N.A.Sakhanenko, G.F.Luger. Failure-driven context switching with logic-based stochastic models.
JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 24(X): 1–end Mon. 2009

Model Failure and Context Switching Using Logic-based Stochastic Models

Nikita A. Sakhanenko[1,2] and George F. Luger[2]

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

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

E-mail: nsakhanenko@systemsbiology.org; luger@cs.unm.edu

Received month day, year.

Abstract This paper addresses parameter drift in stochastic models. We define a notion of context that represents invariant, stable-over-time behavior and we then propose an algorithm for detecting context changes in processing a stream of data. A context change is seen as model failure, when a probabilistic model representing current behavior is no longer able to "fit" newly encountered data. We specify our stochastic models using a first-order logic-based probabilistic modeling language called Generalized Loopy Logic (GLL). An important component of GLL is its learning mechanism that can identify context drift. We demonstrate how our algorithm can be incorporated into a failure-driven context-switching probabilistic modeling framework and offer several examples of its application.

Keywords Context, Failure-driven online learning, Probabilistic reasoning

1 Context-Based Diagnostics

In real-time diagnosis, where observations are given as a data stream, reasoning often has to be performed under strict time constraints with limited amounts of data available at each time step. This diagnostic problem can be simplified by a context-based approach where the data stream is partitioned into stable regions (contexts) and a separate model is built for each context. The complexity of a model representing stable behavior under a context is often considerably reduced since most of the contextually irrelevant information is left out during modeling. Reduced models often require less training data. In this paper we define the notion of context capturing stable data patterns and propose an algorithm for detecting when these contextual patterns change. A context change is identified by model failure when the current model no longer fits the incoming data. Our representation is based on a first-order logic-based probabilistic modeling language that combines the

This work was funded by a US Air Force Research Laboratory SBIR contract (FA8750-06-C0016).

power of first-order logic with the ability to handle uncertainty and noise.

Using a compact logic-based representation supports knowledge-based model construction (KBMC)[1] that represents general knowledge expressed as first-order rules in a specific, contextually relevant probabilistic model. As opposed to using probabilistic graphical models directly, applying logic-based representations facilitates systems diagnosis, since logical rules provide a bird's eye view of a graphical model (built using KBMC) masking some complexity out.

Probabilistic modeling systems that dynamically represent changing data are important for carrying out complex diagnostic reasoning tasks. With the increasing use of remote sensing technology continuously and concurrently collecting large sets of data, it becomes more necessary to develop a methodology for processing noisy data in a timely manner. Since modern sensing systems are often supported by very large sensor networks, the standard approach of collecting and processing all data at a central location is rarely efficient and it becomes necessary to shift aspects of the computation to the sensors where the data are collected. This introduces additional constraints on the running time and memory of the modeling system. The most suitable systems in these cases, we believe, are those that are able to *evolve* to handle rapidly changing pieces of information. There is a limitation, however, that makes current probabilistic modeling unable to support this evolution: many approaches assume that modeling is done only once and that the entire dataset is available ahead of time. In this paper we define *context* and introduce *failure-driven context-switching* probabilistic modeling that incorporates ideas from developmental learning, including assimilation and accommodation[2], to model streams of data from dynamic environments.

In Section 2 we overview related research. In Section 3 we give a definition of context and describe the context-sensitive modeling problem underlying this research. In Section 4 we describe generalized loopy logic (GLL), a first-order logic-based reasoning language we employ to specify contextual models and to perform inferencing over them. In Section 5 we propose an iterative algorithm for online detection of context transitions. In Section 6 we show how our context detection algorithm can be incorporated as a component of a larger context-sensitive modeling system and provide several examples. Finally, in Section 7 we give ideas for future research and conclude.

2 Related Research

Logic-based representations for stochastic modeling have been proposed by a number of researchers. Poole[3] was one of the first to develop an approximate inference algorithm for a Turing complete probabilistic logic language where uncertainty is expressed through sets of mutually exclusive predicates annotated with probabilities. Although successful at employing probability to handle uncertainty and noise in logic-programs, applications maintaining the correct normalization across the rules can be complicated and is left to the user. Haddawy[4] created a first-order probabilistic logic that he used to specify a static class of (propositional) Bayesian networks (BNs) as a knowledge base. Haddawy proposed a provably correct Bayesian network generation algo-

rithm that was later adapted to focus the knowledge base on the relevant information[5, 6]. The approach in [5] is one of the first attempts to explicitly use contextual information in probabilistic modeling. Their logic-based stochastic modeling approach utilizes context as a way to reduce the size of a model.

We see the methods of Haddawy[4], Ngo and Haddawy[5], and Ngo et al.[6] as examples of knowledge-based model construction (KBMC) that uses a query to project a general knowledge base down to a particular model sufficient to answer the query. The work by Glesner and Koller[7] is also largely based on the idea of KBMC. Their approach extends [4] to the temporal case by adding a time argument to the predicates and subsequently mapping the temporal knowledge base to a dynamic Bayesian network. Like Haddawy[4], Glesner and Koller attempt to focus the knowledge base to the relevant information by representing a conditional probability table as a decision tree which is invoked during the model construction step in order to decrease the size of the resulting model.

PRISM[8] and its later extensions[9] offer another method for statistical parameter learning on logic programs represented as a set of Horn clauses. In PRISM, a parameterized program expresses distributions over the set of all possible truth assignments to ground atoms provable from the program. Thus, facts in PRISM programs are probabilistically true, while other formulas representing if-then laws are always true. Sato and Kameya[8] set up a statistical abduction framework where special atoms expressing a probabilistic switch between finite alternatives are abducibles from which explanations are constructed as conjunctions. PRISM attempts to get around combinatorial problems by constraining the allowed programs to guarantee that the formula describing all proofs of a query is a combination of disjoint products, which is limiting for non-context-free grammars. ProbLog[10] extends PRISM by attaching probabilistic labels to all clauses (not just to ground facts) and consequently handling the situation when several facts are simultaneously true. Labels in ProbLog refer to the probability that the corresponding clauses are true, and these probabilities are mutually independent.

Friedman et al.[11] and, later, Getoor et al.[12] proposed probabilistic relational models (PRMs) that differ from other approaches[5, 13, 14] by specifying a probability model using classes of objects rather than simple attributes. For example, an explicitly identified relational structure of PRMs (similar to relational DBs) supports probabilistic dependencies between attributes of related objects. PRMs[11, 12] use maximum likelihood parameter estimation for parameter learning, while structure learning is done through a heuristic search of the best scores in a hypothesis space.

Bayesian logic programs (BLPs) is another knowledge-based model construction approach proposed in [13]. This framework generates Bayesian networks specific for given queries using a set of first-order Prolog-like rules with uncertainty parameters. Kersting and DeRaedt[13] represent a Horn clause as a probability formula such that a Horn clause head is conditioned on the body. The conditional probability distribution quantifying this relation is attached to the formula.

A recent approach called Bayesian knowledge

base (BKB)[15] extends the logic-based probabilistic reasoning to a temporal domain. BKBs can be seen as an extension of BLPs for handling different cyclic and recursive structures and incorporating logical context. Similarly to BLPs, BKBs are mapped to Bayesian networks, however if there are cyclic relationships then BKBs are mapped to two-slice stationary dynamic Bayesian networks.

Another approach called relational dynamic Bayesian networks (RDBNs) extends Bayesian networks (in this case dynamical) to relational domains[16]. In this work Sanghai et al. take first-order formulas extended to perform various aggregation (such as count) and combine them with conditional probability tables represented as first-order probability trees (probability estimation trees extended to the first-order case). This approach can be seen as an extension of a decision tree representation for CPTs proposed by [7]. RDBNs aim at state estimation in temporal domains and employ particle filtering for inference.

As opposed to earlier approaches, such as BLPs, that are based on restricted logic subsets (Horn clauses), Richardson and Domingos[14] propose Markov logic networks (MLNs) that use 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. Kersting and DeRaedt[13], Ngo and Haddawy[5], and various other approaches[7, 16, 15] propose using Bayesian Networks for inference.

In this paper, we choose a significantly different direction than the approach of Richardson and Domingos[14] using both 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 argue that Horn clauses provide modeling power by preserving the expressivity and at the same time supporting the embedding of various heuristics. We use a stochastic language, called Generalized Loopy Logic (GLL), described in detail in section 4, that combines Horn clauses with BNs similarly to BLPs[13]. As opposed to BLPs, GLL uses Markov networks for inference that naturally handles the product distribution combining rule and adds EM-based parameter learning (see section 4.2).

Although our research is motivated by [4, 5], their contextual mechanism cannot reflect all the complexity of the internal structures of data. Ngo and Haddawy define a context of a rule (a Horn clause) specified as a conjunction of logic predicates that are evaluated by a contextual logic program. Shen[15] followed a similar direction and extended each first-order rule representing a relation with predicates describing context. His approach integrates contextual constrains together with a logical program describing relations in a domain, as opposed to [5] where context and the logical knowledge base are evaluated disjointly. A number of other approaches use context indirectly defined through conditional independences[17, 18] or through decision trees[7, 16] to refine the models and improve their performance.

In this paper, we create a more general context for a model as opposed to assigning contextual constraints to each rule independently. We provide a formal specification of context as truth assignments to a specific set of variables that we know

about. The choice of variables is similar to the approach of Pearl[19] and Halpern and Pearl[20] that uses exogenous variables (that are not in the model) to identify a background for the possible causes of an event. Other stochastic logic-based methods can be seen as examples of knowledge-intensive modeling, however they assume that all the data are given at the start of problem solving, and thus cannot be applied efficiently to domains with distinct contextual changes.

In our paper, the definition of context (next section) describes the framework for tracking concept drift and identifying context change.

3 Specifications for Context Modeling

We next introduce the general problem of learning with context-sensitive probabilistic models by first introducing a formal notation. Italic uppercase letters (X, Y,Z) denote *variables*, and italic lowercase letters (x, y, z) represent their instantiated *values*. Similarly, bold uppercase letters ($\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$) represent sets of variables, and bold lowercase letters ($\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$) denote their instantiations.

The *probability distribution of* a set of variables $\mathbf{X}$ is denoted with $Pr(\mathbf{X})$ whose elements are $Pr(\mathbf{x})$. For example, using this notation we can write $\sum_{\mathbf{x}} Pr(\mathbf{x}) = 1$. Similarly, $Pr(\mathbf{X} \mid \mathbf{Y})$ denotes the *conditional probability of* $\mathbf{X}$ *given* $\mathbf{Y}$, which is a table of probability distributions indexed by the instantiations of $\mathbf{Y}$: every $Pr(\mathbf{X} \mid \mathbf{y})$ is a probability distribution over $\mathbf{X}$, each element of which is depicted by $Pr(\mathbf{x} \mid \mathbf{y})$.

Selected features of knowledge and beliefs about a domain are encoded in a *model*, which is a partial view of total information about the domain. We use *probabilistic graphical models*[21] as suitable representations for a model.

Definition 3.1. *Given a (*universal*) set of variables* $\mathbf{V}$, *a* model $\mathcal{M}$ *imposed on* $\mathbf{U} \subseteq \mathbf{V}$ *is a graphical model defined on* $\mathbf{U}$. *Similar to Halpern and Pearl*[20], *the variables in* $\mathbf{U}$ *are called* endogenous variables, *given* $\mathcal{M}$, *and denoted as* $En(\mathcal{M})$. *All the variables that are not in* $\mathcal{M}$ *are called* exogenous variables *and denoted as* $Ex(\mathcal{M})$. *Formally, we have* $Ex(\mathcal{M}) = \mathbf{V} - En(\mathcal{M})$. *Recall that* $\mathcal{M}$ *has a structural component, a graph* $G_{\mathcal{M}}$, *and a parametric component, a set of probability distributions* $\Theta_{\mathcal{M}}$.

Definition 3.2. *A conjunction of truth assignments to some exogenous variables of a model* $\mathcal{M}$ *is called a* context $\mathcal{C}$ *of* $\mathcal{M}$: $\mathcal{C} \equiv V_1 \wedge \ldots \wedge V_n$, *where* $\{V_1, \ldots, V_n\} \subseteq Ex(\mathcal{M})$. *Note that to make the definitions simpler, we assumed that all variables of our models are boolean; this can be relaxed by using general variable assertions instead of truth assignments.*

The idea of a context is to capture the stable invariant behavior 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. Consider a domain in which a system goes through a set of operational contexts over time. One possible modeling approach is to build a single large probabilistic graphical model that will account for different situations corresponding to various system states. This approach suffers from high data requirements and low time/space efficiency due to high complexity of the model. Another possible method is to represent the system with many tiny models that hold for a limited period of time (due to noise in the data). Although this approach requires small amount of data, mem-

ory, and computation per model, this method is often not suitable since we have to frequently replace models which is very costly. The challenge is to find a trade-off between these two extreme approaches. This can be seen as multidimensional optimization: find an optimal collection of small probabilistic models that represent a system in different contexts (situations) accurately and efficiently.

There are several important points to mention. First, each probabilistic model from the collection captures the relationships in the domain that are *relevant* in the corresponding context. Second, we want this model to be robust to noise: *small* fluctuations in the data do not mean the change in the context. Thus, the difference between two contexts (and the difference between the corresponding models) should be *significant*. Third, robustness of the context (which is related to the permitted level of noise) and the level of difference between contexts is domain dependant and closely tied to the cost of switching between the contextual models.

Since the collection of contexts may not be known a priori, we have to find an *optimal* set of contexts improving accuracy and efficiency. Two properties of the set of contexts are accounted for during optimization: (a) context stability and (b) a rate of change of contexts. Consequently, we search for a set of contexts by minimizing the error representing how well each context from the set agrees with associated data and how many context changes are present. The search space is a collection of all possible sets of contexts. The formal constraints on the search are presented next.

Let $\mathbf{D}$ represent a set of observed data. Naturally we assume that the data set is ordered: $\mathbf{D} = \{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_m\}$, where each $\mathbf{d}_i$ is a vector of observations recorded at the ith time step ($i \leq m$) for all observable variables of the system. Given recent observations $\mathbf{d}_i$ for some $1 \leq i < m$, we refer to the successive data vector using the following notation: $s(\mathbf{d}_i) = \mathbf{d}_{i+1}$.

Consider a set of contexts $\mathbf{H} = \{C_1, ..., C_k\}$ from the search space. Each context from $\mathbf{H}$ represents invariant behavior in a possibly *non-continuous* subset of data. Therefore, $\mathbf{H}$ corresponds to some decomposition of a data stream. There are many possible decompositions of $\mathbf{D}$ into k mutually exclusive subsets, which we denote as $\rho(\mathbf{D})$. Consider an element $\rho_i \in \rho(\mathbf{D})$ that decomposes $\mathbf{D}$ into $\mathbf{D}_1, \ldots, \mathbf{D}_k$, where each $\mathbf{D}_j$ corresponds to observations of the stable behavior described by a context C_j. Note that each $\mathbf{D}_i$ consists of data vectors that may not form a continuous time range of observations.

Example. Consider first six time steps ($m = 6$) and assume we are looking for two contexts ($k = 2$). Then $\rho_1 = \{\mathbf{D}_1 = \{\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_4\}, \mathbf{D}_2 = \{\mathbf{d}_3, \mathbf{d}_5, \mathbf{d}_6\}\}$ and $\rho_2 = \{\mathbf{D}_1 = \{\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3\}, \mathbf{D}_2 = \{\mathbf{d}_4, \mathbf{d}_5, \mathbf{d}_6\}\}$ are possible decompositions of $\mathbf{D}$. Note that decomposition ρ_1 is not contiguous.

We now define two error scores associated with data decomposition ρ_i:

$$error'_j(\rho_i) = Pr_{\mathbf{x} \in \mathbf{D}_j}[C_j(\mathbf{x}) = false],$$

$$error''_j(\rho_i) = Pr_{\mathbf{x} \in \mathbf{D}_j}[C_j(s(\mathbf{x})) = false \mid C_j(\mathbf{x}) = true],$$

where $C_j(\mathbf{x})$ is an instantiation of context C_j on a data vector $\mathbf{x}$. $Pr_{\mathbf{x} \in \mathbf{D}_j}$ indicates that the probability is taken over the instance distribution D_j. Informally, score $error'_j(\rho_i)$ indicates the error rate

we expect when applying $\mathcal{C}_j$ to instances drawn from the probability distribution $\mathbf{D}_j$. It captures how much context $\mathcal{C}_j$ disagrees with data set $\mathbf{D}_j$ from data decomposition ρ_i. Given a successful application of $\mathcal{C}_j$ to an instance, score $error''_j(\rho_i)$ indicates the expected error rate when applying $\mathcal{C}_j$ to the next instance. Note that when $error'_j(\rho_i)$ is minimal, $error''_j(\rho_i)$ denotes the amount of instability in the system's behavior described by context $\mathcal{C}_j$ and sampled with data $\mathbf{D}_j$. By summing these two scores across ρ_i we obtain an error score for the data decomposition given the context partition:

$$error(\rho_i) = \sum_{j=1}^{k} [error'_j(\rho_i) + error''_j(\rho_i)].$$

Example. Continuing the earlier example, assume the first context logically holds only on the first three data points and the second context holds only on the last three data points. Then $error(\rho_1) = 1.17$, since $error'_1(\rho_1) = 0.33$, $error'_2(\rho_1) = 0.33$, $error''_1(\rho_1) = 0.5$, $error''_2(\rho_1) = 0$. Similarly $error(\rho_2) = 0$.

Minimizing $error(\rho_i)$ across all data decompositions yields a score $Error_{\mathbf{D}}(\mathbf{H})$ for a context set $\mathbf{H}$ given a data stream $\mathbf{D}$:

$$Error_{\mathbf{D}}(\mathbf{H}) = \min_{\rho_i \in \rho(\mathbf{D})} [error(\rho_i)].$$

For the example above $Error_{\mathbf{D}} = 0$. Note that the problem of estimating the error score of $\mathbf{H}$ is essentially the problem of clustering the data according to some stable contiguous patterns.

Minimizing $Error_{\mathbf{D}}(\mathbf{H})$ over all possible sets of contexts gives us an optimal collection of contexts that represents the stable invariant behavior (with the smallest number of context changes) of the observed system: $\min_{\mathbf{H}}[Error_{\mathbf{D}}(\mathbf{H})]$.

Recall that each element of $\mathbf{H}$ (a context $\mathcal{C}$) corresponds to some model $\mathcal{M}$: there is a connection between $\mathcal{M}$ and $\mathcal{C}$, assuming there is an association (a link in a Bayesian network) between each exogenous variable and endogenous variables. Therefore, we look at context $\mathcal{C}$ as a condition that *constrains* the set of all possible models (structurally and parametrically).

Ultimately, we search for a context partition that results in probabilistic models that most accurately represent the data. Ideally, these models should be as small as possible to reduce the cost of inference over them. Therefore, while minimizing $Error_{\mathbf{D}}(\mathbf{H})$, we want to maximize the probability for each model $\mathcal{M}_c$ constrained by contexts $\mathcal{C} \in \mathbf{H}$:

$$\arg\max[Pr(\mathcal{M}_c \mid \mathbf{D})] \propto \arg\max[Pr(\mathcal{M}_c) Pr(\mathbf{D} \mid \mathcal{M}_c)].$$

The prior probability distribution $Pr(\mathcal{M}_c)$ reflects our belief before seeing any data that the model $\mathcal{M}_c$ imposed by the context $\mathcal{C}$ is correct. This search for the best contextual model fits the minimum description length principle:

$$\arg\min[-\log Pr(\mathcal{M}_c) - \log Pr(\mathbf{D} \mid \mathcal{M}_c)],$$

where the first log corresponds to the number of bits to describe the contextually constrained model and the second log corresponds to the number of bits to describe the data in terms of the model.

To ensure that the contextual models are parsimonious we minimize the in-degree of possible models (minimizing the fan-in/fan-out problem). However, even with structural restrictions finding the best graphical model is NP-complete[22]. If contexts do not constrain corresponding models, then the entire optimization problem de-

scribed above can be reduced to a traditional structure search and parameter estimation for a single model.

In the next section we provide the tools for building models and for performing parameter learning and inference on the specified models.

4 Generalized Loopy Logic

Generalized Loopy Logic (GLL) is a logic-based probabilistic reasoning language that supports our context-based architecture. In section 4.1 we give a description of the language and in section 4.2 we present the inference and learning components of GLL.

4.1 GLL: Language Description

GLL is based on earlier work by Poole[3], Haddawy[4], Getoor et al.[12], and Kersting and DeRaedt[13]. GLL is an extension of the language developed by Pless et al.[23]. GLL is a first-order logic-based Turing-complete stochastic modeling language that improves expressive and reasoning power by combining deterministic and probabilistic approaches. Note that the expressive power of traditional Bayesian networks is constrained to finite domains as in the propositional logic. GLL handles this representational shortcoming through variables capturing general classes of events and relationships. This first-order language combines Horn-clause logic with Bayesian networks in order to represent potentially infinite classes of stochastic relationships such as Markov processes.

A knowledge base constructed using GLL is a set of Prolog-like rules annotated with probabilistic distributions describing the conditional dependences among random variables. The domain of terms is specified using set notation: `head <- {0, 1}` indicates that `head` is either `0` or `1`. A GLL sentence is of the form

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

where $\texttt{body}_i$, $0 \leq i \leq k$, are the predicates that represent corresponding random variables on which a variable `head` is conditionally dependent. Facts (rules when $k = 0$) are used to represent the observations. Note that the size l of the conditional probability table is equal to $arity(\texttt{head}) \times \prod_{i=1}^{k} arity(\texttt{body}_i)$, where $arity(\texttt{x})$ is the number of states of x (2 for `head`). The probabilities are listed for each state of `head` and $\texttt{body}_i$, $0 \leq i \leq k$. For instance, if `head` is a binary term defined above and `body` is defined over $\{0, 1, 2\}$, then a conditional probability distribution $Pr(\texttt{head|body})$ is defined by the clause

```
head|body=[[.5,.5],[.4,.6],[.1,.9]].
```

In GLL, terms can be full predicates with structure and contain Prolog style variables. For instance, the sentence `head(N) = [0.5,0.5]` says that `head` is universally equally probable to take on either of two values. In case when there are multiple rules with unifiable heads, GLL uses the *product distribution* as a combining rule that simply takes the product of the corresponding conditional probability tables. For instance, if we have two sentences, `head = [0.1, 0.9]` and `head = [0.4, 0.6]`, then the resulting probability of `head` will be `[0.07, 0.93]` obtained by normalizing `[0.1*0.4, 0.9*0.6]`. The product distribution combining rule allows GLL to handle rules and facts in the same way, since a fact can be viewed as being annotated with a distribution where the

probability of one state is equal to 1 while the probability of all other states is equal to 0. This combining rule is handled naturally during inference in GLL.

4.2 GLL: Inference and Learning

The following GLL program defines a hidden Markov model (HMM) with four observable time steps:

```
state <- {true, false}
emit <- {hi, low}
state(s(N))|state(N)=[[.9,.1],[.01,.99]]
emit(N)|state(N)=Emit
emit(0) = hi
emit(1) = hi
emit(2) = low
emit(3) = low
```

In this example, at any time step `N` the system can be in one of two states, `true` or `false`, which is represented by the predicate `state(N)`. 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 move to the state `true`. In both states the system can output either `hi` or `low`, which is represented by `emit(N)`. Note how the recursive rule of GLL captures the Markov process between states of the HMM.

The conditional probability of the system's output given a hidden state is denoted with capitalized "`Emit`" indicating that this is a *learnable distribution* that will be estimated. The data for learning is obtained from GLL rules and facts (observations). In our example four facts are added when the variable `N` is bound by the first four time steps (here integers are a shorthand for successors of zero, e.g., `2=s(s(0))`). Even though the variable `emit` is completely determined at each of the four time steps, in general there could be a learnable distribution with no direct evidence. GLL uses an algorithm based on Expectation Maximization[24] for inferring probability distributions and estimating the adjustable parameters.

Generalized Loopy Logic uses the message-passing inference algorithm known as *loopy belief propagation*[21]. As opposed to its predecessor[23], GLL can also use other iterative inferencing schemes including *generalized belief propagation* and *Markov chain Monte-Carlo*.

Similar to [13] GLL constructs an SLD tree from the original program, but then maps the tree to a Markov random field as opposed to mapping it to a Bayesian network[25]. Mapping into a Markov field handles the product distributions arising from goals that unify with multiple heads: if more than one rule unifies with the rule head, then the variable node (corresponding to a ground instance of the head) is connected to more than one cluster node (corresponding to a probability distribution of each rule), which results in a product distribution. One feature of GLL is its control of the depth of the unfolding of recursive rules when mapping into a Markov random field. Figure 1 demonstrates how the GLL program specifying an HMM presented earlier is converted into a Markov field (factor graph). Here 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).

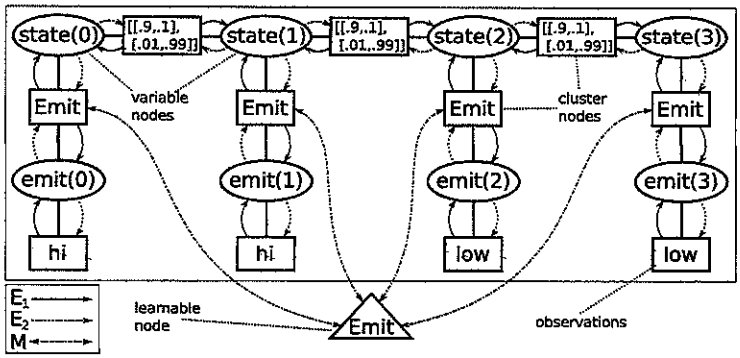

Fig.1. A Markov random field produced by unrolling the GLL program defining a hidden Markov model.

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 (a message E_1 in figure 1) is the product of the conditional probability table (called a local potential) 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 (a message E_2 in figure 1) is the normalized product of all the messages to V except the message from C. This process, iterating until convergence, has been found to be effective for stochastic inference[26] and when applied to an acyclic graph is proved to converge to an optimal solution[21].

A major feature of GLL is its natural support for parameter learning by the assignment of learnable distributions to rules of a GLL program. These parameters are estimated using a variant of the Expectation Maximization (EM) algorithm[24] implemented through the message passing of the loopy belief propagation algorithm. EM estimates learning parameters iteratively, alternating between an expectation (E) step and a maximization (M) step. In the E step, the distribution for the hidden variables is based on their known value and the current estimate of the parameters is found. In the M step, these parameters are re-estimated. Assuming the distribution estimated in the E step is correct, each EM iteration increases the probability of reaching maximum likelihood[24].

More specifically, GLL utilizes the EM algorithm by adding a special kind of node, a learnable node, to a Markov random field (the triangular node in figure 1). Each instance of the cluster node that is to be fitted is connected to the learnable node. By inferencing over the cluster and variable nodes of a Markov field (using loopy belief propagation) GLL computes the messages for the learnable nodes (a message M in figure 1). Applying the propagation algorithm until convergence is equivalent to the E step of the EM algorithm, since it produces an approximation of the expected values. The averaging over all the cluster nodes connected to the learnable node yields a maximum likelihood estimate of the parameters in a 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.

5 The Detection of Context Transitions

In this section we analyze the model failure heuristic and introduce an approximate heuristic-based algorithm for context partitioning.

5.1 Failure-Driven Model Revision

Since contexts correspond to invariant behavior of a system over periods of time, modeling context yields a focused representation of a specific operational mode of the system. In this section we explore the close relationship between context change and model failure, i.e., a new situation when a model no longer fits recent data. A model corresponding to an active context is less robust to context changes than the full model of the system. During a context transition event, when the observed data undergoes a significant qualitative or quantitative change, the current model fails. Thus, we consider model failure to be an indication of a context transition event. We argue that this *failure-driven* approach is suitable for switching between contextual models.

Our approach to the probabilistic modeling of changing contexts is based on ideas from developmental human learning[2]. Piaget suggested that when an unfamiliar situation is presented to a child, she tries to fit it into her current understanding of the world. When this fails, the normal child is able to form new cognitive structures to address the situation. This corresponds to two forms of learning recognized by Piaget[2]: *assimilation* and *accommodation*. We argue that probabilistic inference systems can benefit greatly by emulating these mechanisms.

When new data are available, we check whether the current model fits the dataset well. If it does (i.e., if any changes in the data are not severe), the data are incorporated into the model by updating its probability distribution (model parameters). Otherwise, if the model fails to fit the data, we assume that the system is operating in a new context. We save the current model as it is no longer relevant and choose a new version that accounts for the new data. Here learning by assimilation happens when the model is consistent with new data and it is fine-tuned by assimilating the dataset. Learning by accommodation is when the model is inconsistent with new data, and, in order to account for the dataset, we have to reorganize our model.

Our failure-driven context-switching approach addresses two related and common problems in machine learning: the problems of *over-fitting* and *over-generalization*. When single models are learned on a data set that is not diverse, models tend to become too specific and are said to over-fit and are unable to generalize and account for slightly varying datasets. The converse problem of over-generalization is when a very general model is learned from well distributed and possibly sparse data in the learning stage and, therefore, performs badly on all types of data in the operational stage. When used in probabilistic systems, the mechanisms of assimilation and accommodation along with the notion of context and context change, helps minimize these problems.

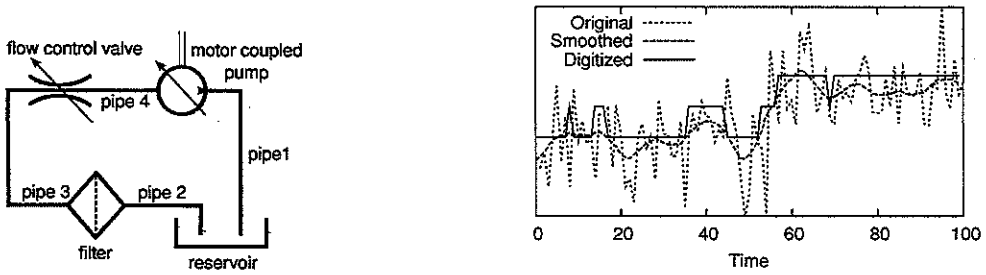

Fig.2. (left) The diagram representing a simplified pump system. (right) The time series of the pressure generated by the pump (*OutPr*) and its smoothed and discretized versions.

5.2 An Example of Context-Revision

Identification of model failure is crucial in context sensitive modeling. 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. When this likelihood is below a certain threshold, then the model fails. Even though the models of the probabilistic system described in this paper are specified by the first-order stochastic language – Generalized Loopy Logic (see section 4), the notion of failure can be extended to any probabilistic graphical model specifying a full joint probability distribution (see section 3 definition 3.1).

To illustrate our failure detection method we consider temporal data obtained from multiple sensors installed on the *mechanical pump system* schematically depicted in figure 2(left).

A water pump draws liquid from a reservoir through a pipe (`pipe1`) and ejects the liquid into another pipe (`pipe4`). The pump is driven by an electrical motor. The liquid, containing contaminants is cleared by a filter and then 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, the emission state of the liquid at different locations, as well as *indicating parameters* such as the rotation rate of the pump and vibration near the motor. One important task is to detect when the filter gets clogged leading to possible cavitation in the system. In order to perform such diagnostic tasks, the knowledge about the system is transformed into a stochastic model using the GLL tool.

The sensory data consists of a time series of three parameters: pressure coming into the 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

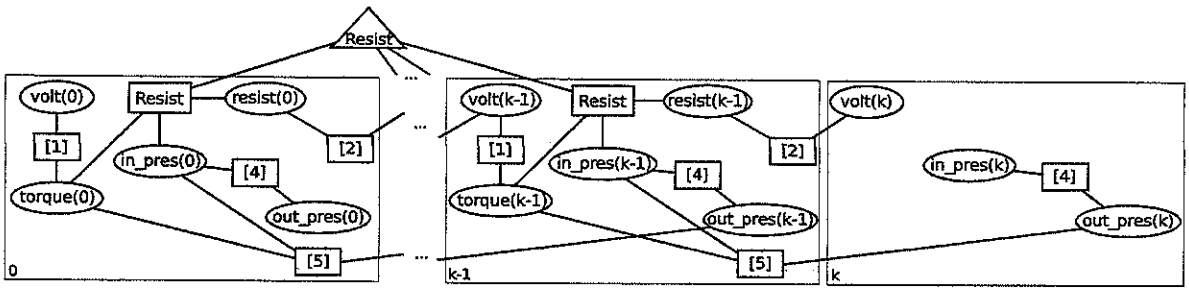

Fig.3. A Markov random field produced by unrolling the GLL program defining a pump system. The round nodes represent random variables, the rectangular nodes represent CPTs, and the triangular node represents learnable distribution. Note that for simplicity we omitted showing the evidence in the diagram.

using a sliding window and discretized. Figure 2(right) illustrates the time series of one of the parameters (*OutPr*) of the pump system.

We selected 35 time steps to train a stochastic model, 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 following is an example of a pump model specified in GLL.

```
...
torque(N)|volt(N)=[1]
volt(N+1)|resist(N)=[2]
resist(N)|torque(N),in_pres(N)=Resist
in_pres(N)|out_pres(N)=[3]
out_pres(N+1)|torque(N),in_pres(N)=[4]
volt(0)=low
in_pres(0)=hi
out_pres(0)=low
...
```

Note that for simplicity we replaced the conditional probability tables with placeholders, e.g., table `[1]`. We assume that we are given the observations of $k+1$ time steps (35 in this case). In figure 3 we show a Markov network built by unrolling the GLL program. Note that unrolling is done through knowledge base model construction: we start with the current observations (at time $k+1$) and recursively apply the clauses of the GLL program until the last observation of the window, which is assumed to be at time 0.

To select an appropriate size of training data we performed a leave-one-out cross-validation for each model trained on the first K time steps of the training data (here K takes values between 5 and 45 since we know that the first 45 time steps came from the same stationary distribution). Figure 4(left) shows that the average prediction error decreases as the size of the training dataset increases and becomes minimal at around 35.

Recall that GLL employs the EM learning algorithm[24] implemented using loopy belief propagation[21] to learn model parameters (see section 4.2). Figure 4(right) demonstrates the dependency of the iterations of the learning algorithm on the training dataset. The fact that learning the model from the training dataset with 35 time steps requires a considerably smaller number of iterations is another indicator of the appropriate size of the training dataset.

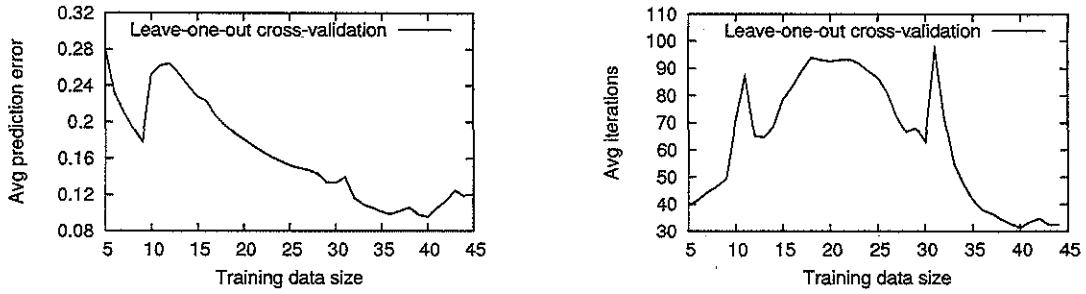

Fig.4. A leave-one-out cross-validation analysis across models trained on data sets with gradually increasing size. (left) Dependence between a model prediction error and the size of the training data. The error is averaged across the range of predicted parameters as well as across iterations of the cross-validation. Notice that the error decreases as the window gets larger than 15, and the error is minimal at around 35. (right) The number of iterations a model takes on average to converge versus the size of training data. Notice the amount of iterations stabilized to a minimum when the size of the training data is greater than 35.

5.3 Detecting Model Failure

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, e.g., in [28] the authors propose a Bayesian formulation of change detection through hidden Markov models. In [29] the authors also follow a Bayesian approach and use Gibbs sampling to detect the changes in the distribution. Finally, in [30] the authors used an EM-based algorithm to learn a Gaussian mixture model, which is related to our approach. In this paper we provide a method that fits naturally into the iterative framework of our context-sensitive probabilistic modeling. Our approach is related to a classifier ensemble for explicit change detection (see [31] for a detailed review). Recall that our failure detection algorithm is proposed in the framework of logic-based probabilistic reasoning as a way of detecting context changes and triggering model switching or adaptation. Table 1 outlines our failure detection algorithm. The idea of the algorithm is to monitor a selected subset of model parameters (triggers) and signal a possible model failure when the parameters of the previously trained model are considerably different than these learned with new data. Recall that GLL model parameters are the CPTs of the GLL clauses. Note that the function *learn_param* (in step 5.3) estimates a single parameter given new data and the model, with the rest of the parameters learned from the training data. This is different from the function *train_model* (in step 1) that estimates all model parameters given data.

We now go through each step of the meta-algorithm of the failure detection in Table 1. In step 1 the model specified by a GLL program is trained on the initial data (see section 4.2 for details). Then, in step 2 and 3 we determine the optimal window and threshold parameters for the failure detection to capture the context change (this step is explained in section 5.4).

1. model M ← *train_model*(gll_program, training_data)
2. window_params ← *find_window*(training_data, model M)
3. threshold ← *find_threshold*(training_data, model M, window_params)
4. current_data ← *slide_window*(window_params)
5. *for each* (trigger_params of model M)
   - 5.1. current_cpt ← *get_cpt*(trigger_param)
   - 5.2. model M' ← *make_learnable*(model M, trigger_param)
   - 5.3. new_cpt ← *learn_param*(model M', current_data)
   - 5.4. difference ← *frobenius_norm*(current_cpt, new_cpt)
   - 5.5. *if* (difference > threshold) *then* failure ← true
6. *if* (not failure) *then* go to 4

Table 1: A high-level description of the failure detection algorithm. The algorithm is initialized in steps 1, 2, 3 and proceeds by iteratively selecting a data window (in step 4) and checking for model failure in the specified window (in step 5). If no break-down is detected, the algorithm slides the data window further along the data stream. Note that in steps 1 and 5.3 the algorithm employs the EM-based learning of GLL (see section 4.2) and also uses optimization techniques for selecting appropriate window and threshold parameters (see section 5.4).

In step 4 the algorithm reads the input data. In step 5 we traverse all the trigger parameters of our model in order to detect the context change. Specifically, in step 5.1 we store the current CPT of a trigger parameter according to our trained model M. In step 5.2 the algorithm makes a temporary model M' by replacing the CPT of the trigger parameter with a learnable distribution. In step 5.3 we apply the EM-based learning of GLL (see section 4.2) to estimate the learnable distribution based on the new data. In step 5.4 the Frobenius norm is used to compare the original distribution stored in current_cpt and the distribution estimated from the new data (new_cpt). The algorithm indicates a context change if at least one trigger parameter shows a considerable deviation from the original distribution. Note that not only is the detection algorithm in Table 1 controlled by the size of the data window and the size of the window shift, but it is also regulated by the threshold indicating model failure.

In general the problem of finding the appropriate window and threshold parameters can be seen as a two-dimensional error minimization problem. Given window/threshold parameters Θ, consider $error_1 = Pr[FD(M, \Theta, D_{nofail}) = true]$, a Type I error representing the rate with which our failure detector FD signals the failure of model M on the data D_{nofail}.

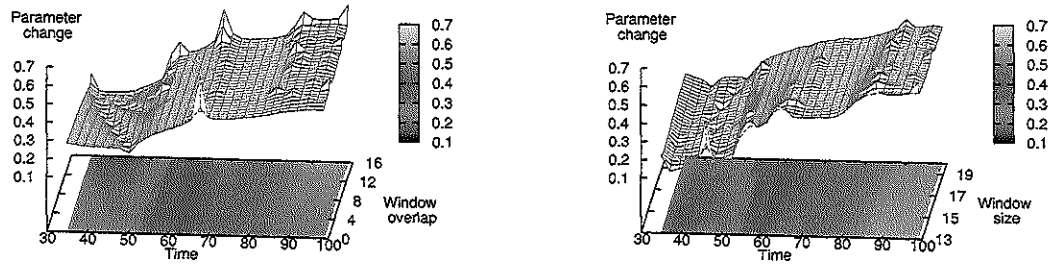

Fig.5. Performance of the failure detector for a single model parameter (*Resist*) across (left) different window overlaps given a window of size 17, and (right) different window sizes given a window overlap of 12 data points. The X axis corresponds to time, the Y axis corresponds to the size of (left) the window overlap and (right) the sliding window, and the Z axis is the difference between the expected and the predicted values of the model parameter (*Resist* in this case) measured by the Frobenius norm.

Note that since D_{nofail} is the data from the same stationary distribution as the training data for the model M, no failure is expected. $error_2 = Pr[FD(M, \Theta, D_{fail}) = false]$ is a Type II error that shows how frequently FD misses model failure. Ultimately, we would like to find parameters Θ that would minimize $error_1$ and $error_2$.

Minimizing $error_1$ 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 the minimization of $error_1$ returns a subset of possible window/threshold parameters. Given the third subset of the training data on which the detector is *expected* to signal failure, we can perform a similar minimization routine to further constrain the parameter set.

Figure 5(left) shows the performance of the failure detector (trained on data from the pump system) for a model parameter corresponding to resistance of the pump (*Resist*). The detection algorithm slides a window of 17 data points through the data stream starting from the 35th time step (since we used the first 35 data points of the stream to train the model). The data stream has a breakdown at around time step 54, when a valve of the pump system is closed causing less flow coming into the pump and increasing the pump's resistance. Figure 5(left) shows that choosing an overlap between consecutive windows affects the choice of the threshold: selecting 0.3 as a threshold in case of overlap 12 accurately captures the breakdown, however the same threshold does not work for overlap 10.

Figure 5(right) shows how the failure detector performance changes depending on the sliding window size, assuming the consecutive windows overlap by 12 points. It can be seen that the smaller the window, the more prone to data noise the failure detection becomes. On the other hand, larger windows produce smoother, more stretched out results.

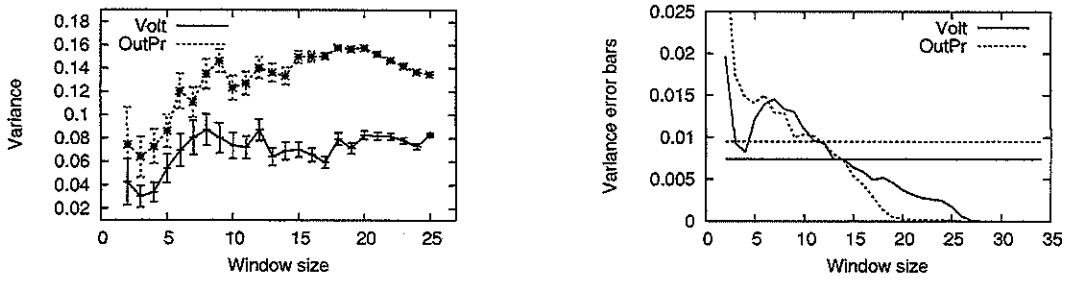

Fig.6. Average variance (left) and the corresponding error bars (right) of subsets of the training data of the pump model plotted for various subset size (between 2 and 25). Two plots are shown for random variables *Volt* (voltage at the motor of the pump system) and *OutPr* (pressure coming out of the pump). We would like to select a window big enough for the changes in variance to be below the level depicted in (right) by the horizontal lines for each variable.

We see from figure 5 that larger windows produce a failure detection *lag*, when model failure is identified long after the break-down has occurred. Additionally, larger windows demand more computational power. On the other hand, smaller windows result in a higher likelihood of a false positive error.

5.4 Using Variance to Optimize Window/Threshold Parameters

In general, without an appropriate data set, minimizing Type II error ($error_2$) is a challenging problem. The problem becomes even more difficult if the difference between the distributions which create model failure is small. 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, a data window, 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.

Figure 6(left) shows the average variance of data windows with increasing size randomly selected from the training dataset. Figure 6(right) demonstrates 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. Thus, we set the window size to 12 or larger (25 is the optimal). Automatically, this can be done by selecting the window as soon as the error bars drop below a certain level, as the window size increases.

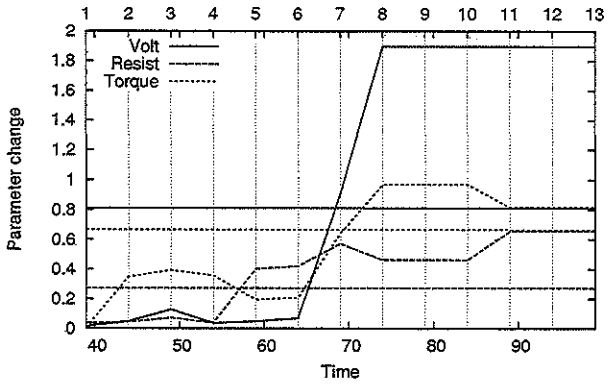

Fig.7. Model failure detection for the pump model. Each horizontal line corresponds to a failure threshold: once a corresponding distribution change goes above this threshold, the failure detector signals a model break-down. The grid corresponds to window shifts.

Once the window size is set, the failure threshold can be found by computing an average difference (e.g., a Frobenius norm) between the current value of a 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 a failure threshold.

5.5 Application of Failure Detection

Figure 7 illustrates the performance of the failure detector on the sensory data for the pump model plotted for three model parameters: motor voltage, *Volt*, pump resistance, *Resist*, and motor torque, *Torque*. Note that in this example each parameter has its own failure threshold, which brings more flexibility into the detection process, since some parameters change less gradually (such as *Volt*), while other deviate considerably (like *Torque*). The thresholds were automatically identified using the method described above.

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

When the model is large, failure detection in general can be very expensive. We specify a small subset of *trigger* parameters, whose changes are seen as most important by the domain experts and indicative of model failure. Instead of checking for failure in the entire model, only this small set of trigger parameters is monitored. Full-fledged failure detection is engaged once a change in a trigger parameter is discovered. Since different parameters give different detecting performance, it might be useful to employ a combination of these. Two-layered failure detection can be used, for example, when a parameter that is sensitive to data noise but useful in detecting early failure (*Resist* in figure 7) can trigger an alert mode, in which case a more stable parameter (such as *Voltage*) is analyzed to confirm the detected model break-down.

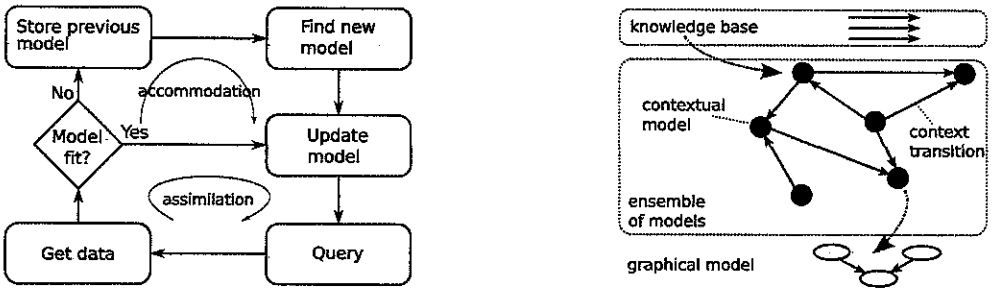

Fig.8. (left) The flow chart of the failure-driven architecture incorporating two discovery modes, assimilation and accommodation. (right) A diagram of the relationship between three levels of representation and control. At the top level resides a knowledge base represented as a set of Horn clauses. Given input data and a query, the relevant components of the knowledge base are instantiated resulting in a specific graphical model (a Markov random field) which is a part of the bottom level. The intermediate level contains a database of graphical models from the bottom level, each of which captures a specific contextual information encountered so far. Each specific graphical model (an entry in the database depicted with a black node in the figure) is connected with at least one other model through a context transition that has been encountered previously.

6 Modeling Dynamic Data: Example

In this section we show how our method for the detection of context changes supports a framework for context-sensitive probabilistic modeling in the analysis of dynamical data collected over time.

6.1 A Context-Switching Architecture

Probabilistic modeling systems that dynamically represent frequently changing data are important for monitoring complex tasks. Dynamical systems employing distributed sensing technology also introduce additional constraints on the running time and memory of the modeling system. The most suitable systems in these cases, we believe, are those that are able to *evolve* to handle rapidly changing pieces of information.

Following a failure-driven approach described in section 5.1, a possible architecture of a context-sensitive probabilistic modeling system is described by the flow chart diagram in figure 8(left). Context switching mechanisms, which are among the main components of the system, employ two forms of learning within the architecture – *assimilation* and *accommodation*. When new data are available, the system checks whether the current model fits the dataset well. If it does, the data are incorporated into the model by updating its probability distributions. Otherwise, if the model still fails to fit the data, the system saves the current model and searches for a new version of the model that will account for the new data.

The operation of the system, see figure 8(left), when the condition "*Model fit?*" holds, corresponds to learning by assimilation: the model is consistent with the new data and it is fine-tuned by assimilating the dataset. Conversely, when this condition does not hold, the system employs learning by accommodation: the model is inconsistent

with new data, and in order to account for the dataset, we have to reorganize the model.

Thus a key component of our context-sensitive modeling system is an ensemble of contextual models, figure 8(right). Interconnected contextual models managed by domain knowledge, the top layer, correspond to vertices and edges of the structure of the ensemble. Graphical models, the vertices of the ensemble's structure, constitute the lowest layer. The system incrementally populates the ensemble of models by applying our failure-driven methodology.

6.2 Context Switching: Pump Example

Our context change detection algorithm is implemented in Scheme[32]. The prototype is applied to the data obtained from the mechanical pump system shown in figure 2. In particular, two types of tests were performed: *Test A*, a test on detecting a transition from normal behavior to a context when the filter is clogged, and *Test B*, a test on choosing an appropriate model in the ensemble of models once the context change is detected.

In *Test A* an active model was trained on the data from the mechanical pump system operating normally. The normal behavior of the pump system continued until the 48th time step when the flow valve (see figure 2) was partially closed to simulate a clogged filter. The system was subsequently halted after a total of 100 data steps. During this time the active model was continuously checked for failure. The results for the three model parameters, voltage at the motor, resistance at the pump, and torque at the motor, are plotted in figure 9(left).

Failure was detected using a window of 17 time steps that was sliding 5 steps at a time. As seen in figure 9(left), the modeling system successfully identified the context change. Note however that the context change was captured after the actual break-down had occurred, at steps 55-60 as opposed to the 48th step, due to the choice of conservative window parameters from the set of possible parameters trained earlier. An attempt to capture the context change sooner by employing a smaller window (size 15) with a smaller overlap (10) was not successful (data are not shown): our failure detection method prematurely, around the 25th time step, signals a context change due to the high noise level in the data stream. A possibly more robust extension to the context switching mechanism might be to consider several windows of different sizes: small windows can be used to alert the system of a possible context change, while large windows can then confirm this change.

In *Test B* we trained 6 distinct models, corresponding to various operational contexts of the pump system. The contexts consist of normal operational behavior, behavior under a highly clogged filter, under a slightly clogged filter, behavior when a pump is misaligned, when a shaft between a motor and the pump is misaligned, and when one of the gears has a chipped tooth.

We then considered the data stream of the pump system operating normally until the 70th time step when the flow valve was almost closed (simulating a highly clogged filter) and reopened again after the 120th time step. The failure detection method was invoked for each of the six trained models.

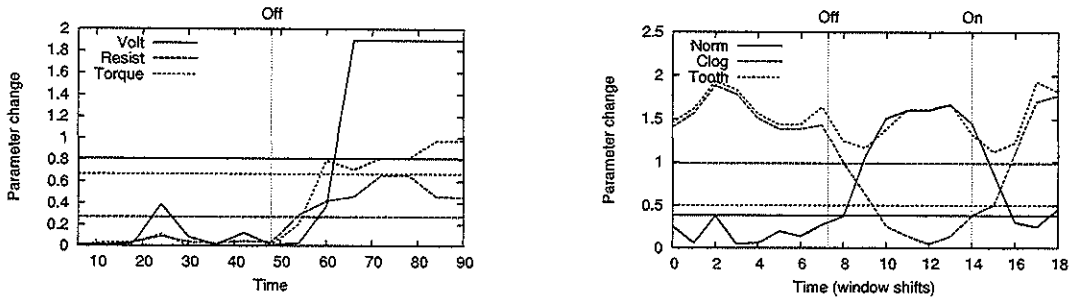

Fig.9. (left) Failure detection on three model parameters (*Volt*, *Resist*, *Torque*) in the data stream when the pump system initially operates normally, but then breaks down at the 48th time step (when the flow valve is partially closed). The prediction is performed using a sliding window of 17 points with 12 point overlaps. A vertical line *Off* shows an actual system break down, while horizontal lines correspond to failure thresholds for the corresponding (same style) model parameters. 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. (right) An illustration of an experiment where the pump system starts operating normally and the valve is partially closed (at the *Off* time step), which is then opened back (at the *On* time step). The figure 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, and a model (*Tooth*) capturing the situation when a gear tooth is chipped.

Figure 9(right) illustrates the results of failure detection for a model parameter corresponding to voltage at the motor (*Volt*) for three models *Norm*, *Clog*, and *Tooth*, corresponding to contexts of normal operational behavior, behavior when a flow valve is partially closed, and behavior when a gear tooth is chipped. Note that the model *Norm* shows the smallest difference between predicted and true model parameters before time *Off* (when the flow valve is turned), which then peeks after the turn, and drops down after time *On* (when the valve is reopened). On the other hand, the model *Clog* presents almost the opposite behavior: it shows a very large parameter difference before the turn of the valve, which steeply reduces once the valve is partially closed, and increases back after the valve is reopened. Note that other models do not show such distinct behavior, for example the model *Tooth* in figure 9(right) exhibits a constant large (above the corresponding threshold) difference between true and predicted parameters.

Since the model *Norm* has the smallest error at the beginning of the data stream, the context-sensitive modeling system employing our context change detection method selects this model as initially active. As soon as the first context change (*Off* in figure 9(right)) is identified, the system then switches the active model to *Clog*, since it has the smallest error among the six models. Consequently, the modeling system returns the active model back to *Norm* after the second context change is detected.

7 Conclusions and Future Directions

In this paper we considered a logic-based probabilistic modeling approach called GLL. While the probabilistic component of GLL handles noise and uncertainty, the logic component lets the model designer compactly represent the underlying knowledge through a set of first-order rules. In this paper we characterize the notion of context and propose a GLL-based framework that uses model failure to represent a complex real-time diagnostic problem as a set of simpler context-specific knowledge-focused reasoning tasks. The framework uses a failure-driven approach of switching between minimal models corresponding to contexts, motivated by research in developmental psychology[2].

There are a number of important advantages of using the context-based failure-driven modeling approach to real-time diagnostic reasoning. First, there is no need to reconstruct a complete distribution of all possible data, thus less data is needed for training a diagnostic model. Usually, during probabilistic reasoning, system dynamics are assumed to be stationary and invariant over the *entire* training data set. While reasoning about a system whose dynamics change according to states of the external environment and where little a priori knowledge is given, every possible aspect of the world must be explicitly represented for the training data and learning algorithm to capture all hidden relationships.

One implication of minimal model context switching is that when a traditional knowledge base changes, the learned general model is discarded as no longer true and a new one must be constructed from scratch. Using our approach, by splitting the domain into contexts we are able to construct smaller models with reduced complexity capturing only relevant, currently present-in-the-data relationships. Such small models assume stationary behavior and require only a minimum amount of training data. This fact also helps to reduce the overfitting problem.

A second advantage of our approach is that non-stationary behavior is handled by using context transitions and the swapping of models. The different operational contexts of a system are captured by different small models representing local (referring to a context) stationary behavior in the data. Combining these models together by swapping a currently active model with another model provides a way to handle global non-stationary behavior in the data.

Finally, for the domain expert, there can be more meaningful diagnoses due to the underlying logical knowledge base and domain focusing. Since the contextual models are knowledge-focused (representing only relevant information due to KBMC) and are associated with a subset of the first-order knowledge base, the analysis of these models and their contextual differences can be much more meaningful to a domain expert.

There are a number of directions for future research. Even though it is linked to a probabilistic model, only deterministic context changes are currently allowed. A major extension would be to allow stochastic context transitions that will potentially increase the reasoning power of our framework. Other research directions include extending the method for detecting model failure, e.g., by using several sliding windows of various sizes to increase the sensitivity of the method while keeping the rate of false-positive error low.

Interested readers can find the GLL software supporting failure-driven context-modeling at `http://www.cs.unm.edu/~sanik/Support/gll.tgz`.

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 of Management Sciences, Inc. and Tom Caudell, Lance Williams, and Roshan Rammohan at the University of New Mexico. We also thank three anonymous reviewers for their critiques. This research was funded by an Air Force Research Laboratory SBIR contract (FA8750-06-C0016).

References

[1] Wellman M P, Breese J S, Goldman R P. From knowledge bases to decision models. *Knowledge Engineering Review*, 1992, 7(1): 35–53.

[2] Piaget J. Piaget's theory. *Handbook of Child Psychology*, ed. Mussen P, 4th edition, New York: Wiley 1983.

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

[4] Haddawy P. Generating Bayesian Networks from Probability Logic Knowledge Bases. In *Proc. of 10th Conf. on Uncertainty in AI*, July, 1994, pp. 262–269.

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

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

[7] Glesner S, Koller D. Constructing Flexible Dynamic Belief Networks from First-Order Probabilistic Knowledge Bases. In *Proc. of the European Conference on Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, 1995, pp. 217–226.

[8] Sato T, Kameya Y. PRISM: a language for symbolic-statistical modeling. In *Proc. of the 15th Intl. Joint Conf. on AI (IJCAI)*, 1997, pp. 1330–1335.

[9] Sato T, Kameya Y. New advances in logic-based probabilistic modeling by PRISM. In *Probabilistic Inductive Logic Programming*, 2008, pp. 118–155.

[10] DeRaedt L, Kimmig A, Toivonen H. ProbLog: a probabilistic Prolog and its application in link discovery. In *Proc. of the 20th Intl. Joint Conf. on AI (IJCAI)*, 2007, pp. 2468–2473.

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

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

[13] Kersting K, DeRaedt L. Bayesian Logic Programs. In *Proc. of 10th Int. Conf. on ILP*, 2000, pp. 138–155.

[14] Richardson M, Domingos P. Markov Logic

Networks. *Machine Learning*, 2006, 62(1–2): 107–136.

[15] Shen Y D. Reasoning with Recursive Loops under the PLP Framework. *ACM Trans. on Computational Logic*, 2008, 9(4): 1–30.

[16] Sanghai S, Domingos P, Weld D. Relational dynamic Bayesian networks. *Artificial Intelligence Research*, 2005, 24: 759–797.

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

[18] Sanscartier M J, Neufeld E. Identifying Hidden Variables from Context-Specific Independencies. In *Proc. of FLAIRS-07 Conference*, 2007, pp. 472–477.

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

[20] Halpern J, Pearl J. Causes and Explanations: A Structural-Model Approach — Part 1: Causes. In *Proc. of 17th Conf. on Uncertainty in AI (UAI-01)*, 2001, pp. 194–202.

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

[22] Chickering D. Learning bayesian networks is NP-complete. In *Proc. of AI and Stat.*, 1995.

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

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

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

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

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

[28] Dayanik S, Goulding C, Poor H V. Joint Detection and Identification of an Unobservable Change in the Distribution of a Random Sequence. *Information Sciences and Systems*, 2007, 68–73.

[29] Steyvers M, Brown S. Prediction and Change Detection. *Advances in Neural Information Processing Systems*, 2006, 18: 1281–1288.

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

[31] Kuncheva L I. Classifier ensembles for detecting concept change in streaming data: Overview and perspectives. In *Proc. 2nd Workshop SUEMA (ECAI)*, 2008, pp. 5–10.

[32] Dybvig R K. The Scheme Programming Language. The MIT Press, 2009.

Nikita A. Sakhanenko After receiving his master's degree in Mathematics from the Novosibirsk State University in Russia, Nikita Sakhanenko shifted his attention to Computer Science. He received a master's degree and then a PhD in Computer Science from the University of New Mexico in 2008,. His dissertation focuses on logic-based probabilistic modeling applied to sets of streaming data. Contexts are defined as stable-over-time components of a stochastic model and parameter drift is seen as an indicator of model failure.

Currently, Nikita Sakhanenko has a postdoctoral research appointment at the Institute of Systems Biology in Seattle Washington. At the Institute he is working on modeling and prediction of genes and gene interactions associated with different diseases, participating in the development of a biological complexity metric identifying modules of genes, and continuing his research in logic-based probabilistic modeling.

George F. Luger Since 1979 George Luger has been a Professor in the UNM Computer Science Department. His two master's degrees are in pure and applied mathematics. He received his PhD from the University of Pennsylvania in 1973, with a dissertation focusing on the computational modeling of human problem solving performance in the tradition of Allen Newell and Herbert Simon.

George Luger had a five-year postdoctoral research appointment at the Department of Artificial Intelligence of the University of Edinburgh in Scotland. In Edinburgh he worked on several early expert systems, participated in development and testing of the Prolog computer language, and continued his research in the computational modeling of human problem solving performance.

At the University of New Mexico, George Luger has also been made a Professor in the Psychology and Linguistics Departments, reflecting his interdisciplinary research and teaching in these areas. His most recent National Science Foundation supported research is in diagnostic reasoning, where he has developed stochastic models, mostly in an extended form of Bayesian Belief Networks. His book, Cognitive Science, was published by Academic Press in 1994.

His AI book, Artificial Intelligence: Structures and Strategies for Complex Problem Solving (Addison-Wesley 2008) is now into its sixth edition. His publications may be found at: `http://www.cs.unm.edu/~luger/`