Averaged Probabilistic Relational Models

Daniel Wright

June 3, 2002

Abstract

Most real-world data is stored in relational form. In contrast, most statistical learning methods work with "flat" data representations, forcing us to convert our data into a form that loses much of the relational structure. The recently introduced framework of *Probabilistic Relational Models* (PRMs) allows us to represent probabilistic models over multiple entities that utilize the relations between them. However, for extremely large domains it may be impossible to represent every object and every relation in the domain explicitly. We propose representing the domain as an *Averaged PRM* using only "schema-level" statistical information about the objects and relations, and present an approximation algorithm for reasoning about the domain with only this information. We present experimental results showing that interesting inferences can be made about extremely large domains, with a running time that does not depend on the number of objects.

1 Introduction

Bayesian Networks have been shown to be a powerful tool for representing statistical patterns in real-world domains. They exploit the conditional independencies between variables in the domain to reduce the size of the representation, and provide this compact representation in a form that is easy to work with.

However, many real-world domains are highly structured, and are typically represented in relational models. For these domains, *Probabilistic Relational Models (PRMs)* (Koller & Pfeffer[4]) are more suitable. These models extend the standard attribute-based Bayesian network representation to incorporate a much richer relational structure. They allow properties of any entity to depend probabilistically on properties of other *related* entities. The model represents a generic dependence for a *class* of objects, which is then instantiated for particular sets of entities and relations between them.

The PRM framework assumes that the relational structure - the relational links between entities - is background knowledge. This assumption requires that the domain be small enough that all objects can be enumerated, and thus their relational structure can be provided in a database.

Getoor *et al.* [1] provide a framework for specifying a probabilistic model of the relational structure in addition to the attributes of objects in the domain. However, their approach still assumes that the domain is small enough to be explicitly enumerated. Thus, in order to reason with the model, they produce a Bayesian network that is at least as large as the domain is, and use existing techniques for Bayesian networks.

Notice, however, that in many cases we as humans are able to reason about extremely large domains without considering each object individually. Consider the following example: in a hypothetical university we have students who can be enrolled in courses and professors who can teach courses. We can reason that making one Computer Science course harder will increase the chance that students majoring in Computer Science (and thus likely to be taking that course) will be stressed, and thus that they will increase the pressure they put on the professors teaching the other courses that they are taking. Since most of the courses they are taking are probably Computer Science courses, which are probably taught by professors in the Computer

Science department, the net result is to increase the likelihood that Computer Science professors are highly stressed.

Here, without data about exactly which students take which classes and without enumerating each student or class, we are still able to make some useful inferences, at, essentially, a schema level. The key insight here is that, while reasoning, we do not differentiate between individual students, and thus can reason about them as a group. Intuitively, since we have no data to differentiate between students, we expect that our reasoning will come to the same conclusions for each one. Thus, while we might be able to make inferences such as "a student is more likely to be stressed if we know that he is a Computer Science major" that apply equally to all students in our domain, we cannot say that student A is more likely to be stressed than student B. Thus, since all our information is symmetric in the students, we can efficiently reason about all those students as a group.

Regular PRMs do not enforce this symmetry in the representation. Indeed, they intend for different data to be provided for each object, and are thus not suitable for this kind of inference.

Our goal is to create a language that can describe our knowledge about a relational domain such as the hypothetical university in the form of a probability distribution over all possible databases conforming to a schema and allow efficient reasoning with this distribution. It is important that this language enforce this symmetry of information about groups of objects, so that individual objects in the groups need never be enumerated.

2 Bayesian Networks

At its core, the problem we are trying to solve amounts to representing a joint probability distribution over a finite set of random variables $A_1, ..., A_n$, and performing inference on that distribution. The joint distribution over a set of variables is exponentially large in the number of variables, but we can utilize the structure of the domain to represent it more efficiently. This general problem has been well studied, and we will build on existing methods, specifically Bayesian Networks, to solve this particular instance of the problem.

A Bayesian Network [6] provides a (possibly empty) set of parent variables (Pa(A_i)) for each variable A_i. Then, it provides a *conditional probability distribution (CPD)* for each variable A_i given its parents. This CPD can be viewed as a function that, given an assignment of values to Pa(A_i), returns a probability distribution over the domain of A_i.

Then, to find the probability of a particular assignment of values to all of the variables $A_1 = a_1, ..., A_n = a_n$, we take the product for each i of the probability that $A_i = a_i$ provided by the CPD of A_i, given the assignment to Pa(A_i).

For any desired joint distribution over $A_1, ..., A_n$, it is possible to create a Bayesian Network that represents the distribution, but the network could itself be exponentially large. However, in real-world domains it is usually possible to exploit independence properties between the variables to find a compact representation of the distribution as a Bayesian Network.

Figure 1 shows a simple example of a Bayesian Network, including the CPDs in tabular form. In this sample domain, a man is at work deciding whether to go home to check if his house has been burgled. The house has an alarm which may activate with probability 0.8 if the house is burgled, and if the alarm activates his neighbor would call him at work with probability 0.7. However, if there is an earthquake, it could also trigger the alarm, with probability 0.3, but if there is an earthquake he will probably hear about it over the radio. There is also a small chance that the alarm malfunctions and activates with no apparent cause, or the house could be burgled even if there is an earthquake, giving a very high chance that the alarm activates.

This network represents a full joint probability distribution over the five variables. For example, one instantiation of these variables may be that there is no earthquake and no radio call, but there is a burglary and the alarm does activate, and the neighbor does call. The probability of this instantiation occurring is:

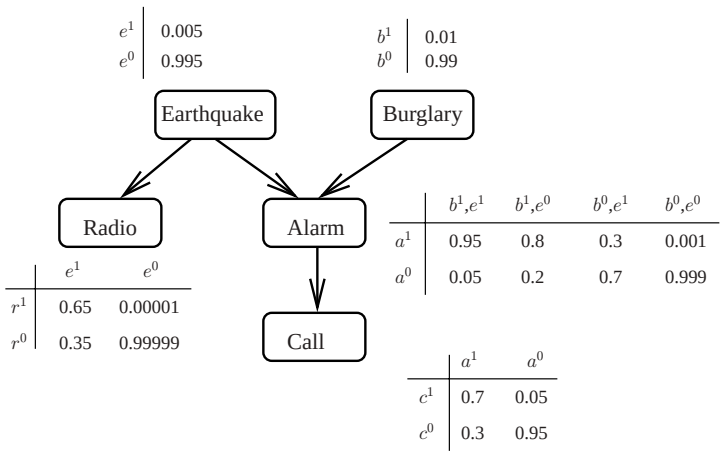

Figure 1: Alarm Bayesian Network

$$P(\text{Earthquake} = e^0) \times P(\text{Burglary} = b^1) \times P(\text{Call} = c^0 \mid \text{Earthquake} = e^0) \times$$
$$P(\text{Alarm} = a^1 \mid \text{Burglary} = b^1, \text{Earthquake} = e^0) \times P(\text{Call} = c^1 \mid \text{Alarm} = a^1)$$
$$= 0.995 \times 0.01 \times 0.99999 \times 0.8 \times 0.7$$

Usually, however, we will want to perform more complicated queries than simply asking the probability of a specific instantiation of all variables. The specific types of query we will consider in this paper are all of the following form: provide evidence of the state of some subset of the variables, and query the probability distribution of one of the remaining variables, given the evidence. We will allow this evidence to be "soft evidence" - that is, evidence of a specific variable is in the form of a probability distribution over that variable's domain.

A naive approach to answering these queries is to construct the full joint distribution from the Bayesian Network, apply the evidence to that distribution and sum out the result over the domain of the queried variable. Of course, as this approach does not take advantage of Bayesian networks' compact representation, it is intractable in even small networks. There are far more efficient algorithms for computing the exact result of such queries [6], allowing tractable exact inference in moderate sized networks. However, in general, exact inference in Bayesian networks is NP-hard, and for the types of networks we will be dealing with in this thesis, exact inference is not tractable. Instead, we must use approximate inference algorithms. The algorithm we will use in this thesis is known as loopy belief propagation (LBP).

3 Loopy Belief Propagation (LBP)

LBP can be used to approximate the result of queries on a Bayesian network in nearly linear time. Recent work by Yedidia *et al*. [3] provides some theoretical results justifying LBP as an approximation algorithm.

To perform LBP, we use a standard conversion to a pairwise Markov network. For each variable A in the Bayesian Network, we have two corresponding nodes in the pairwise Markov network – we have a "variable node" denoted by $v(A)$ whose domain is the same as that of the variable, and a "compound node" denoted by $c(A)$ whose domain is the joint domain of all of the node's parents. These nodes are connected by an edge containing the variable's CPD as a potential. For each of A's parents (say, B), we have an

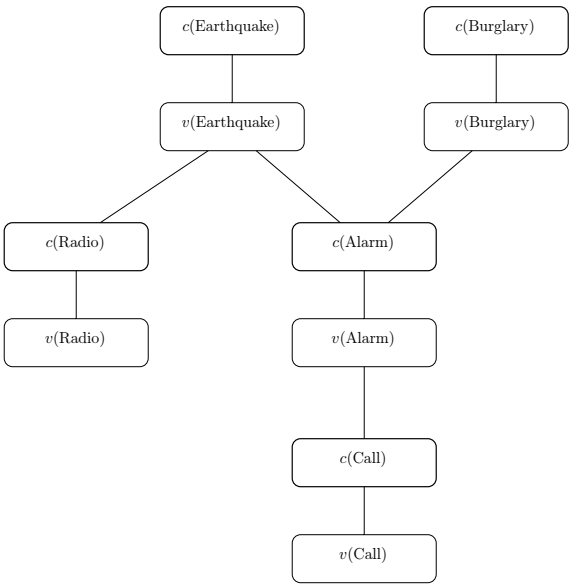

Figure 2: Pairwise Markov Network for Alarm Bayesian Network

edge between $v(B)$ and $c(A)$ with the edge potential simply being the identity mapping between the two domains. Figure 2 shows the conversion of the Alarm Bayesian network into a pairwise Markov network.

Next, we introduce messages along each edge in both directions which work as follows. If nodes C and D are connected by an edge, there is an outgoing message from node C to node D which is a factor over the domain of C, which we will refer to as $o_{C \to D}(C)$, and an incoming message (also from C to D) which is a factor over the domain of D, and which we will refer to as $i_{C \to D}(D)$. These messages are simply related by the potential on the edge between C and D (the potential is a factor over the joint domain of C and D). $i_{C \to D}(D)$ is simply $o_{C \to D}(C)$ multiplied by the potential and summed out over C. Since edges are undirected, there are also the messages $o_{D \to C}(D)$ and $i_{D \to C}(C)$. Figure 3 shows the messages along the edge between $c(A)$ and $v(A)$.

Loopy belief propagation starts by initializing all outgoing messages to uniform factors of 1s, and computing the corresponding incoming messages. For each outgoing message, $o_{C \to D}(C)$, let $\mathcal{S}_C$ be the set of nodes E such that there is an edge between C and E. Then, in each iteration $o_{C \to D}(C)$ is set to $\prod_{E \in \mathcal{S}_C - D} i_{E \to C}(C)$, and the incoming messages corresponding to these new outgoing messages are recomputed. The algorithm repeats this process until the values of the messages converge[1]. The final approximate belief at each node is the product of all of its incoming messages after the last iteration.

4 Averaged Probabilistic Relational Models (Averaged PRMs)

In this thesis, we will present an *Averaged Probabilistic Relational Model (Averaged PRM)*, which defines a template for a probability distribution over a database representing a relational domain. Unlike a regular Probabilistic Relational Model (as described in Koller & Pfeffer [4]), an Averaged PRM does not contain information about the exact relationships between objects in the domain – it only contains information about potential relationships and the probabilities that those relationships exist. Intuitively, by reducing the amount

[1]Convergence is not guaranteed, but several results show that when LBP does converge, the approximation has some validity. See Yedidia *et al.* [3]

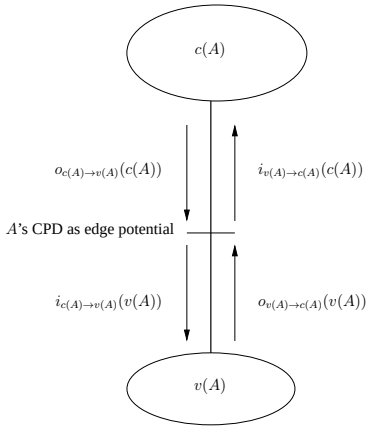

Figure 3: The messages between the compound node and variable node of a variable

of information available to the model, we can improve the performance of inferences done using that model. In this case, we wish to be able to make approximate inferences over the model described by the Averaged PRM in time less than linear in the amount of data that would be required to fully represent the relations between all objects.

The Averaged PRM is intended to be used in conjunction with soft evidence about entire classes of objects in the hypothetical database and not the database itself.

Our definition of an Averaged PRM is based on the definition of PRMs with Existence Uncertainty given by Getoor *et al.* [1]

Relational Schema A schema for a relational model describes a set of *classes*, $\mathcal{X} = X_1, \ldots, X_n$. Each class is associated with a set of *descriptive attributes* and a set of *reference slots*.[2] The set of descriptive attributes of a class X is denoted $\mathcal{A}(X)$. Attribute A of class X is denoted $X.A$, and its domain of values is denoted $V(X.A)$.

We assume here that domains are finite, for example, the Student class might have the descriptive attributes *Stress*, with domain {low, high}.

The set of reference slots of a class X is denoted $\mathcal{R}(X)$. We use $X.\rho$ to denote the reference slot ρ of X. Each reference slot ρ is typed: the domain type of $\mathrm{Dom}[\rho] = X$ and the range type $\mathrm{Range}[\rho] = Y$, where Y is some class in $\mathcal{X}$. A slot ρ denotes a function from $\mathrm{Dom}[\rho] = X$ to $\mathrm{Range}[\rho] = Y$. For example, we might have a class IsEnrolled with the reference slots *student* whose range is the class Student and *course* whose range is the class Course. For each reference slot ρ, we can define an *inverse slot* ρ^{-1}, which is interpreted as the inverse function of ρ.

It is useful to distinguish between an *entity* and a *relationship*, as in entity-relationship diagrams. In our language, classes are used to represent both entities and relationships. Thus, a relationship such as IsEnrolled, which relates students to courses, is also represented as a class, with reference slots to the class Student and the class Course. We do dot allow entities to have reference slots, and for the duration of this thesis we will assume that each relation has precisely two reference slots, both of which point to entities. This is not as restrictive as it may seem, as we will show later how to construct dependencies between attributes in different entities that pass through relations. Also, for each pairing of the two entities that the relation's reference slots point to, there can be at most one object in the relation (that is, the two reference

[2]There is a direct mapping between our notion of class and the tables in a relational database: descriptive attributes correspond to standard table attributes, and reference slots correspond to foreign keys (key attributes of another table).

Student	
name	stress
fred	low
ginger	high
bing	low

Course	
number	workload
CS221	high
CS105	low

IsEnrolled		
student	course	exists
fred	CS221	false
fred	CS105	true
ginger	CS221	true
ginger	CS105	true
bing	CS221	true
bing	CS105	false

Figure 4: An instantiation of the relational schema for a simple school domain

slots can serve as a "key" for this relation). So in our example, a student can only be enrolled in a course once – there can only be one IsEnrolled object that references student Jane and cs221.

We use $\mathcal{X}_\mathcal{E}$ to denote the set of classes that represent entities, and $\mathcal{X}_\mathcal{R}$ to denote those that represent relationships. We use the generic term *object* to refer both to individual elements of entities and to individual elements of relationships.

The semantics of this language is straightforward. An instantiation $\mathcal{I}$ specifies the set of objects in each class X, and the values for each attribute and each reference slots of each object.

For example, Figure 4 shows an instantiation of our simple school schema. It specifies a particular set of students, classes and IsEnrolled relations, along with values for each of their attributes and references.

For $x \in \mathcal{I}(\mathrm{Range}[\rho])$, we use $x.\rho^{-1}$ to denote the set of relation objects $\{y \in \mathcal{I}(X) \ : \ y.\rho = x\}$.

As discussed in the introduction, our goal in this thesis is to construct probabilistic models over instantiations. To do so, we need to provide enough background knowledge to circumscribe the set of possible instantiations. Friedman *et al.* [2] assume that the entire relational structure is given as background knowledge. In other words, they assume that they are given a *relational skeleton*, σ_r, which specifies the set of objects in all classes, as well as all the relationships that hold between them (in other words, it specifies the values for all of the reference slots). In our simple school example, the relational skeleton would specify which students and courses existed, and which IsEnrolled object existed – specifying which students were enrolled in which courses. It would not specify other attributes like course workload or students' grades in courses.

Instead, we add to the schema only information about the number of objects in each entity. Then, as in Getoor *et al.* [1], we will introduce into the model all relation objects that can *potentially* exist. That is, for each pairing of possible values for the relation's two reference slots, we will create one object. We associate with them a special binary variable that indicates whether that object actually exists or not. This binary variable is a descriptive attribute like any other, so we will add it to the schema for the relation class. Thus, in our example, we would add the attribute IsEnrolled.*exists*, and would then create IsEnrolled objects for each possible pairing of a course and a student. Then, student a would be enrolled in course b iff the IsEnrolled object b with c.*enrollingStudent* $=$ a and c.*enrolledCourse* $=$ b had c.*exists* $= true$.

Probabilistic Model An Averaged Probabilistic Relational Model (Averaged PRM) specifies a probability distribution over all instantiations $\mathcal{I}$ of the relational schema. It consists of the qualitative dependency structure, $\mathcal{S}$, and the parameters associated with it, $\theta_\mathcal{S}$. The dependency structure is defined by associating with each attribute $X.A$ a set of *parents* $\mathrm{Pa}(X.A)$. Note that the *exists* attribute in the relation classes is treated like a regular attribute, and can have parents in the same way that other attributes in the relation classes can.

Each parent of $X.A$ has the form $X.\tau.B$ where τ is either empty, a single slot ρ or an inverse of a slot, ρ^{-1}. (PRMs also allow dependencies on longer *slot chains*, but we have chosen to omit those in Averaged PRMs.)

To understand the semantics of this dependence, note that if τ is empty or a single slot, then $x.\tau.A$ refers to an attribute of a single object, and the dependence is defined as in a Bayesian Network. If τ is an inverse slot ρ^{-1}, then $x.\tau.A$ is a multi-set of values S in $V(X.\tau.A)$, and the dependence is on the combined value of all of those attributes.

The quantitative part of the PRM specifies the parameterization of the model. Given a set of parents for an attribute, we can define a local probability model by associating with it a *conditional probability distribution (CPD)*. For each attribute we have a CPD that specifies $P(X.A \mid \text{Pa}(X.A))$.

Definition 1: An *Averaged Probabilistic Relational Model (Averaged PRM)* Π for a relational schema $\mathcal{S}$ is defined as follows. For each class $X \in \mathcal{X}$ and each descriptive attribute $A \in \mathcal{A}(X)$, we have a set of *parents* $\text{Pa}(X.A)$, and a *conditional probability distribution (CPD)* that represents $P_\Pi(X.A \mid \text{Pa}(X.A))$. ∎

Π specifies a distribution over a set of instantiations $\mathcal{I}$ consistent with $\mathcal{S}$:

$$P(\mathcal{I} \mid \Pi, \mathcal{S}) = \prod_{x \in \mathcal{S}(X)} \prod_{A \in \mathcal{A}(x)} P(x.A \mid \text{Pa}(x.A)) \tag{1}$$

Here $\mathcal{S}(X)$ is the set of object associated with X by the schema (recall that the schema includes with each class the number of objects in that class).

For this definition to specify a coherent probability distribution over instantiations, we must ensure that our probabilistic dependencies are acyclic, so that a random variable does not depend, directly or indirectly, on its own value. For this purpose, we use a *class dependency graph* (see Friedman *et al*. [1]), which describes all possible dependencies among attributes. In this graph, we have an (intra-object) edge $X.B \rightarrow X.A$ if $X.B$ is a parent of $X.A$. If $X.\rho.B$ is a parent of $X.A$, and $Y = \text{Range}[\rho]$, we have an (inter-object) edge $Y.B \rightarrow X.A$. If the dependency graph of $\mathcal{S}$ is acyclic, then it defines a legal model for any relational skeleton σ_r [2].

Example Suppose a student has a *stress* attribute which we want to depend on the *workload* attribute of courses that he is enrolled in. Now Student and Course are both entity classes, but we can create a relation class IsEnrolled, with reference slots IsEnrolled.*enrolledCourse* and IsEnrolled.*enrollingStudent* that represents the fact that the student is enrolled in a specific course.

Our model does not allow an attribute in one entity to depend directly on an attribute in another entity, so we cannot allow Student.*stress* to depend directly on Course.*workload*. However, we can get around this problem by adding a "dummy" attribute to the IsEnrolled class as follows. We add an attribute IsEnrolled.*addedWorkload* to IsEnrolled, and have it depend on IsEnrolled.*enrolledCourse*.*workload* with the identity CPD – that is, this new attribute in the relation class will always be identical to the attribute in the entity class. Now, Student.*stress* can depend on the workload of all enrolled classes by adding the set Student.*enrollingStudent*$^{-1}$.*addedWorkload* to its parents. That is, Student.*stress* depends on the new dummy attribute in all of the relation objects representing courses that this student is enrolled in.

The IsEnrolled.*exists* attribute can have the same kinds of dependencies as other attributes. For example, if a student is most likely to be taking classes offered by the same department as his major, IsEnrolled.*enrollingStudent*.*major* and IsEnrolled.*enrolledCourse*.*department* would be parents of IsEnrolled.*exists*.

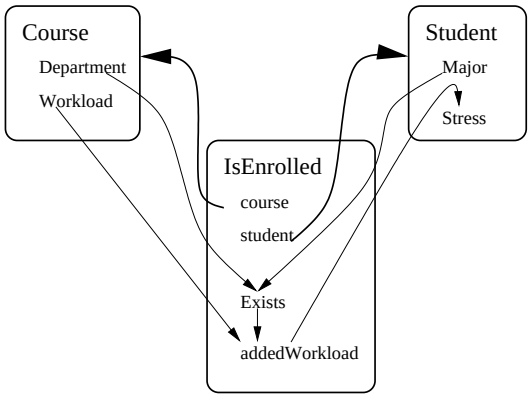

Figure 5: The schema for the school example, showing attribute dependencies

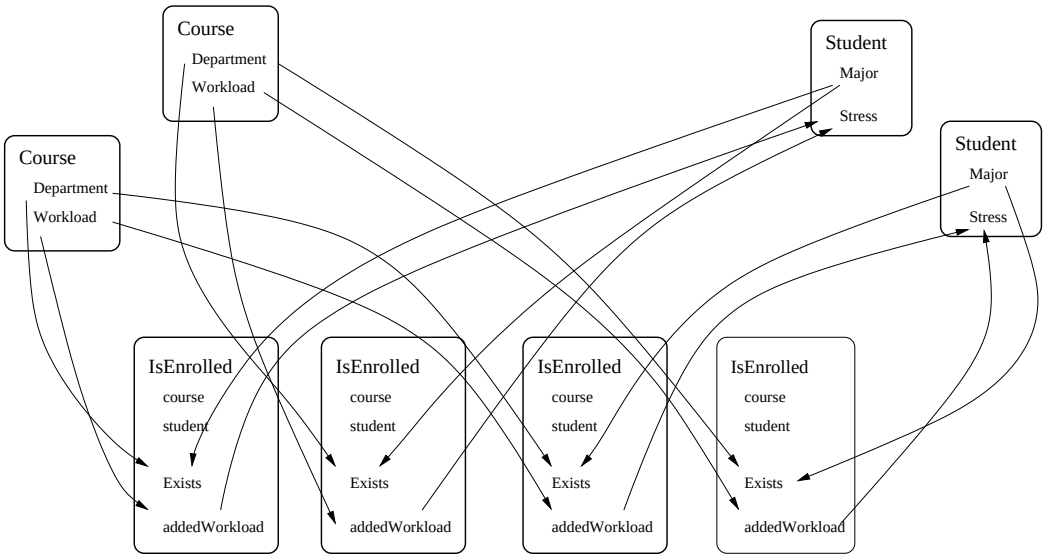

Figure 6: An example of the school network unrolled with two students and two courses

5 CPDs

Attributes in relation classes can have CPDs of any form – their sets of parents are small and well defined by the model. That is, for an attribute $X.A$ where $X \in \mathcal{X}_{\mathcal{R}}$, the set $\text{Pa}(X.A)$ defines a precise small number of parents for $X.A$, so we can define a probability distribution for $X.A$ for each assignment of values to its parents.

For some attributes in entity classes, this may also be the case – if all of an attribute's parents are in the same class, then we can use any regular model for that attribute's CPD. However, if an attribute $X.A$ (with $X \in \mathcal{X}_{\mathcal{E}}$), and $X.A$ has a dependency passing through an inverse reference slot $X.\rho^{-1}.A'$, this translates into $X.A$ having an arbitrary number of parents, so a regular tabular CPD cannot be defined, and would be intractably large if we did know exactly how many parents there were.

Thus, we must use a form of aggregate CPD. This CPD must compactly represent a dependence on a set of variables, all with the same domain, where the size of the set may vary. We will require that this dependence be symmetric in the parent variables – that is, for any two distinct $y_1, y_2 \in \sigma_r(X.\rho^{-1})$, the belief over a specific $x.A$ should not change if the beliefs over $y_1.A'$ and $y_2.A'$ are swapped (assuming no other paths of influence exist between $X'.\rho^{-1}.A'$ and $X.A$).

For the purposes of our inference algorithm below, we will also require that there is a "null-value" in the parents' domain. We will then use this null-value to model relations with the *exists* attribute set to false. In our example above, we would modify the IsEnrolled.*addedWorkload* attribute of the IsEnrolled relation such that, if IsEnrolled.*exists* is true, IsEnrolled.*addedWorkload* is identical to IsEnrolled.*enrolledCourse*.*workload*, but if IsEnrolled.*exists* is false, IsEnrolled.*addedWorkload* is null. Then, we can make the Student.*stress* depend on the set Student.*enrollingStudent*$^{-1}$.*addedWorkload*.

The CPD of Student.*stress* should treat *addedWorkload* values of null as not affecting the student's stress level because the student is not taking that class.

In general, consider the case where an attribute of an entity class $X.A$ depends on an attribute in a relation X' via an inverse slot $X.\rho^{-1}.A'$, and for specific objects $x \in \sigma_r(X)$ and $x' \in x.\rho^{-1}$, $x'.A' = null$. Then, CPD of $x.A$ should return the belief of $x.A$ given that the relationship specified by x' does not exist.

In section 9.4, we will show specific results for the noisy-or CPD for parents through inverse slots, and some general results applicable to all CPDs fulfilling the above criteria.

6 Inference

The primary purpose of this model is for performing inferences given evidence. Specifically, we will deal with queries that ask for the posterior belief over an attribute of an object, given the schema and some evidence about the model.

Traditionally, inference on a regular PRM is done by "unrolling" the model to form a large Bayesian network over all the variables of all the objects. That is, for each object attribute of object in each class, create one node in the Bayesian network, and set the parents of these nodes to correspond to their parents in the PRM. Then, standard Bayesian network inference algorithms may be applied. Clearly, this Bayesian network can perform any inference that is possible with the original PRM.

Unfortunately, exact inference in a Bayesian network is exponential in the worst case, and the networks produced in this manner are typically so highly connected that exact inference is intractable. However, there are many approximate inference algorithms for Bayesian networks, and we will specifically consider loopy belief propagation (LBP) [5]. This is a good solution for the networks created by regular PRMs, and it offers a running time that is close to linear in the number of objects in the model.

This is not a feasible solution for Averaged PRMs, because it involves at least enumerating every object in the domain to create the Bayesian network, even before beginning the inference. Instead, we must exploit

the symmetry of the domain in our approach.

However, even with this symmetry, performing exact inference is not feasible for general Averaged PRMs (though of course there may be certain models for which there exists an efficient specialized algorithm). In the worst case, all attributes of all objects in an Averaged PRM can be correlated, and thus in order to perform exact inference, in would be necessary to create factors over the joint domains of all variables associated with a specific class attribute. These factors would be intractably large.

We can, however, approximate the result of queries on an Averaged PRM efficiently, and the remainder of this thesis presents an algorithm to do so. Our algorithm considers the result of a standard approximate inference algorithm, Loopy Belief Propagation, applied to the unrolled Bayesian network that would be generated for a regular PRM. While we can not generate this network, the symmetry of the Averaged PRM gives us enough information about the structure of the Bayesian network for us to analytically compute the result that would be generated if Loopy Belief Propagation were applied to it.

7 Evidence

Clearly we would like to be able to introduce evidence into the model. However, in order to preserve the symmetry of the model, we must make several restrictions on the form of this evidence.

Specifically, we do not allow evidence of attributes of specific objects - all evidence must be given at the schema level. That is, we can give evidence for an $X.A$ attribute of a class, which is then applied to all $x.A$ variables where $x \in \sigma_r(X)$. This means that we must give the same evidence to all members of a class, we cannot choose a specific member to have different evidence.

This is, however, not as restricting as it may seem. We can still represent models where different members of an entity class have different evidence by splitting the class into new classes that are identical to the old one, except with different evidence. These new classes must then have the same connectivity as the old class, so for every relation class that referenced the original class, we must create another relation that is identical except that it references the new class. After splitting these entity classes, it is possible to apply evidence to the new subclasses individually. It is also possible to apply evidence to the new subclasses of the relation classes.

Clearly, splitting classes increases the number of classes in the model substantially. For example, if we split two entity classes into n and m subclasses respectively and there was a relation joining those two entities, we must split that relation into nm subclasses. Thus, it is clear that this model works well only in cases where most of the evidence is grouped such that it can be applied to an entire class.

We must be careful modifying the CPDs for existing attributes when we split a class. Specifically, consider the case where there is an entity class X with an attribute $X.A$, and a relation class Y with a reference slot $Y.\rho = X$ and an attribute $Y.A'$, and $X.\rho^{-1}.A'$ is in the parent set of $X.A$. Now assume that we split another entity class that Y references, and thus have to split Y. Then, we will have two relation classes – Y and Y', with $Y.\rho = X$ and $Y'.\rho' = X$ and *both* of these sets are parents of $X.A$. We will require that the CPD of $X.A$ be symmetric in elements of both of these sets.

We will allow evidence to be in the form of "soft evidence", which is advantageous because it allows us to express information of the form "a survey showed that 60% of students are highly stressed" without splitting classes at all. Soft evidence is provided as a probability distribution over the domain of the attribute. The effect of applying soft evidence to a single attribute of a single object is equivalent multiplying the posterior belief of the joint distribution over the entire network by that evidence and normalizing the resulting belief. The effect of applying soft evidence to an attribute of a *class* (as is required) is defined to be equivalent to applying the soft evidence to that attribute of each object in the class individually.

Source nodes	Destination nodes	Outgoing Set	Incoming Set	Type	Set stored

For each attribute $X.A$:

$C(X.A)$	$V(X.A)$	$M^o_{C(X.A)\to V(X.A)}$	$M^i_{C(X.A)\to V(X.A)}$	one-to-one	incoming
$V(X.A)$	$C(X.A)$	$M^o_{V(X.A)\to C(X.A)}$	$M^i_{V(X.A)\to C(X.A)}$	one-to-one	outgoing

For $X.A$ and $X'.A'$ in the same class (either entity or relation):

$C(X.A)$	$V(X'.A')$	$M^o_{C(X.A)\to V(X'.A')}$	$M^i_{C(X.A)\to V(X'.A')}$	one-to-one	incoming
$V(X'.A')$	$C(X.A)$	$M^o_{V(X'.A')\to C(X.A)}$	$M^i_{V(X'.A')\to C(X.A)}$	one-to-one	outgoing

For $X.A$ in an entity class with parent $X'.A'$ in a relation:

$C(X.A)$	$V(X'.A')$	$M^o_{C(X.A)\to V(X'.A')}$	$M^i_{C(X.A)\to V(X'.A')}$	one-to-many	incoming
$V(X'.A')$	$C(X.A)$	$M^o_{V(X'.A')\to C(X.A)}$	$M^i_{V(X'.A')\to C(X.A)}$	many-to-one	outgoing

For $X.A$ in a relation class with parent $X'.A'$ in an entity:

$C(X.A)$	$V(X'.A')$	$M^o_{C(X.A)\to V(X'.A')}$	$M^i_{C(X.A)\to V(X'.A')}$	many-to-one	incoming
$V(X'.A')$	$C(X.A)$	$M^o_{V(X'.A')\to C(X.A)}$	$M^i_{V(X'.A')\to C(X.A)}$	one-to-many	outgoing

Figure 7: The message sets involved if an attribute $X.A$, has a parent $X'.A'$, and how the properties vary depending on whether they are in entity or relation classes

8 Loopy Belief Propagation for Averaged PRMs

We will now consider representing the above model as a Bayesian Network and performing loopy belief propagation (LBP) on that network. While the size of the network we are considering will make even LBP too slow, in the following section we will show that the result of performing LBP in this network is so well structured that we can compute the same result without actually enumerating all messages or even all nodes in the network.

The Bayesian Network representation is the intuitive one – each attribute for each object, $x.A$, corresponds to a variable in the Bayesian Network. The variables' set of parents corresponds to the set of parents defined by the Averaged PRM, as does each CPD.

From this Bayesian Network, a pairwise Markov network is created as described in section 3.

8.1 Sets of messages

For each attribute $X.A$ in the schema, there is already an associated set of variables of the form $x.A$ for $x \in \sigma_r(X)$. We will use the notation $V(X.A)$ to refer to the set of all variable nodes associated with these variables, and the set $C(X.A)$ to refer to the associated set of compound nodes.

Then, for each of these variables, there is an edge in the pairwise Markov network from the associated edge between $v(x.A)$ and $c(x.A)$, and the messages associated with this edge. Define the four sets of such messages to be $M^o_{V(X.A)\to C(X.A)}$, $M^i_{V(X.A)\to C(X.A)}$, $M^o_{C(X.A)\to V(X.A)}$ and $M^i_{C(X.A)\to V(X.A)}$. We will call these sets *one-to-one* because no two messages in a single set have the same source or destination node.

Now, if $X.A' \in \text{Pa}(X.A)$, then, for each $x \in \sigma_r(X)$, there are variables $x.A$ and $x.A'$, and there is the edge between $v(x.A')$ and $c(x.A)$, and the messages associated with these edges. Consider, for example, each outgoing message $o_{v(x.A')\to c(x.A)}(v(x.A'))$, and notice that each is a factor over the same domain –

the domain of $X.A'$. Later, we will show that at each step in loopy belief propagation, all of these messages in fact have the same value. We will refer to this set of messages at $M^o_{V(X.A')\to C(X.A)}$. Similarly, we define the set of incoming messages in the same direction as $M^i_{V(X.A')\to C(X.A)}$. And of course since each edge has messages in both directions, there are also the sets $M^o_{C(X.A)\to V(X.A')}$ and $M^i_{C(X.A)\to V(X.A')}$. Note that all of these sets are also one-to-one.

Suppose we have a relation X and an entity $X.\rho$. If $X.\rho.A' \in \text{Pa}(X.A)$, then for each $x \in \sigma_r(X)$, there are variables $x.A$ and $x.\rho.A'$, there is the edge between $v(x.\rho.A')$ and $c(x.A)$ and there are the messages associated with this edge. These messages are associated with the message sets $M^o_{V(X.\rho.A')\to C(X.A)}$, $M^i_{V(X.\rho.A')\to C(X.A)}$, $M^o_{C(X.A)\to V(X.\rho.A')}$ and $M^i_{C(X.A)\to V(X.\rho.A')}$ respectively. Note that these message sets are not one-to-one. $M^o_{V(X.\rho.A')\to C(X.A)}$ and $M^i_{V(X.\rho.A')\to C(X.A)}$ are *one-to-many*, because there are many different relation objects $x \in \sigma_r(X)$ such that $x.\rho$ refers to the same entity object. In fact, since we were careful about limiting the structure of the schema, we know that if there is a relation between two entities, there is precisely one object in that relation for each pair of objects in the entities (though the exists attribute may be true or false for an arbitrary number of them). Thus, if X is a relation and $X.\rho$ refers to the entity class X', then for each object $x' \in \sigma_r(X')$, there are precisely $|X|/|X'|$ objects $x \in \sigma_r(X)$ such that $x.\rho = x'$. Similarly, $M^o_{C(X.A)\to V(X.\rho.A')}$ and $M^i_{C(X.A)\to V(X.\rho.A')}$ are *many-to-one*.

Finally, suppose we have an entity class X and a relation class $X.\rho^{-1}$. If $X.\rho^{-1}.A' \in \text{Pa}(X.A)$, then for each $x \in \sigma_r(X)$, there are variables $x.A$ and $x.\rho^{-1}.A'$, and there is a edge between $v(x.\rho^{-1}.A')$ and $c(x.A)$ and there are the messages associated with this edge. $M^o_{V(X.\rho^{-1}.A')\to C(X.A)}$ and $M^i_{V(X.\rho^{-1}.A')\to C(X.A)}$ are *many-to-one*, because $x.\rho^{-1}$ refers to a set of relation objects, and the construction above is set up to require that if x and x' are distinct objects in $\sigma_r(X)$, then $x.\rho^{-1}$ and $x'.\rho^{-1}$ are disjoint. Then, using the same argument as before, there are $|X.\rho^{-1}|/|X|$ messages for each x object. Similarly, $M^o_{C(X.A)\to V(X.\rho^{-1}.A')}$ and $M^i_{C(X.A)\to V(X.\rho^{-1}.A')}$ are *one-to-many*.

8.2 Equal Beliefs

Intuitively, we feel that for each set of messages defined above, the model definition is symmetric for all messages in that set; so as long as we start with all values of messages within each set being equal, after each iteration they should still be equal to each other. In this section, we provide a formal statement of this intuition, and in later sections we provide both a proof that it is correct and a method to calculate the new values in each set of messages after each iteration.

For a given variable node set $V(A)$, let the incoming messages to each specific node $v(a)$ be partitioned into sets as described above, with the sets named $M^i_{C(B_1)\to V(A)}, ..., M^i_{C(B_n)\to V(A)}$. Now assume that all elements of $M^i_{C(B_i)\to V(A)}$ are equal at the beginning of an iteration of LBP, for each $i = 1, ..., n$. Then, at the end of that iteration, all elements of $M^o_{V(A)\to C(B_i)}$ are equal, for each $i = 1, ..., n$.

For a given compound node set $C(B)$, let the outgoing messages from variable nodes to each specific node $c(b)$ be partitioned into sets as described above, with the sets named $M^o_{V(A_1)\to C(B)}, ..., M^o_{V(A_n)\to C(B)}$ Now assume that all elements of $M^o_{V(A_i)\to C(B)}$ are equal at the beginning of an iteration of LBP, for each $i = 1, ..., n$. Then, at the end of that iteration, all elements of $M^i_{C(B)\to V(A_i)}$ are equal, for each $i = 1, ..., n$.

Note that all the sets mentioned here are incoming to variable nodes or outgoing from variable nodes – they are all factors over the domains of the variable node. The above statement does not hold for sets incoming to or outgoing from compound nodes because those messages are over the joint domain of the variable's parents. Since each message is associated with a different parent, when expanded over the joint domain, they have different values. That is, even if $o_{v(a_1)\to c(b)}(v(a_1))$ and $o_{v(a_2)\to c(b)}(v(a_2))$ are equal, $i_{v(a_1)\to c(b)}(c(b))$ and $i_{v(a_2)\to c(b)}(c(b))$ are not. However, since the CPD is required to be symmetric in the

parents, we can show that once the outgoing messages in a given set are summed out over just their single parent's domain, they are all equal to each other.

Now, clearly if we initialize all messages to contain all 1's, then all messages in any given set (of the sets mentioned above) are equal at the start of the first iteration of message passing, and thus also after each iteration after that.

9 Analytically computing the results of LBP

In the section above we showed that, when running belief propagation on the pairwise Markov network with all messages initialized to all 1's and all messages passed in parallel each iteration, all messages in a given set will have the same value after each iteration.

This insight leads to the following algorithm:

- For each set of messages, store one copy of the value that we showed must be common to all messages in the set.
- Using these values directly, compute what each value must be after the subsequent iteration (how to do this depends on what functions are used for the CPDs and will be explained later).
- Continue this process until convergence, or as many iterations as would have been used if it were feasible to perform a regular belief propagation algorithm on this network.
- With the resulting values of the messages, it is possible to compute the belief over any one node, as it would be had regular belief propagation been performed.

Since each compound node has a very large domain, and outgoing messages from it and the incoming messages to it are factors over the same domain, we do not want to ever actually store the value of even one of these messages. To accomplish this, notice an important fact about the structure of the pairwise Markov network. Specifically, it is a bipartite graph - all edges connect a compound node to a variable node. We mentioned earlier that given an edge between A and B, $o_{A\to B}(A)$ and $i_{A\to B}(B)$ hold essentially the same useful information. Given $o_{A\to B}(A)$ we could compute $i_{A\to B}(B)$ by multiplying by the edge potential and summing out. Thus, if A is a compound node, and we have a way of computing $i_{A\to B}(B)$, we do not ever need to instantiate $o_{A\to B}(A)$. Similarly, if B is instead the compound node, then we can store all information stored in $i_{A\to B}(B)$ by just storing $o_{A\to B}(A)$. By doing this, we only store messages that are factors over variable nodes, which are much smaller than the exponentially large factors over compound nodes.

The primary step in the above algorithm is: given the values for each set of messages in one iteration, calculate the values for each set in the next iteration. How to do this varies depending on the function used for the CPD of each attribute (which translates into the potential on the edge between a variable node and its compound node), and for many functions is intractable. We will show how to efficiently perform the computation exactly for noisy-or CPDs and how to approximate the result for other models.

9.1 Messages to variable nodes

First, given the values of all incoming message sets to variable nodes at the beginning of an iteration, we wish to compute the values of the outgoing message sets from variable nodes.

For a given $v(x.A)$ with $x \in \sigma_r(X)$, we know each incoming message from the previous iteration, and each is over the child node's small domain. We can then compute the belief over this node as follows:

start with a factor over the domain of X.A initialized to all ones as an accumulated belief. For each one-to-one or one-to-many incoming message set with $V(X.A)$ as the destination, we know there is precisely one corresponding incoming message into $v(x.A)$ and we know its value, so we can multiply that value into the accumulated belief.

For each many-to-one (say n-to-one) message set with $V(X.A)$ as the destination, we know there are precisely n corresponding incoming messages to $v(x.A)$, all with the same known value. Thus, instead of multiplying the accumulated belief by the same message n times, we can multiply it by the message raised to the power of n (where we define raising a message to the power of n to mean raising each entry in that message to the power of n). Then this accumulated belief is the belief for each variable node in the set $V(X.A)$, and we have computed it in time proportional to the number of *sets* of messages with destinations in this node set, times the size of the node's domain. Note: for improved numerical precision, it is preferable to work with the logarithm of each entry in the message, so raising the message to the power of n is replaced by multiplying the logarithm of each component of the message by n.

Now, to compute the values of the corresponding outgoing message sets, for each outgoing message set (be it one-to-one, many-to-one, or one-to-many), we know that one outgoing message in that set corresponds to one incoming message from the corresponding incoming message set, so the value of that outgoing message set is the belief over the node divided by the value of the corresponding incoming message set. Thus, given the values of all the incoming message sets to variable node sets, we can compute the the values of the corresponding outgoing message sets.

The remaining task is harder; given the value of the outgoing message sets (from variable node sets) we must compute the value of the corresponding incoming message sets that would be computed by LBP. Since LBP would require factors exponential in the number of objects, and we have a very large number of objects, clearly this must be done analytically.

This is more complicated, and in some cases we will have to approximate the result. Since this computation can be done for each set of compound nodes independently, we will show the computation for an arbitrary given object. That computation can then be done once and applied to the entire class of objects.

9.2 One-to-one and one-to-many messages to compound nodes

If all of the incoming messages to a compound node are one-to-one or one-to-many, then the domain of that compound node will not be intractably large. This corresponds to attributes whose parents are all in the same class, without passing through a reference slot, and attributes whose parents do pass through a reference slot but not an inverse reference slot. In these cases, while the size of the joint domain of the attribute's parents is still exponential in the number of parents, there is a relatively small number of parents, so it is acceptable to work with messages that are factors over this domain, given that the model is of a reasonable size. Thus, we can use simple brute force as we did for the variable nodes.

In this case, we make no restriction on the form of the CPD of $X.A$ – we will convert the CPD into a table and use it as such.

9.3 Many-to-one messages to compound nodes

The single remaining case that we must deal with is when a compound node has many-to-one incoming messages. This corresponds to the case where there is an entity X, and a relation X', where X' has a reference slot $X'.\rho$ that refers to X, and the attribute $X.A$ has as a parent $X'.\rho^{-1}.A'$. Then, the domain of $c(x.A)$ for $x \in \sigma_r(X.A)$ will be the joint domain over *all* $x'.A'$ variables for $x' \in \sigma_r(X')$, and clearly this is intractably large.

9.3.1 Single Parent set

Consider the special case where $X.A$'s only parents are $X.\rho^{-1}.A'$. For simplicity, we assume $X.\rho^{-1}.A'$ has the domain $\{false, true\}$.

Thus, all of the messages in $M^o_{V(X.\rho^{-1}.A')\to C(X.A)}$ are equal and over the domain $\{false, true\}$. We can represent the value of each such message as $\{1-p, p\}$.

Let the domain of $x.A$ ($x \in \sigma_r(X)$) be the set of values $a_1, ..., a_v$, and let the message $i_{c(x.A)\to v(x.A)}(v(x.A))$ be $\{m_1, ..., m_v\}$ where m_j is the component of the message corresponding to $x.A = a_j$. Similarly, let the outgoing message $o_{v(x.A)\to c(x.A)}(v(x.A))$ be $\{m'_1, ..., m'_v\}$. Also, let there be n elements in $X' = X.\rho^{-1}$.

Now the CPD of $X.A$ gives us $P(x.A \mid x'_1.A', ..., x'_n.A')$. However, since the CPD is symmetric in its parents, we know that this probability depends only on the number of parents that are set to $true$. So the CPD can be more compactly represented as a function $F_j(k)$ which is the probability that $x.A = a_j$ given that precisely k of $x.A$'s parents $x'_1.A', ..., x'_n.A'$ are set to $true$.

Then, we can write:

$$m_j = \sum_{i=0}^{n} \binom{n}{i} p^i (1-p)^{n-i} F_j(i) \tag{2}$$

Unfortunately, for large n, this can still be hard to compute exactly for a general F_j. We will show how to compute it for certain specific CPDs in section 9.4.

For the messages back to the parents, from the compound node note that each message is the product of all but one of the outgoing messages from the parents (expanded over the whole domain) and the outgoing message from the variable node, all summed out, and reduced to that parent's domain. We can then utilize the above formula to construct the message back to the parent. Let m^t_j be the j'th component of the message that would have been passed if we were given evidence that one parent was true, that is:

$$m^t_j = \sum_{i=0}^{n-1} \binom{n-1}{i} p^i (1-p)^{n-i-1} F_j(i+1)$$

Similarly, let m^f_j be the component of the message that would have been passed if we were given evidence that that component of the message was false. Then,

$$m^f_j = \sum_{i=0}^{n-1} \binom{n-1}{i} p^i (1-p)^{n-i-1} F_j(i)$$

Then, the incoming message to the parents is

$$\left\{ \sum_{j=0}^{v} m^f_j m'_j, \sum_{j=0}^{v} m^t_j m'_j \right\} \tag{3}$$

For the sample CPDs below, we will only show how to calculate or approximate the message to the variable node of the child attribute. In each case, the same calculation or approximation can be applied to the above formula to construct the messages back to the parent attributes.

9.3.2 Multiple Parent Sets

In the case where $X.A$ has parents through several inverse slots, $X.\rho_1^{-1}.A', ..., X.\rho_s^{-1}.A'$, recall that we made the restriction that the domain of each of the attributes pointed to by these slots was the same, and that the CPD of $X.A$ was still symmetric between these parent slots (so, if the values of one attribute of an object in $x.\rho_i^{-1}$ and an object in $x.\rho_j^{-1}$ are switched, the belief over $x.A$ should not change). Thus, the value of $x.A$ depends on the sum of the number of parents true in each set.

Now, let the messages in $M^o_{V(X.\rho_i^{-1}.A')\to C(X.A)}$ be represented by $\{1-p_i, p_i\}$, and let there be n_i objects in $X.\rho_i^{-1}$. Then, we have

$$m_j = \sum_{i_1=0}^{n_1} \cdots \sum_{i_s=0}^{n_s} \left(\prod_{k=1}^{s} \binom{n_k}{i_k} p_k^{i_k} (1-p_k)^{n_k - i_k} \right) F_j \left(\sum_{k=1}^{s} i_k \right) \tag{4}$$

9.3.3 Both many-to-one and one-to-one parents

Since only attributes in entity classes can have aggregate parents, and only attributes in reference classes can have parents through regular reference slots, we do not have to consider the case when a compound node has both incoming many-to-one and one-to-many messages. We do, however, have to consider the case when it has incoming many-to-one and one-to-one messages. This corresponds to the case where an attribute in an entity class has parents in the same class, as well as parents through an inverse reference slot.

In this case, we still require that, provided that the values of the parent attributes in the same object remain fixed, the belief CPD of $X.A$ is symmetric in the parents through the inverse reference slot. This leads naturally to the following general form of CPD: for each assignment of values to the parents in the same object, the CPD provides a function of the values of the parents through the reference slots that conforms to exactly the same restrictions as mentioned in the previous sections.

Then, to compute the message from the compound node to the variable node, multiply all the incoming messages from parents in the same class together to get a factor over the possible assignments of values to those parents. Then, for each of those possible assignments compute the associated function over the parents through the inverse reference slot, and average the distributions returned by these functions weighted by the computed factor. Computing the messages back to the parents is similar, as described earlier in section 9.3.1.

9.4 Efficient propagation through CPDs

We will show how to compute the exact message into the variable node assuming only many-to-one parents, and how to approximate the message for general CPDs. The methods we show can be generalized to the other cases as described above.

9.4.1 Noisy-or

For a noisy-or CPD, we have $n+1$ noise parameters $\lambda_0, ..., \lambda_n$, and

$$P(x.A = false \mid x'_1.A', ..., x'_n.A') = (1-\lambda_0) \prod_{i:x'_i.A'=true} (1-\lambda_i)$$

To ensure that the CPD is symmetric, we set $\lambda_i = 1-r$ for each $i > 0$, and set $\lambda_0 = 1-k$. Also, for this case we must have that the domain of $X.A$ is also $\{false, true\}$.

Then, $F_0(i) = kr^i$. In this case, there is a simple solution:

$$
\begin{aligned}
m_0 &= \sum_{i=0}^{n} \binom{n}{i} p^i (1-p)^{n-i} k r^i \\
&= k(1-p)^n \sum_{i=0}^{n} \binom{n}{i} \left(\frac{pr}{1-p}\right)^i \\
&= k(1-p)^n \left(1 + \frac{pr}{1-p}\right)^n \\
&= k(1-p+pr)^n
\end{aligned}
$$

And $m_1 = 1 - m_0$.

This solution generalizes to the multiple parent set case with the formula:

$$m_0 = k \prod_{i=1}^{s} (1 - p_i + p_i r)^{n_i}$$

9.4.2 Generalizations of noisy-or

A CPD is said to exhibit *independence of causal influence* if it can be represented as follows:

For each parent $x'_i.A'$, there is a hidden variable $x'_i.B'$ with the single parent $x'_i.A'$, and $x.A$'s CPD is a deterministic function of the values of each $x'_i.B'$ variable, as well as one $x.noise$ variable.

The natural representation of such a model in an Averaged PRM is to explicitly introduce the $x'_i.B'$ variables by creating an $X'.B'$ attribute with $X'.A'$ as its only parent, and also to create a $X.noise$ attribute. Then, the CPD of $X.A$ becomes a deterministic function of $X'.B'$ and $X.noise$.

Now the CPD of $X'.B'$ works just like any other CPD of an attribute whose only parent is in the same class. Also, $X.noise$ is handled as a parent of $X.A$ as described above in section 9.3.3. Thus, all that remains is to show how to compute the incoming message into $v(x.A)$, given the beliefs of each $v(x'_i.B')$ node, without considering the $X.noise$ node.

This must be done separately for each deterministic function required. For example, the deterministic-or function can be computed directly as follows (note that this is a special case of the noisy-or function):

Let each $x'_i.B'$ variable have a binary domain. If each is true with probability p and false with probability $1-p$, then the false-component of the message is $(1-p)^n$. In the more general case where there are s sets of parents, the formula generalizes to $\prod_{j=1}^{s}(1-p_j)^{n_j}$.

9.4.3 Approximations

Unfortunately, most CPDs cannot be represented so simply, and for them there is often no closed-form formula. Instead, we must approximate the result.

One approach is to approximate the binomial distribution with a Gaussian distribution, giving:

$$m_j \approx \frac{1}{2\pi} \sum_{i=0}^{n} \frac{1}{\sqrt{np(1-p)}} e^{\frac{-(i-np)^2}{2np(1-p)}} F_j(i)$$

This approximation can be computed in linear time (assuming all intermediate values can be represented by a single machine variable). We can also improve the approximation by using a standard statistical skew-normal approximation instead of a regular Gaussian.

For large values of n, this summation can be approximated by an integral if $F_j(i)$ is continuous. This integral can then be approximated using standard techniques; in the case of a single many-to-one incoming message set, Gaussian quadrature is particularly well suited to this task.

Thus, we have shown how to analytically simulate each iteration of LBP, and thus have completed all steps for the algorithm described earlier. In the case where all dependencies through inverse slots use a noisy-or CPD, we have shown how to compute exactly the same results as would be computed by LBP. For other CPDs, numerical approximations can be used. Note that none of these methods depend on the number of objects in the domain – only on the number of classes. If a particular class has one hundred or one billion objects, the algorithm will run in exactly the same amount of time.

10 Results

We first evaluated our model by running sample queries on the example school domain described earlier, to validate that the results produced were meaningful.

As a simple example, the posterior belief that Student.*stress* $= high$ increases if we provide soft evidence to Course.*workload* indicating prior knowledge that most courses have a high workload. Similarly, if we split Course.*workload* into a main group for which we have no evidence, and a smaller group (containing only one object) for which we have evidence that that class has a high workload, the posterior that Student.*stress* $= high$ increases as expected.

Slightly more subtle results are obtained when we apply evidence to the Student.*stress* attribute. When we apply soft evidence that most students are stressed, the posterior belief that Course.*workload* $= high$ increases, but also, the posterior belief that IsEnrolled.*exists* $= true$ increases, indicating that the students are probably taking more classes than was previously believed. Thus, we have used the evidence about a descriptive attribute of the students to make inference about the relational structure of the domain itself.

Next, we split Course into two groups – one with soft evidence that the courses mostly had a low workload, and the other with evidence that they mostly had a high workload. We also split Student into two groups – one with evidence of low stress and the other with evidence of high stress. Now, as per the procedure described earlier for splitting entities, IsEnrolled is split into four groups – one for each pairing of the new Course and Student groups. Inference on this model showed that high-stress students are more likely to be taking high-workload courses, and low-stress students are more likely to be taking low-workload courses, just as expected. In this model, since students do not have a hard restriction on the number of courses they take, the inference also shows that a high-stress student is actually more likely to be taking a given low-workload course than a low stress student, but this makes sense because a high-stress student is probably taking more courses than a low-stress student, as shown by the previous experiment.

Recall that the *exists* attributes of relation courses can have parents. This allows for some interesting reasoning patterns. Specifically, given that one student is stressed with high probability, and one course is in the Computer Science department, if we supply evidence that that course has a high workload, the probability that the student is a Computer Science major increases.

The above simple examples used small domains, with only 50 courses and 100 students. To test the scalability of the algorithm, we increased the number of courses to 5 million and the number of students to 1 billion, giving 5 quadrillion IsEnrolled objects. With this size of domain, we received the same intuitive results as above, with no increase in running time.

Figure 8 shows the running time of inference in an Averaged PRM compared to conventional LBP in the unrolled PRM, as the number of objects increases, using one specific implementation of each algorithm. As expected, conventional LBP take approximately linear time in the number of objects, while inference in the Averaged PRM takes constant time.

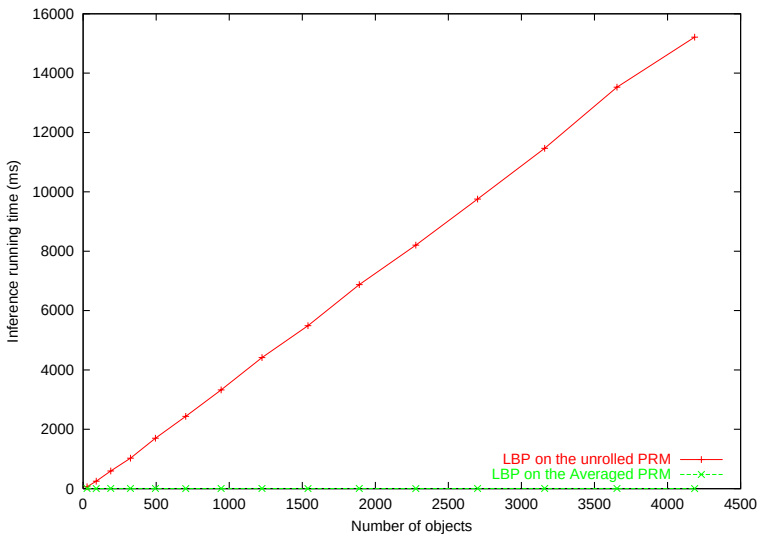

Figure 8: Running time of LBP in an unrolled PRM and in an Averaged PRM

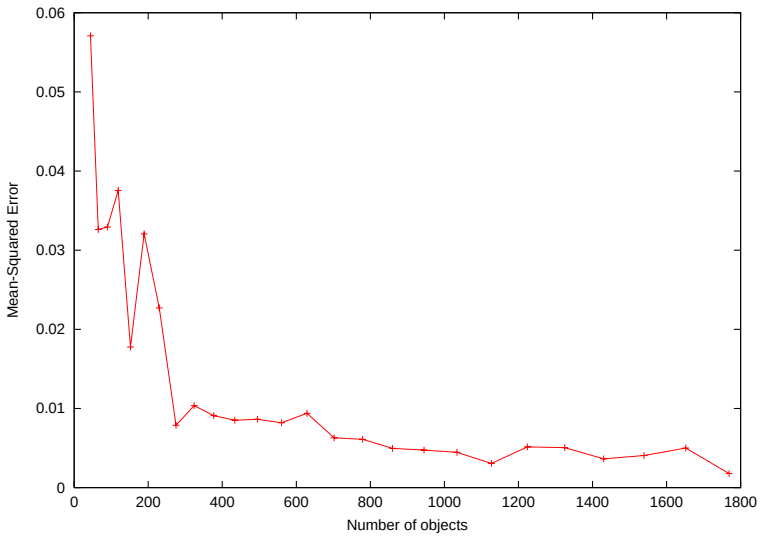

Figure 9: Error of Averaged PRM estimate given the full link structure as the number of objects increases

We have shown mathematically that the Averaged PRM LBP algorithm gives the same result as performing LBP on the unrolled network described by the Averaged PRM. However, this is not the same as comparing it to LBP in a PRM that has a database containing the exact relational structure of domain and precise evidence for individual variables. Of course, since an Averaged PRM has less information available, we cannot expect exactly the same results. However, it is interesting to consider how much accuracy is lost by ignoring the large database and using only averaged statistics, as the Averaged PRM does. Figure 9 shows the mean-squared error using an Averaged PRM on multiple randomly produced datasets of different sizes, with the relation *exists* probability adjusted to keep the expected number of relations connected to each entity object constant. As might be expected, for small domains it is important to consider the exact structure and evidence, while for large domains the details of the exact structure and evidence are less important.

11 Issues with Loopy Belief Propagation

LBP is an approximate inference algorithm. While we showed that the algorithm presented above computes exactly the same result as LBP, clearly it must suffer from the same limitations as LBP. Recent work by Yedidia *et al*. [3] has provided some theoretical validation of the approximation computed by LBP, and has also proposed an enhanced version of the algorithm, generalized belief propagation (GBP). Since this algorithm is implemented in a manner very similar to LBP, it seems appropriate to attempt to modify our algorithm to compute the results that GBP would have computed if applied to the entire unrolled Bayesian network associated with the Averaged PRM. We refer the reader to Yedidia *et al*. for details on Generalized Belief Propagation, and provide here a suggestion as to how it may be applied to our algorithm.

Essentially, in GBP, the nodes are grouped into regions, and messages are passed between entire regions at a time. For each object, we create one GBP region containing all of the nodes associated with attributes of that object. Also, we define a pair of objects to be connected if there is at least one edge that passes from a node in one object's region to a node in the other object's region. Then, for each pair of connected objects, we create a region that contains all the nodes on edges that connect the two objects.

Now, the factor over each region associated with an entity object can be intractably large, because the regions includes the compound node node for nodes that have parents through inverse slots. We can overcome this issue in the same way as we did for regular loopy belief propagation. Instead of representing the entire factor over all nodes associated with the entity object, we leave out the compound node nodes associated with parents through inverse slots from the message and from the object's factor, and then analytically compute the incoming message summed out to just the remaining nodes. Exploring the details of this enhanced algorithm is left for future work.

12 Discussion and Conclusion

In this thesis, we have proposed Averaged Probabilistic Relational Models (Averaged PRMs) as a representation for abstract information about extremely large domains – domains so large that we cannot hope to enumerate all objects in them. We have also presented an algorithm to perform approximate inference using Averaged PRMs efficiently, with running time dependent only on the schema of the model and not the number of objects in the domain. We have presented results showing that this algorithm can answer queries of interest, with inferences matching those made by human intuition.

The accuracy of our approximation is fundamentally based on a reduction to the now standard loopy belief propagation algorithm, which has been well analyzed by Yedidia *et al.* [3]. Yedidia *et al.* also generalized LBP to improve its accuracy at the cost of reduced performance. Those generalizations can be applied to our algorithm, with the same tradeoffs.

Methods for describing similar domains have been presented before by Getoor *et al*. [1], but those approaches assumed that the domains were small enough that it was feasible to enumerate all objects, and used approximate inference algorithms that are linear in the size of the domain. They can, however, allow more precise descriptions of the domain than our models can, because it is not necessary for them to maintain strict symmetry between groups of objects.

However, due to the similarity between our model and that used by Getoor *et al*., there is a possibility of using their methods to learn a schema from a small amount of data, and then using that schema in an Averaged PRM to make inferences about a large domain with a similar structure.

The ability to reason about extremely large domains in this manner presents many exciting opportunities. In future work, we hope to increase the expressiveness of the model, perhaps by providing a hybrid combination of Averaged PRMs and regular PRMs. We also plan on exploring more fully how to improve the accuracy of inference results using generalized belief propagation.

Acknowledgments This work was done in collaboration with my advisor, Daphne Koller, and with Eran Segal.

References

[1] Getoor, L., Friedman, N., Koller, D., & Taskar, B. (2001). Learning Probabilistic Models of Relational Structure. *Proc. ICML*.

[2] Friedman, N., Getoor, L., Koller, D., & Pfeffer, A. (1999). Learning probabilistic relational models. *Proc. IJCAI*.

[3] Yedidia, J.S., Freeman W.T. & Weiss, Y. (2000). Bethe free energy, Kikuchi approximations and belief propagation algorithms *Proc. NIPS*.

[4] Koller, D., & Pfeffer, A. (1998). Probabilistic frame-based systems. *Proc. AAAI*.

[5] Murphy, K., & Weiss, Y. (1999). Loopy belief propagation for approximate inference: an empirical study. *Proc. UAI*.

[6] Pearl, J. (1988). Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann, San Francisco.