Gaia – A Software Framework for Temporal Probabilistic Reasoning

P. Wiggers, L. J. M. Rothkrantz

Key Words: *Temporal probabilistic reasoning; (dynamic) Bayesian networks; speech and language processing.*

Abstract. *In this paper we present our general purpose framework for probabilistic temporal reasoning that is specifically designed for effcient processing in domains with large state spaces. We discuss the design and the features of the toolkit. Features include: sparse representations of probability tables, a fast algorithm for inference with probability tables, lazy evaluation of probability tables, algorithms for calculations with treeshaped distributions, the ability to change distributions on the fly, and a generalisation of dynamic Bayesian networks that we developed. Finally, we discuss the construction of a speech recognizer with the toolkit that was used to validate our implementation.*

1. Introduction

In recent years, the interest in probabilistic reasoning over temporal processes has increased. The applications have moved from toy problems to real domains which are characterised by large state spaces and huge amounts of data. For example, Markov chains are extensively used in statistical natural language processing [22] (where they are called n-grams) as are weighted finite state machines. Google's pagerank is another example of a Markov chain [7]. Hidden Markov models (HMMs) are the dominating paradigm in speech recognition and have been applied in handwriting recognition and gesture recognition. Dynamic Bayesian networks are a popular tool for motion and object tracking [17,20]. Among other things, they have been applied to multimodal sensor fusion [34], mobile robot navigation [8], prediction of driving behaviour [24], modelling of physiological processes [18]and gene expression recognition [29].

Moreover, dynamic Bayesian networks have been shown to be a generalisation of the models mentioned above [35] as well as of others such as Kalman filters [31]. Unlike these 'flat' models, dynamic Bayesian networks allow a factored representation of state. In many domains this leads to much sparser encoding and hence to more efficient processing. In addition, formulating different models in the same framework enables simple model combination and removes the need to develop new inference algorithms for specialised models. Following this line of reasoning, dynamic Bayesian networks have been introduced in domains as speech recognition [43,3] and language modelling [40,39].

Developing models for such domains is only useful if an efficient implementation is available that makes experimentation simple and efficient. A number of toolkits exists. Unfortunately, these systems are typically designed for research on probabilistic reasoning as such or tuned for particular domains. They all lack features that are needed to deal with the large state spaces that are implied by applications such as natural language processing, in which the states of a variable are typically words or grammatical categories.

In this paper we describe $Gaia$, our general purpose framework for temporal probabilistic reasoning that is designed for efficient processing in domains with large state spaces. In sections 2 and 3 we give a short review of Bayesian networks and dynamic Bayesian networks. In section 4 we discuss related work. The features and the design of our framework are discussed in section 5. Section 6 presents a speech recognizer we implemented to validate the framework. We conclude with a discussion of future extensions of the framework.

2. Bayesian Networks

Bayesian networks originate in artificial intelligence as a method for reasoning with uncertainty based on the formal rules of probability theory [28,9]. A Bayesian network represents the joint probability distribution over a set of random variables $X_1, X_2 \ldots X_N$. It consists of two parts:

1. A directed acyclic graph (DAG) G, i.e. a directed graph without any directed cycles. There exists a one to one mapping between the variables in the domain and the nodes of G, i.e. every node v_i in G represents exactly one variable X_i and every variable X_i is represented by exactly one node v_i. The directed arcs in the network represent the direct dependencies between variables. The absence of an arc between two nodes means that the variables corresponding to the nodes do not directly depend on each other.

2. A set of conditional probability distributions. With each variable X a conditional probability distribution $P(X|Pa(X))$ is associated, that quantifies how X depends on $Pa(X)$, the set of variables represented by the parents of node V in G representing X.

Applying the chain rule of probability theory and the independence assumptions made by the network, we can write the joint probability distribution represented by the network in factored form as a product of the local probability distributions:

$$(1)\quad P(X_1, X_2 \ldots X_N) = \prod_{i=1}^{N} P(X_i|Pa(X_i))$$

The computational complexity of the right hand side of (1) may be much more efficient than that of the full joint representation.

2.1. Inference

Inference in Bayesian networks is the process of calculating the probability of one or more random variables given some evidence, i.e. computing $P(X_Q|X_E{=}x_E)$ where X_Q is a set of query variables and X_E is a set of evidence variables. Let X_H be the set of variables that are not part of the query and that have not received any evidence:

(2) $X_H = \{X_1, X_2 \ldots X_N\} \setminus (X_Q \cup X_E)$,

then

$$(3)\quad P(X_Q|X_E) = \frac{P(X_Q,X_E)}{P(X_E)} = \frac{\sum_{X_H} P(X_H, X_Q, X_E)}{\sum_{X_H,X_Q} P(X_H, X_Q, X_E)}.$$

For given evidence the denominator is a constant, therefore we can write:

$$(4)\quad P(X_Q|X_E) = \alpha \sum_{X_H} P(X_H, X_Q, X_E).$$

Inference thus comes down to marginalisation over X_H followed by normalisation. Directly applying equation (4) is intractable for most networks, but a number of efficient algorithms, such as message passing [28], variable elimination [12,1] and the junction tree algorithm [33,21] exist that exploit the independence of variables in a network. Approximate algorithms sacrifice accuracy for speed, examples include loopy belief propagation and stochastic sampling methods [32,36].

2.2. Learning

For Bayesian networks both the structure as well as the probability distributions can be learned from data. The networks may be fully or partially observed during learning. Networks are partially observed if some data is missing or because some nodes are hidden. Learning of partially observed networks can be done using the expectation-maximisation (EM) algorithm [13], but other methods such as gradient descent have been used as well [26].

3. Dynamic Bayesian Networks

When modelling temporal processes, we want to specify relations between variables in time. Typically, the number of time steps in such a process is not known beforehand, i.e.the number of variables is potentially infinite. Dynamic Bayesian networks (DBNs) [11] offer a concise way to model processes that evolve over time. A DBN is defined by two Bayesian networks: a prior $P(X_1)$ and a transition model that defines how the variables at a particular time depend on the nodes at the previous time steps.

$$(5)\quad P(X_t|X_{t-1}) = \prod_{i=1}^{N} P(X_t^i|Pa(X_t^i)),$$

were X_t^i is the i^{th} variable in slice t. The parents of a node can either be in the current or in a previous time slice. Typically, first order Markov assumptions are made, i.e. the nodes in a time slice only depend on the nodes in the previous time slice. If there is an arc from X_{t-1}^i to X_t^i the node is called persistent.

3.1. Inference

As DBNs are Bayesian networks all inference algorithms for Bayesian networks can be used. However, this is often not feasible as for example for the junction tree algorithm one would have to unroll the entire network, which for most networks will not fit into memory, that is, if one happens to know the length of the network beforehand at all. Online algorithms have been developed that unroll the network slice-by-slice. We briefly discuss the most important algorithms.

The Frontier Algorithm [43,26] is equivalent to variable elimination with a specific ordering. In a forward pass, a variable N is multiplied onto the frontier, when all its parents are already in the frontier as this will guarantee that N is independent from other variables in the past. Similarly, a node is removed, i.e. marginalized out, from the frontier when all its children are in the frontier. In the backward pass nodes are added and removed in the opposite order.

The Interface Algorithm creates a junction tree for an unrolled 1½ - slice DBN which consists of a time slice and all interface nodes from the preceding slice. It then imposes the restriction that all the nodes in the forward interface must belong to one clique. The junction trees can be glued together via their interfaces. Inference can be performed in each tree separately and then messages are passed through the interfaces, first forwards and then backwards.

Approximate Inference. The Boyen-Koller algorithm [6] is a factored version of the interface algorithm. It approximates the joint distribution over the interface as the product of smaller terms (marginals). The accuracy of the algorithm depends on the number of clusters that is used to represent the interface.

Using a single cluster corresponds to exact inference. The Boyen-Koller algorithm performs exact inference in a two-slice DBN, sometimes this approximation is still intractable. The Factored Frontier algorithm [26] uses a more aggressive factorisation by approximating the frontier by a product of marginals.

Particle filtering [14] is a simple and eficient sampling algorithm for DBNs that computes N samples in parallel. The algorithm focuses the set of samples on the high-probability regions of the state space by allowing only the N most likely samples to propagate to the next slice. Combinations of discrete approximation methods and stochastic sampling also exist [15].

4. Related Work

Several toolkits that implement dynamic Bayesian networks or generalisations thereof exist. dHugin [23] adds temporal reasoning to the popular commercial Hugin [2] shell. It assumes that DBNs obey the Markov property, i.e. a variable only depends on variables in the current or in the previous time slice. The stucture of time slices can vary. An exact junction-tree based inference routine that unrolls the network for k slices at time as well as forward sampling are provided.

The Bayes net toolbox (BNT) [27] is a free, open-source library intended for research purposes. It is implemented in Matlab because of the ease with which it can handle Gaussian random variables. On the down side the choice for a high level language makes the toolbox relatively slow and limits the size of the networks that can be processed. A DBN is represented with a prior and a transitional network, so that only first-order Markov processes can be modelled. Several inference algorithms for static Bayesian networks are provided, each of which makes different trade offs between accuracy, generality, simplicity and speed. The conditional probabilities of the defined variables can be continuous or discrete. Parameter learning is supported as well. Currently, the toolbox lacks online inference and learning,and does not include prediction.

The probabilistic network library (PNL) [19] is a C++

version of BNT implemented by Intel's research lab in Saint Petersburg. It does not yet support all functionality of the Bayes net toolbox.

The graphical models toolkit (GMTK) [4] is a freely-available toolkit written in C++ that is specifically designed for DBN-based speech recognition. Models have to be defined in GMTKL a flexible but complex specification language. In this language a DBN definition consists of several frames each of which can define different variables and relations between variables. The first N frames form the prologue and are used at the beginning of the network, the last M frames, called the epilogue, are used for the final M frames in the network. The frames between N and M form the repeating substructure. The toolkit has many desirable features, such as sparse representations of CPDs, tree-shaped CPDs,continuous observation distributions, switching parents, beam search and generalised EM training. It supports smoothing and Viterbi inference using the online Frontier algorithm. But because of the epilogue in the network definition the length of an input sequence has to be known in advance.

The Structural Modeling, Inference and Learning Engine (SMILE) [16] is a platform independent library of C++ classes that implements Bayesian networks and influence diagrams. Recently, support for temporal reasoning has been added [18].

5. The Gaia Framework

Unlike the frameworks discussed above,our framework is developed with the tasks of speech and language processing in mind. Although it should be noted that its applicability is not limited to those domains. Specifically, it can deal with very small probabilities and a large number of states while the data representations and inference algorithms have been optimised for sparse distributions. It features, tree-shaped distribution that can exploit reoccurring substructure and implement interpolation and smoothing schemes. Data sparsity can be battled by parameter sharing among several nodes in the network. The framework supports switching parents, i.e. variables of which the values are used to decide on which of its (other) parents a variable will be conditioned, such as interpolation weights. Furthermore, the framework implements a generalisation of DBNs that allows the structure of the network to change over time and k-th order Markov relations between time slices.

The toolkit consists of a collection of general purpose tools for inference and learning of DBNs. The functionality behind these tools is implemented in a common library implemented in C++ that makes it easy to change or reuse parts of the software and to experiment with different algorithms and data structures. The library has a layered structure. At the lowest level it provides general purpose classes, such as input and output routines and a class that can represent very small probabilities. The core of the system is formed by classes that implement efficient mathematical operations on multidimensional probability tables. The top layer is responsible for construction of and inference in dynamic Bayesian networks.

The remainder of this section discusses these layers in more detail focusing on the algorithms and data structures that we developed.

5.1. Dealing with Small Probabilities

At the lowest layer two data structures implement the mathematical concepts of probabilities and likelihoods and their operations. A likelihood is defined as a positive real value that can result from an operation on probabilities. Compared to a simple floating point representation these representations are more robust in the face of underflow, which is a huge problem for models, such as HMMs and DBNs, that multiply long sequences of probabilities together. They lift the burden of normalising or worrying about underflow from higher layers.

The first method accomplishes this in the classical way of representing a probability as its logarithm[37]. This has the additional advantage that multiplication and division become faster since they reduce to addition and subtraction. However, addition and subtraction become more complex and slower.

The second approach extends its range compared to double precision floating points. It represents its value as a mantissa in the interval (0.5,1)and an integer exponent. The value follows from: $\text{mantissa}*2^{\text{exponent}}$. This approach takes more space than the first and is slower for multiplication and division, but when it comes to addition and subtraction it is considerably faster.

5.2. Likelihood Tables

The second layer of the library introduces multidimensional tables of likelihoods. These tables are the core of the library. They are used to implement multinomial probability tables, to store intermediate calculations and as accumulators in learning. To make low level parameter sharing possible, tables consist of two parts: the table itself with an associated list of cardinalities and a domain of variables. As long as the cardinalities match, the same table can be used with multiple domains.

Tables are implemented in several ways, giving a network designer the possibility to find a good balance between space and time requirements. The straightforward implementation has one entry for every tuple of values of variables in its domain. For small tables this is efficient as one does not explicitly have to save the indices of values in the table and as entries are ordered any value can be found in constant time.

However, for many tables this representation will take too much space. *Sparse tables* offer an alternative. They only store non-zero values together with their indices. For sparse distributions this can result in enormous space savings at the cost of increased search time. To minimise the latter indices are stored in order. This allows our inference algorithm that processes tables as a whole to accesses the indices (semi) sequentially. It is possible to specify default values different from zero for sparse tables. Only probabilities that are different from the default value will then be saved. Our default implementation of sparse tables uses red-black trees, but alternative implementations using hash tables or tries exist.

Deterministic variables have only one possible value for every configuration of their parent variables (that thus must have probability 1). As a consequence, there is no need to save the probabilities, only the non-zero values have to be saved. The current implementation is similar to that of sparse tables, with the difference that it only saves indices. The resulting structure is strictly speaking more general than described above, as it is

possible to have more than one value for the same 'parent configuration'. This allows to represent evidence that only certain states are possible.

Constant tables are used to implement uniform distributions, i.e. they store a single probability that applies to all values.

Calculations with likelihood tables proceed as described by [21]. Multiplication of two tables is defined as pairwise multiplication of elements that have equal values for all common variables. Equation (6) gives an example.

(6)

	A
A	0.2
	0
	0.8

×

		A		
B		0.3	0.5	0.7
		0.3	0.5	0.1
		0.4	0	0.2

=

		A		
B		0.06	0	0.56
		0.06	0	0.08
		0.08	0	0.16

Marginalisation projects a table to a lower dimension, by summing over all values of the variables that are marginalised out for every tuple of values of the remaining variables. Equation (7) shows how a three-dimensional table is projected to a one-dimensional table.

(7) $\sum_{BC}$

		A		
B	C	0.012	0	0.056
		0.048	0	0.504
	C	0.036	0	0
		0.024	0	0.08
	C	0.04	0	0.16
		0.04	0	0

=

	A
A	0.62
	0.14
	0.24

An inference algorithm has been developed to multiply any number of tables and project the result to a subdomain of the joint domain in one operation. The algorithm uses the domains of the tables to deal with overlapping variables. It also exploits observed values to avoid unnecessary calculations and to keep the size of the resulting table as small as possible. Since probability table manipulation makes up the bulk of the work in probabilistic inference a lot of attention has been dedicated to optimising this part of the library. For example special memory pools are used for fast allocation and deallocation of probability tables.

(8)

	A	
C	0.2	0.4
	0.3	0.1

,

	B	
D	0.1	0.3
	0.2	0.4

The algorithm systematically iterates through all values of the joint domain of the input tables. Variables are ordered in all tables. All values of a lower ranking variable are processed before the value of a higher ranking variable is increased. For example, in (8) the order might be A, B, C, D. When multiplying these tables a structure as shown in *figure 1*is created. Initially, all variables will have value zero and the upper left corners of the tables will be multiplied, then D will increase its value. As D has only two values, it will be reset to its start position in the third step, while the value of C is increased by one.

As a consequence, all input tables are processed in sequential blocks (non-overlapping input tables are processed sequentially). This approach is fast in the face of buffers, caches and paging. It guarantees that intermediate results never take more space than the end result. In particular, if no projection to the output domain is needed, the output table will be built sequentially.

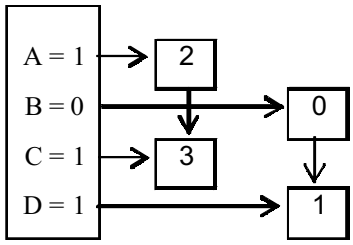

Figure1. Visualization of the multiplication algorithm

To limit the amount of searching in the tables, a reference to its direct parent within the domain of a table is kept for every variable, as this is the starting point for this variable within the set of values of lower ranking variables. For example, *figure1*shows the situation where the last value of the first table (0.1) is multiplied with the second value of the second table (0.2). In the next step, the value of B will become one and the values of C and D will become zero. The squares in the *figure* represent pointers that the algorithm keeps into the probability tables. On increasing the value of B its pointer will be moved to the position following the position of the next variable in this table, D in this case, thus it will become 2. In general, the table might be sparse, so it will use this position as a starting point to search for the first non-zero entry that has B = 1. Next all variables (lexicographically) further down the ranking will copy the positions of the first variable with a value lower than or equal to B, i.e. C copies the position of A (0) and D copies the new position of B (2). Observed variables are skipped completely.

The algorithm only processes values that are non-zero in the output table. Rather than increasing the value of a variable by one in every step, the input tables are consulted to find the next common non-zero probability within the current set of parent values. For summation the algorithm moves to the next value for which one of the tables has a non-zero probability.

5.3. Lazy Evaluation

As discussed in section 2 inference in Bayesian networks comes down to a sequence of multiplications and marginalisations of likelihood tables. Most inference techniques determine the order of operations based on the structure of the network, without taking the shape of the probability distributions into account. In addition, this is typically done before any evidence, that may introduce additional independence relations, has been observed. As a consequence, the order is not always efficient. [25] introduced lazy evaluation in the junction tree algorithm to make better use of evidence and probability table structure.

We implemented a similar lazy evaluation approach, but at the level of likelihood tables. This has the advantage that it can be used with any inference algorithm. In this approach multiplication of tables is deferred until it really is necessary, i.e. if a variable in the domain of the table is marginalised out. Rather than a table a set of tables is used. Multiplication is a very fast operation as it simply adds the table to the set. Upon marginalisation or summation all tables in the set whose domains contain variables that are marginalised out are selected. Multiplication of these tables and marginalisation are done in a single operation to avoid the construction of (large) intermediate

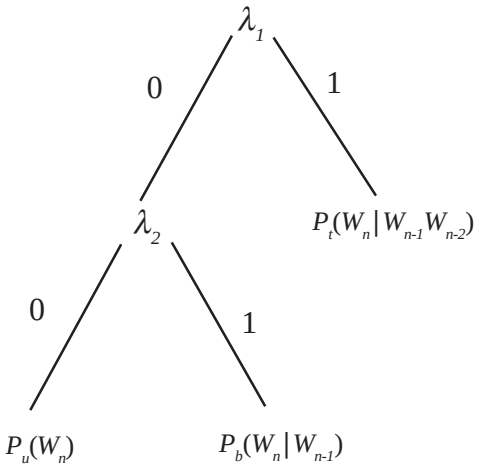

Figure 2. A decision tree representation of an interpolated trigram. *Wi* corresponds to the *i*-th word of a sentence. λ_1 and λ_2 are interpolation weights.

tables. Note that if the variables of a particular subtable are never looked at, that table will never really be processed. In Bayesian networks this may occur if certain subparts of the network are not needed in an inference process (barren nodes).

5.4. Tree-shaped Likelihood Tables

If a table contains reoccurring substructure or if several tables share the same substructure an even more efficient re presentation of tables is possible. *n*-ary decision trees are be used for this purpose[5,4]. Every node in the tree represents a random variable, it has a list of values that are connected to either another node or to a set of tables as described above. A table or subtree of tables can be shared by several trees, allowing for very flexible parameter sharing. *Figure 2* shows a tree representation of an interpolated trigram $P(W_i|W_{i-1}W_{i-2}\lambda_1\lambda_2)$. If $\lambda_1 = 0$ the word variable W_i is independent of W_{i-2}, if $\lambda_2 = 0$ as well W_i is also independent of the previous word W_{i-1}.

Unlike [5,30] who use the tree-structures to alter the network structure we use the tree-shaped CPDs directly. Although altering the network structure may lead to smaller cliques and hence to faster processing, our approach, in combination with lazy evaluation, has the advantage that whole sequences of operations may be skipped as on observation of a value of a switching variable all subtrees associated with other values can be removed at once. We developed algorithms that use decision tree manipulations for multiplication and marginalisation of these switching tables.

Tree multiplication. Multiplication and division of trees comes down to merging trees. Multiplication creates a new tree by attaching the right-hand-side tree below every leaf of the left-hand-side tree and subsequently removing unreachable paths. Subtrees are non-reachable when their root question already appears higher up in the tree with a different answer value.

The new tree has the tables of the right-hand-side-tree as its leafs. To obtain the product the corresponding leafs of the left-hand-side tree are multiplied onto the leafs of the new tree (The construction of the new tree guarantees that there is exactly one path in the old tree for every path in the new tree). *Figure 3* illustrates tree multiplication. A missing link, such as the link corresponding to B = 0, C = 1 in the left-most tree in the *figure*, means that all corresponding probabilities are zero.

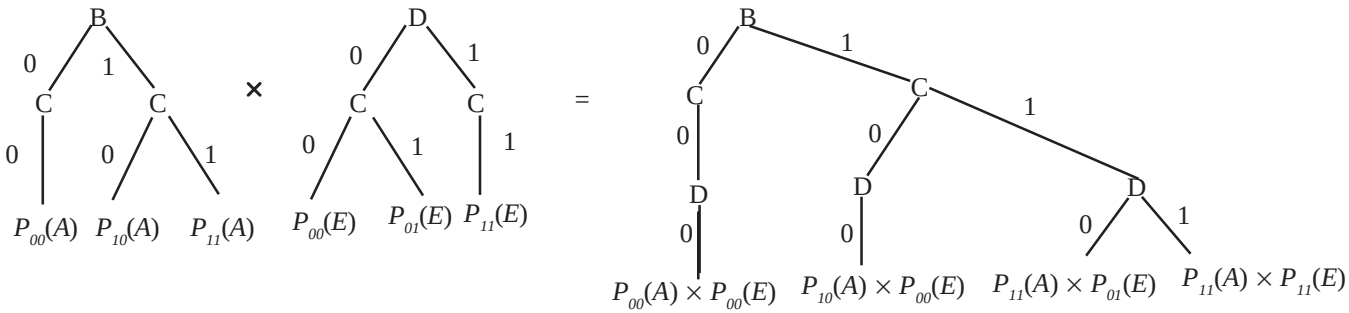

Figure 3. Multiplication of trees representing probability distributions P (ABC) and P (CDE)

Tree marginalisation. Marginalisation processes the tree in (depth first) post-order. First leaf tables are marginalised and subsequently their parent nodes. Marginalising a tree node means summing over its subtrees. Since leafs are marginalised before their parents these node variables, that act as observed variables in the leafs ,will be removed from the leaf tables. All leaf tables will thus have the same domain the moment the node is marginalised. An example of tree marginalisation is given in *figure 4*.

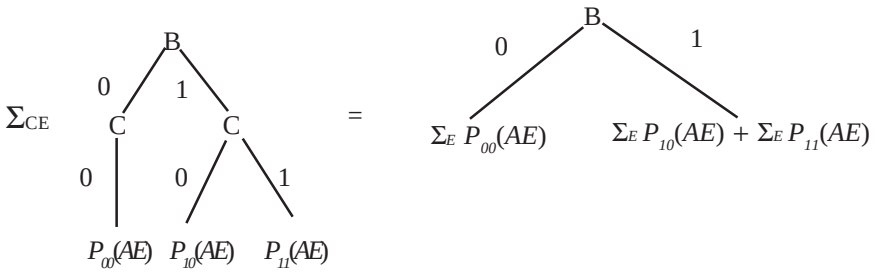

Figure 4. Marginalising variables C and E out of a tree representing probability distribution P(ABCE). Marginalisation of leafs proceeds as usual, marginalisation of intermediate nodes requires summation of subtrees.

5.5. Potentials and Distributions

Tree-shaped tables are used to represent both potentials and multinomial distributions. Potentials are tables of likelihoods that represent intermediate results in an inference procedure. Multinomial distributions on the other hand can only contain probabilities. In addition, distributions can have accumulators that are used when learning the parameters of a network. Accumulators can be shared across distributions. For example, when learning the parameters of a DBN, the accumulators are shared across time slices. But parameters can also be tied for learning across other variables. Continuous observation distributions, such as Gaussian mixtures are also supported. Upon observation these distributions provide one probability for every set of parent values. Hence the result is a potential over the parent variables that is processed as any other potential.

5.6. Network Structure

The upper layer of the library implements a generalisation of dynamic Bayesian networks. To increase the flexibility of the system this layer consists of three components: an abstract definition of DBNs that is not bound to any inference algorithm, an interface to the outside world that implements high level inference technique independent functionality such as learning and an inference engine. Different inference engines can be plugged into the system without changing any of the other structures.

Whereas standard DBNs have a repeating substructure, our networks can have any number of subnetworks, called chapters, that each have a repeating substructure. The last chapter will repeat indefinitely, all other chapters must have a predefined length. The substructure that repeats within a chapter may span several time slices. In addition, a chapter can define static variables, that do not have temporal dynamics. It is also possible to define static variables at the network level. Those correspond to the contemporal nodes of [18]. Following [18], we also provide a separate chapter with static variables that is attached at the end of the network.

To every variable in the network a distribution is attached. Variables with the same name in different time slices and chapters share the same distribution. The parents of a variable can be in any preceding slice or chapter, i.e. k-th order relations are allowed.

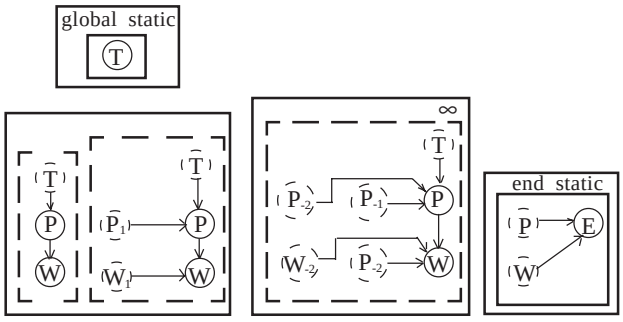

Figure 5. A generalised DBN definition of a language model in which the words W are conditioned on their part of speech P that are conditioned on the type of speech T. The outer rectangles represent chapters, dashed rectangles represent slices in non-static chapters that have repeating substructure. Dotted nodes are references to parent nodes in the expanded network.

The user of the system provides the chapters of the network and a template of the repeating slices in the network as well as the (tree-shaped) distributions in XML format .The DBN interface is responsible for expanding (unrolling) this definition into a network and checking it for consistency.

Figure 5 shows an example of a generalised DBN definition for a language processing model that assigns probabilities to word sequences (where sequences that are likely sentences should get high probabilities and random word sequences should have low probabilities) in which the words W are conditioned on their part of speech P, such as verb, noun or determiner, that are in turn conditioned on the type of speech T, i.e. whether it is conversational speech or more formal speech. *Figure 6* shows the corresponding expanded network.

The rectangles in *figure 5* denote chapters. The top-most chapter contains global static nodes. It is assumed that the type

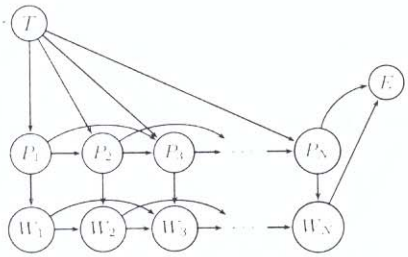

Figure 6. The unrolled network (N slices) corresponding to *figure 5*

of speech is constant for the whole word sequence. Therefore, the type of speech random variable is placed in this chapter. The other chapters are combined left-to-right in the expanded network. The first chapter defines slices for the first two time steps, represented by rectangles with dotted lines. The number in the upper right corner gives the length of this chapter. As the chapter has a length of two, the slices in it will not be repeated.

After the first two time steps, the expanded network will continue with the second chapter. This chapter contains a single slice that will be repeated as long as there are input values.

The dotted circles are place holders for parents of a node that are defined in other slices in the expanded network. For example all dotted T nodes refer to the global T variable and the P_{-1} node in the second chapter refers to the part of speech node in the previous slice. Note that for the third time step this is a variable in the first chapter and after that it is a node in a previous instantiation of the second chapter. When the whole word sequence is processed the final chapter, that contains an end of sequence node, is attached to the network.

Figure 5 illustrates the advantage of subdividing a network in chapters. There is no need to repeat the type of speech and end of sequence nodes in every time step as is the case in standard DBNs. This is not just a convenience for the designer of the network, but also allows for faster inference.

5.7. Inference Engine

The toolkit supports filtering, smoothing, prediction and Viterbi inference. For the latter it can be specified which variables should be marginalised (summed) out and which should be maximised. It is possible to run any of these algorithms in interactive mode to inspect any variables in any time slice during inference.

We currently implemented the Frontier algorithm and the Interface algorithm for exact online inference and the Boyen-Koller and factored frontier algorithm for approximate inference. The generalised structure using chapters and k-order Markov relations complicates things a bit as for different time steps a variable (in a particular chapter) may have a different set of children. In addition, as the same variable may appear in different chapters, the last child of a variable may change. This is solved by splitting up the network in additional chapters in such a way that the variables in a particular chapter have the same children in all future slices that can be reached from slices in the chapter.

In a preprocessing step the Frontier engine establishes the elimination order based on a number of rules :1) A node can be added to the frontier when all of its parents are in the frontier. 2) A node can be removed when all its children are in the frontier. 3) Sum marginalisations are added before max marginalisations. 4) Heuristics or a search algorithm can be used to order unconstrained nodes. When an input sequence is processed these operations are added to a processing queue. Different operations such as filtering, prediction and smoothing may alter the order of the operations in the queue. Furthermore, if one wants to consult the value of particular variables the order of the operations in the queue can be altered on the fly.

5.8. Learning

For parameter learning the expectation-maximisation algorithm is used. It has been implemented using a forward and a backward pass, enabling the use of any inference engine that implements smoothing. To allow for eficient incremental model improvement, e.g. update parameter weights, it is possible to specify which variables will be updated in a training run and which not. An optional damping factor can be set that combines the distribution and the newly learned distribution. At every time slice intermediate results have to be saved in the forward pass, so the space requirements of the algorithm may quickly get out of hand. Therefore, the Island algorithm [44] has been implemented. This algorithm saves messages at C island points during the forward pass (including the first and last slice), resulting in $C-1$ subproblems on which the algorithm is then called recursively. If messages are saved every $\sqrt{T}$ steps the space complexity will be reduced to $O\,(Slog_C T)$, the cost of this is a increase in time complexity from $O(T)$ to $O\,(Tlog_C T)$.

The learning tool can work in distributed mode. Each processor learns a part of the data. A central thread of the program combines the accumulators of the other programs.

5.9. Context-dependent Distributions

A special feature of the toolkit is that the distributions in the network can be changed on the fly. This is especially useful when training with partial information. For example, recordings used for training in speech recognition are typically annotated at the word level. Using a dictionary one can find the sequence of phonemes that are being spoken (although due to pronuncation variation and dialects this will usually not be a perfect match with the sounds that are being spoken) but it is unkown when a particular word or phoneme starts or ends, therefore this information cannot simply be used as evidence for specific variables (at a specific point in time) during training.

However, using the context feature one can change the word or phoneme distributions in a speech recognizer network such as shown in figure7 to match the actual transcription for

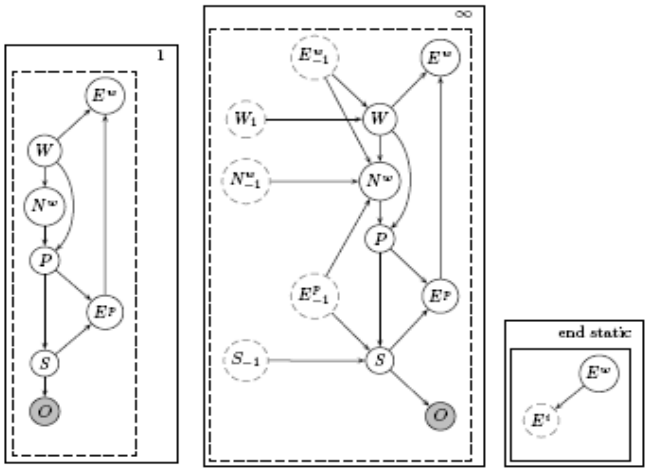

Figure 7. A speech recognizer implemented with Gaia. O is the observation vector, W represents words, P the phonemes and the S are subphonemic states. The E nodes are binary nodes that signal when a level is allowed to change state.

every recording, i.e. encode the word and/or sounds and the order of the words, while leaving the transitions between the phonemes to be learned.

5.10. Pruning

Inspired by the common practice in hidden Markov models and probabilistic grammars to prune large parts of the search space to achieve real-time performance, we implemented beam pruning that sets all probabilities that fall below some treshold relative to the highest probability in a potential to zero. For Bayesian networks pruning is seldomly used. Speed ups are achieved using approximate algorithms. The advantage of pruning over such methods is that pruning focuses on the high probability paths through a network given the observations. Stochastic approximate inference techniques such as particle filtering do focus on high probability paths. However, to achieve real-time performance the number of paths through the model that will be sampled will be sparse.

6. Validation

To test the correctness of our mplementation we reimplemented a speech recognizer using the toolkit and compared that to the original implementation of that recognizer in terms of hidden Markov models. The original system was trained on the Dutch Polyphone dataset [10] using HTK [42] as described in [41]. In a preprocessing phase all audio files were encoded using 12 mel frequency cepstral coefficients (MFCCs) and the total energy in the signal. Those features together with their first and second order derivatives, were calculated every 10 ms using a window of 25 ms.

For this test we used 43 simple 3 state left-to-right hidden Markov models that represented the phonemes of Dutch and models for silence and mouth noises. A 39-dimensional Gaussian distribution over the input is attached to every state.

A vocabulary of 150 words was selected, pronunciations for those words are obtained by combining the phoneme models using the pronunciation dictionary of the Polyphone corpus. For recognition those words were combined using a ordloop language model, i.e. all words are considered equally likely and every utterance can consist of a sequence of words. Our test set consisted of 181 files from the Polyphone set that were not used for training.

The structure of the DBN speech recognizer is shown in *figure 7*. In this model the W nodes represent words, i.e. the values of the W variable are the 150 words in our vocabulary. As before we used a uniform distribution over the words. The P nodes represent phonemes. We used the same 43 phonemes as in the baseline system. N^{ω} is a counter variable that keeps track of the phoneme position within a word. Therefore $P(P \mid W, N^{\omega})$ is a deterministic distribution that encodes that pronunciation dictionary. As before every phoneme has three substate. This is represented by the S variable. Only transitions from a state to itself or to its direct predecessor are non-zero. The O variable represents the state dependent Gaussian distributions over the input. E^{p} is binary variable that signals that the end of a phoneme is reached. This is only possible if the last state of a phoneme is reached. In the same way E^{ω} indicates the end of a word. $P(E^{\omega} \mid W, N^{\omega}, E^{p})$ is deterministic, the end of a word is reached if the end of its last phoneme is reached. The end chapter of the model contains a variable E^{i} which signals the end of the input sequence. It is a child variable of E^{ω} and can only get value 1 if E^{ω} is one as well. To make sure that the recognizer will come up with a valid word sequence the value of E^{i} is clamped to 1. As long a the end of the word is not reached the value of the word variable will remain the same in the next time slice. If the end of the word is reached the next word will be chosen according to a uniform distribution. In a similar way, the next subphonemic stated is determined. The original system recognized 77% of all utterances in the test set correctly, the DBN system using the Gaia inference engine (most probable explanation usign the Frontier algorithm) recognized 75% of all utterances. The difference in results is due to small differences in the calculations.

7. Summary and Conclusion

In this paper we presented Gaia, our computational framework for temporal probabilistic reasoning. We discussed several features of the framework that enable efficient processing of models with large state spaces. Novel contributions include a highly efficient algorithm for inference with probability tables, the combination of lazy evaluation at the probability table level and tree-shaped distributions that exploit independencies in the network that are not used by junction tree based inference algorithms such as the Frontier and Interface algorithm and a generalisation of dynamic Bayesian networks that consists of several chapters each of which contains a repeating substructure that can span several slices in the time domain. The network and the corresponding distributions can be fully specified in an easy to read XML format, which allows for rapid model constructions and experimentation.

We successfully implemented a speech recognizer using the framework in order to validate our inference algorithms . The framework has been applied in the domains of speech recognition and language modelling where it enabled us to experiment with models that contained several variables with hundreds to thousands of states in every time slice [40,38].

In the future we plan to further extend our framework with stochastic sampling algorithms and with algorithms for structure learning. We will also introduce parameterized network defnitions that can be used to instantiate several copies of a subnetwork at runtime. For example if one has a system in which combines sensor information the subnetwork that is responsible for the sensor information can be instantiated as many times as there are sensors in an actual situation.

References

1. Aji, R., S. M. McEliece. The Generalized Distributive Law. – *IEEE Transactions on Information Theory*, 46 (2), March 2000, 325-343.

2. Andersen, S. K., K. G. Olesen, F. V. Jensen, and F. Jensen. HUGIN – a Shell for Building Bayesian Belief Universes for Expert Systems. Eleventh International Joint Conference on Artificial Intelligence (ijcai89), 2, 1989, 1080-1085.

3. Bilmes, J. Natural Statistical Models for Automatic Speech Recognition. PhD Thesis, Dept. of EECS, University of California, Berkeley, May 1999.

4. Bilmes, J. and G. Zweig. The Graphical Models Toolkit: An Open Source Software System for Speech and Time-series Processing. Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing, 2002.

5. Boutilier, C., N.Friedman, M.Goldszmidt, and D.Koller. Context-specific Independence in Bayesian Networks. Uncertainty in Artificial Intelligence, 1996, 115-123.

6. Boyen, X. and D. Koller. Tractable Inference for Complex Stochastic Processes. Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, 1998, 33-42.

7. Brin, S. and L. Page. The Anatomy of a Large-scale Hypertextual Web Search Engine. – *Computer Networks and ISDN Systems*, 30 (1-7),1998, 107-117.

8. Chella, A., S. Vitabile , and R. Sorbello. A Vision Agent for Mobile Robot Navigation in Time-variable Environments. 11th International Conference on Image Analysis and Processing (ICIAP'01), 2001.

9. Cowell,R. G., A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. Probabilistic Networks and Expert Systems. Springer,1999.

10. Damhuis, M., T. Boogaart, C. In't Veld, M. Versteijlen, W. Schelvis, L. Bos, and L. Boves. Creation and Analysis of the Dutch Polyphone Corpus. Proceedings of the International Conference on Spoken Language Processing, ICSLP'94, Yokohama, Japan,1994, 1803-1806.

11. Dean, T and K. Kanazawa. A Model for Reasoning about Persistence and Causation. – *Computational Intelligence*, 5 (3), 1989, 142-150.

12. Dechter, R. Bucket Elimination: A Unifying Framework for Reasoning. – *Artificial Intelligence*,113 (1-2), 1999, 41-85.

13. Dempster, A. P., N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the em Algorithm. – *Journal of the Royal Statistical Society. Series B (Methodological)*, 39 (1), 1977, 1-38.

14. Doucet, A. On Sequential Simulation-based Methods for Bayesian Filtering. Technical Report CUED/F-INFENG/TR 310, Department of Engineering, Cambridge University,1998.

15. Doucet, A., N. de Freitas, K. Murphy, and S. Russell. Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks. UAI '00 (Uncertainty in AI), 2000.

16. Druzdzel, M. J. SMILE: Structural Modeling, Inference, and Learning Engine and GeNle: a Development Environment for Graphical Decision-theoretic Models. AAAI '99/IAAI' 99, Menlo Park, CA, USA,1999, 902-903.

17. Huang, T., D. Koller, J. Malik, G. H. Ogasawara, B. Rao, S. J. Russell, and J. Weber. Automatic Symbolic Traffic Scene Analysis Using Belief Networks. National Conference on Artificial Intelligence, 1994, 966-972.

18. Hulst, J. Modeling Physiological Processes with Dynamic Bayesian Networks. Master's Thesis, Man-Machine Interaction Group, Delft University of Technology, September 2006.

19. Intel Corporation. Probabilistic Network Library – User Guide and Reference Manual, March 2004.

20. Intille, S. S., and A. F. Bobick. Recognizing Planned, Multiperson Action. Computer Vision and Image Understanding: CVIU, 81 (3), 2001, 414-445.

21. Jensen, F. V. Bayesian Networks and Decision Graphs. Statistics for Engineering and Information Science Series. Springer Verlag, 2001.

22. Jurafsky, D., and J. H. Martin. Speech and Language Processing. Prentice Hall, 2000.

23. Kjaerulff, U. dhugin: a Computational System for Dynamic Time-sliced Bayesian Networks. – *International Journal of Forecasting*, 11 (1), March 1995 89-111.

24. Kumagai, T., and M. Akamatsu. Prediction of Human Driving Behavior Using Dynamic Bayesian Networks. IEICE Transactions on Information and Systems, E89-D (2), 2006.

25. Madsen, A. and F. Jensen. Lazy Propagation: a Junction Tree Inference Algorithm Based on Lazy Evaluation. Artificial Intelligence, 113, 1999, 203-245.

26. Murphy, K. Dynamic Bayesian Networks: Representation, Inference and Learning. PhD Thesis, University of California, Berkeley, 2002.

27. Murphy, K. P. The Bayes Net Toolbox for Matlab. Computing Science and Statistics, 33, 2001, 331-350.

28. Pearl, J. Probabilistic Reasoning in Intelligent Systems – Networks of Plausible Inference. Morgan Kaufmann Publishers, Inc., 1988.

29. Perrin, B. -E., L. Ralaivola, A. Mazurie, S. Bottani, J. Mallet, and F. dAlché Buc. Gene Networks Inference Using Dynamic Bayesian Networks. – *Bioinformatics*,19 (Suppl. 2):ii138-ii148,2003.

30. Pfeffer, A. Sufficiency, Separability and Temporal Probabilistic Models. UAI, 2001.

31. Roweis, S., and Z. Ghahramani. A Unifying Review of Linear Gaussian Models. – *Neural Computation*,11 (2), 1999, 305-345.

32. Russell, S. and P. Norvig. Artificial Intelligence: A Modern Approach. Prentice Hall, Englewood Cliffs, NJ, 1995.

33. Shafer, G. R., and P. P. Shenoy. Probability Propagation. Annals of Mathematics and Artificial Intelligence, 2, 1990, 327-351.

34. Singhal, A. and C. Brown. Dynamic Bayes Net Approach to Multimodal Sensor Fusion. SPIE Conference on Sens or Fusion and Decentralized Control, 3209, Pittsburgh, PA, 1997.

35. Smyth, P., D. Heckerman, and M. Jordan. Probabilistic Independence Networks for Hidden Markov Probability Models. – *Neural Computation*, 9, 1997, 227-269.

36. Srinivasan, R. Importance Sampling – Applications in Communications and Detection. Springer Verlag, Berlin, 2002.

37. Alphen, P. van. HMM-based Continuous-speech Recognition. PhD Thesis, University of Amsterdam, 1992.

38. Wiggers, P. Modelling Context in Automatic Speech Recognition. PhD Thesis, Delft University of Technology, 2008.

39. Wiggers, P., and L. J. M. Rothkrantz. Dynamic Bayesian Networks for Language Modeling. ILecture Notes in Artificial Intelligence 4188: Text, Speech and Dialogue 2006, September 2006.

40. Wiggers, P., and L. J. M. Rothkrantz. Topic-based Language Modeling with Dynamic Bayesian Networks. Proceedings of Interspeech 2006, Pittsburgh, Pennsylvania, September 2006, 1866-1869.

41. Wiggers, P., J. C. Wojdel, and L. J. M. Rothkrantz. Development of a Speech Recognizer for the Dutch Language. Proceedings of 7th Annual Scientific Conference on Web Technology, New Media, Communications and Telematics Theory, Methods, Tools and Applications (EUROMEDIA), 2002, 133-138.

42. Young, S., G. Evermann, T. Hain, D. Kershaw, G. Moore, J. Odell, D. Ollason, D. Povey, V. Valtchev, and P. Woodland. The HTK Book (for HTK Version 3.2.1), 2002.

43. Zweig, G. Speech Recognition with Dynamic Bayesian Networks. PhD Thesis, Computer Science Division, University of California at Berkeley, 1998.

44. Zweig, G. and M. Padmanabhan. Exact Alpha-beta Computation in Logarithmic Space with Application to Map Word Graph Construction. Proceedings of ICSLP'00, Beijing, China, 2000.

Manuscript received on 19.03.2010

Drs. dr. L. L. M. Rothkrantz *is Professor of Computer science at the Netherlands Defence Academy and at Delft University of Technology. He has a MSc degree in Mathematics and Psychology and received his PhD in Mathematics at the University of Amsterdam. His research interest are multimodal communication, signal processing and sensor technology.*

Contacts:
Netherlands Defense Academy
Mekelweg 4, 2628CD
Delft, The Netherlands
e-mail: l.j.m.rothkrantz@tudelft.nl

Dr. ir. ***Pascal Wiggers*** *received is Master Degree in Computer Science in 2001 and his PhD degree in 2008 from Delft University of Technology in the Netherlands. He currently is an assistant professor in the Man-Machine Interaction group of the faculty of Electrical Engineering, Mathematics and Computer Science at the same university.*
His research focuses on graphical models, modeling of context in multimodal recognition, multimodal fusion and emotion recognition.

Contacts:
Man-Machine Interaction Group
Delft University of Technology
Delft, The Netherlands
e-mal: p.wiggers@tudelft.nl