Probabilistic Entity-Relationship Models, PRMs, and Plate Models

David Heckerman HECKERMA@MICROSOFT.COM
Christopher Meek MEEK@MICROSOFT.COM
One Microsoft Way, Redmond, WA 98052

Daphne Koller KOLLER@CS.STANFORD.EDU
Computer Science Department, Stanford, CA 94305

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.

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, several graphical languages for relational data have emerged including plate models (e.g., Buntine, 1994; Spiegelhalter, 1998) and probabilistic relational models (PRMs) (e.g., Friedman, Getoor, Koller, and Pfeffer, 1999. These models are to relational data what ordinary graphical models (e.g., directed-acyclic graphs and undirected graphs) are to flat data.

In this paper, we introduce a new graphical model for relational data—the probabilistic entity-relationship (PER) model. This model class is more expressive than either PRMs or plate models. We concentrate on a particular type of PER model—the directed acyclic probabilistic entity-relationship (DAPER) model—in which all probabilistic arcs are directed. 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 (in the expanded tech report, Heckerman, Meek, and Koller, 2004) 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. Our hope is that this demonstration of similarity will foster greater communication and collaboration among statisticians who mostly use plate models and computer scientists who mostly use PRMs.

We in fact began this work with an effort to unify traditional PRMs and plate models. In the process, we discovered that it was important to make both entities and relationships (concepts discussed in detail in the next section) first class objects in the language. We in turn discovered an existing language that does this—the entity-relationship (ER) model—a commonly used model for the abstract representation of database structure. We then extended this language to handle probabilistic relationships, creating the PER model.

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. Conse-

Appearing in *Proceedings of the 21st International Conference on Machine Learning*, Banff, Canada, 2004. Copyright 2004 by the first author.

quently, 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. ER Models

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[1]; 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.[2] 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.

A relationship is a tuple of pointers to entities—an indication that those referenced entities are somehow related. 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 1a. 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 1b.

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

[2] 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.

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 1a applied to the skeleton in Figure 1b are shown in Figure 1c. In what follows, we use *ER model* to mean both the *ER diagram*—the graph in Figure 1a—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.

3. PER Models

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 2a. The model extends the ER model in Figure 1 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 2a). 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 2c shows the DAG (structure) generated by the application of the DAPER model in Figure 2a to the skeleton in Figure 2b. (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 2a will be acyclic. In general, as we show in Heckerman et al. (2004), 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 Heckerman et al. (2004).

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

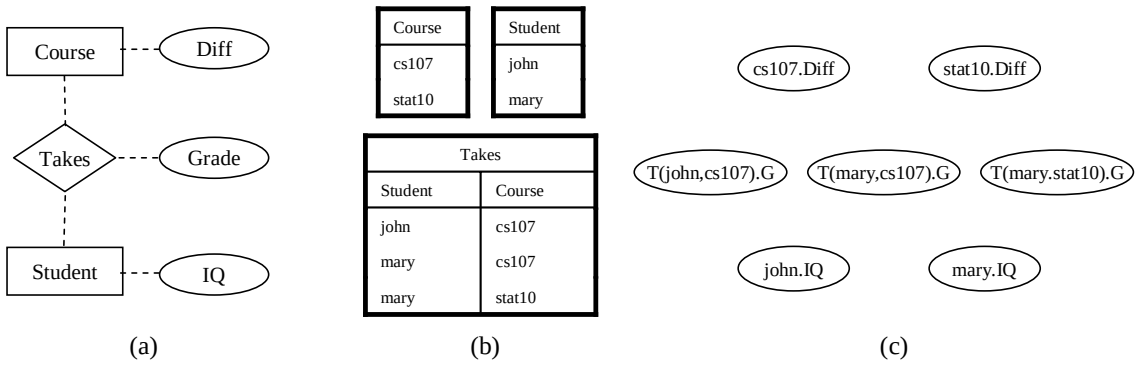

Figure 1. (a) An entity-relationship (ER) model depicting the structure of a university database. (b) An example skeleton for the entity and relationship classes in the ER model. (c) The attributes defined by the application of the ER model to the skeleton. The attribute names are abbreviated.

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 Heckerman et al. (2004).

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.

4. Plates 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 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 Heckerman et al., 2004).

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 2a is shown in Figure 3a. The two rectangles are the plates corresponding to the Student and Course entity classes. The single relationship class between Student and Course—Takes—is repre-

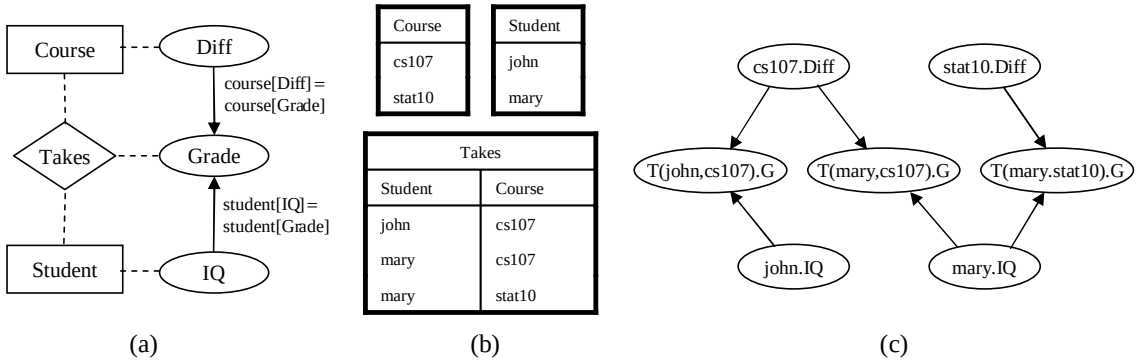

Figure 2. (a) A directed acyclic probabilistic entity-relationship (DAPER) model showing that a student's grade in a course depends on both the student's IQ and the difficulty of the course. (b) An example skeleton for the entity and relationship classes in the ER model (the same one shown in the previous figure). (c) The DAG model (structure) defined by the application of the DAPER model to the ER skeleton.

sented 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 Heckerman et al., 2004).

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-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 2a is shown in Figure 3b. (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.

5. Details

In this short discussion, we have omitted many of the technical details of DAPER models as well as important facets of modeling relational data including the use of restricted relationships, self relationships, and probabilistic relationships. In addition, we have not described several important classes of PER model that expand into graphical models other than traditional DAG models. These topics are covered in Heckerman et al. (2004).

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.

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

[Friedman et al., 1999] Friedman, N., Getoor, L.,

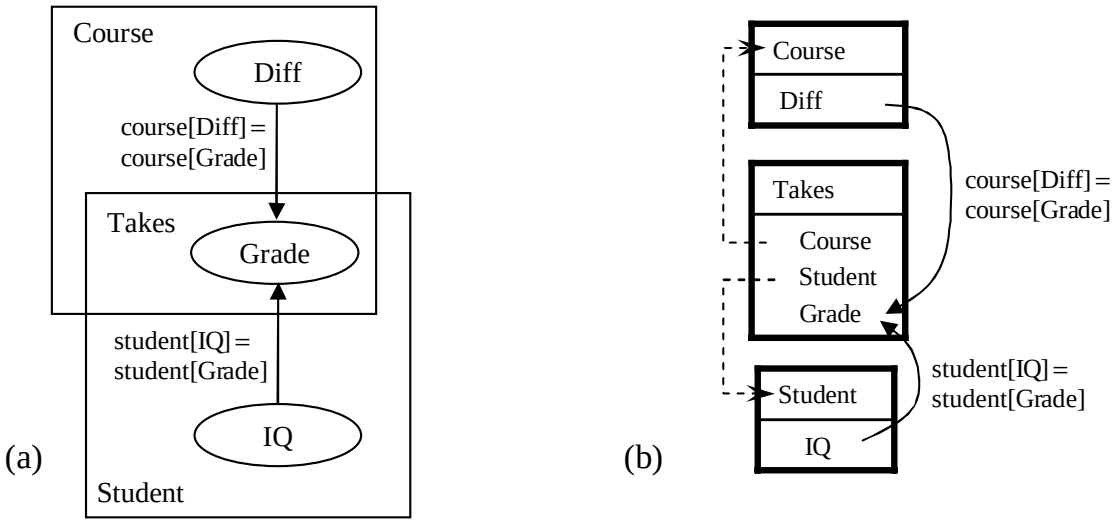

Figure 3. A plate model (a) and probabilistic relational model (b) corresponding the DAPER model in Figure 2a.

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.

[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.

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

[Heckerman et al., 2004] Heckerman, D., Meek, C., and Koller, D. (2004). Probabilisitic models for relational data. Technical Report MSR-TR-2004-30, Microsoft Research, Redmond, WA. http://research.microsoft.com/research/pubs/view.aspx?tr_id=734.

[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.

[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.