In (a) the data is noisy during the 35-45 time interval and noise is reduced after the 45th time step. COSMOS switches to model $MC_1$, whilem in (b) COSMOS switches to model $MC_2$.

combined from both contexts. The contextual models are also compared to another structurally more complex model trained on the combined data set. Figure 16(a) shows extrapolation in context 1 and figure 16(b) shows extrapolation in context 2.

Note that COSMOS and the two models perform similarly when the data is noisy (context 1), however COSMOS outperforms the other models as the data noise decreases (the end of context 1 as well as of context 2). The models trained on combined data overfit that data; this is further supported by the information in Table 1. Leave-one-out cross-validation is performed on the training data set for Context 1 and Context 2, see Table 1.

In order to illustrate the running time complexity of COSMOS, we compared the number of iterations needed to converge for COSMOS (on average) as well as for the two large models. The iterations are given in Table 2. Note that COSMOS models on average require fewer iterations to convergence because

these models are smaller and more focused. Table 3 shows the average number of iterations the learning algorithm took to converge during the leave-one-out cross-validation.

| Model | Context 1 (Var) | Context 2 (Var) |
|---|---|---|
| $M_1$ | 0.13546 (0.027373) | 0.10175 (0.043229) |
| $M_2$ | 0.14649 (0.029093) | 0.091425 (0.046431) |
| *COSMOS* | 0.13340 (0.045253) | 0.091472 (0.049549) |

Table 1
The average difference (L2 distance) calculated during leave-one-out cross-validation. The distance is computed between predicted values and noisy data.

| Model | $M_1$ | $M_2$ | *COSMOS* |
|---|---|---|---|
| Iterations | 97 | 100 | 47 |

Table 2
The number of iterations of the parameter estimation algorithm before convergence. The models are trained using an EM-based algorithm implemented with loopy belief propagation.

| Model | Context 1 | Context 2 |
|---|---|---|
| $M_1$ | 99 | 95 |
| $M_2$ | 99 | 98 |
| *COSMOS* | 63 | 57 |

Table 3
The average number of iterations required during each iteration of leave-one-out cross-validation.

## 5 Conclusions and Future Directions

In this paper we described COSMOS, a system for context-sensitive probabilistic modeling. This system uses transfer-function diagrams to capture organizational, hierarchical, and representational constraints in system components. COSMOS handles the complexity issues found in modeling dynamic environments by focusing on only relevant parts of the environment. Since each context represents some stable, invariant behavior over a time period, it can be represented by a considerably smaller model, focused on this relevant knowledge. Such knowledge-focused models provide accurate results for a specific context, yet require less training information. COSMOS models non-stationary behavior of the dynamic environment by sequentially applying contextual models which represent subsets of stable behavior.

Although represented by a stochastic model, a context, as currently defined, can change only deterministically (represented as a finite state machine in section 3.4). Extending the definition of context to allow stochastic context transitions (replacing a finite state machine with a probabilistic one) will potentially increase the reasoning power of COSMOS. This extension is linked to the general direction of developing more robust domain knowledge representations. Other directions for future research include extending the mechanism for detecting context transition events, for example, choosing trigger variables, combining various sliding windows, etc.

## 6 Acknowledgements

The research presented in this paper is part of the PhD dissertation of the first author under the supervision of the second. We thank Carl Stern from Management Sciences, Inc. and Roshan Rammohan for support and numerous discussions. We also acknowledge the help of Tom Caudell and Lance Williams, who served on this PhD committee. The first author was supported by the Air Force Research Laboratory SBIR contract (FA8750-06-C0016).

## References

[1] J. Barwise, Conditionals and Conditional Information, On Conditionals (1986) 21–54.

[2] S. Dayanik, C. Goulding, H. V. Poor, Joint Detection and Identification of an Unobservable Change in the Distribution of a Random Sequence, Information Sciences and Systems (2007) 68–73Issue of 41st Annual Conference on Volume.

[3] A. Dempster, N. Laird, D. Rubin, Maximum likelihood from incomplete data via the EM algorithm, Journal of the Royal Statistical Society, Series B (Methodological) 39 (1) (1977) 1–38.

[4] N. Friedman, L. Getoor, D. Koller, A. Pfeffer, Learning Probabilistic Relational Models, 1999, pp. 1300–1307, proc. of 16th Intl. Joint Conf. on AI (IJCAI).

[5] L. Getoor, N. Friedman, D. Koller, A. Pfeffer, Learning Probabilistic Relational Models, Relational Data Mining (2001) 307–335.

[6] A. Gopnik, C. Glymour, D. M. Sobel, L. E. Schulz, T. Kushnir, D. Danks, A theory of causal learning in children: Causal maps and Bayes nets, Psychological Review 111 (1) (2004) 3–32.

[7] N. Granott, K. W. Fischer, J. Parziale, Bridging to the unknown: a transition mechanism in learning and development, Microdevelopment: transition processes in development and learning.

[8] P. Haddawy, Generating Bayesian Networks from Probability Logic Knowledge Bases, Morgan Kaufmann, 1994, pp. 262–269, proc. of 10th Conf. on Uncertainty in AI.

[9] J. Halpern, J. Pearl, Causes and Explanations: A Structural-Model Approach — Part 1: Causes, San Francisco, CA: Morgan Kaufmann, 2001, pp. 194–202, proc. of 17th Conf. on Uncertainty in AI (UAI-01).

[10] D. Jensen, J. Neville, Schemas and Models, 2002, pp. 56–70, proceedings of the First SIGKDD Workshop on Multi-Relational Data Mining (MRDM-2002), University of Alberta, Edmonton, Canada.

[11] K. Kersting, L. DeRaedt, Bayesian Logic Programs, 2000, pp. 138–155, proc. of 10th Int. Conf. on ILP.

[12] K. Kersting, L. DeRaedt, Basic Principles of Learning Bayesian Logic Programs, Tech. Rep. 00174, albert-Ludwigs University at Freiburg (2002).

[13] D. Koller, A. Pfeffer, Object-Oriented Bayesian Networks, 1997, pp. 302–313, proc. of the 13th Conf. on UAI.

[14] H. Langseth, O. Bangso, Parameter Learning in Object-Oriented Bayesian Networks, Annals of Mathematics and Artificial Intelligence 32 (1–4) (2001) 221–243.

[15] K. Laskey, S. Mahoney, Network Fragments: Representing Knowledge for Constructing Probabilistic Models, 1997, pp. 334–340, proc. of the 13th Conf. on UAI.

[16] G. F. Luger, Artificial intelligence: Structures and Strategies for Complex Problem Solving, Addison-Wesley, 2009.

[17] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, B. Schölkopf, An introduction to kernel-based learning algorithms, IEEE Transactions on Neural Networks 12 (2) (2001) 181–201.

[18] K. P. Murphy, Y. Weiss, M. Jordan, Loopy Belief Propagation for Approximate Inference: An Empirical Study, in: Uncertainty in Artificial Intelligence, 1999, pp. 467–475.

[19] L. Ngo, P. Haddawy, Answering queries from context-sensitive probabilistic knowledge bases, Theoretical Computer Science 171 (1–2) (1997) 147–177.

[20] L. Ngo, P. Haddawy, R. A. Krieger, J. Helwig, Efficient Temporal Probabilistic Reasoning via Context-Sensitive Model Construction, Computers in Biology and Medicine 27 (5) (1997) 453–476.

[21] N. J. Nilsson, Teleo-Reactive Programs and the Triple-Tower Architecture, Electronic Transactions on Artificial Intelligence 5 (2001) 99–110.

[22] F. V. J. O. Bangsø, J. Flores, Plug and play object oriented Bayesian networks, LNAI 3040, 2004, pp. 457–467, proc. of the 10th Conf. of the Spanish Assoc. for AI.

[23] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, 1988.

[24] J. Pearl, Causality: Models, Reasoning, and Inference, Cambridge University Press, 2000.

[25] J. Piaget, Piaget's theory, Handbook of Child Psychology 1.

[26] D. J. Pless, C. Chakrabarti, R. Rammohan, G. F. Luger, The Design and Testing of a First-Order Stochastic Modeling Language, International Journal on Artificial Intelligence Tools 15 (6) (2006) 979–1005.

[27] M. Pollak, Optimal Detection of a Change in Distribution, The Annals of Statistics 13 (1) (1985) 206–227.

[28] D. Poole, Logic Programming, Abduction and Probability: a top-down anytime algorithm for estimating prior and posterior probabilities, New Generation Computing 11 (3–4) (1993) 377–400.

[29] M. Richardson, P. Domingos, Markov Logic Networks, Machine Learning 62 (1–2) (2006) 107–136.

[30] M. J. Sanscartier, E. Neufeld, Identifying Hidden Variables from Context-Specific Independencies, AAAI Press, 2007, pp. 472–477, proceedings of FLAIRS-07 Conference.

[31] D. L. Silver, R. Poirier, Context-Sensitive MTL Networks for Machine Lifelong Learning, AAAI Press, 2007, pp. 628–633, proceedings of FLAIRS-07 Conference.

[32] X. Song, M. Wu, C. Jermaine, S. Ranka, Statistical Change Detection for Multi-Dimensional Data, 2007, pp. 667–676, proceedings of the 13th Intnl. Conf. on Knowledge Discovery and Data Mining (KDD'07).

[33] S. Srinivas, Modeling techniques and algorithms for probabilistic model-based diagnosis and repair, Ph.D. thesis, KSL, CS Dept., Stanford University (1995).

[34] M. Steyvers, S. Brown, Prediction and Change Detection, Advances in Neural Information Processing Systems 18 (2006) 1281–1288.

[35] S. Thrun, Lifelong learning: A case study, Tech. Rep. CMU-CS-95-208, cS Dept., Carnegie Mellon University (1995).

[36] P. Turney, The Identification of Context-Sensitive Features: A Formal Definition of Context for Concept Learning, 1996, pp. 53–59, proceedings of Workshop on Learning in Context-Sensitive Domains at the 13th ICML.

[37] T. A. van Dijk, Discourse, context and cognition, Discourse Studies 8 (1) (2006) 159–177.