Probabilistic Models for Relational Data

David Heckerman*, Christopher Meek, and Daphne Koller**
Microsoft Research and **Stanford University
heckerma@microsoft.com, meek@microsoft.com, koller@cs.stanford.edu

March 2004

Abstract

We introduce a graphical language for relational data called the probabilistic entity-relationship (PER) model. The model is an extension of the entity-relationship model, a common model for the abstract representation of database structure. We concentrate on the directed version of this model—the directed acyclic probabilistic entity-relationship (DAPER) model. The DAPER model is closely related to the plate model and the probabilistic relational model (PRM), existing models for relational data. The DAPER model is more expressive than either existing model, and also helps to demonstrate their similarity. In addition to describing the new language, we discuss important facets of modeling relational data, including the use of restricted relationships, self relationships, and probabilistic relationships. Many examples are provided.

Keywords: relational modeling, relational data, plate models, probabilistic relational models, graphical models

1 Introduction

For over a century, statistical modeling has focused primarily on "flat" data—data that can be encoded naturally in a single two-dimensional table having rows and columns. The disciplines of pattern recognition, machine learning, and data mining have had a similar focus. Notable exceptions include hierarchical models (e.g., Good, 1965) and spatial statistics (e.g., Besag, 1974). Over the last decade, however, perhaps due to the ever increasing volumes of data being stored in databases, the modeling of non-flat or *relational data* has increased significantly. During this time, two graphical languages for relational data have emerged: plate models (e.g., Buntine, 1994; Spiegelhalter, 1998) and probabilistic relational models

*Corresponding author. One Microsoft Way, Redmond, WA, 98052.

(PRMs) (e.g., Friedman, Getoor, Koller, and Pfeffer, 1999; Getoor, Friedman, Koller, and Pfeffer, 2002). These models are to relational data what ordinary graphical models (e.g., directed-acyclic graphs and undirected graphs) are to flat data.

Although graphically quite different, the two languages are similar in their ability to represent or express probabilistic relationships (i.e., conditional independence) in relational data. Nonetheless, plate models have been used almost exclusively by statisticians, whereas PRMs have been used mostly by computer scientists. In this paper, we demonstrate the similarity among these languages with the hope of fostering greater communication and collaboration among researchers in these two disciplines.

We demonstrate the similarity between the plate model and PRM by introducing a third language—the probabilistic entity-relationship (PER) model—that is similar to both. This new model class is an extension of the entity-relationship (ER) model, a common model for the abstract representation of database structure. As we shall see, this language not only serves to make connections between plate models and PRMs, but it also enhances the expressiveness of these languages. Unlike plates models or PRMs, PER models make relationships first class objects in the language. This feature of the language makes it particularly easy to model relational data.

In this paper, we concentrate on a particular type of PER model—the directed acyclic probabilistic entity-relationship (DAPER) model—in which all probabilistic arcs are directed, although we discuss extensions near the end of the paper. It is this version of PER model that is most similar to the plate model and PRM. We define new versions of the plate model and PRM such their expressiveness is equivalent to the DAPER model, and then compare the new and old definitions. Consequently, we both demonstrate the similarity among the original languages as well as enhance their abilities to express conditional independence in relational data.

In Section 2, we review (ordinary) directed acyclic graphical models. In Section 3, we present the basics of our relational language. First, we introduce the PER model by example. Then, we introduce our definitions for the plate model and PRM by providing a mapping from any DAPER model to an equivalent plate model and from any DAPER model to an equivalent PRM. In Section 4, we examine DAPER models in detail, examining issues such as the representation of restricted relationships, self relationships, and probabilistic relationships. In Sections 5 and 6, we examine plate models and PRMs, respectively, and discuss how our definitions differ from and improve upon the existing ones. In Section 7, we describe extensions to the DAPER model including versions that encode undirected and hyperedge models. In the Appendix, we provide formal definitions and theorems. Throughout the paper, we concentrate on the PER model as a language for representing conditional independence in relational data. We pay little attention to issues of learning such models from data or of performing probabilistic inference with such models.

We should emphasize that the languages we discuss are neither meant to serve as a database schema nor meant to be built on top of one. In practice, database schemas are built up over a long period of time as the needs of the database consumers change. Consequently, schemas for real databases are often not optimal or are completely unusable as the basis for statistical modeling. The languages we describe here are meant to be used as statistical modeling tools, independent of the schema of the database being modeled.

This work borrows heavily from concepts surrounding PRMs described in (e.g.) Friedman et al. (1999) and Getoor et al. (2002). Where possible, we use similar nomenclature, notation, and examples.

2 Background: Graphical Models

As mentioned, we shall concentrate on directed models in this paper. Accordingly, we first review (ordinary) directed acyclic models.

A directed acyclic graphical (DAG) model for a finite set of attributes $\mathbf{X} = (X_1, \ldots, X_n)$ with joint distribution $p(\mathbf{x})$ has two components: (1) a directed acyclic graph—sometimes referred to as the *structure* of the model—that encodes a set of conditional independencies among the attributes, and (2) a collection of local distributions. The nodes in the directed acyclic graph are in one-to-one correspondence with the attributes in $\mathbf{X}$. To keep notation simple, we use X_i to refer to the node corresponding to attribute X_i. Whether X_i refers to an attribute or node will be clear from context. The absence of arcs in the directed acyclic graph encode probabilistic independencies that allow the joint distribution for $\mathbf{X}$ to be written as

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i|\mathbf{pa}_i), \quad (1)$$

where $\mathbf{pa}_i$ are the attributes corresponding to the parents of node X_i. The *local distributions* of the DAG model is the set of conditional probability distributions $p(x_i|\mathbf{pa}_i)$, $i = 1, \ldots, n$. Thus, a DAG model for $\mathbf{X}$ specifies the joint distribution for $\mathbf{X}$.

An example DAG model structure for attributes (X, Y, Z, W) is shown in Figure 1a. The structure (i.e., the missing arcs) encode the independencies (1) X and Z are independent given Y, and (2) (Y, Z) and W are independent given X. We note that DAG models can be interpreted as a *generative model* for the data. In our example, we can generate a sample for (X, Y, Z, W) by first sampling X, then Y and W given X, and finally Z given Y.

As we shall see, when working with relational data, it is often necessary to express

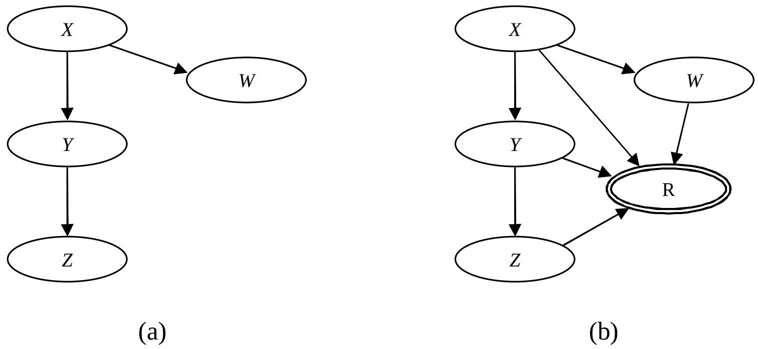


Figure 1: (a) A DAG model. (b) A similar DAG model with an added restriction among the attributes.

constraints or *restrictions* among attributes. Such restrictions can be encoded in a DAG model, which we review here.

As a simple example, suppose we have generative story for binary (0/1) attributes X, Y, Z, and W that can be described by the DAG model structure shown in Figure 1a. In addition, suppose we know that at most two of these attributes take on the value one. We can add this restriction to the model as shown in Figure 1b. Here, we have added a binary node named R. Associated with this node (not shown in the figure) is a local distribution wherein $R = 1$ with probability one when at most two of its parents take on value one, and with probability zero otherwise. To encode the restriction, we set $R = 1$. Note that R is a *deterministic attribute*. That is, given the parents of R, R is known with certainty. As is commonly done in the graphical modeling literature, we indicate deterministic nodes with double ovals.[1]

Assuming that the restriction always holds—that is, R is always equal to one—it is not meaningful to work with the joint distribution $p(x, y, z, w, r)$. Instead, the appropriate distribution to make inferences with is

$$p(\mathbf{x}|r = 1) = p(x)\ p(y|x)\ p(z|y)\ p(w|x)\ p(r = 1|x, y, z, w). \quad (2)$$

Readers familiar with directed factor-graph models (Frey, 2003) will recognize that this distribution for (X, Y, Z, W) can be encoded by a directed factor-graph model in which node R is replaced by the factor $f(x, y, z, w) = p(r = 1|x, y, z, w)$. More generally, the

[1] DAG models can also be used to encode "soft" restrictions. For example, if we know that zero, one, two, three, and four of the attributes $\mathbf{X}$ take on the value one with probabilities p_0, p_1, p_2, p_3, and p_4, respectively, we can encode this soft restriction using the DAG model structure in Figure 1b where R is no longer deterministic and has the appropriate local probability distribution.

factor-graph model is perhaps a more natural model for situations having both a generative component and restrictions. In this paper, however, we use the DAG representation of restrictions so that we remain within the class of DAG models and thereby simplify the presentation.

3 The Basic Ideas

Before we describe languages for the statistical modeling of relational data, we begin with a description of a language for modeling the data itself. The language we discuss is the *entity-relationship (ER) model*, a commonly used abstract representation of database structure (e.g., Ullman and Widom, 2002). The creation of an ER model is often the first step in the process of building a relational database. Features of anticipated data and how they interrelate are encoded in an ER model. The ER model is then used to create a relational schema for the database, which in turn is used to build the database itself.

It is important to note that an ER model is a representation of a database structure, not of a particular database that contains data. That is, an ER model can be developed prior to the collection of any data, and is meant to anticipate the data and the relationships therein.

When building ER models, we distinguish between entities, relationships, and attributes. An *entity* corresponds to a thing or object that is or may be stored in a database or dataset[2]; a *relationship* corresponds to a specific interaction among entities; and an *attribute* corresponds to a variable describing some property of an entity or relationship. Throughout the paper, we use examples to illustrate concepts.

Example 1 A university database maintains records on students and their IQs, courses and their difficulty, and the courses taken by students and the grades they receive.

In this example, we can think of individual students (e.g., john, mary) and individual courses (e.g., cs107, stat10) as entities.[3] Naturally, there will be many students and courses in the database. We refer to the set of students (e.g., {john,mary,...}) as an *entity set*. The set of courses (e.g., {cs107,stat10,...}) is another entity set. Most important, because an ER model can be built before any data is collected, we need the concept of an *entity class*—a reference to a set of entities without a specification of the entities in the set. In our example, the entity classes are Student and Course.

[2]In what follows, we make no distinction between a database and a dataset.

[3]In a real database, longer names would be needed to define unique students and courses. We keep the names short in our example to make reading easier.

A relationship is a list of entities. In our example, a possible relationship is the pair (john, cs107), meaning that john took the course cs107. Using nomenclature similar to that for entities, we talk about relationship sets and relationship classes. A *relationship set* is a collection of like relationships—that is, a collection of relationships each relating entities from a fixed list of entity classes. In our example, we have the relationship set of student-course pairs. A *relationship class* refers to an unspecified set of like relationships. In our example, we have the relationship class Takes.

The IQ of john and the difficulty of cs107 are examples of *attributes*. We use the term *attribute class* to refer to an unspecified collection of like attributes. In our example, Student has the single attribute class Student.IQ and Course has the single attribute class Course.Diff. Relationships also can have attributes; and relationship classes can have attribute classes. In our example, Takes has the attribute class Takes.Grade.

An ER model for the structure of a database graphically depicts entity classes, relationships classes, attribute classes, and their interconnections. An ER model for Example 1 is shown in Figure 2a. The entity classes (Student and Course) are shown as rectangular nodes; the relationship class (Takes) is shown as a diamond-shaped node; and the attribute classes (Student.IQ, Course.Diff, and Takes.Grade) are shown as oval nodes. Attribute classes are connected to their corresponding entity or relationship class, and the relationship class is connected to its associated entity classes. (Solid edges are customary in ER models. Here, we use dashed edges so that we can later use solid edges to denote probabilistic dependencies.)

An ER model describes the potential attributes and relationships in a database. It says little about actual data. A *skeleton for a set of entity and relationship classes* is specification of the entities and relationships associated with a particular database. That is, a skeleton for a set of entity and relationship classes is collection of corresponding entity and relationship sets. An example skeleton for our university-database example is shown in Figure 2b.

An ER model *applied to a skeleton* defines a specific set of attributes. In particular, for every entity class and every attribute class of that entity class, an attribute is defined for every entity in the class; and, for every relationship class and every attribute class of that relationship class, an attribute is defined for every relationship in the class. The attributes defined by the ER model in Figure 2a applied to the skeleton in Figure 2b are shown in Figure 2c. In what follows, we use *ER model* to mean both the *ER diagram*—the graph in Figure 2a—and the mechanism by which attributes are generated from skeletons.

A skeleton still says nothing about the values of attributes. An *instance for an ER model* consists of (1) a skeleton for the entity and relationship classes in that model, and (2) an assignment of a value to every attribute generated by the ER model and the skeleton.

That is, an instance of an ER model is an actual database.

Let us now turn to the probabilistic modeling of relational data. To do so, we introduce a specific type of probabilistic entity-relationship model: the directed acyclic probabilistic entity-relationship (DAPER) model. Roughly speaking, a DAPER model is an ER model with directed (solid) arcs among the attribute classes that represent probabilistic dependencies among corresponding attributes, and local distribution classes that define local distributions for attributes. Recall that an ER model applied to a skeleton defines a set of attributes. Similarly, a DAPER model applied to a skeleton defines a set of attributes as well as a DAG model for these attributes. Thus, a DAPER model can be thought of as a language for expressing conditional independence among unrealized attributes that eventually become realized given a skeleton.

As with the ER diagram and model, we sometimes distinguish between a *DAPER diagram*, which consists of the graph only, and the *DAPER model*, which consists of the diagram, the local distribution classes, and the mechanism by which a DAPER model defines a DAG model given a skeleton.

Example 2 In the university database (Example 1), a student's grade in a course depends both on the student's IQ and on the difficulty of the course.

The DAPER model (or diagram) for this example is shown in Figure 3a. The model extends the ER model in Figure 2 with the addition of arc classes and local distribution classes. In particular, there is an *arc class* from Student.IQ to and arc class from Takes.Grade and from Course.Diff to Takes.Grade. These arc classes are denoted as a solid directed arc. In addition, there is a single local distribution class for Takes.Grade (not shown).

Just as we expand attribute classes in a DAPER model to attributes in a DAG model given a skeleton, we expand arc classes to arcs. In doing so, we sometimes want to limit the arcs that are added to a DAG model. In the current problem, for example, we want to draw an arc from attribute c.Diff for course c to attribute Takes(s, c').Grade for course c' and any student s, only when $c = c'$. This limitation is achieved by adding a *constraint* to the arc class—namely, the constraint course[Diff] = course[Grade] (see Figure 3a). Here, the terms "course[Diff]" and "course[Grade]" refer to the entities c and c', respectively—the entities associated with the attributes at the ends of the arc.

The arc class from Student.IQ to Takes.Grade has a similar constraint: student[IQ] = student[Grade]. This constraint says that we draw an arc from attribute s.IQ for student s =student[IQ] to Takes(s', c).Grade for student s'=student[Grade] and any course c only when $s = s'$. As we shall see, constraints in DAPER models can be quite expressive—for example, they may include first-order expressions on entities and relationships.

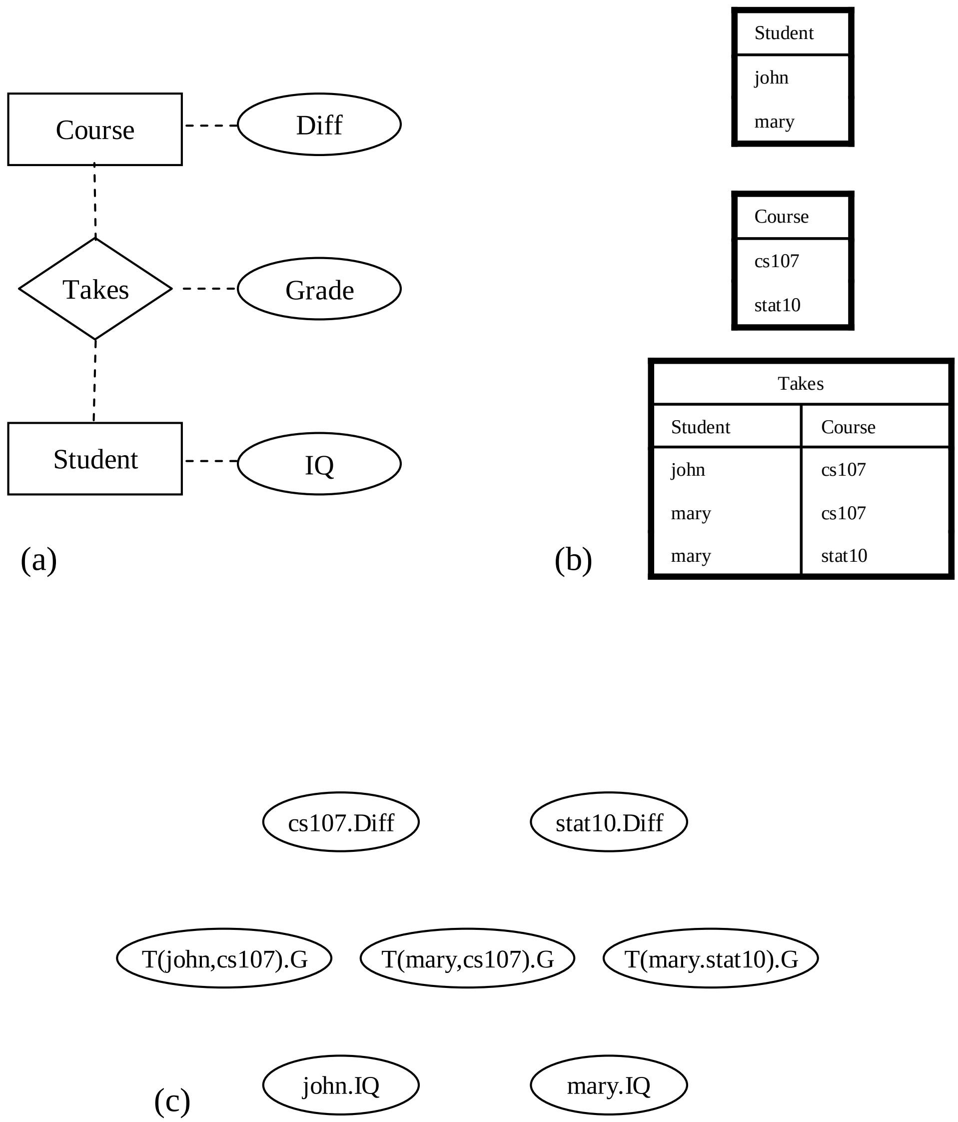


Figure 2: (a) An entity-relationship (ER) model depicting the structure of a university database. (b) An example skeleton for the entity and relationship classes in the ER model. (c) The attributes defined by the application of the ER model to the skeleton. The attribute names are abbreviated.

Figure 3c shows the DAG (structure) generated by the application of the DAPER model in Figure 3a to the skeleton in Figure 3b. (The attribute names in the DAG model are abbreviated.) The arc from stat10.Diff to Takes(mary,cs107).Grade (e.g.) is disallowed by the constraint on the arc class from Course.Diff to Takes.Grade.

Regardless of what skeleton we use, the DAG model generated by the DAPER model in Figure 3a will be acyclic. In general, as we show in the Appendix, if the attribute classes and arc classes in the DAPER diagram form an acyclic graph, then the DAG model generated from any skeleton for the DAPER model will be acyclic. Weaker conditions are also sufficient to guarantee acyclicity. We describe one in the Appendix.

In general, a *local distribution class* for an attribute class is a specification from which local distributions for attributes corresponding to the attribute class can be constructed, when a DAPER model is expanded to a DAG model. In our example, the local distribution class for Takes.Grade—written p(Takes.Grade|Student.IQ, Course.Diff)—is a specification from which the local distributions for Takes(s, c).Grade, for all students s and courses c, can be constructed. In our example, each attribute Takes(s, c).Grade will have two parents: s.IQ and c.Diff. Consequently, the local distribution class need only be a single local probability distribution. We discuss more complex situations in Section 4.

Whereas most of this paper concentrates issues of representation, the problems of probabilistic inference, learning local distributions, and learning model structure are also of interest. For all of these problems, it is natural to extend the concept of an instance to that of a *partial instance*; an instance in which some of the attributes do not have values. A simple approach for performing probabilistic inference about attributes in a DAPER model given a partial instance is to (1) explicitly construct a ground graph, (2) instantiate known attributes from the partial instance, and (3) apply standard probabilistic inference techniques to the ground graph to compute the quantities of interest. One can improve upon this simple approach by utilizing the additional structure provided by a relational model—for example, by caching inferences in subnetworks. Koller and Pfeffer (1997), for example, have done preliminary work in this direction. With regards to learning, note that from a Bayesian perspective, both learning about both the local distributions and model structure can be viewed as probabilistic inference about (missing) attributes (e.g., parameters) from a partial instance. In addition, there has been substantial research on learning PRMs (e.g., Getoor et al., 2002) and much of this work is applicable to DAPER models.

We shall explore PER models in much more detail in subsequent sections. Here, let us examine two alternate languages for relational data: plate models and PRMs.

Plate models were developed independently by Buntine (1994) and the BUGS team (e.g., Spiegelhalter 1998) as a language for compactly representing graphical models in

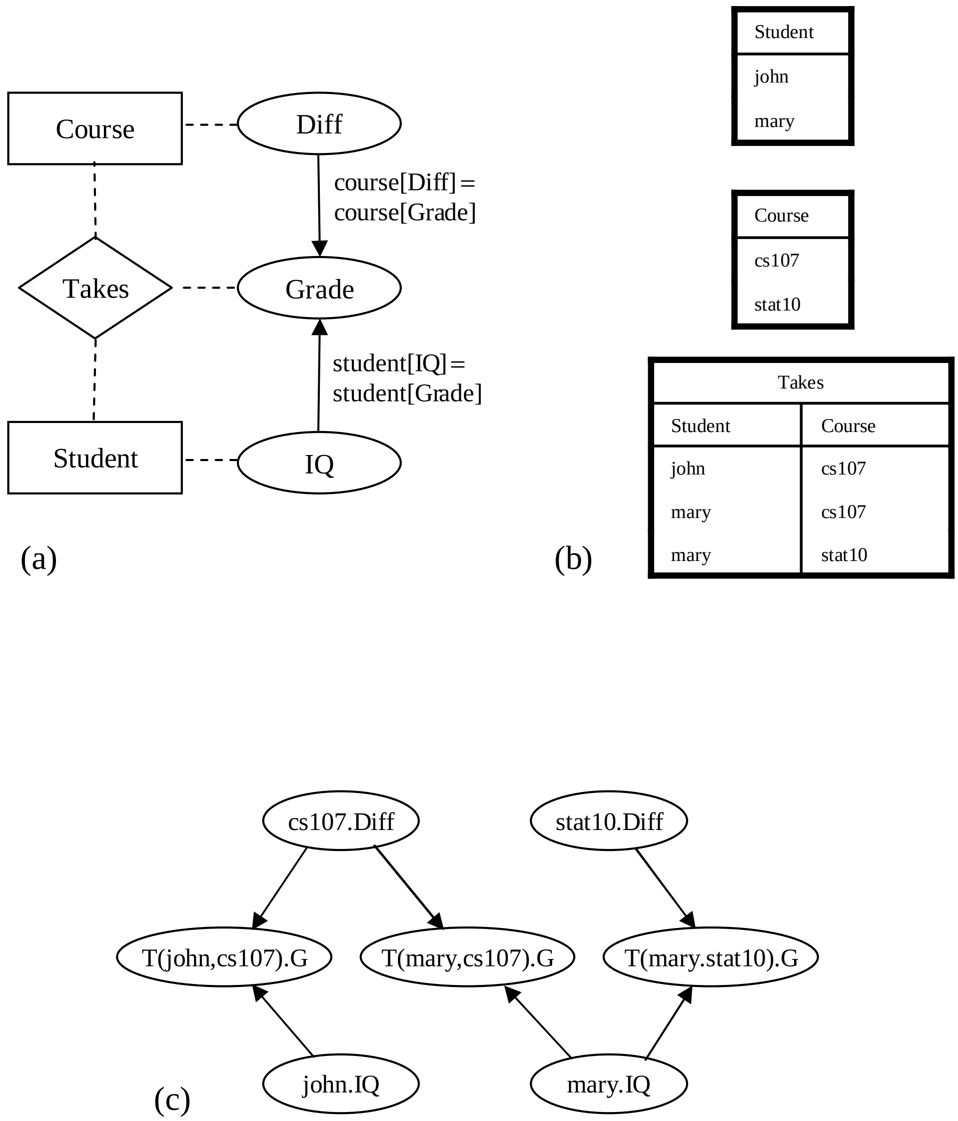


Figure 3: (a) A directed acyclic probabilistic entity-relationship (DAPER) model showing that a student's grade in a course depends on both the student's IQ and the difficulty of the course. The solid directed arcs correspond to probabilistic dependencies. These arcs are annotated with constraints. (b) An example skeleton for the entity and relationship classes in the ER model (the same one shown in the previous figure). (c) The DAG model (structure) defined by the application of the DAPER model to the ER skeleton.

which there are repeated measurements. We know of no formal definition of a plate model, and so we provide one here. This definition deviates slightly from published examples of plate models, but it enhances the expressivity of such models while retaining their essence (see Section 5).

According to our definition, plate and DAPER models are equivalent. The invertible mapping from a DAPER to plate model is as follows. Each entity class in a DAPER model is drawn as a large rectangle—called a *plate*. The plate is labeled with the entity-class name. Plates are allowed to intersect or overlap. A relationship class for a set of entity classes is drawn at the named intersection of the plates corresponding to those entities. If there is more than one relationship class among the same set of entity classes, the plates are drawn such that there is a distinct intersection for each of the relationship classes. Attribute classes of an entity class are drawn as ovals inside the rectangle corresponding to the entity but outside any intersection. Attribute classes associated with a relationship class are drawn in the intersection corresponding to the relationship class. Arc classes and constraints are drawn just as they are in DAPER models. In additon, local distribution classes are specified just as they are in DAPER models.

The plate model corresponding to the DAPER model in Figure 3a is shown in Figure 4a. The two rectangles are the plates corresponding to the Student and Course entity classes. The single relationship class between Student and Course—Takes—is represented as the named intersection of the two plates. The attribute class Student.IQ is drawn inside the Student plate and outside the Course plate; the attribute class Course.Diff is drawn inside the Course plate and outside the Student plate; and the attribute class Takes.Grade is drawn in the intersection of the Student and Course plate. The arc classes and their constraints are identical to those in the DAPER model.

Probabilistic Relational Models (PRMs) were developed in (e.g.) Friedman et al. (1999) explicitly for the purpose of representing relational data. The PRM extends the *relational model*—another commonly used representation for the structure of a database—in much the same way as the PER model extends the ER model. In this paper, we shall define directed PRMs such that they are equivalent to DAPER models and, hence, plate models. This definition deviates from the one given by (e.g.) Friedman et al. (1999), but enhances the expressivity of the language as previously defined (see Section 6).

The invertible mapping from a DAPER model to a directed PRM (by our definition) takes place in two stages. First, the ER-model component of the DAPER model is mapped to a relational model in a standard way (e.g., Ullman and Widom, 2002). In particular, both entity and relationship classes are represented as tables. Foreign keys—or what Getoor et al. 2002 call *reference slots*—are used in the relationship-class tables to enocde the entity-

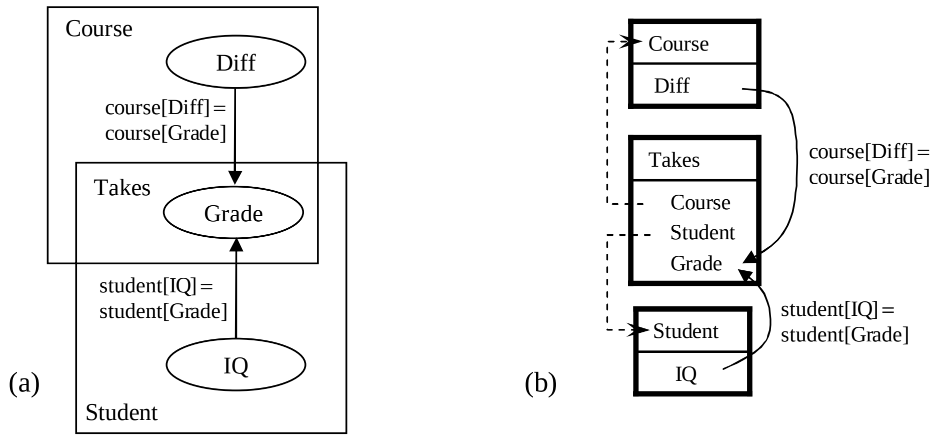


Figure 4: A plate model (a) and probabilistic relational model (b) corresponding the DAPER model in Figure 3a.

relationship connections in the ER model. Attribute classes for entity and relationship classes are represented as attributes or columns in the corresponding tables of the relational model. Second, the probabilistic components of the DAPER model are mapped to those of the directed PRM. In particular, arc classes and constraints are drawn just as they are in the DAPER model.

The directed PRM corresponding to the DAPER model in Figure 3a is shown in Figure 4b. (The local distribution for Takes.Grade is not shown.) The Student entity class and its attribute class Student.IQ appear in a table, as does the Course entity class and its attribute class Course.Diff. The Takes relationship and its attribute class Takes.Grade is shown as a table containing the foreign keys Student and Course. The arc classes and their constraints are drawn just as they are in the DAPER model.

4 Probabilistic Entity-Relationship Models

We now examine DAPER models in detail. After reviewing the fundamentals, we discuss the representation of restricted relationships, self relationships, and probabilistic relationships.

In what follows, we use the following conventions in our notation. We use either capitalized friendly names (e.g., Student, Course) or tokens (e.g., E) for entity classes. We use non-capitalized friendly names or abbreviations (e.g., student[Grade], s) for corresponding entities. Similarly, we use capitalized friendly names (e.g., Takes) or tokens (e.g., R) for relationship classes. We use (e.g.) $R(s, c)$ to say that entities s and c are a relationship associated with the relationship class R. We use X to refer to an arbitrary class when the distinction between an entity and relationship class is unimportant. We use expressions such as $X.A$ to represent a attribute class of class X, and $x.A$ to represent an (ordinary) attribute of entity x.

4.1 Fundamentals

A DAPER model can be viewed as a macro language—a language that, given a skeleton, expands to a DAG model. We use the term *ground graph* to refer to the structure of the DAG model created by the expansion of a DAPER model given a skeleton. An important part of this expansion is the drawing of arcs in the ground graph. Because the DAPER model is so compact, a mechanism is needed to constrain the drawing of arcs. Without such a mechanism, important conditional independence relations could not be expressed. As we have seen, this mechanism in a DAPER model takes the form of constraints on arc classes. To better understand how these constraints work, consider the following four related examples.

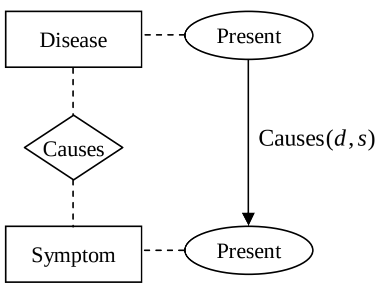
Example 3 A database contains diseases and symptoms for a given patient. Every disease is a potential cause of every symptom.

The DAPER model for this example is shown in Figure 5a. The entity classes Disease and Symptom have attribute classes Disease.Present and Symptom.Present, respectively, and there are no relationship classes. In the diagram, the arc class from Disease.Present to Symptom.Present has no constraint. Because there is no constraint, the ground graph generated by the application of this DAPER model to any given skeleton is a full bipartite graph. The bipartite graph generated by the DAPER model applied to a skeleton in which there are three diseases and three symptoms is shown in Figure 5b.

We give this example first to emphasize that arc classes need not have constraints. Now, let us see what happens when we include such constraints.

Example 4 Extending Example 3, suppose a physician has identified the possible causes of each symptom.

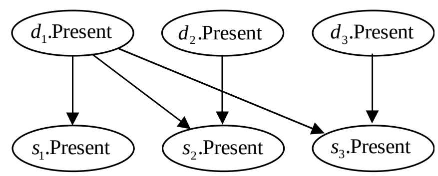
The DAPER model for Example 4 is shown in Figure 6a. With respect to the model in Figure 5a, there is now the relationship class Causes, where Causes(d, s) is true if the physician has identified disease d as a possible cause of symptom s. Also new is the constraint Causes(d, s) on the arc class. This constraint says that, when we expand the DAPER model to a DAG model given a skeleton, we draw an arc from d.Present to s.Present only when



(a)

Causes	
Disease	Symptom
d_1	s_1
d_1	s_2
d_1	s_3
d_2	s_2
d_3	s_3

(b)



(c)

Figure 6: (a) A DAPER model for incomplete bipartite graph of diseases and symptoms. (b) A possible skeleton identifying diseases, symptoms, and potential causes of symptoms. (c) A DAG model resulting from the expansion of the DAPER model to the skeleton.

Causes(d, s) holds. Note that, in the diagram we use "d" and "s" to refer to the entities associated with Disease.Present and Symptom.Present, respectively. In what follows, we will continue to make strong abbreviations as in this example, although such abbreviations are not required and may be undesirable for computer implementations of the PER language.

In the next two examples, we consider more complex constraints.

Example 5 Extending Example 3 in a different way, suppose the physician has identified both primary (major) and secondary (minor) causes of disease.

The DAPER model for Example 5 is shown in Figure 7a. There are now two relationship classes—Primary (1^o) Causes and Secondary (2^o) Causes—between the two entity classes, and the constraint is a disjunctive one: $1^o\text{Causes}(d, s) \vee 2^o\text{Causes}(d, s)$. This constraint

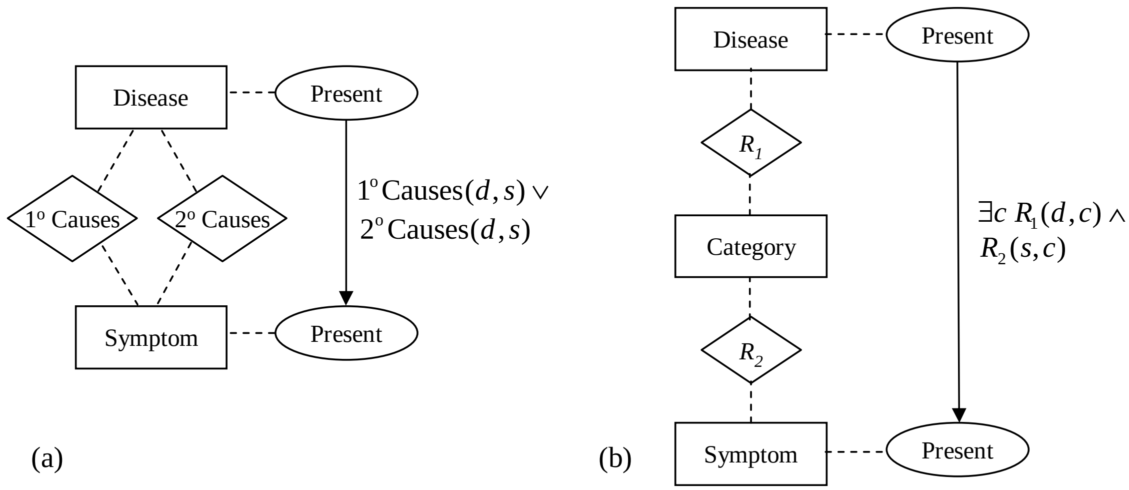


Figure 7: (a) A disjunctive constraint. (b) A constraint containing the existence quantifier.

says that, when the DAPER model is expanded to a DAG model given a skeleton, an arc is drawn from d.Present to s.Present only when d is a primary and/or secondary cause of s.

Example 6 Extending Example 3 in a different way, suppose that both diseases and symptoms have category labels—labels drawn from the same set of categories. The possible causes of a symptom are diseases that have at least one category in common with that symptom.

The DAPER model for this example is shown in Figure 7b. Here, we have introduced a third entity class—Category—whose entities have relationships with Disease and Symptom. In particular, $R1(d, c)$ holds when disease d is in category c; and $R2(s, c)$ holds when symptom s in category c. In this model, the arc class has the constraint $\exists c R1(d, c) \wedge R2(c, s)$, where c is an arbitrary entity in Category. Thus, when the DAPER model is expanded to a DAG given a skeleton, an arc will be drawn from d.Present to s.Present only when d and s share at least one category.

To understand how constraints are written and used in general, consider a DAPER model with an arc class from $X.A$ to $Y.B$. When this model is expanded to a ground graph given a skeleton, depending on the constraint, we might draw an arc from $x.A$ to $y.B$ for any x and y in the skeleton. To determine whether we do, we look at the tail and head entities associated with this putative arc. The *tail entities* of the putative arc from $x.A$ to $y.B$ is the set of entities associated with x. If X is an entity class, then the tail entity is just the entity x. If X is a relationship class, then the tail entities are those entities in the relationship tuple x. Similarly, the *head entities* of this arc is the set of entities associated with y. For example, given the DAPER model and skeleton in Figure 3 for the university database, the tail and head entities of the putative arc from john.IQ to Takes(john,cs107).Grade are (john) and (john,cs107), respectively. A *constraint* on the arc class from $X.A$ to $Y.B$ in a DAPER model is any first-order expression involving entities and relationship classes in the DAPER model such that the expression is bound when the tail and head entities are taken to be constants. To determine whether we draw an arc from $x.A$ to $y.B$, we evaluate the first-order expression using the tail and head entities of the putative arc. It must evaluate to true or false. We draw the arc from $x.A$ to $y.B$ only if the expression is true. Continuing with the same university database example, let us determine whether to draw an arc from john.IQ to Takes(john,cs107).Grade. The relevant constraint—"student[IQ] = student[Grade]"—references the tail entity student[IQ] = john and the head entity student[Grade] = john. Thus, the expression evaluates to true and we draw the arc.

Next, let us consider the local distribution class. A *local distribution class* for attribute class $X.A$ is any specification from which the local distributions for attribute $x.A$, for any entity or relationship x in class X, may be constructed. In Figure 3c, each attribute for a student's grade in a course has two parents—one attribute corresponding to the difficulty of the course and another corresponding to the IQ of the student. Consequently, the local distribution class for Takes.Grade in the DAPER model can be a single (ordinary) local distribution. In general, however, a more complicated specification is needed. For example, in the ground graph of Figure 6c, the attribute s_1.Present has one parent, whereas the attributes s_2.Present and s_3.Present have two parents. Consequently, the local distribution class for Symptom.Present must be something more than a single local distribution.

In general, a local distribution class for $X.A$ may take the form of an enumeration of local distributions. In our example, we could specify a local distribution for every possible parent set of s.Present for every symptom s in every possible skeleton. Of course, such enumerations are cumbersome. Instead, a local distribution class is typically expressed as a canonical distribution such as noisy OR, logistic, or linear regression. Friedman et al. (1999) refer to such specifications as *aggregators*.

So far, we have considered only DAPER models in which all attributes derive from attributes classes. In practice, however, it is often convenient to include (ordinary) attributes in a DAPER model. For example, in a Bayesian approach to learning the conditional probability distribution of Takes.Grade given Student.IQ and Course.Diff in Example 2, we may add to the DAPER model an ordinary attribute θ corresponding to this uncertain distribution as shown in Figure 8a. (If Grade is binary, e.g., θ would correspond to the parameter of a binomial distribution.) The ground graph obtained from this DAPER model applied to the skeleton in Figure 8b is shown in Figure 8c. Note that the attribute θ appears only once in the ground graph and that, because there is no annotation on the arc class from θ to Takes.Grade, there is an arc from θ to each grade attribute.

Although this view makes DAPER models easy to understand, formally, we do not allow such models to contain (ordinary) attributes. Instead, we specify that, for any DAPER model, (1) there is an entity class—Global—that is not drawn; (2) for any skeleton, this entity class has precisely one entity; and (3) every attribute class not connected explicitly to some visible entity class is connected to Global. This view is equivalent to the informal one just presented, but leads to simpler definitions and notation in our formal treatment of DAPER models in the Appendix.

4.2 Restricted Relationships

We now consider restricted relationships or, more precisely, restricted relationship classes. *A relationship class R in an ER (or PER) model is restricted* when some skeletons for the entity and relationship classes of the ER model are prohibited. In practice, many ER models contain restricted relationship classes; and graphical notation has been developed for common restrictions (e.g., Ullman and Widom, 2002). Similarly, restricted relationship classes are an extremely useful tool for modeling with PER models. In this section, we consider several examples.

Example 7 A binary outcome O is measured on patients in multiple hospitals. Each patient is treated in exactly one hospital. It is believed that outcomes in any given hospital h are i.i.d. given binomial parameter $h.\theta$; and that these binomial parameters are themselves i.i.d. across hospitals given hyperparameters α.

A DAPER model for this example is shown in Figure 9a. Here, entity classes Patient and Hospital are related by the relationship class In. The ground graph for a skeleton containing m hospitals and n_i patients in hospital i is shown in Figure 9b. This ground graph is the DAG model (structure) of what is often called a hierarchical model in the Bayesian literature (e.g., Gelman, Carlin, Stern, and Rubin, 1995).

In this example, the relationship class In is restricted in the sense that (patient,hospital) pairs are many to one—each patient is in exactly one hospital. This restriction is represented graphically by a curved arrowhead on the edge from In to Hospital in Figure 9a. The curved arrowhead is a standard notation in the language of ER models (e.g., Ullman and Widom, 2002); and we adopt this same notation for PER models. In general, given an ER or PER model with relationship class R connecting entity classes $E_1, \ldots, E_n$, if know-

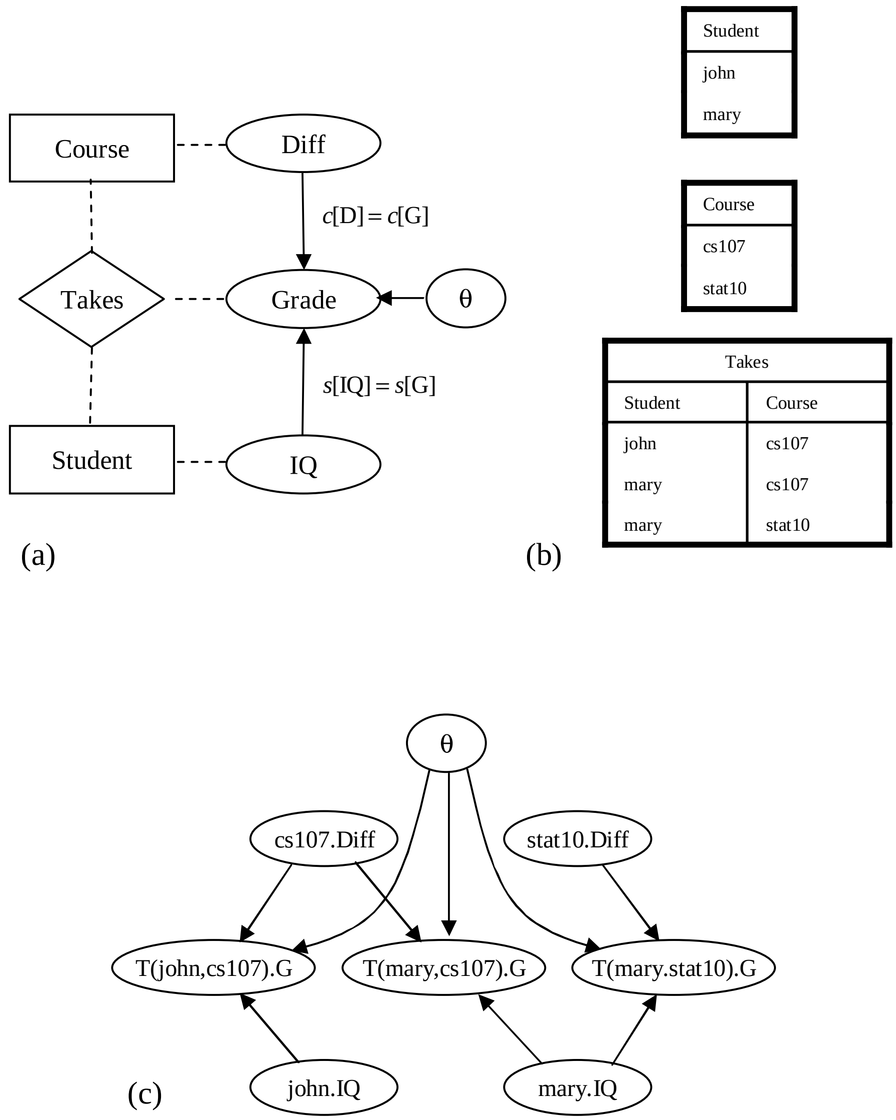


Figure 8: A modification to Figure 3 in which the local distribution for Takes.Grade given Student.IQ and Course.Diff is uncertain. (a) The DAPER model. (b) A skeleton (identical to the one in Figure 3). (c) The ground graph.

ing entities in classes $E_1, \ldots, E_{i-1}, \ldots, E_{i+1}, \ldots, E_n$ uniquely determines entity E_i for any allowed skeleton, then a curved arrowhead is attached to the edge from R to E_i.

Note that, due to the many-to-one restriction in this problem, we could equivalently attach the attribute class O to In rather than to Patient. A DAPER model equivalent to the one in Figure 9a is shown in Figure 9c.

Example 8 The occurrence of words in a document are used to infer its topic. The occurrence of words are mutually independent given document topic. Document topics are i.i.d. given multinomial parameters θ_t. The occurrence of word w in a document with topic t is i.i.d. given t and binomial parameters $\theta_{w|t}$.

This example is commonly referred to a binary Naive-Bayes classification (e.g., McCallum and Nigam, 1998). A DAPER model for this problem is shown in Figure 10. The entity classes Document and Word are related by the single relationship class F. The attribute classes are Document.Topic representing the topic of a document, Word.$\theta_{w|t}$ representing the set of binomial parameters $\theta_{w|t}$ for a word, and F(d, w).In representing whether word w is in document d. The relationship class F is restricted to be a `Full` relationship class. That is, in any allowed skeleton, all pairs (document,word) must be represented.[4] We indicate this restriction on the DAPER diagram by placing the annotation `Full` next to the relationship class. As we shall see in what follows, the `Full` restriction is useful in many situations.

4.3 Self Relationships

Self relationships are relationships that relate like entities (and perhaps other entities as well). *A self-relationship class* is one that contains self relationships. Examples of self relationship classes are common in databases: people are managers of other people, cities are near other cities, timestamps follow timestamps, and so on. ER models can represent self relationships in a natural manner. The extension to PER models is also straightforward, as we illustrate with the following three examples.

Example 9 In the university-database example (Example 2), a student's grade in a course depends on whether an advisor of the student is a friend of a teacher of the course.

The ER model for the data in this example is shown in Figure 11a. With respect to the ER model in Figure 2a, Professor is a new entity class and Advises, Teaches, and F are

[4] In a practical database implementation, this relationship would be encoded sparsely, despite the Full restriction. That is, relationship (d, w) would be stored in the database only when word w appears in document d.

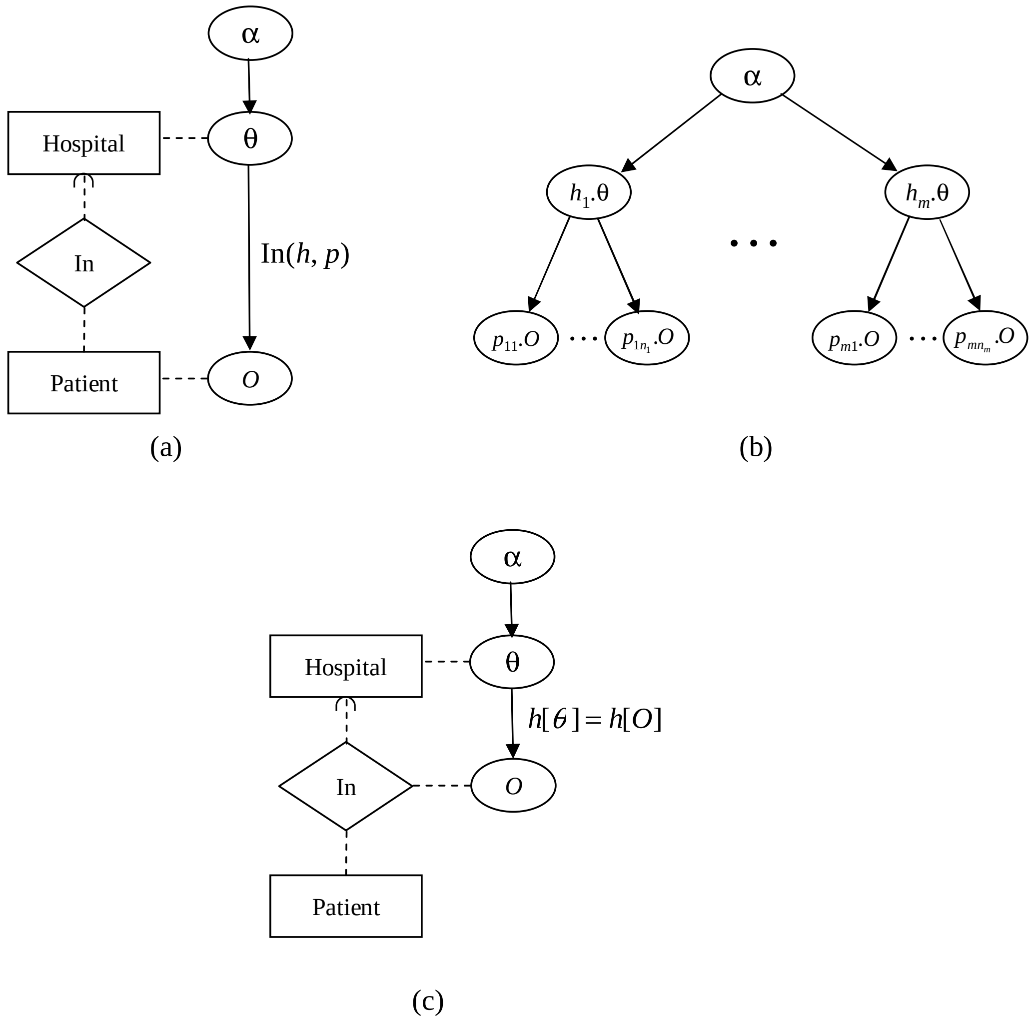


Figure 9: (a) A DAPER model for patient outcomes across multiple hospitals (Example 7). (b) The ground graph (a hierarchical model structure) for a skeleton containing m hospitals and n_i patients in hospital i applied to the DAPER model in (a). (c) A DAPER model equivalent to the one in (a).

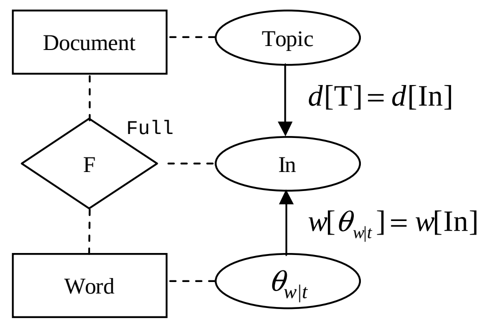


Figure 10: A DAPER model for binary Naive-Bayes document classification.

new relationship classes. Advises(p, s) means that professor p is an advisor of student s. Teaches(p, c) means that professor p teaches course c. (Students may have more than one advisor; and courses may have more than one teacher.)

The relationship class F is introduced to model whether one professor is a friend of another. F is our first example of a self-relationship class—it contains relationships between professor pairs. The two dashed lines connecting F and the Professor entity class in the diagram indicate that F is a self-relationship class. F has one attribute class F.Friend, where the attribute F(p, p_f).Friend is true if professor p_f is a friend of professor p. Note that F has the `Full` constraint so that we can model whether any one professor is a friend of another. Also note that F(p_1, p_2).Friend may be true while F(p_2, p_1).Friend may be false.

The DAPER model for this example, including the new probabilistic relationship between F.Friend and Takes.Grade, is shown in Figure 11b. The constraint on the arc class from F.Friend to Takes.Grade is Teaches$(p, c) \wedge$ Advises(p_f, s). Thus, in any ground graph generated from this model, there is an arc from attribute F(p, p_f).Friend to attribute Takes(s, c).Grade whenever a teacher of the course is p and an advisor of the student is p_f—precisely the additional dependence described in the example.

In the diagram, note that the relationship class F has the label "F(p, p_f)". The ordered pair (p, p_f) following F is introduced to unambiguously identify the different *roles* of the entity class in the self relationship. In this case, "p" and "p_f" refer to the roles of professor and professor's friend, respectively. This added notation in DAPER models is needed for the unambiguous specification of constraints. For example, suppose we had written the constraint on the arc class from F.Friend to Takes.Grade as Teaches$(p_f, c) \wedge$ Advises(p, s). This constraint means something different than the previous one—namely, that the student's grade depends on whether the student's *teacher* is a friend of the course's *advisor*.

Although not a standard convention for ER models, we allow an alternative representation for self relationships. Namely, we allow entity classes participating in a self-relationship class to be copied. The DAPER model in Figure 11b drawn with this alternative convention is shown in Figure 11c. Here, there are two instances of the Professor entity class named "Professor (Teacher)" and "Professor (Advisor)." Note that copying allows us to annotate the role that each copy of the entity class plays in the self-relationship class. Models drawn with this copy convention are sometimes (but not always) more transparent. A similar convention is used in PRMs (e.g., Friedman et al., 1999).

Example 10 A hidden Markov model (HMM) has hidden attributes slice.H, observed attributes slice.X, and uncertain parameters θ_h and $\theta_{x|h}$.

A DAPER model for such an HMM is shown in Figure 12a. The only entity class in the model is Slice. Its entities correspond to the time slices in the HMM. The only relationship class in the model—Next—is a restricted, self-relationship class. Next(s, s_{+1}) holds precisely when time slice s_{+1} immediately follows time slice s. Thus, Next is an example of a relationship class whose constraint induces a total order on its entities. We use `Order` to annotate this restriction. The attributes H and X correspond to the hidden and observed attributes in the HMM, respectively. The attribute classes θ_h and $\theta_{x|h}$ (connected to the Global entity class, which is not shown) represent the uncertain distributions.

Because arc classes can have constraints, DAPER models may contain arc classes that are *self arcs*—arcs whose head and tail nodes are the same.[5] In this example, the self arc is used to represent the Markov chain of hidden attributes H. Another graphical model—Markov transition diagrams—use self arcs in the much the same way. When a self arc appears in a DAPER model, it is not clear which way to draw arcs when expanding the model to a DAG model. In our example, do we draw arcs from $s.H$ to $s_{+1}.H$, or in the opposite direction? To remove the ambiguity, we use bar–hat notation. In this example, the constraint is written Next$(\bar{s}, \hat{s}_{+1})$ indicating that the arc in drawn *from* $s.H$ *to* $s_{+1}.H$. In general, we use a bar and hat to denote head and tail entities, respectively.

When this DAPER model is expanded to a ground graph, the attribute $s_0.H$—where s_0 corresponds to the first time slice—has no parents. In contrast, the attribute $s.H$ where s corresponds to any other slice has one parent. Consequently, the local distribution class for Slice.H may be specified by two (ordinary) local distributions: $p(s_0.H)$ and $p(s_{i+1}.H|s_i.H)$ for $i > 0$.

A DAPER model using the copy convention for the HMM is shown in Figure 12b. Note that the attribute class Slice.X need be represented in only one copy of the entity class.

[5] We use the term "self arc" to refer both to arc classes and to arcs. The use will be clear from context.

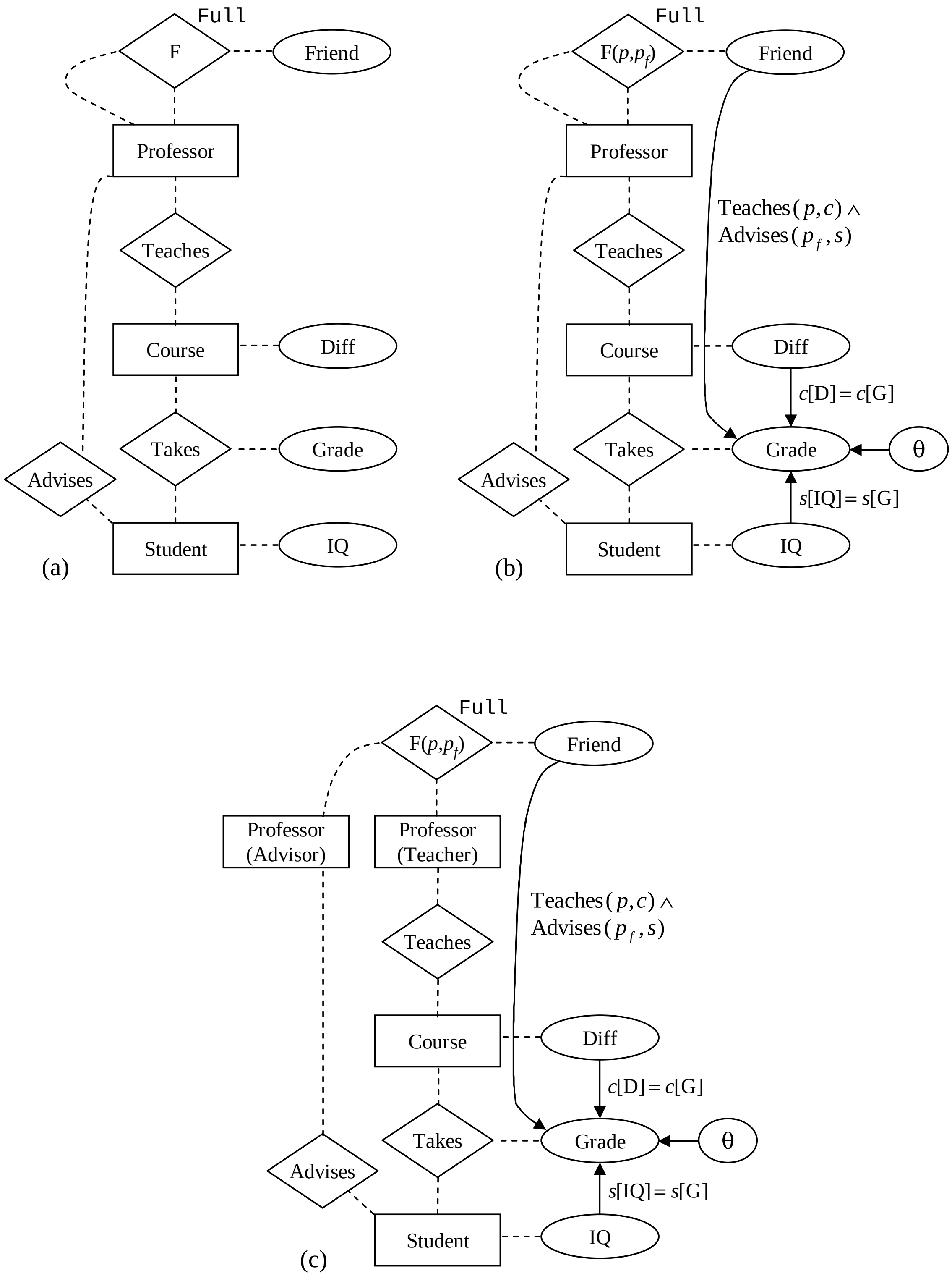


Figure 11: (a) An ER model showing Student, Course, and Professor entities and relationships among them. (b) A DAPER model showing that a student's grade in a course depends on whether the course's teacher likes the student's advisor. (c) The same model in (b) in which the Professor entity class has been copied.

The probabilistic dependencies between $s.H$ and $s.X$, for all slices s, are captured by the inclusion of X in one copy. Also note that, in this example and in any diagram where the copy convention is used, the bar–hat notation is not needed.

Example 11 A gene is transmitted through inheritance. The gene-allele frequencies θ are uncertain.

A DAPER model for this example is shown in Figure 13a. The model contains a single entity class Person and a single three-way, restricted, self relationship class Family. The relationship Family(p_c, p_m, p_f) holds when child p_c has mother and father p_m and p_f, respectively. The relationship class has the `2DAG` constraint, meaning that each child has at most two parents and can not be his or her own ancestor. The constraint on the single arc class indicates that only the gene of a child's mother and father influences the gene of the child. Note that the local distribution class for Gene has three components: (1) $p(\text{gene}|\text{no parents}) = \theta$, (2) $p(\text{gene}|\text{one parent})$, and (3) and $p(\text{gene}|\text{two parents})$. Figure 13b shows the same model in which the entity class Person appears three times.

When a DAPER model contains self relationships, its expansion can produce an invalid DAG model—in particular, one with a ground graph that contains directed cycles. For example, suppose we have a DAPER model where entity class E has a self relationship class R, and $E.A$ has a self arc with no constraint. Then when we expand this model given a skeleton containing $R(e, e)$, the ground graph will contain the self arc from $e.A$ to $e.A$. In general, we need to insure the ground graph is ayclic given all skeletons under consideration. In the Appendix, we describe sufficient conditions (including the absence of self relationships) that guarantee the acyclicity of ground graphs. In general, to determine whether DAPER model produces only acyclic ground graphs for a given set of skeletons, one can check each ground graph individually.

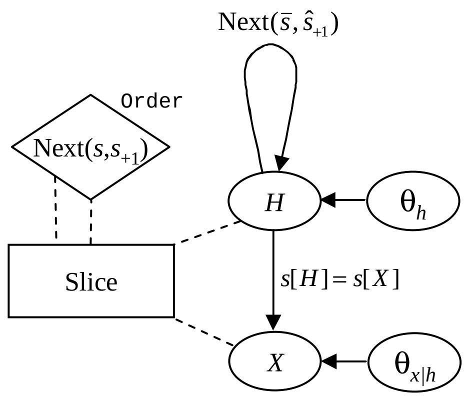
4.4 Probabilistic Relationships

In many situations, relationships may be uncertain or random. In this section, we consider several examples and how they are represented with DAPER models.

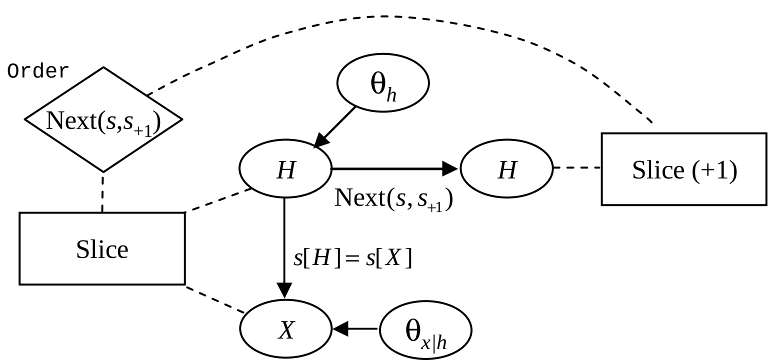
Example 12 (Relationship existence) A database contains academic papers and citations for a subset of those papers. Using the citations we have, we model how the topics of two papers influence whether one paper cites the other.[6]

If each paper in the database came with its citations, we could model this database with the ER model shown in Figure 14a. Here, the single (copied) entity class Paper has the

[6] We assume that citation lists for papers are missing at random.



(a)



(b)

Figure 12: (a) The DAPER-model representation of a hidden Markov model. (b) The same model in which Slice is copied.

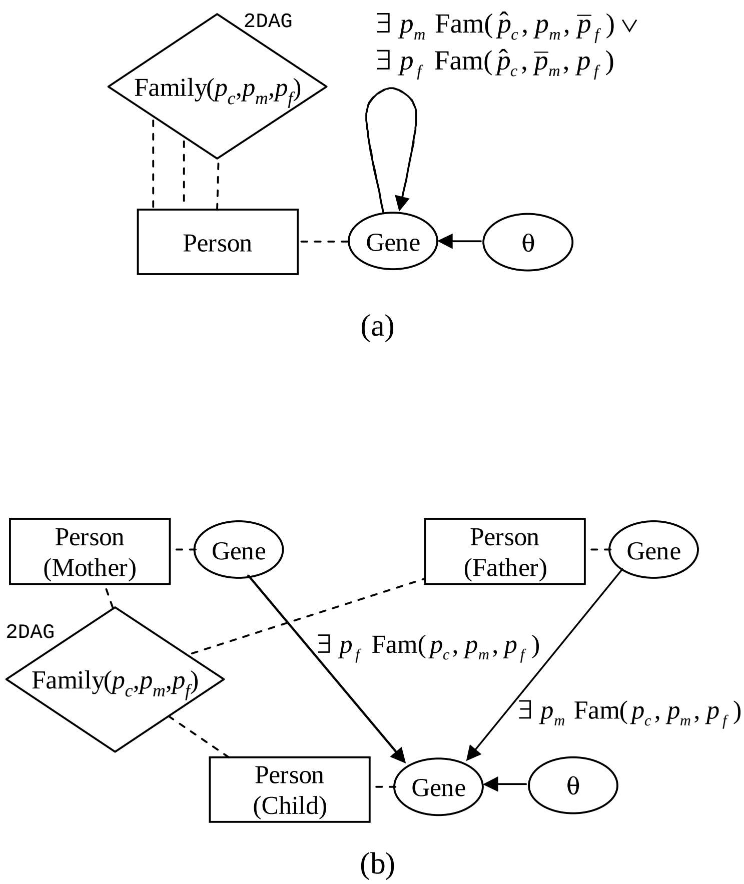


Figure 13: (a) The DAPER model for gene transmission through inheritance. (b) The same model in which Person is copied.

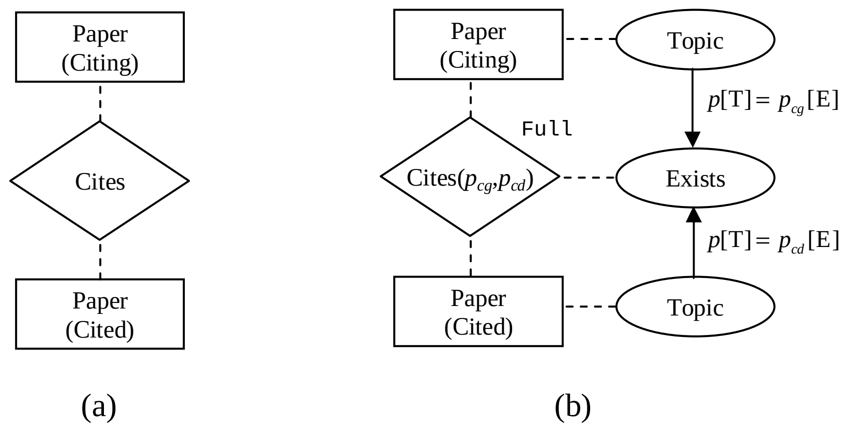


Figure 14: (a) An ER model for a citation database. (b) A DAPER model for the situation where citations are uncertain.

self-relationship Cites, where Cites(p_{cg}, p_{cd}) holds when p_{cg} is the citing paper and p_{cd} is the cited paper. In our example, however, we are uncertain about the citations of papers whose citations have not been recorded. That is, we are uncertain about the relationships in the relationship class Cites. To model this uncertainty, we use a DAPER model in which Cites is a `Full` relationship class with attribute class Cites.Exists, where Cites$(p_{cg}, p_{cd}).Exists$ is true when paper p_{cg} cites paper p_{cd}. In addition, to model how the topics of two papers influence this existence, we add the attribute class Paper.Topic and the arc classes as shown in Figure 14b.

In general, if we have a relationship class R that is uncertain, we model it in a DAPER model by making that relationship class `Full` and adding the attribute class R.Exists. Getoor et al. (2002) discuss this type of uncertainty under the name *existence uncertainty* and use a similar mechanism to represent it in PRMs.

In many situations, relationship classes can be both probabilistic and restricted. In the remainder of this section, we consider two examples.

Example 13 Modifying Example 12, we now know that the database was constructed such that contains at most ten citations from the bibliography of any paper.[7]

The DAPER model in Figure 15 shows the DAPER model for this example, where the Cites relationship class is both uncertain and restricted. As discussed in Section 2, we encode the restrictions using instantiated deterministic nodes. With respect to Figure 14b, we have

[7]We assume that citations above ten in number were censored at random.

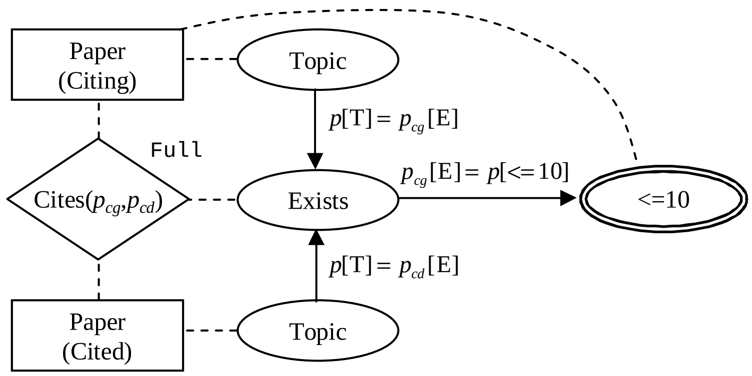


Figure 15: A DAPER model for the situation where citations are uncertain and limited to ten per paper.

added a binary, attribute class $Paper. <= 10$. The double oval associated with this attribute class indicates that this attribute expands to deterministic attributes in a ground graph. In particular, a ground graph attribute $p. <= 10$ will have parents Cites$(p_{cg}, p_{cd}).Exists$, for all p_{cd}, and will be true exactly when ten or fewer of these parents are true. To encode the restriction, we set $p. <= 10$ to true for every p when performing inference in the ground graph.

Example 14 (Partial Relationship Existence) Modifying Example 12 once again, the citation database now has a complete set of citations, but some of citations are so garbled that the identity of some of the cited papers are uncertain.

One way to think about this uncertainty is that the relationships Cites(p_{cg}, p_{cd}) are uncertain only in their second argument. Getoor et al. (2002) refer to this uncertainty as *reference uncertainty* and present a special mechanism for representing it in PRMs. We take an approach that, although equivalent to their method, uses only concepts that we have already discussed.

A DAPER model for this example is shown in Figure 16. With respect to the DAPER model in Figure 14b, we have added the entity class Cites, and the relationship classes R_1 and R_2 between Paper and Cites. An entity pair in Cites correspond to a citation—a citing and a cited paper. $R_1(p_{cg}, c)$ holds when paper p_{cg} is the citing paper in c, and $R_2(p_{cd}, c)$ holds when p_{cd} is the cited paper in c. The relationship class R_1 is a restricted (many-to-one) relationship class. In contrast, the relationship class R_2 is probabilistic relationship class, restricted to be `Full`. The uncertainty in this relationship class is encoded with the

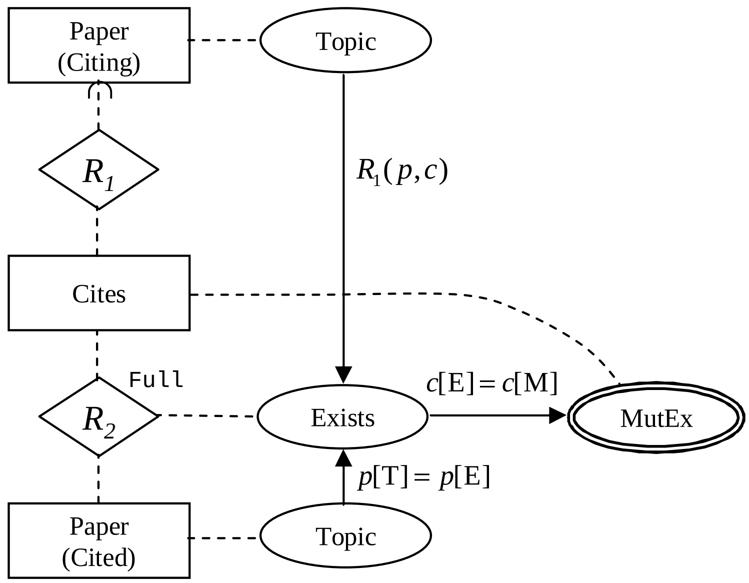


Figure 16: A DAPER model for the situation where only the cited papers are uncertain.

attribute class R_2.Exists, where $R_2(p_{cd}, c)$.Exists is true precisely when citation c cites paper p_{cd}. To model the restriction that the possible cited papers of c are mutually exclusive, we first introduce the deterministic, attribute class Cites.MutEx. In any ground graph obtained from this DAPER model, $c.MutEx$ will be true exactly when one of its parents $R_2(p_{cd}, c)$.Exists is true. For any inference we perform with the ground graph, we set $c.MutEx$ to true for every citation c.

5 Plate Models

In this section, we revisit our definition of the plate model, give examples, and describe how our definition differs from previously published examples.

As discussed in Section 3, we define the plate model by giving an invertible mapping from DAPER to plate model. Thus, the two model types are equivalent in the sense that they can represent the same conditional independence relationships for any given skeleton.

Summarizing the mapping from DAPER to plate model given in Section 3, entity classes are drawn as large named rectangles called plates; a relationship class for a set of entity classes is drawn at the named intersection of the corresponding plates; attribute classes are drawn inside the rectangle corresponding to its entity or relationship class; and arc classes and constraints are drawn just as they are in DAPER models. For example, as

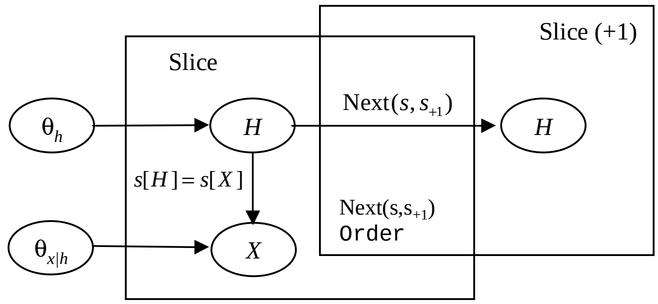


Figure 17: A plate model for an HMM corresponding to the DAPER model in Figure 12b.

we have discussed, the DAPER model in Figure 3a has the corresponding plate model in Figure 4a. As another example, the DAPER model for the HMM shown in Figure 12b has the corresponding plate model in Figure 17. Note that, because plate models represent relationship classes as the intersection of plates, plates (corresponding to entity classes) must be copied when the model contains self-relationship classes.

The plate model corresponding to the DAPER model for the patient-hospital example in Figure 3a is shown in Figure 18a. In this plate model, there are no attributes in the Patient plate outside the intersection. Thus, one can move the Patient plate fully inside the Hospital plate, yielding the diagram in Figure 18b. We allow this *nesting* in our framework. Furthermore, plates may be nested to arbitrary depth. This convention corresponds to one found in published examples of plate models.

There are three differences between plate models as we have defined them and *traditional plate models*—plates models as they have been described in the literature. In all three cases, our definition provides a more expressive language. One, in traditional plate models, a arc class emanating from a attribute class in a plate can not leave that plate. Given this constraint, any arc class from attribute class $E.X$ must point either to attribute class $E.Y$ or to attribute class $R.Y$, where R is nested inside E.

Two, when a traditional plate model is expanded to a ground graph, arcs are drawn only between attributes corresponding to the same entity. To be more precise, consider a plate model containing the arc class from $E.X$ to $E.Y$. In a traditional plate model, the arc class implicitly has the constraint $e[X] = e[Y]$. Similarly, consider a plate model containing the arc class from $E.X$ to $R.Y$ where R is nested inside E, possibly many levels deep. Because R in nested inside E, for any relationship $r \in R$, the entities associated with r must uniquely determine an $e \in E$. Let $\mathbf{r}(e)$ be the set of the relationships r that uniquely determine e. Now, when this traditional plate model is expanded to a ground graph, arcs are drawn

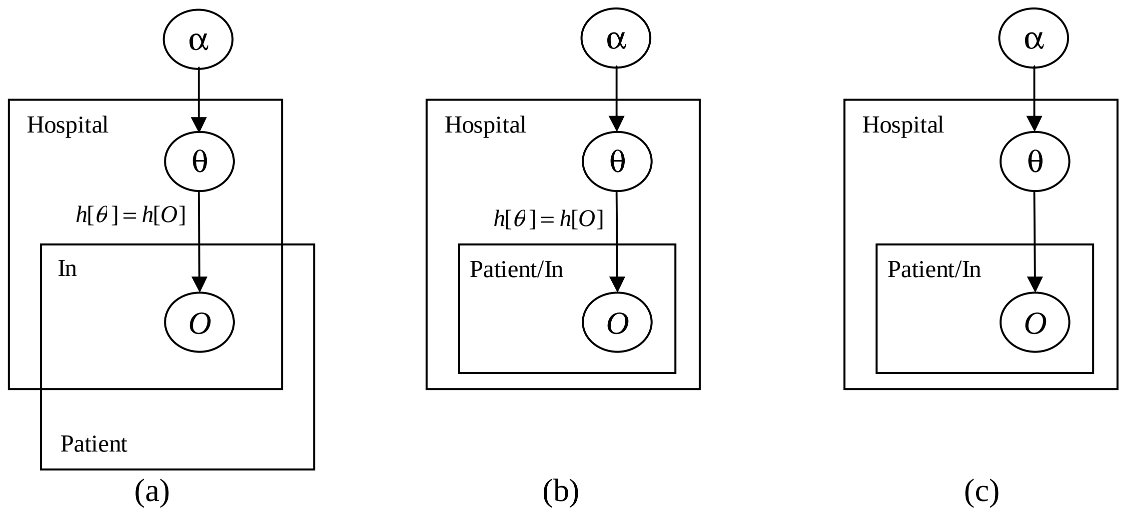


Figure 18: (a) A plate model corresponding to the DAPER model in Figure 12a. (b) An equivalent plate model illustrating the graphical convention of nesting. (c) A traditional plate model, equivalent to the one in (b), in which the constraint $h[\theta] = h[O]$ is implicit.

from $e.X$ to $r.Y$ only when $r \in \mathbf{r}(e)$. As an example, consider Figure 18c, which shows the traditional plate model for the patient–hospital example. Here, E=Hospital, R=In, and $\mathbf{r}(h) = \cup_p\{(h,p)\}$ for all hospitals h. Thus, the arc class from Hospital.θ to In$(h,p).O$ has the constraint $h[\theta] = h[O]$. This constraint is implicit (see Figure 18c).

Three, traditional plate models contain no arc-class constraints other than the implicit ones just described.

The DAPER and plate model (as we have defined them) are equivalent. Nonetheless, in some situations, a DAPER model may be easier to understand than an equivalent plate models, and vice versa. When there are many entity and relationship classes (plates and intersections), DAPER models are often easier to understand. In particular, drawing intersections when there are many plates can be difficult (although not impossible; see Gil, Howse, Kent, and Taylor, 2000). In contrast, when there are few entities and the nesting convention can be used, plates are often easier to understand.

6 Probabilistic Relational Models (PRMs)

In this section, we examine directed PRMs.

Recall that, as in the case of the plate model, we have specified an invertible mapping from a DAPER model to a directed PRM. Thus, DAPER models, plate models, and directed PRMs are equivalent. As described earlier, the mapping from a DAPER to directed PRM takes place in two stages: the ER-model component of the DAPER model is mapped to a relational model, and then the probabilistic component of the DAPER model is mapped to the directed PRM. In the first stage, entity classes are mapped to tables; relationship classes are mapped to tables with foreign keys making the connections to entities; and attribute classes are mapped to attributes (columns) in relational tables. In the second stage, arc classes and constraints are drawn just as they are in DAPER model.

There is one important difference between the directed PRM by our definition and the *traditional PRMs* as defined by Friedman et al. (1999). The difference is not in the relational-model component. This component for a PRM and traditional PRM are identical. Rather, the difference lies in how the probabilistic component is specified. In our PRM, the probabilistic component is a graphical augmentation of the relational model. In a traditional PRM, the probabilistic component takes the form of a list of arc classes. To illustrate this difference, compare the PRM in Figure 4b with the corresponding traditional PRM in Figure 19. In the latter figure, the arc classes pointing to Takes.Grade are specified in a separate list consisting of Takes.Course.Diff $\rightarrow$ Takes.Grade and Takes.Student.IQ $\rightarrow$ Takes.Grade.

The terms Takes.Course.Diff and Takes.Student.IQ are examples of what Friedman et al. (1999) call *slot chains*. In general, a slot chain is a sequence of foreign key (or inverse foreign key) references. The linear nature of slot chains makes them less expressive than the first-order constraints in (our) PRMs. For example, in Example 9 where a student's grade in a course depends on whether the course's teacher likes the student's advisor (Example 9), there are two "relationship paths" from F.Friend to Student.Grade: one through Advises and one through Takes. This double path cannot be represented by a slot chain.

In practice, we find both DAPER models and PRMs easy to understand. Database designers who prefer ER models over relational models may prefer DAPER models over PRMs, and vice versa. We note, however, that the purpose of DAPER models and PRMs is not the implementation of mechanisms for data storage, but rather the modeling of probabilistic dependencies. Consequently, even those who prefer to design databases with relational models may prefer the DAPER model for probabilistic modeling, as DAPER models elevate the relationship class to a first-class object in the language.

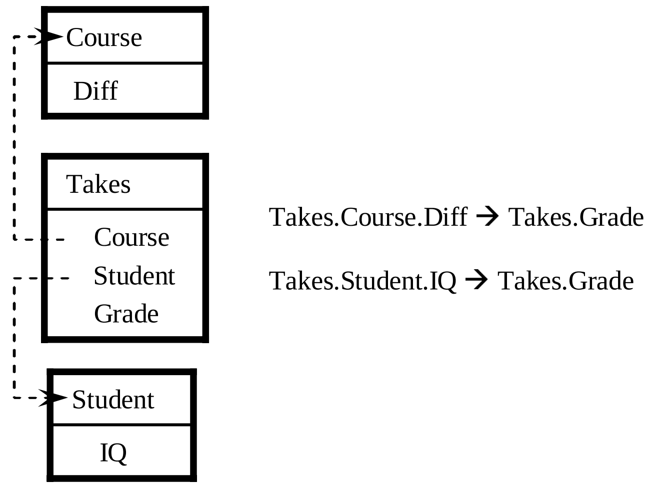


Figure 19: A traditional PRM corresponding to the model in Figure 4b.

7 Extensions and Future Work

In this paper, we have concentrated on the DAPER model, a model that expands into a DAG model given a skeleton. In this section, we examine classes of PER models that expand into graphical models other than traditional DAG models. Many of the ideas here are preliminary and provide opportunities for future work.

An important limitation of traditional graphical models is their inability to represent context-specific independence. An example of such independence is the pair of independencies (1) X and Z are independent given $Y = 0$, and (2) X and Z are dependent given $Y = 1$. Many extensions to graphical models have been developed that can represent particular classes of context-specific independence including decision-tree/DAG-model hybrids (e.g., Boutlier, Friedman, Goldszmidt, and Koller, 1996), contingent DAG models (Fung and Shachter, 1990), and similarity networks (e.g., Heckerman, 1991).

Let us consider a variation on contingent DAG models that uses notation slightly different from that in Fung and Shachter (1990). To understand this model class, consider the context-specific independence described in the previous paragraph: X and Z are independent given $Y = 0$, but dependent given $Y = 1$. Figure 20 shows a contingent DAG model (structure) for this independence. This contingent DAG model has a *state constraint* on the arc from Y to Z that reads $X = 1$. This constraint means that there is a dependence of Y on Z only when $X = 1$. In general, state constraints in contingent DAG models function much the way constraints do in DAPER models. In DAPER models, constraints are first-

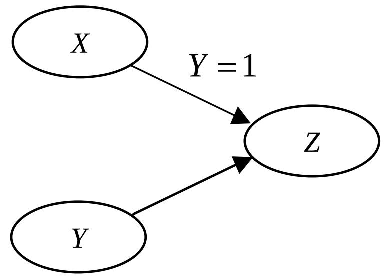


Figure 20: A contingent DAG model (structure) showing the context-specific independence X and Z are independent given $Y = 0$, but dependent given $Y = 1$.

order expressions over entities that control the expansion to a DAG model. In contingent DAG models, state constraints are Boolean expressions over attribute–states that control the expression of conditional independence.

Now consider the contingent DAPER model—a model that expands to a contingent DAG model. The model is identical to an ordinary DAPER model, except that arc classes are now annotated with an order pair. The first component of the ordered pair is a constraint just as is found in the ordinary DAPER model. The second component is a *state constraint class* that specifies the state constraints to be written during the expansion to a contingent DAG model. The state constraint class is a Boolean expression over attribute-states that may take head and tail entities as arguments.

Example 15 (Identity Uncertainty) We have video images of multiple cars of different colors. We know how many cars there are and have zero or more observations of each car's color, but we are uncertain about what observations go with what cars.

Pasula and Russell (2001) describe this example as having *identity uncertainty.* We can represent this example using the contingent DAPER model in Figure 21a. The two entity classes, Car and Observation, are related by the relationship class Of, where $\text{Of}(o, c)$ holds when observation o corresponds to car c. The probabilistic relationship Of has the many-to-one restriction: an observation is associated with exactly one car. As in previous examples, the many-to-one restriction is represented by the `Full` relationship class Of, together with the attribute class Of.Exists and the deterministic node MutEx (which is set to true). The arc class from Car.Color to Observation.Color is annotated with the ordered pair $(\text{Of}(o, c), \text{Of}(o, c).\text{Exists} = \text{true})$. The first component says that we draw an arc from c.Color to o.Observation only when $\text{Of}(o, c)$ is true. (In this case, this constraint is vacuous because the relationship class F is `Full`.) The second component says that, when we draw

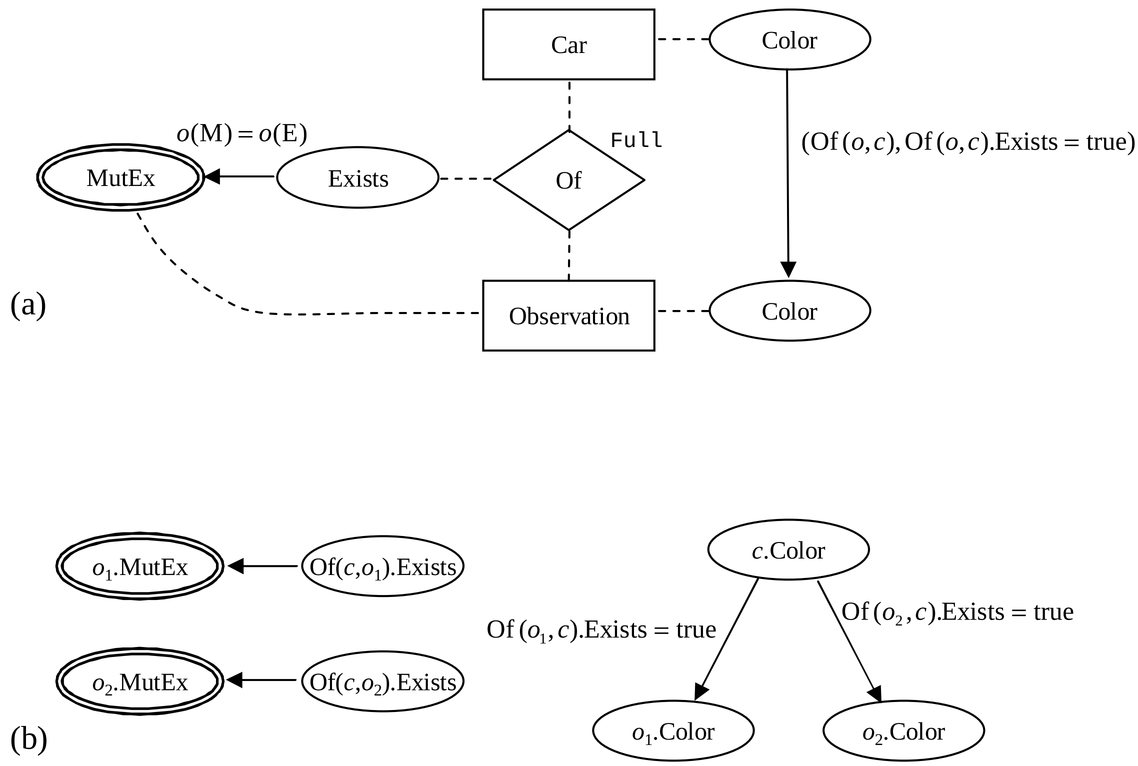


Figure 21: (a) A contingent DAPER model for Example 15, and example of identity uncertainty. (b) A contigent DAG model resulting from the expansion of the model in (a) given a skeleton containing one car and two observations.

such an arc, we add to it the state constraint $\text{Of}(o, c).\text{Exists} = \text{true}$. Figure 21b shows the expansion of this contigent DAPER model to a contingent DAG model for a skeleton containing one car and two observations. Note that, because there is only one car, the MutEx nodes are redudant and can be omitted.

In this example, we know how many cars there are. If we do not, we can place a probability distribution on the number of cars and stipulate that the DAPER model in Figure 21a should be applied to each possible number of cars.

Let us now discuss possibilities for relational modeling with undirected models. A commonly used (non-relational) undirected model is the undirected graphical (UG) model. This model class has more than one definition—definitions that coincide only for positive distributions (Lauritzen, 1996). Here, we define an UG for attributes $\mathbf{X}$ with joint distribution $p(\mathbf{x})$ as a model having two components: (1) an undirected graph (the model structure) whose nodes are in one-to-one correspondence with $\mathbf{X}$, and (2) a collection of non-negative clique functions $\phi_m(\mathbf{x}_m)$, $m = 1, \ldots, M$, where m indexes the maximal cliques of the graph and $\mathbf{X}_m$ are the attributes in $\mathbf{X}$ in the mth maximal clique, such that

$$p(\mathbf{x}) = c \prod_{m=1}^{M} \phi_m(\mathbf{x}_m), \tag{3}$$

The term c is a normalization constant. As is the case for the DAG model, the UG model for $\mathbf{X}$ defines the joint distribution for $\mathbf{X}$. The clique functions are sometimes called *potentials*.

An UG model for (X, Y, Z) is shown in Figure 22a. The graph has a single maximal clique consisting of all three attributes, and hence represents an arbitrary distribution for these attributes.

A related but more general undirected model is the hierarchical log-linear graphical (HLLG) model. An HLLG model is a model having two components: (1) a undirected hypergraph (the model structure) whose nodes are in one-to-one correspondence with $\mathbf{X}$, and (2) a collection of potentials $\phi_h(x_h)$, $h = 1, \ldots, H$, where h indexes the hyperarcs of the graph and $\mathbf{x}_h$ are the attributes in $\mathbf{X}$ of the hth hyperarc, such that

$$p(\mathbf{x}) = c \prod_{h=1}^{H} \phi_h(\mathbf{x}_h). \tag{4}$$

Again, an HLLG model for $\mathbf{X}$ defines the joint distribution for $\mathbf{X}$. In this paper, we represent a hyperarc as a triangle connecting multiple nodes with undirected edges. For example, Figure 22b shows an HLLG model with a single hyperedge.

By virtue of Equations 3 and 4, both UG and HLLG model structures define factorization constraints on distributions. In this sense, HLLG models are more general than UG models. That is, given any UG model structure, there exists an HLLG model structure that can encode the same factorization constraints, but not vice versa. For example, the UG structure in Figure 22a has the equivalent HLLG model structure shown in Figure 22b. In contrast, the HLLG model structure shown in Figure 22c, encodes the factorization constraint

$$p(x, y, z) = c\ \phi_1(x, y)\ \phi_2(y, z)\ \phi_3(x, z),$$

which cannot be represented by an UG model structure. Also, we note that the factorization constraints of any HLLG model can be encoded with a factor-graph model (Kschischang, Frey, Loelinger, 01) in which all potentials are non-negative.

Turning to relational modeling, let us consider the hierarchical log-linear probabilistic entity-relationship (HELPER) model. A model in this class expands into an HLLG model. Like the DAPER model, a HELPER model is an extension of an ER model. In contrast to the DAPER model, the probabilistic component of a HELPER model is expressed as hyperedge classes and potential classes on those hyperedges. Hyperedge classes are expanded to

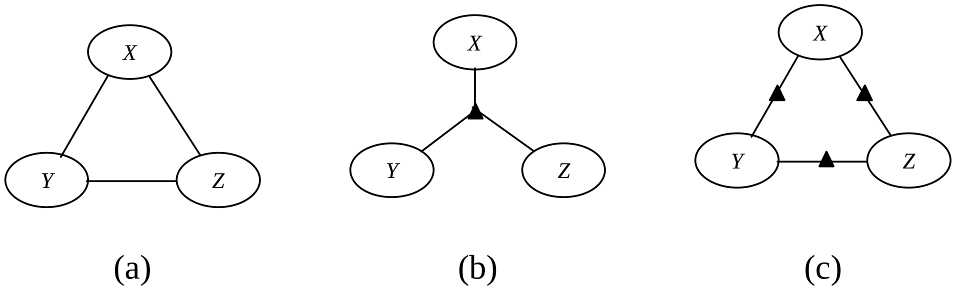


Figure 22: (a) An UG model structure. (b) An equivalent HLLG model structure. (c) An HLLG model that encodes pairwise interactions.

an hyperedges according to constraints. These constraints, in turn, may be any first-order expression that is bound given the entities associated with the endpoints of the hyperedge.

Example 16 An arbitrary hierarchical log-linear graphical model with at most two-way interactions.

The HELPER diagram for this example is shown in Figure 23a. There is a single entity class Variable corresponding to the attributes in the hierarchical log-linear model, a single attribute class X, and a single self relationship class Neigh, where Neigh(v_1, v_2) if $v_1.X$ and $v_2.X$ have a pairwise interaction. The only hyperedge class in the model is a self edge that connects Variable.X with itself. The constraint on this hyperedge class is such that $v_1.X$ and $v_2.X$ will be neighbors in the ground graph only when Neigh(v_1, v_2) holds. Note that the Neigh relationship class is restricted to be upper triangular so that the expanded graph has no self arcs and has at most one arc between any two attributes.

A sample skeleton for three attributes and the resulting hierarchical log-linear model is shown in Figures 23a and b, respectively.

Whereas HLLG models have a natural relational counterpart, UG models do not. To understand this point, imagine a PER model that expands to an UG model. Such a model would need a mechanism for specifying potentials in the ground graph. Such potentials, however, are not defined until the maximal cliques of the ground graph are determined, and these cliques will depend on the skeleton used to expand the PER model.

Finally, there are numerous classes of graphical models that we have not yet explored, including mixed directed and undirected models (e.g., Lauritzen, 1996), directed factor-graph models (e.g., Frey, 2003), influence diagrams (e.g., Howard and Matheson, 1981), and dependency networks (e.g., Heckerman, Chickering, Meek, Rounthwaite, and Kadie,

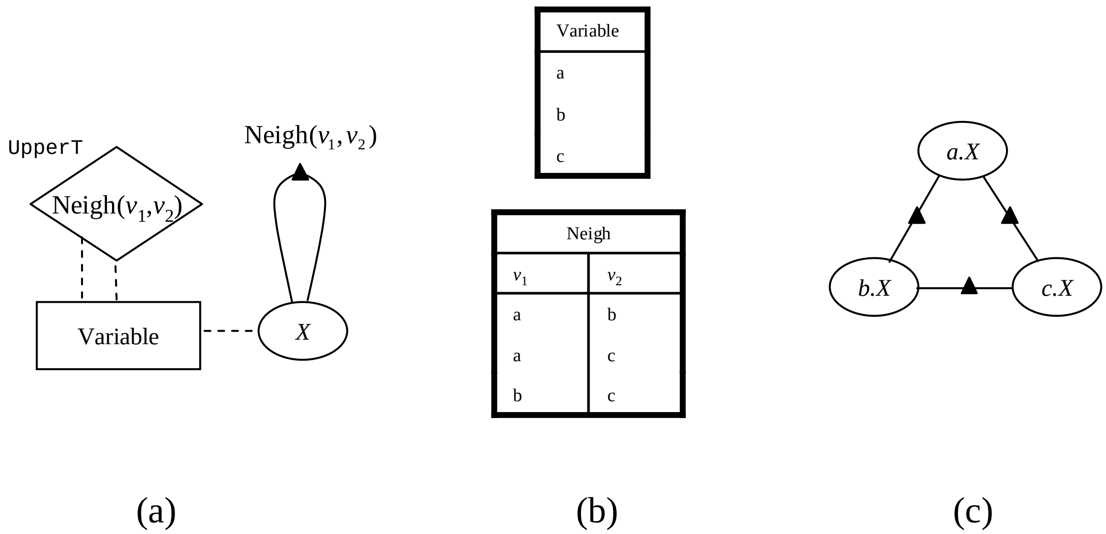


Figure 23: (a) An HELPER model for an arbitrary hierarchical log-linear model with at most two-way interactions. (b) An example skeleton. (c) The hierarchical log-linear model resulting from the model in (a) applied to the skeleton in (b).

2000). The development of PER models that expand to models in these classes also provide opportunities for research.

Acknowledgments

We thank David Blei, Tom Dietterich, Brian Milch, and Ben Taskar for useful comments.

References

[Besag, 1974] Besag, J. (1974). Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society, B*, 36:192–236.

[Boutlier et al., 1996] Boutlier, C., Friedman, N., Goldszmidt, M., and Koller, D. (1996). Context-specific independence in Bayesian networks. In *Proceedings of Twelfth Conference on Uncertainty in Artificial Intelligence,* Portland, OR, pages 115–123. Morgan Kaufmann.

[Buntine, 1994] Buntine, W. (1994). Operations for learning with graphical models. *Journal of Artificial Intelligence Research*, 2:159–225.

[Frey, 2003] Frey, B. (2003). Extending factor graphs so as to unify directed and undirected graphical models. In *Proceedings of Nineteenth Conference on Uncertainty in Artificial Intelligence,* Acapulco, Mexico, pages 257–264. Morgan Kaufmann.

[Friedman et al., 1999] Friedman, N., Getoor, L., Koller, D., and Pfeffer, A. (1999). Learning probabilistic relational models. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence,* Stockholm, Sweden, pages 1300–1309. International Joint Conference on Artificial Intelligence.

[Fung and Shachter, 1990] Fung, R. and Shachter, R. (1990). Contingent belief networks.

[Gelman et al., 1995] Gelman, A., Carlin, J., Stern, H., and Rubin, D. (1995). *Bayesian Data Analysis*. Chapman and Hall.

[Getoor et al., 2002] Getoor, L., Friedman, N., Koller, D., and Pfeffer, A. (2002). Learning probabilistic relational models of link structure. *Journal of Machine Learning Research*, 3:679–707.

[Gill et al., 2000] Gill, J., Howse, J., Kent, S., and Taylor, J. (2000). Projections in venn-euler diagrams. In *Proceedings of the IEEE Symposium on Visual Languages (VL2000),* Seattle, WA, pages 119–126. IEEE Computer Society Press.

[Good, 1965] Good, I. (1965). *The Estimation of Probabilities.* MIT Press, Cambridge, MA.

[Heckerman, 1991] Heckerman, D. (1991). *Probabilistic Similarity Networks.* MIT Press, Cambridge, MA.

[Heckerman et al., 2000] Heckerman, D., Chickering, D., Meek, C., Rounthwaite, R., and Kadie, C. (2000). Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1:49–75.

[Howard and Matheson, 1981] Howard, R. and Matheson, J. (1981). Influence diagrams. In Howard, R. and Matheson, J., editors, *Readings on the Principles and Applications of Decision Analysis*, volume II, pages 721–762. Strategic Decisions Group, Menlo Park, CA.

[Koller and Pfeffer, 1997] Koller, D. and Pfeffer, A. (1997). Object-oriented Bayesian networks. In Geiger, D. and Shenoy, P., editors, *Proceedings of Thirteenth Conference on Uncertainty in Artificial Intelligence,* Providence, RI, pages 302–313. Morgan Kaufmann, San Mateo, CA.

[Kschischang et al., 2001] Kschischang, F., Frey, B., and Loeliger, H. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47:498–519.

[Lauritzen, 1996] Lauritzen, S. (1996). *Graphical Models*. Claredon Press.

[McCallum and Nigam, 1998] McCallum, A. and Nigam, K. (1998). A comparison of event models for naive Bayes text classification. In *Workshop on Learning for Text Categorization and the Fifteenth Conference of the American Association for Artificial Intelligence,* Madison, WI.

[Pasula and Russell, 2001] Pasula, H. and Russell, S. (2001). Approximate inference in first-order probabilistic languages. In *IJCAI*, pages 741–748.

[Spiegelhalter, 1998] Spiegelhalter, D. (1998). Bayesian graphical modelling: A case-study in monitoring health outcomes. *Applied Statistics*, 47:115–134.

[Ullman and Widom,] Ullman, J. and Widom, J. *A First Course in Database Systems*. Prentice Hall, Upper Saddle River, NJ.

A Appendix

We use $\mathcal{E}$ and $\mathcal{R}$ to denote the set of entity and relationship classes, respectively. We use E and R (sometimes with subscripts) to denote an entity and relationship class, respectively, and X to denote an arbitrary class in $\mathcal{E} \cup \mathcal{R}$. We use $\sigma(E)$ and $\sigma(R)$ denote an entity and relationship set, respectively, and $\sigma(X)$ to denote an arbitrary $\sigma(E)$ or $\sigma(R)$. We use e and r to denote a particular entity and relationship, respectively, and x to denote an arbitrary entity or relationship. We use $X.A$ to denote the attribute class A associated with class X, and $\mathcal{A}(X)$ to denote the set of attribute classes associated with class X. We use $x.A$ to denote an attribute associated with entity or relationship x, and $\mathcal{A}(x)$ to denote the set of attributes associated with x. Each attribute class and attribute is associated with a *domain*—a set of possible values. The domain of $x.A$ is the same as the domain of $X.A$ for every $x \in X$.

First, we define the entity-relationship model in the following series of definitions.

Definition 1 *An* entity-relationship (ER) diagram for entity classes $\mathcal{E}$, relationship classes $\mathcal{R}$, and attribute classes $\mathcal{A}$ *is a graph in which rectangular nodes correspond to entity classes, diamond nodes correspond to relationship classes, and oval nodes correspond to attribute classes of entity or relationship classes. The node corresponding to a relationship class among entities $E_1, \ldots, E_n \in \mathcal{E}$ is connected to the nodes corresponding to these entities* *with a dashed edge. attribute classes corresponding to an entity or relationship class are connected to this class with dashed edges.*

Definition 2 *A* skeleton for entity classes $\mathcal{E}$ and relationship classes $\mathcal{R}$*—denoted* $\sigma_{\mathcal{ER}}$*—consists of (1) an entity set* $\sigma(E)$ *for every* $E \in \mathcal{E}$ *and (2) a relationship set* $\sigma(R)$ *for every* $R \in \mathcal{R}$ *that is consistent with any constraints imposed by the relationship classes.*

Definition 3 *An* entity-relationship (ER) model for entity classes $\mathcal{E}$, relationship classes $\mathcal{R}$, and attribute classes $\mathcal{A}$ *is an ER diagram for* $\mathcal{E}$*,* $\mathcal{R}$*, and* $\mathcal{A}$ *that defines a set of (ordinary) attributes* $\mathcal{A}(\sigma_{\mathcal{ER}})$ *for any skeleton* $\sigma_{\mathcal{ER}}$*. In particular, attribute* $x.A$ *is in* $\mathcal{A}(\sigma_{\mathcal{ER}})$ *if and only if there is an* X *in* $\mathcal{E} \cup \mathcal{R}$ *and an* $x \in \sigma(X)$ *such that* A *is in* $\mathcal{A}(X)$*.*

Definition 4 *An* entity-relationship instance for an ER model for $\mathcal{E}$, $\mathcal{R}$, and $\mathcal{A}$*—denoted* $\mathcal{I}_{\mathcal{ERA}}$*—consists of (1) a skeleton* $\sigma_{\mathcal{ER}}$ *and (2) a value for every attribute in* $\mathcal{A}(\sigma_{\mathcal{ER}})$*.*

Now we consider domains wherein attributes may be probabilistic and define the DAPER model through the following series of definitions.

Definition 5 *Given an entity or relationship class* X *with entity or relationship* x*,* the ordered set of entities $\mathbf{e}(x)$ associated with x *is as follows. If* X *is an entity class, then* $\mathbf{e}(x) = (x)$*. If* X *is a relationship class containing relationships* $R(e_1, \ldots, e_n)$*, then* $\mathbf{e}(x) = (e_1, \ldots, e_n)$*.*

Note that the set $\mathbf{e}(x)$ is ordered to preserve roles associated with self relationship classes.

Definition 6 *Given an ER model with attribute classes* $X.A$ *and* $Y.B$*,* the constraint $\mathcal{C}_{AB}(\mathbf{e}(x), \mathbf{e}(y))$ for the ordered pair $(X.A, Y.B)$ *is a first-order expression that is bound when the elements of* $\mathbf{e}(x)$ *and* $\mathbf{e}(y)$ *are taken as constants. The atoms of this expression have the form* $R(e_1, \ldots, e_n)$ *where* R *is a relationship class connected to entity classes* $E_1, \ldots, E_n$ *or a predefined relationship class such as equality, less than, greater than, and first.*

Definition 7 *A* directed probabilistic entity-relationship (DPER) diagram *for entity classes* $\mathcal{E}$*, relationship classes* $\mathcal{R}$*, and attribute classes* $\mathcal{A}$ *consists of (1) an ER model for* $\mathcal{E}$, $\mathcal{R}$, *and* $\mathcal{A}$*, and (2) a set of* arc classes *drawn as solid directed arcs corresponding to probabilistic dependencies. There can be at most one arc class from attribute class* $X.A$ *to attribute class* $Y.B$*; and any arc class may have a constraint* $\mathcal{C}_{AB}(\mathbf{e}(x), \mathbf{e}(y))$*. The set of arc classes pointing to* $X.A$ *is the* parent class *of* $X.A$*, denoted* $\mathcal{PA}(X.A)$*.*

Definition 8 *A* ground graph *for a DPER diagram and skeleton $\sigma_{\mathcal{ER}}$ for $\mathcal{E}$, $\mathcal{R}$, and $\mathcal{A}$ is a directed graph constructed as follows. For every attribute in $\mathcal{A}(\sigma_{\mathcal{ER}})$, there is a corresponding node in the graph. For any attribute $x.A \in \mathcal{A}(\sigma_{\mathcal{ER}})$, its parent set $\mathbf{pa}(x.A)$ are those attributes $y.B \in \mathcal{A}(y)$ such that there is an arc class from $Y.B$ to $X.A$ and the expression $\mathcal{C}_{AB}(\mathbf{e}(x), \mathbf{e}(y))$ is true.*

Definition 9 *Given $\Sigma_{\mathcal{ER}}$, a set of skeletons for $\mathcal{E}$, $\mathcal{R}$, and $\mathcal{A}$, a DPER diagram for $\mathcal{E}$, $\mathcal{R}$, and $\mathcal{A}$ is acyclic with respect to $\Sigma_{\mathcal{ER}}$ if, for every $\sigma_{\mathcal{ER}} \in \Sigma_{\mathcal{ER}}$, the ground graph for the DPER diagram and $\sigma_{\mathcal{ER}}$ is acyclic.*

Theorem 1 *If the probabilistic arcs of a DPER diagram for $\mathcal{E}$, $\mathcal{R}$, and $\mathcal{A}$ form an acyclic graph, then the DPER diagram is aylic with respect to $\Sigma_{\mathcal{ER}}$ for any $\Sigma_{\mathcal{ER}}$.*

Proof: Suppose the theorem is false. Consider a cyclic ground graph for some skeleton. Denote the attributes in the cycle by $(x_1.A_1 \to x_2.A_2 \to \ldots \to x_n.A_n)$ where $x_1.A_1 = x_n.A_n$. For each attribute $x_i.A_i$ there is an associated attribute class $X_i.A_i$. From Definition 8, we know that there must be an edge from $X_i.A_i \to X_{i+1}.A_{i+1}$. Because $X_1.A_1 = X_n.A_n$, there must be a cycle in the DPER diagram, which is a contradiction. Q.E.D.

Friedman et al. (1999) prove something equivalent.

Definition 10 *A* directed acyclic probabilistic entity-relationship (DAPER) model *for entity classes $\mathcal{E}$, relationship classes $\mathcal{R}$, attribute classes $\mathcal{A}$, and skeletons $\Sigma_{\mathcal{ER}}$ consists of (1) an DPER diagram for $\mathcal{E}$, $\mathcal{R}$, and $\mathcal{A}$ that is acyclic with respect to every $\sigma_{\mathcal{ER}} \in \Sigma_{\mathcal{ER}}$, and (2) a* local distribution class*—denoted $P(X.A|\mathcal{PA}(X.A))$—for each attribute class $X.A$. Each local distribution class is collection of information sufficient to determine a local distribution $p(x.A|\mathbf{pa}(x.A))$ for any $x.A \in \mathcal{A}(\sigma_{\mathcal{ER}})$. For every $\sigma_{\mathcal{ER}} \in \Sigma_{\mathcal{ER}}$, the DAPER model specifies a DAG model for $\mathcal{A}(\sigma_{\mathcal{ER}})$. The structure of this DAG model is the ground graph of the DPER diagram for $\sigma_{\mathcal{ER}}$. The local distributions of this DAG model are the local distributions $p(x.A|\mathbf{pa}(x.A))$.*

An immediate consequence of Definition 10 is that, given $\mathcal{D}$, a DAPER model for $\mathcal{E}$, $\mathcal{R}$, $\mathcal{A}$, and $\Sigma_{\mathcal{ER}}$ and a skeleton $\sigma_{\mathcal{ER}} \in \Sigma_{\mathcal{ER}}$, we can write the joint distribution for $\mathcal{A}(\sigma_{\mathcal{ER}})$ as follows:

$$p(\mathcal{I}_{\mathcal{ERA}}|\sigma_{\mathcal{ER}}, \mathcal{D}) = \prod_{X \in \mathcal{E} \cup \mathcal{R}} \prod_{x \in \sigma(X)} \prod_{A \in \mathcal{A}(X)} p(x.A|\mathbf{pa}(x.A)). \tag{5}$$

In the remainder of this section, we describe a condition weaker than the one in Theorem 1 that guarantees the creation of acyclic ground graphs from a DAPER model. In this discussion, we use $R(e_1, \ldots, e_n)$ to denote a particular relationship in a relationship set $\sigma(R)$.

Definition 11 *A relationship class R is a* self-relationship class with respect to entity class E *if a relationship in R contains two or more references to entities in the entity class E.*

Definition 12 *A* projected pairwise self-relationship class *is obtained from a self-relationship class by projecting two of the entities in the relationships that are from the same entity class.*

For example, the Family relationship class is a self-relationship class that can be projected into the Father-Child relationship class and the Mother-Child relationship class; and both are projected pairwise self-relationship classes.

Definition 13 *Given skeleton $\sigma_{\mathcal{ER}}$ for $\mathcal{E}$ and $\mathcal{R}$, a relationship set σR for a self-relationship class R is* acyclic *if there exists a projected pairwise self-relationship class R' for some entity set E containing entities $e_1, \ldots, e_n$ such that $R'(e_1, e_2), \ldots, R'(e_{n-1}, e_n)$ and $R'(e_n, e_1)$. If a relationship set is not cyclic, it is* acyclic*.*

Definition 14 *A arc class in a DAPER model is called a* self arc *if both the head and tail of the arc are the same attribute class.*

Theorem 2 *If (1) the arc classes excluding the self arcs of a DAPER diagram for $\mathcal{E}$, $\mathcal{R}$, and $\mathcal{A}$ form an acyclic graph, (2) every self arc has a constraint involving a self-relationship class, (3) no constraint on a self arc is disjunctive, and (4) $\sigma(R)$ for every self-relationship class R is acyclic for every $\sigma_{\mathcal{ER}} \in \Sigma_{\mathcal{ER}}$, then the DPER diagram is ayclic with respect to $\Sigma_{\mathcal{ER}}$.*

Proof: Suppose the theorem is false. Consider a ground graph G for some skeleton $\sigma_{\mathcal{ER}}$ containing a cycle. Further consider DPER diagram D' obtained from the original DPER diagram with all self arc classes removed. By condition 1, D' is acyclic. Let G' be the ground graph obtained from D' applied to $\sigma_{\mathcal{ER}}$. Given D' is acyclic and Theorem 1, G' must be acyclic.

Let S be the edges in the cycle that appear in graph G but not G'. S must be non-empty, because G is cyclic whereas G' is not. By construction, each arc in S is associated with a self arc in the DPER diagram; and hence the endpoints of each arc must have the same attribute class. Consequently, the arcs in S must form a cycle lest D' would be cyclic. Because the arcs in S form a cycle, each arc in S must be associated with the same self arc class.

Given conditions 2 and 3, we know that the self arc class associated with the cycle has a non-disjunctive constraint involving a self-relationship class—say R. Because S is cyclic, $\sigma(R)$ must be cyclic, contradicting condition 4. Q.E.D.