Model Construction Algorithms For Object Oriented Probabilistic Relational Models

Catherine Howard[1] and Markus Stumptner[2]

[1]Electronic Warfare and Radar Division
Defence Science and Technology Organisation
PO Box 1500, Edinburgh, South Australia, 5111
catherine.howard@dsto.defence.gov.au

[2]Advanced Computing Research Center
University of South Australia
Adelaide, South Australia, 5095
mst@cs.unisa.edu.au

Abstract

This paper presents three new algorithms for the automatic construction of models from Object Oriented Probabilistic Relational Models. The first two algorithms are based on the knowledge based model construction approach while the third is based on an Object Oriented Bayesian Network instance tree triangulation method. We discuss the strengths and limitations of each of the algorithms and compare their performance against the knowledge based model construction and Structured Variable Elimination algorithms developed for Probabilistic Relational Models.

1. Introduction

Our research focuses the use of automated reasoning techniques to produce battlespace situation assessments. These situation assessments provide dynamic decision support to tactical military commanders. Situation Assessments are defined as "persistent representations of the relationships of interest between objects of interest in the area of interest" in the battlespace (Lambert 2003). Such relationships of interest can include physical, temporal, spatial, organizational, perceptual and functional relationships.

Bayesian Networks (BNs) are a popular technique that have been used in many existing tactical military decision support systems to reason about causal and perceptual relationships between objects in the battlespace (e.g. Das, Grey and Gonslaves 2002). However, they have been shown to be inadequate for reasoning about large, complex domains (Pfeffer 1999; Wright 2002) because of their lack of flexibility and inability to take full advantage of domain structure or reuse. This lack of flexibility is of particular relevance to producing situation assessments because the variables relevant to reasoning about a situation are highly dependent on the domain and the user intentions. Object Oriented Probabilistic Relational Models (OPRMs) were recently developed to address some of the limitations of BNs in modelling complex military domains (Howard and Stumptner 2005a/b). This paper presents three new algorithms for the automatic construction of models from OPRMs. Section 2 outlines OPRMs and presents a simple university example, which is used in the discussion (Section 3) and evaluation (Section 4) of the model construction algorithms. We return to our real world application in Section 3.3, where we discuss the advantages and limitations of the various model construction algorithms for this application.

2. OPRMs

OPRMs (Howard and Stumptner 2005a) specify a template for the probability distribution over a knowledge base (KB), where a knowledge base is defined as consisting of a set of OPRM classes and instances, a set of inverse statements and a set of instance statements.

Definition: An Object-Oriented Probabilistic Relational Model, Ξ, is a pair (RC, PC) of a relational component RC and a probabilistic component PC. The RC consists of:

- A set of classes, $\mathbf{C} = \{C_1, C_2, \ldots, C_n\}$, and possibly a partial ordering over $\mathbf{C}$ that defines the class hierarchy;
- A set of named instances, $\mathbf{I} = \{I_1, I_2, \ldots, I_n\}$, which represent instantiations of the classes;
- A set of descriptive attributes, $\Delta_{\mathbf{C}} = \{\delta_1, \delta_2, \ldots, \delta_n\}$, for each class C in $\mathbf{C}$. Attribute δ_x of class C_1 is denoted $C_1.\delta_x$. Each descriptive attribute δ_x has a domain type $Dom[\delta_x] \in \mathbf{C}$ and a range type $Range[\delta_x]=Val[\delta_x]$ where $Val[\delta_x]$ is a predefined finite enumerated set of values, i.e., $Val[\delta_x] = \{Val_1, Val_2, \ldots, Val_n\}$;
- A set of complex attributes, $\Phi_{\mathbf{C}} = \{\phi_1, \phi_2, \ldots, \phi_n\}$, for each class C in $\mathbf{C}$. Attribute ϕ_x of class C_1 is denoted $C_1.\phi_x$. Complex attributes represent functional relationships between instances in the knowledge base. Each complex attribute ϕ_x has a domain type $Dom[\phi_x] \in \mathbf{C}$ and a range type $Range[\phi_x] \in \mathbf{C}$ for some class C in $\mathbf{C}$.

The PC consists of:

- A conditional probability model for each descriptive attribute δ_x, $P(\delta_x | \mathbf{Pa}[\delta_x])$ where $\mathbf{Pa}[\delta_x] = \{Pa_1, Pa_2, \ldots, Pa_n\}$ is the set of parents of δ_x. These probability models may be attached to particular instances or inherited from classes.

2.1 The University Domain Example

Our OPRM model is designed to evaluate the promotion prospects of university academics based upon their teaching skills, brilliance, productivity and the impact of their publications; the latter is affected by the standard and prestige of the conferences and is summarized by the node Aggregate(Papers). An aggregate attribute is a descriptive attribute that summarizes a property of a set of related instances (Howard and Stumptner 2005a). The model contains three classes, $\mathbf{C}$= {Lecturer, Paper, Conference}, shown in Figure 1. The set of descriptive attributes for the Lecturer class, for example, is $\Delta_{\mathbf{Lecturer}}$={Productivity, Tired, Brilliance, Teaching Skills, Aggregate(Papers), WillGetPromoted} while the set of complex attributes is $\Phi_{\mathbf{Lecturer}}$={Papers}.

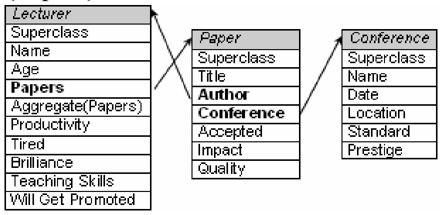


Figure 1. The classes and attributes of the university OPRM. These classes will be exactly the same in the equivalent PRM.

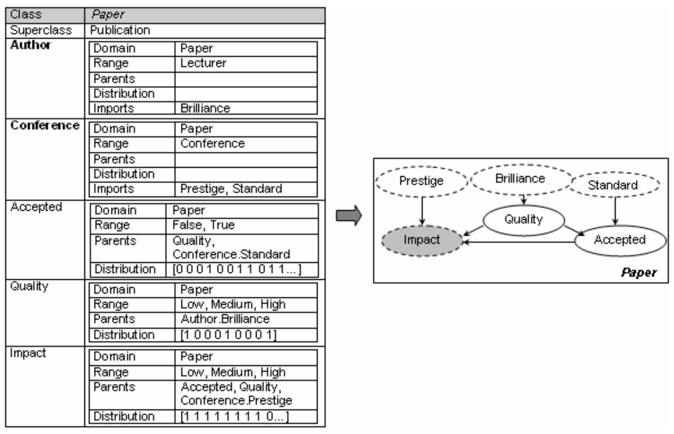
The model is the simplest form of OPRM, where the complete relational structure is known. This means that the set of objects and the set of potential relationships between them are known and there is no uncertainty about the structure of the model. Given this relational structure, the OPRM specifies a probability distribution over the attributes of the instances of the model. The unique names assumption is employed which means that each object in the knowledge base is assumed to have a unique identifier (i.e. there is no uncertainty about the identity of the objects).

2.2 OPRMs versus PRMs

OPRMs follow the principles of Probabilistic Relational Models (PRMs), first developed by (Koller and Pfeffer 1998) and later refined by (Getoor 2002; Pasula 2003). Both PRMs and OPRMs integrate probabilistic information with a frame-based representation system. In both languages, inference is performed by dynamically constructing the 'equivalent' Object Oriented Bayesian Networks (OOBNs) from the probabilistic information contained in the frames. For example, Figure 2 shows the BOOBN structure generated for the Paper class. An OOBN is defined as a BN fragment containing *output*, *input*, and *encapsulated* nodes. The input and output variables form the *interface* of the class. The interface encapsulates the internal variables of the class, d-separating them from the rest of the network (Bangsø 2004).

The key difference between PRMs and OPRMs is the type of OOBN constructed from the probabilistic information. PRM algorithms construct a Koller/Pfeffer OOBN (KPOOBN) (Koller and Pfeffer 1997) while OPRM algorithms construct a Bangsø OOBN (BOOBN) (Bangsø 2004).

The main difference between the two OOBN frameworks is that BOOBNs introduce the use of *reference nodes* and *reference links* to overcome the problem that no node inside a class can have parents outside the class. A reference node is a special type of node that points to a node in another scope (called the *referenced* node). A reference node is bound to its referenced node by a reference link. In the BOOBN framework, all input nodes are reference nodes.



Class	*Paper*	
Superclass	Publication	
Author	Domain	Paper
	Range	Lecturer
	Parents	
	Distribution	
	Imports	Brilliance
Conference	Domain	Paper
	Range	Conference
	Parents	
	Distribution	
	Imports	Prestige, Standard
Accepted	Domain	Paper
	Range	False, True
	Parents	Quality, Conference.Standard
	Distribution	[0 0 0 1 0 0 1 1 0 1 1...]
Quality	Domain	Paper
	Range	Low, Medium, High
	Parents	Author.Brilliance
	Distribution	[1 0 0 0 1 0 0 0 1]
Impact	Domain	Paper
	Range	Low, Medium, High
	Parents	Accepted, Quality, Conference.Prestige
	Distribution	[1 1 1 1 1 1 1 1 0...]

Figure 2. The BOOBN structure generated for the Paper class.

These reference nodes provide several important benefits. Firstly, the use of reference nodes means that the interface of an OPRM class is fully specified once the class is defined. The interface for a class C, $\kappa_{\mathbf{C}}$, is the set of input attributes, $\alpha_{\mathbf{C}}$, and the set of output attributes, $\beta_{\mathbf{C}}$, $\kappa_{\mathbf{C}}=\alpha_{\mathbf{C}}\cup\beta_{\mathbf{C}}$. In the university example, $\kappa_{\mathbf{Paper}}$= {Brilliance,Standard,Prestige,Impact}. Since KPOOBNs do not use reference nodes, a PRM class does not have a single, clearly defined interface. The interface depends on how other objects refer to it in the particular query under consideration. For example, the Imports facet of the Papers attribute in the Lecturer class defines the Paper attributes the Lecturer class has access to, namely Paper.Impact. However, from within Lecturer, it is not known how Paper depends on the Lecturer (or any other) class. This information is available only within the Paper class. Figs 3 and 4 illustrate this difference between OPRMs and PRMs.

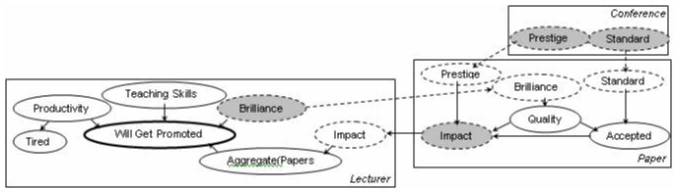


Figure 3. The university OPRM class model showing the probabilistic (solid) and reference (dashed) relationships. Gray dashed nodes are output nodes, dashed nodes are input nodes.

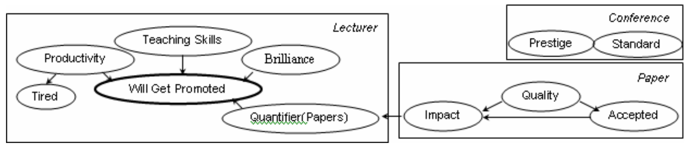


Figure 4. The equivalent PRM for the university domain. This diagram is drawn from the point of view of the Lecturer class and as such, it is not known how other classes depend on attributes from the Lecturer class.

The fact that a PRM class does not have a single, clearly defined interface means that the model construction algorithms used with PRM knowledge bases need to determine the interfaces of the classes and instances on the fly (Pfeffer 1999; Pfeffer et al 1999).

Secondly, the use of reference nodes enables the BOOBN framework to have a more intuitive definition of inheritance for the modeling domain. The KPOOBN inheritance definition corresponds to *contravariance* while the BOOBN definition corresponds to *covariance*.

3. OPRM Model Construction Algorithms

We will first describe Model Construction using the following conventions. The general format of an attribute name is $I.\gamma.\delta$, where I is the instance, γ is a possibly empty slot chain that consists of complex attributes, and δ is the descriptive attribute at the end of the slot chain. The elements of a slot chain will be members of Φ_C. For descriptive attributes, $\gamma = \varnothing$ and for complex attributes, $\gamma \neq \varnothing$. Take as an example an instance Paper[1] of the Paper class in Figure 2. Using this notation, the descriptive attribute Paper[1].Accepted of this instance breaks down into I = Paper[1], $\gamma=\varnothing$ and δ=Accepted, while the complex attribute Paper[1].Author.Brilliance breaks down into I = Paper[1], γ=Author and δ=Brilliance. In the case where we know the attribute to be a complex attribute, we replace the $I.\gamma.\delta$, notation with $I.\phi.\delta$. Finally, the range of the attribute $I.\gamma.\delta$ is denoted by $VT[I.\gamma.\delta]$

In the following, we denote the set of encapsulated attributes by $\xi=\Phi-\kappa=\{\xi 1, \xi 2,\ldots, \xi n\}$, the set of aggregate attributes by $\varepsilon=\{\varepsilon 1, \varepsilon 2,\ldots, \varepsilon n\}$, and the set of indirect attributes by $\varpi=\{\varpi 1, \varpi 2,\ldots, \varpi n\}$. The selector attribute for a given indirect attribute $Ix.\phi$ is denoted by $Ix.sel(\phi)$, and the number attribute over a multi-valued $Ix.\phi x$ by $Ix.num(Ix.\phi x)$. An instance statement in the KB about attribute ϕ of instance I is denoted by $SI(I.\phi)$, and the set of instance statements for instance I is denoted by S(I).

3.1 The Knowledge Based Model Construction Algorithms

3.1.1 Query Dependant KBMC Algorithm

At design time, the dependency model for each class is specified. In general terms, at run time, starting with the list of query variables, the KBMC algorithm uses the knowledgebase to backward-chain along the dependency relationships to construct a flat Bayesian Network on which it performs inference using a junction tree algorithm. More specifically, the algorithm maintains a list of nodes to be processed (*nodes*), which initially contains the set of query variables $\sigma=\{\sigma_1, \sigma_2,\ldots, \sigma_n\}$. During each iteration, the algorithm removes the first node, n, from *nodes* (1) and adds this node to the list of nodes for the complete model, *nodes*$_N$. The algorithm then creates a node, n_{Pa}, for each parent pa of n (2). When a node is created, its range, its parents and its CPD must be specified. If n_{Pa} is a descriptive attribute, it is simply added to *nodes* (10). If n_{Pa} is a complex attribute (i.e. of the form $I_x.\phi.\delta$), the knowledge base instance statements are searched to find any instance statements relating to attribute $I_x.\phi$. If $I_x.\phi$ is assigned a named instance I_y in the instance statements, (for example, in the university model, Paper[1].Author=Gump), I_y is assigned to I_z (3). If there is no named instance in the KB, a generic, unnamed instance, I_g, is created and added to the KB. This generic instance is assigned to I_z (4). A node is created for $I_z.\delta$ and added to *nodes* (5). In cases where n_{Pa} is an indirect or aggregate attribute, n_{Pa} is added to *nodes* (7,9) after a complete list of parents has been generated (6,8). How this parent list is generated depends on the type of n_{Pa}.

```
Algorithm KBMC($\sigma$,**KB**)
  Initialize nodes$\leftarrow\sigma$
  **while** nodes$\neq\varnothing$ **do**
(1)    n=first(nodes)
       nodes$_N\leftarrow$nodes$_N\cup$n  /*node list for complete model */
       nodes$\leftarrow$nodes$-$n
       **forall** Pa$\in$**Pa[n] do**  /* for all parents of n */
           **if** Pa$\notin$(nodes$\cup$nodes$_N$) **then**
(2)            create node $n_{Pa}$ for Pa, denoted $I_x.\gamma.\delta$
               **if** $\gamma\neq\varnothing$ and $I_x.\phi\notin\varpi$ **then**
                   **if** $\exists S_I(I_x.\phi)$ in KB | $I_x.\phi\leftarrow I_y$ in $S_I(I_x.\phi)$ **then**
(3)                    $I_z\leftarrow I_y$
                   **else**
                       create $I_g$, where $VT[I_g]=VT[I_x.\phi]$
                       add $I_g$ to KB
(4)                    $I_z\leftarrow I_g$ **end if**
                   **if** $I_z.\delta\notin$nodes and $I_z.\delta\notin$nodes$_N$ **then**
                       create node $n_z$ for $I_z.\delta$
(5)                    nodes$\leftarrow$nodes$\cup n_z$ **end if**
               **else if** $\gamma\neq\varnothing$ and $I_x.\phi\in\varpi$ **then**
                   **Val**[I.sel($\phi$)]=VT[I.sel($\phi$)]
(6)                /* generate **Pa**[$n_{Pa}$] */
                   **forall** Val$\in$**Val**[I.sel($\phi$)] **do**
                   **if** Val$\in$**I then**
                           let $I_y$ denote Val
                           **Pa**[$n_{Pa}$]$\leftarrow$**Pa**[$n_{Pa}$]$\cup I_y.\delta$ **end if**
                       **if** Val$\in$**C then**
                           let C denote Val
                           create $I_g$ and add $I_g$ to KB
                           **Pa**[$n_{Pa}$]$\leftarrow$**Pa**[$n_{Pa}$]$\cup I_g.\delta$ **end if**
                   **end forall**
(7)                nodes$\leftarrow$nodes$\cup n_{Pa}$
               **else if** $\gamma=\varnothing$ and $I_x.\delta\in\varepsilon$ **then**
                   $I_x.\delta$ is an aggregate attribute over $I_x.\phi_x$
                   **if** $\exists$ number uncertainty **then**
                       $n_{max}=max(VT[I_x.num(I_x.\phi_x)])$ **end if**
(8)                /* generate **Pa**[$n_{Pa}$] */
                   **if** $\exists S_I(Pa[n_{Pa}])$ in KB **then**
                       let $\mathbf{I_y}$ denote the instances |
                       $Pa[n_{Pa}]\leftarrow I_y$ in $S_I(Pa[n_{Pa}])$
                       $n_{actual} = |\mathbf{I_y}|$
                       **forall** $I_y\in\mathbf{I_y}$ **do**
                           **Pa**[$n_{Pa}$]$\leftarrow$**Pa**[$n_{Pa}$]$\cup I_y.\delta$ **end forall**
                       **if** $\exists$number uncertainty$\vee n_{max}>n_{actual}$ **then**
                           **for** i =m+1 **to** n **do**
                               create instance $I_g$
                               where $VT[I_g]=VT[I_x.\phi]$
                               add $I_g$ to KB
                               **Pa**[$n_{Pa}$]$\leftarrow$**Pa**[$n_{Pa}$]$\cup I_g.\delta$ **end for**
                       **end if end if**
(9)                nodes$\leftarrow$nodes$\cup n_{Pa}$
               **else if** ($\gamma=\varnothing$and $I_x.\delta\notin\varepsilon$) **then**
(10)               nodes$\leftarrow$nodes$\cup n_{Pa}$ **end if**
       **end if end forall end while**
   Create Bayesian Network B from nodes$_N$
**end** KBMC
```

Algorithm 1: OPRM KBMC algorithm.

As expected, this algorithm is very similar to the PRM KBMC algorithm. However, there are two important differences. Firstly, this algorithm is query specific while the PRM KBMC algorithm is not. The second difference involves the way that complex attributes are handled. In the PRM KBMC algorithm, when a complex attribute Ix.ϕ.δ is encountered, a node Iz.δ is created and added to the Bayesian network as a parent of the complex attribute Ix.ϕ.δ and the CPD of Ix.ϕ.δ is set to reflect this relationship. Iz is either a named instance Iy (if there exists an instance statement in KB for such that Ix.ϕ is assigned the value Iy) or a generic instance Ig where VT[Ig]=VT[Ix.ϕ]. In the BOOBN framework however, reference nodes by definition cannot have parents. Therefore in the OPRM KBMC algorithm, a node is created for Iz.δ and this node is added to the network and the Ix.ϕ.δ node is not. Figure 5 provides an example of the resulting network. As will be seen in Figure 7, this technique results in fewer nodes in the model.

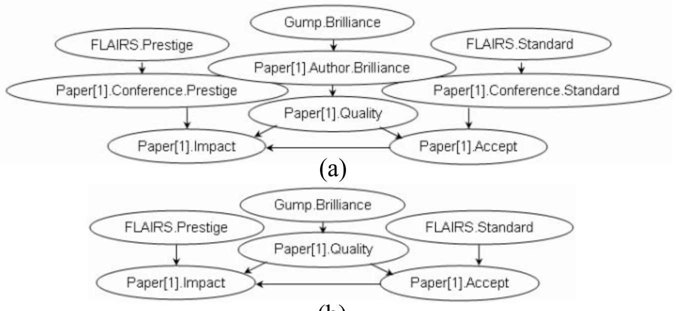


Figure 5. An example of part of the network produced by the (a) PRM KBMC algorithm and (b) OPRM KBMC algorithm where FLAIRS is the conference to which Paper[1] is submitted and Gump is the author of the paper.

3.1.2 Query Independent KBMC Algorithm

The objective driving the modeling process during our real world application of deriving situation assessment is to determine which of the relationships of interest to the user hold at any given time given the observations and a priori data. The overall task of producing situation assessments is relatively constant, but the set of concepts relevant to performing this task varies dynamically. The information in the knowledgebase flows directly from the observations. That is, for our application, most of the information we need to represent and reason about will be in the form of instances in the KB. Therefore we require that the model construction algorithms must operate over the KB to dynamically construct a situation specific model based on the observations. A query driven construction process is clearly not appropriate for this application.

The OPRM KBMC algorithm can be made query independent by changing the initial value of *nodes* to be the set of all descriptive attributes from all instances in the KB i.e. $\Delta_{KB}=\{\Delta_{I1,}\Delta_{I2,\ldots}\Delta_{In}\}$. Using this initial list, the algorithm produces a *situation* rather than *query* specific model.

3.2 The Junction Tree Construction Algorithm

At design time, the dependency model for each class is specified and a BOOBN is created for each class. This network is then translated directly into a 'local' junction tree. An interface clique is created, which consists of all nodes in the class's interface,κ. This interface clique is connected into the local junction tree and any loops created during this process are removed. Thus at design time, each class is 'precompiled' into a local junction tree and these local junction trees are stored in a cache. At run time, whenever an instance of a class is created, the appropriate local junction tree is instantiated. The KB is searched for any instance statements applicable to the instance under consideration and any required corrections to the local junction tree are made. A root clique for the model is created which contains all the nodes in all the instances interfaces. Each local instance junction tree is then connected to the root clique to create the 'global junction tree'. Inference can then be performed using this global junction tree.

We denote the unrooted junction tree for class C by JT_{UC}, the rooted junction tree for class C by JT_{RC}, the set of cliques for the class C by $\Omega_C=\{\omega_1, \omega_2,\ldots, \omega_n\}$, and correspondingly use Ω_{UC} for the set of cliques in the JT_{UC}, Ω_{RC} for the set of cliques in the JT_{RC} and $\Omega_{KB}=\{\Omega_{C1}, \Omega_{C1},\ldots, \Omega_{Cn}\}$ for the set of cliques in the KB. The interface clique for the class is denoted by ω_κ and the root clique for the KB by ω_R.

```
Algorithm JC(σ,KB)
Design Time
forall $C \in \mathbf{C}$ do
  $nodes_C \leftarrow \Delta_C \cup \kappa_C$
  Create Bayesian Network $B_C$ from $nodes_C$
  Create $JT_{UC}$ from $B_C$; Create $JT_{RC}$ from $JT_{UC}$
  $\Omega_{RC} \leftarrow \Omega_{UC}$ and $JT_{RC} \leftarrow JT_{UC}$
  $\omega_\kappa=\{n|n\in\kappa\}$
  $\Omega_{RC} \leftarrow \Omega_{RC} \cup \omega_\kappa$
  Calculate $d_{ij} \mid d_{ij}=1 \Leftrightarrow$ node j occurs in clique i
  forall $\omega_x \in \Omega_{RC} \mid \omega_x \neq \omega_\kappa$ and $(d(\omega_x,:) \cap d(\omega_\kappa,:)) \neq \emptyset$ do
      /* where $d(\omega_x,:)$ denotes all nodes in clique $\omega_x$ */
      $JT_{RC}(\omega_{\kappa,}\omega_x)=1$ and $JT_{RC}(\omega_{x,}\omega_\kappa)=1$ end forall
  Compute separator matrix $\psi$ for $JT_{RC}$ where
          $\psi_{xy}=d(\omega_x,:) \cap d(\omega_y,:)$
  if $\neg$acyclic($JT_{RC}$) then remove loops end if
end forall
Runtime
forall $I \in \mathbf{I}$ do
  $nodes_I \leftarrow nodes_C$ where C=VT[I], $JT_{RI}=JT_{RC}$ and $\Omega_I=\Omega_C$
  if $S_I(I) \neq \emptyset$ then
      forall $S \in \mathbf{S_I}(I)$ where $S=S_I(I.\phi)$ do
          if $\exists S_I(I.\phi)$ in KB then
              let $\mathbf{I_y}=\{I_{y1,}I_{y2,\ldots}I_{yn}\}$ denote the instances |
              $I.\phi \leftarrow I_y$ in $S_I(I.\phi)$
              forall $I_y \in \mathbf{I_y} \mid I.\phi.\delta \in nodes_I$ and $I_y.\delta \notin nodes_I$ do
                  create node for $I_y.\delta$
                  $nodes_{NN} \leftarrow nodes_{NN} \cup I_y.\delta$
                  forall $\omega \in \Omega_I \mid I.\phi \in \omega$,
                  forall $\psi_{xy} \in \psi \mid I.\phi \in \psi_{xy,}$
                  forall $x \in nodes_I \mid I.\phi.\delta \in \mathbf{Pa}[x]$ do
                      replace $I.\phi.\delta$ with $nodes_{NN}$ end forall
                  $nodes_I \leftarrow nodes_I - I.\phi$
                  $nodes_I \leftarrow nodes_I \cup nodes_{NN}$
              end forall end if end forall end if
  $nodes_{KB} \leftarrow nodes_{KB} \cup nodes_I$
  $\Omega_{KB} \leftarrow \Omega_{KB} \cup (\Omega_I - \omega_\kappa)$
  $\omega_R \leftarrow \omega_R \cup \kappa_I$
end forall
$\Omega_{KB} \leftarrow (\Omega_{KB} \cup \omega_R)$
Create $\psi_{KB}$ from $\psi_I$
end JC
```

Algorithm 2: OPRM Junction Tree Construction algorithm.

3.3 Real-World Application and Discussion

There are two facets to the structure provided by the OPRM language that model construction algorithms should exploit. They should exploit the structure of the domain by exploiting the fact that the internal state of the classes/instances are encapsulated from the remainder of the network via their interface. Algorithms should also facilitate reuse where possible.

The first limitation of the OPRM KBMC algorithm is that while the OPR models take advantage of the structure of the domain, the KBMC algorithm does not, nor does it facilitate reuse. All structure gained by using OPRMs to represent observations and prior knowledge is lost as soon as this information is translated into a flat BN.

The second limitation is that the algorithm is query driven. If the set of query variables changes, the entire model construction process must be run again and a different model will result. While this may not be detrimental to the current university example, it can be a critical problem for our real world application domain of deriving situation assessments from observations. This application is data driven.

As discussed in Section 3.1.2, the OPRM KBMC algorithm can be made query independent. However, as shown in Figure 6, depending on the model and the query, the query specific algorithm can significantly outperform the non-query specific algorithm.

To solve a query on class C, using the KB, the PRM Structured Variable Elimination (SVE) algorithm constructs a local BN for C consisting of a node for each attribute of C in addition to special output and projection nodes. When the algorithm comes across a complex attribute in C (of type C′), it eliminates it by performing a recursive call that generates a temporary local flat BN for the class C′ and applies standard variable elimination techniques to compute the factor over the query variables of class C′, given the input variables. This factor is then used in the BN representing class C.

By taking this structured approach to the elimination of complex attributes in classes, SVE takes advantage of the structure of the domain. However, when dealing with instances, Pfeffer et al reason that there may be situations where the instance interfaces no longer encapsulate the protected attributes from the rest of the network, which means that the recursive technique cannot be used. For all instances, SVE copies the instance information into a top level object T in the knowledgebase and uses this object to construct one flat BN containing all attributes of all the instances using a backward chaining algorithm. Inference is then performed using this flat BN. By copying this information into a flat structure, SVE fails to take advantage of the structure of the domain for instances. And while there is a cache to help reuse of computations for classes, there is no mechanism for reuse for instances.

SVE produces a query specific network and as such the model construction time is dependant on the query asked (simpler queries result in shorter model construction times) and because of its recursive nature, a complete model of the situation never exists.

For our application domain, the OPRM JC algorithm has several advantages. Firstly it produces a situation rather than query specific model. Secondly, unlike SVE, even when constructing models involving mainly instances, the algorithm exploits the structure of the domain and facilitates reuse of the class models. The main disadvantage of the ORPM JTC algorithm, as seen in Fig 9b, is that its model construction times are longer than SVEs for the same number of nodes in the network.

4. Experimental Results

The OPRM presented in Section 3 and the equivalent PRM were used to evaluate the performance of the query specific and non-query specific OPRM KBMC algorithms and the OPRM JC algorithm against each other and the PRM KBMC and SVE algorithms (Figures 6-9). The knowledgebase consisted of one Lecturer instance: Lecturer Gump and a varying number of Paper and Conference instances. The query variable used in most comparisons was Gump.WillGetPromoted. However in the comparison of the query specific and non-query specific OPRM KBMC algorithms the Paper[1].Impact query was used. Because the Gump.WillGetPromoted query involves nearly all the attributes in the university OPRM, the difference in performance of the two algorithms for this query was minimal. However, for the Paper[1].Impact query, regardless of the number of instances in the knowledge base, the query specific algorithm produced a BN with only six nodes (Gump.Brilliance, FLAIRS.Standard, FLAIRS.Prestige, Paper[1].Quality, Paper[1].Accepted, Paper[1].Impact) while the number of nodes in model produced by the non-query specific algorithm depended on the number of instances in the KB.

For all algorithms, the model construction time measured the time taken for the algorithm to complete its model construction process and did not include the time taken to perform inferencing. For example, the SVE algorithm, which is divided into the initialization, factor construction and variable elimination phases, was timed from the beginning of the initialization phase to the end of the factor construction phase.

The first plot in Figure 8 shows the number of instances versus the number of nodes (SVE x;OPRM +) and number of factors (o) produced by the algorithms. While SVE produces a node for every attribute of the class or instance, it calculates factors only for those attributes of direct relevance to the query, so in order to compare its performance against the OPRM algorithms, we have plotted Model Construction Time versus the number of Factors, rather than the number of nodes. The number of factors produced by the SVE algorithm is the same as the number of nodes produced by the OPRM JTC and OPRM KBMC algorithms.

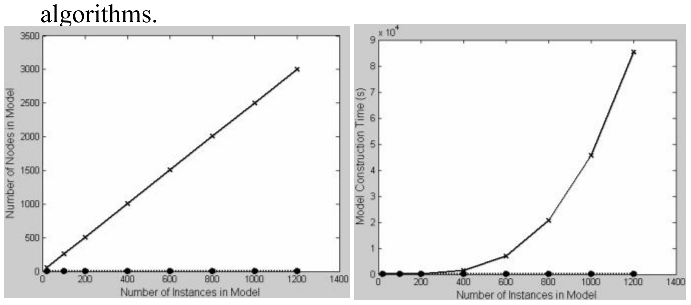


Figure 6. Comparison of the query specific (circles) and non-query specific (crosses) OPRM KBMC algorithms.

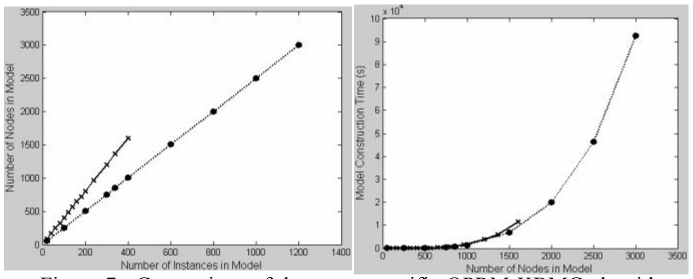


Figure 7. Comparison of the query specific OPRM KBMC algorithm (circles) with the PRM KBMC algorithm (crosses).

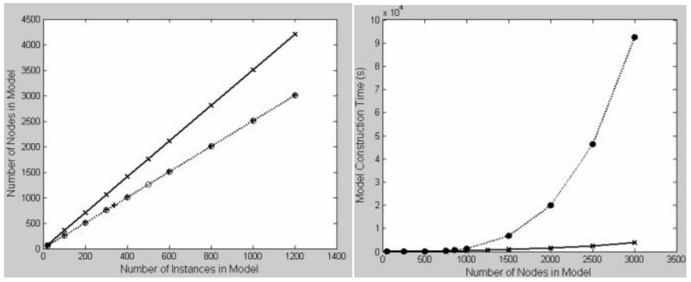


Figure 8. Comparison of the query specific OPRM KBMC algorithm (circles) with the PRM SVE algorithm (crosses).

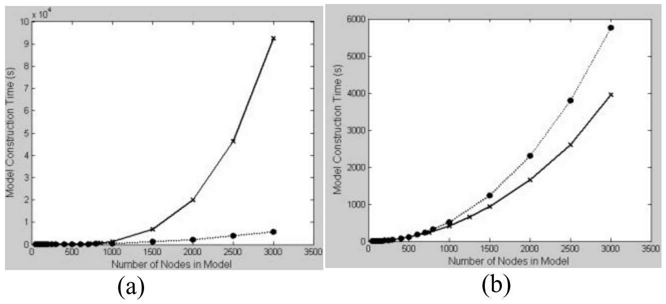

(a) (b)

Figure 9. Comparison of the OPRM JC algorithm (circles) with the (a) query specific OPRM KBMC algorithm (crosses) and (b) PRM SVE algorithm (crosses).

All algorithms were implemented in MATLAB and executed using a Pentium III, 1.2GHz. To illustrate the differences between a query specific and situation specific approach, consider running the following group of queries: σ={Paper[1].Impact}, σ={Paper[2].Impact} up to σ={Paper[100].Impact} in succession on a knowledgebase consisting of one Lecturer instance and 100 Paper and Conference instances. The time taken for the OPRM JC algorithm to construct a model on which it can perform these queries is 122.48 seconds, while the time taken for the SVE algorithm to construct its models to answer the same queries (as the query variable changes each time, the SVE algorithm will construct a different model for each query) is 8705 seconds.

5. Conclusions

Based on our OPRM language that extends prior work on OOBNs, we have presented three model construction algorithms for OPRMs and compared their performance with the model construction algorithms developed for PRMs. While the SVE algorithm outperforms the OPRM JC algorithm for individual larger models, the JC algorithm has several advantages over SVE for our application domain, in particular the production of a model that is reusable for multiple queries, and OO style reuse of class models. As the most expensive part of the SVE algorithm is in the variable elimination phase, which was not part of the model construction time measurements, we aim to measure the time taken to perform inference for both the algorithms in future experiments. We also aim to examine a number of extensions such as the use of approximate algorithms for very large models.

6. References

Lambert, D.A. 2003. Grand Challenges of Information Fusion. In *Proc. 6th Intl Conf on Information Fusion*.

Das, S., R. Grey, and P. Gonsalves. 2002. Situation Assessment via Bayesian Belief Networks. in *Proc. 5th Intl Conf on Information Fusion*. Annapolis, MD, USA.

Pfeffer, A.J. 1999. Probabilistic Reasoning for Complex Systems. Ph.D. diss, Stanford University.

Wright, E., et al. 2002. Multi-Entity Bayesian Networks for Situation Assessment. In Proc. 5th Intl Conf on Inf. Fusion.

Howard, C. and M. Stumptner. 2005a. Situation Assessments Using Object Oriented Probabilistic Relational Models. In *Proc. 8th Intl Conf on Information Fusion*. Philadelphia, USA.

Howard, C. and M. Stumptner. 2005b. Situation Assessment with Object Oriented Probabilistic Relational Models. In *Proc. 7th Intl Conf on Enterprise Information Systems*.

Koller, D. and A. Pfeffer. 1998. Probabilistic Frame-Based Systems. In *Proc. AAAI-98*. Madison, Wisconsin.

Getoor, L. 2002. Learning Statistical Models From Relational Data. Ph.D. diss, Stanford University.

Pasula, H.M. 2003. Identity Uncertainty. Ph.D. diss, Dept. of Computer Science, UC Berkeley.

Bangsø, O. 2004. Object Oriented Bayesian Networks. PhD diss, Dept of Computer Science, Aalborg University.

Koller, D. and A. Pfeffer. 1997. Object-Oriented Bayesian Networks. In *Proc. 13th Conf. on Uncertainty in Artificial Intelligence (UAI-97)*. Providence, Rhode Island.

Pfeffer, A., et al. 1999. SPOOK: A System for Probabilistic Object Oriented Knowledge Representation. In *Proc. 15th Conf. on Uncertainty in Artificial Intelligence (UAI-99)*.