Decision-Guidance Management Systems (DGMS): Seamless Integration of Data Acquisition, Learning, Prediction, and Optimization[*]

Alexander Brodsky
Department of Computer Science
George Mason University, Fairfax, Virginia
& Adaptive Decisions, Inc., Rockville, Maryland

X. Sean Wang
Department of Computer Science
The University of Vermont
Burlington, Vermont

ABSTRACT

A Decision Guidance Management System (DGMS) is a productivity platform for fast development of applications that require a closed-loop data acquisition, learning, prediction, and decision optimization. This paper introduces the DGMS concept, and the first DGMS data model with its query language, DG-SQL. The DGMS data model is an extension of the relational model with probability distributions over a set of attributes as random variables. DG-SQL supports a seamless integration of (1) querying the data collection and construction of learning sets, (2) learning from the learning sets, using parameterized transformers and optionally defining an estimation utility, such as sum of squares of errors, to be minimized; (3) probabilistic prediction and simulation, using expressions that involve random variables, such as expectation, variance and probability of a logical formula; and (4) stochastic or deterministic optimization, where search space is defined as a set of feasible non-deterministic query evaluations.

1. INTRODUCTION

Increasing number of applications require predicting behavior of a complex system and making decisions to move the system towards desirable outcomes. In such applications, predictions and decisions are to be made in the presence of large amounts of dynamically collected data and learned uncertainty models.

There has been extensive research in the areas of operations research, mathematical and constraint programming, machine learning and data mining, and database systems. However, there are no cohesive frameworks, algorithms and systems that unify the models and computational paradigms of all the components. Without unification of models for the related but different tasks, users are forced to express their knowledge of the underlying domain multiple times using different mathematical abstractions.

We call a system that supports a closed-loop data acquisition, learning, prediction and decision optimization, a *decision-guidance database*, and a productivity tools platform for fast development of decision-guidance databases - *decision-guidance management system* (DGMS). This paper introduces the concept of DGMS and proposes the first data model and an integrated DGMS query language, DG-SQL, which supports seamless integration of (1) construction of learning sets; (2) learning; (3) probabilistic prediction and simulation; and (4) stochastic or deterministic optimization. In DG-SQL all of these activities are based on the same *dependence graph* and *transformers* model which represent users' domain knowledge.

The remainder of this paper is organized as follows. Section 2 discusses decision-guidance challenges in supply chains. Section 3 outlines key DGMS components. Section 4 gives an extension of the relational data model with stochastic attributes and transformers. Sections 5–9 describe, respectively, (a) the notion of transformers and the dependence graph, (b) the DG-SQL prediction and simulation queries, (c) the syntax and semantics for DG-SQL decision optimization queries, (d) how transformers' parameters are learned from learning sets, also expressed as DG-SQL queries, and (e) the data acquisition challenges and solutions. Section 10 further reviews related work. Finally, Section 11 concludes the paper with a brief discussion of future work.

2. DECISION GUIDANCE CHALLENGES IN SUPPLY CHAINS

Guiding decisions in supply chains can significantly reduce costs, and increase revenues and profits. There has been significant advances in quantitative models for supply chains, and optimization methods (e.g., see a collection [33] and text [31]). However, building software systems that use the quantitative models for supply chains to make better decisions typically involves a major effort of modeling, design and development. Besides modeling, attention has be paid to the automatic translation of different specialized models to the languages of mathematical programming solvers (e.g., [4, 36]). Still, the result of such efforts are systems that are difficult to modify or extend, especially when there is a large amount of data that the system can take advantage of.

[*]Work of Wang was partially supported by the NSF grant IIS-0415023.

Consider the elements of a typical supply chain in the consumer packaged goods industry (CPG). The supply chain decisions are typically driven by demand for individual products in different markets. The demand can be viewed as a stochastic function of price, and additional market characteristics. Especially for seasonable products, demand may be extremely time sensitive, and loosing a market window may drastically reduce demand. Decisions on prices of products is the task of price-revenue optimization. Higher prices may reduce the demand, yet the revenue may or may not increase. Maximizing revenue is a typical goal in price-revenue optimization. However, especially when there is significant fluctuations of raw material prices and production costs, a better (and a more complex) goal may be optimizing profitability that takes in the account the cost side.

Distribution is another component of supply chain. In some markets, there are multiple distribution choices, which makes the decision on which products are distributed, where to, by which distributor and from which distribution center adds complexity to business decision makers.

Transportation of products from manufacturing facilities to distribution centers may involve multiple providers, such as truck- and less-then-truck-load carriers, and sea and air carriers. The decision which transportation providers should be used is an additional layer of decision guidance.

Manufacturing products may involve multiple manufacturing facilities, often overseas. Deciding what quantities of which products should be manufactured where is another layer of decision making. Often, companies of consumer goods, such as Nike, put together a collaborative supply chain involving the components discussed. Typically, such companies would not buy the products from manufacturers, but rather contract them for manufacturing service. This way, they can negotiate better prices with raw materials' suppliers, and also take the uncertainty in raw materials prices from manufacturers. This, in turn, allows them to get cheaper manufacturing prices.

Sourcing of raw materials involves another layer of decisions on which raw materials should be purchased in what quantities from which suppliers (and transported to manufacturing facilities). Raw materials' prices involve uncertainty, and may depend on a variety of market characteristics. Sometimes, companies mitigate the risk related to raw materials' price fluctuation by buying option and futures contracts ahead of their supply chain cycle.

Let us assume that we are interested to develop a decision guidance solution for an instance of a supply chain described above. Let us further assume that we would like to make the decision choices outlined above so that they would satisfy the capacity, timing and other constraints, and would maximize the profit margin. To model the problem, we need to mathematically capture the profit margin as a (possibly stochastic) function of the decision choices (quantities, selection of distributors, transportation providers, manufacturers and suppliers etc). We also need express the capacity, timing and other constraints. The expression of the profit margin involves, in turn, the models of demand (as a stochastic function of prices etc.), distribution (cost as a function of distribution choices), transportation (cost as a function of selected transportation providers and quantities), manufacturing and sourcing. Such modeling is typically done using a modeling language such as AMPL. Then, the application software would typically integrate a mathematical programming solver, such as ILOG CPLEX, to which problem instances, say represented in AMPL will be submitted.

This approach assumes the knowledge of the deterministic or stochastic models, such as the demand model, which may not be known a priori. However, there may be significant historic and statistical data which can be leveraged to "learn" the stochastic demand function, using, for example, regression analysis techniques. The system challenge here is that different mathematical abstractions and modeling languages are used to describe the learning problems. Whereas, the same domain knowledge is being used for both optimization and learning.

Often, business decision makers are interested in prediction tasks (which are less than optimization). For example, for a specific instance of a supply chain, including the chosen product prices, business managers may want to predict the expected profit, or to estimate the probability of a total loss to exceed $2M. Technically, prediction involves computing expectations of random variables, and probabilities of logical formulae that involve random variables. The system challenge here, again, is that different mathematical abstractions, languages and software tools are used for this task. The focus of DGMS is to unify the models for domain knowledge representation, and developing a unified language for answering learning, prediction, optimization and data acquisition questions.

3. DGMS ARCHITECTURE AND COMPONENTS

We anticipate that a typical DG database will be organized in a number of components, as depicted in Figure 1. The *raw database* is to contain a regular relational database, as is, which is dynamically modified via its interaction with users and other systems. The *learning set views* are for storing typical, or user-defined, organization of the raw data for the purpose of learning. The *domain knowledge* is to include correlation dependency relationships among relational attributes. It can also include different working hypotheses used in DG reasoning. The *learned knowledge* component provides the learned rules, typically using the learning sets prepared in the learning set views. The *prediction views* and *decision optimization views* are for storing system- or user-defined queries that are useful for different prediction and decision optimization activities. The *DG-SQL* is the query language and processing component that provides the unified access and definition to all the different views, i.e., learning set views, prediction views, and decision optimization views, as well as provides a means for system to derive learned knowledge and for user to issue ad-hoc queries. DG-SQL will have the ability to incorporate the domain knowledge and learned knowledge in its queries, and deduce data acquisition requirements to external data sources.

From a user perspective, a DGMS is to be used in iterative, loop-back *phases*. In the first phase, raw database and the domain knowledge are used to define the data sets that are

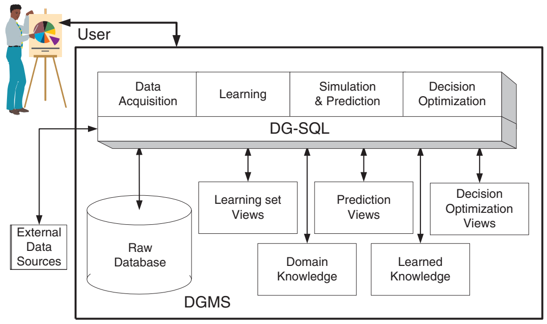

Figure 1: DGMS architecture and components

used to derive learned knowledge. In the second phase, the learning and domain knowledge is used for prediction and simulation. The results from prediction and simulation will be used in the third phase, in which decision optimization queries are issued. Finally, data acquisition queries are used in a feedback loop to direct the system on how to gain more raw data to reduce uncertainty in decisions. The views that are stored in the systems can be used as intermediate results, or to hand over results to different users in the next phases, thereby supporting division of duties.

To support the above architecture and intended use of the DGMS system, this paper makes the following contributions. First, we introduce a stochastic relational data model, in which database relations may have *stochastic* attributes. A relational tuple assigns numeric values to regular attributes, and associates them with a probability distribution function (pdf) over the stochastic attributes. The function that defines a pdf given regular attributes is defined using the notion of a *dependence graph* and *transformers*, introduced in the paper. A *dependence graph* is composed of a set of *connectors* which have input and output attributes representing random variables. As in a Bayesian network, a connector represents an independence assumption: under the condition that all inputs to a connector are instantiated, the output random variables are dependent on nothing but the inputs. Unlike in a Bayesian network, a connector in the dependency graph may have an associated *transformer*. A *transformer* is a program that computes outputs from inputs, and in which numeric variables are polymorphic and correspond to either a numeric value or a random variable, with a particular pdf. For example $x := y + 2 * z$ can be interpreted as defining a random variable x from previously defined random variables y and z. Also, some parameters in a transformer may be declared *learned*, to indicate that they are unknown at the time of transformer's definition, but can potentially be *learned* if a learning set is provided. Thus, a connector and a corresponding transformer represent partial knowledge held by the user on how output random variables are computed from the inputs. For example, a *Demand* stochastic transformer may capture the knowledge on how (a random variable for) market *demand* for a particular product is computed from the product's *price* and a *market-index*; a *Manufacturing* stochastic transformer may capture the knowledge on how (random variables for) manufacturing cost, and the quantity of ingredients are computed from the quantities of the products to be manufactured. Similarly, deterministic transformer *profit* may capture the knowledge that it is the *total-revenue* minus the *total-cost*.

Second, we propose a unified DG-SQL language, which is an extension of SQL. To support predictions, DG-SQL provides (stochastic or deterministic) transformers to define new attributes, which can be viewed as random variables. It also provided operations on random variables, including computing their expectations and standard deviations, and the probability of a logical formula. Such expressions, in turn, can be used in the SELECT and the FROM clauses of DG-SQL. Similarly to predictions, DG-SQL supports simulation. Given a random variable (in one of the attributes), it allows a random selection of a value, according to its pdf. For example, a prediction DG-SQL query can ask to find all markets in which, given the planned products' prices and manufacturing quantities, the revenue expectation is above $2M and the likelihood of financial loss (i.e., negative profit) is bounded by 5%. Or, a simulation DG-SQL query (possibly run many times) can ask to find, for all revenue generating markets, their profitability which is randomly selected using the probability distribution implicitly defined by the *Profit* and other related transformers.

To support decision optimization, DG-SQL queries (and thus a cascade of views) allow to designate some attributes as *non-deterministic* (called ND-nulls) and impose ASSERT statements that constrain them using a logical formula similar to one in the WHERE clause. An optimization DG-SQL query also specifies an aggregation function, such as sum, on a number of attributes defined in the SELECT clause to be minimized or maximized. Semantically, each selection of values into ND-nulls may or may not satisfy the ASSERT conditions, and may lead to a different aggregation computed by the query. A DG-SQL query that involves ND-nulls essentially defines a set of non-deterministic query evaluations, each corresponding to a selection of allowed values into ND-nulls. The optimization semantics of DG-SQL amounts to (1) finding an *optimal* non-deterministic query evaluation, i.e., one that produces the minimal or maximal, as requested, aggregation answer, and then (2) computing the query with values for ND-nulls corresponding to the optimal evaluation path. Note that the search space in DG-SQL optimization queries is defined implicitly through a set of non-deterministic query evaluations, rather then by arithmetic constraints over numeric constraint variables. As an example, an optimization DG-SQL query can ask to find, for each market, products' prices and production quantities that will maximize expected market profitability, while bounding the likelihood of financial loss by 5%.

To support learning, DG-SQL query defines a transformer program with parameters designated as *learned* (called L-nulls). The transformer is associated with an optimization DG-SQL query that computes a learning set, and, from it, an aggregate, such as the sum of squares of errors, to be minimized. The transformer is used in the optimization query in the computation of the aggregate, and its L-nulls are interpreted as non-deterministic values to be optimally instantiated. For example, in the *Manufacturing* transformer mentioned above the coefficients that describe how much of each ingredient is required for 1 unit of each product may not be known (and thus designated as L-nulls), but rather learned from historical data.

4. STOCHASTIC RELATIONAL DATA MODEL

Our basic data model is relational that is extended with probabilistic attributes.

Definition *An n-ary* stochastic relational schema*, or simply* s-schema*, is a set of* n *pairs* $S = \{A_1 : T_1, \ldots, A_n : T_n\}$ *with* $A_i \neq A_j$ *if* $i \neq j$*, where each* A_i $(i = 1, \ldots, n)$ *is an* attribute name*, or simply* attribute*, and each* T_i $(i = 1, \ldots, n)$ *is a* type name *or* domain*. The attributes* $A_1, \ldots, A_n$ *are partitioned into two sets: the* regular attributes S_{Reg} *and the* probabilistic attributes S_{Prob}.

When $T_1, \ldots, T_n$ are understood in an s-schema, we may abbreviate the s-schema as $S = \{A_1, \ldots, A_n\}$. Semantically, each T_i is associated with a domain, denoted $Dom(T_i)$ or simply $Dom(A_i)$. Without loss of generality, we assume $S_{Reg} = \{A_1, \ldots, A_k\}$ and $S_{Prob} = \{A_{k+1}, \ldots, A_n\}$ for some $0 \leq k \leq n$. With this notation, $S_{Reg} = \emptyset$ when $k = 0$, and $S_{Prob} = \emptyset$ when $k = n$.

Definition *Given an n-ary s-schema* $S = \{A_1, \ldots, A_n\}$*, an* s-instance *of* S *is a finite set of* s-tuples *over* S*, where each s-tuple* t *over* S *consists of two elements* t_{Reg} *and* t_{Prob} *such that* t_{Reg} *is a (regular) tuple over the attributes* S_{Reg} *and* t_{Prob} *is an* $(n-k)$*-dimension* probability density function (PDF) *over the domain* $Dom(A_{k+1}) \times \cdots \times Dom(A_n)$*, where* $\{A_{k+1}, \ldots, A_n\} = S_{Prob}$.

We shall use r to denote an s-instance of S, and t an s-tuple of r, consisting of t_{Reg} and t_{Prob}. In special cases when $S_{Reg} = \emptyset$ or $S_{Prob} = \emptyset$, we let t_{Reg} or t_{Prob}, respectively, be the empty tuple (i.e., 0-ary tuple).

Alternatively, we may write an s-tuple in the form $t = \langle a_1, \ldots, a_k, pdf \rangle$. Mathematically, the *pdf* takes $A_{k+1}, \ldots, A_n$ as its random variables. Semantically, an s-tuple of the above form gives the conditional probability density function $pdf(A_{k+1}, \ldots, A_n | \; A_1 = a_1, \ldots, A_k = a_k)$.

Note that in the above, we implicitly assumed that $Dom(A_j)$, $k+1 \leq j \leq n$, are dense domains, but we can easily accommodate discrete domains.

5. DEPENDENCE GRAPH AND TRANSFORMERS

The dependence among attributes is an important consideration for stochastic relational models. We model the dependencies with the help of a *dependence graph*. Syntactically, a dependence graph captures the relationship among attributes, including statistical correlation and independence. Also, it guides the design of transformers that relate the values of these attributes.

In order to describe dependency graph and related concepts in the sequel, we consider a part of a supply chain, a company ABC, that manufactures and whole sells a number of perishable products in a number of markets. Assume that ABC's management makes weekly decisions on pricing and the production quantity of its perishable products for each of the target markets. Product demand depends on its pricing and the characteristics of the market where it is sold, and is uncertain. Higher price in a particular market may reduce the demand, so the revenue may or may not rise. Producing less than the actual demand would miss a revenue opportunity, whereas producing more than the actual demand would waste the perishable products, i.e., the cost of ingredients and manufacturing. The cost of ingredients also involves uncertainty. Thus, even if the revenue goes up due to a change in pricing, the profit may not. Furthermore, ABC's manufacturing capacity is constrained, and so sometimes producing more of product A may be at the expense of producing less of product B.

If ABC's weekly profit probability distribution could be accurately predicted as a function of product prices, manufacturing quantities and markets' economic characteristics, we could use the prediction function computation in simulating and trying out ABC's potential decisions. Or, if the prediction function could be expressed in a reasonable analytical form, ABC's decision problem could be formulated (and hopefully solved) as one of stochastic or deterministic programming. However, the prediction function may not be explicitly known, but needs to be learned from ABC's past decisions and historical market information.

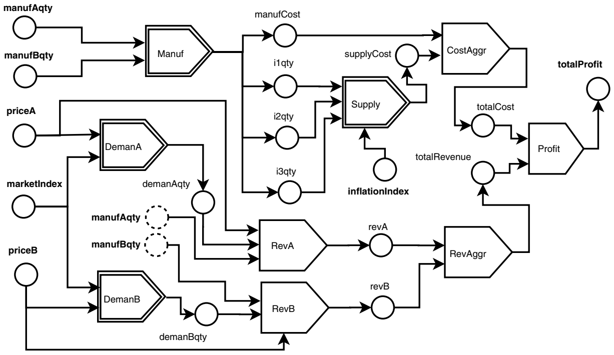

Figure 2: A dependence graph

The dependence graph for ABC is exemplified in Figure 2. Circles correspond to attributes, such as *priceA*, *manufAqty*, ı2qty, and *totalProfit*. (In the graph, the dashed circles are duplicates of other attributes.) Boxes correspond to dependence connectors, such as *Manuf*, *DemandA*, *Supply*, *Profit*. Single-line boxes correspond to *deterministic* connectors. For example, the *CostAggr* connector suggests that the value of its output attribute *totalCost* is deterministically dependent on its input attributes *manufCost* and *supplyCost*, i.e., there is exactly one value of *totalCost* that correspond to the pair of input costs (which, in this case, is the sum of the two costs). Double-line boxes correspond to *stochastic* connectors. For example, dependence connector *DemandA* suggests that its output attribute *demandAqty* is stochastically dependent on the values of its input attributes *priceA* and *marketIndex*, i.e., the probability distribution function (pdf) of the random variable *demandAqty* is determined given the values for its input attributes *priceA* and *marketIndex*. Implicitly, it also says that, given the input values, the random variable *demandAqty* is independent of anything else. More formally,

Definition *A* dependence graph *is a triple* $\Delta = (\mathcal{A}, \Lambda, \mathcal{E})$*, where* $\mathcal{A}$ *is a set of attributes,* Λ *is a set of* connectors*, and* $\mathcal{E}$ *is a set of edges of form* (A, λ) *or* (λ, A)*, where* A *is in* $\mathcal{A}$ *and* λ *and in* Λ*. A dependence graph* $\Delta = (\mathcal{A}, \Lambda, \mathcal{E})$ *is said to be* acyclic *if (1) each attribute node* A *in* $\mathcal{A}$ *has at most one incoming edge (i.e., at most one edge of the form* (λ, A) *for each* A*), and (2) the graph is acyclic if it is taken as directed graph.* We also define the notion of dependency among attributes as follows:

Definition *An attribute* B *is said to be* dependent *on an attribute* A *if there is a directed path from* A *to* B *in the dependence graph. (In particular,* A *is dependent on* A *itself for any attribute.)* For example, *manufCost*, *totalCost* and *totalProfit* are each dependent on *manufAqty*.

We will also need the notion of *complete dependence*. Intuitively, attribute *totalCost* is *completely dependent* on attributes *manufCost* and *supplyCost*, in the sense that these two are all the input attributes of the connector *CostAggr*, of which *totalCost* is output. Furthermore, *totalCost* is *completely dependent* on attributes *manufCost*, *i1qty*, *i2qty*, *i3qty*, and *inflationIndex*, because, intuitively, *supplyCost* is completely dependent on *i1qty*, *i2qty*, *i3qty*, and *inflationIndex* due to connector *Supply*. We omit the formal definition of complete dependence.

Correlation and independence can be derived from a dependence graph as follows:

Definition *The* probabilistic correlation *defines a binary relation given as follows: (1) If an attribute is dependent on another, then these two attributes are probabilistically correlated. (2) Two output attributes of the same probabilistic connector are correlated. (3) The transitive closure of the above two correlation definition gives all the correlated pairs. When two attributes are not correlated, they are probabilistically* independent*.*

As mentioned earlier, a dependence graph also guides the design of *transformers* connecting values of dependent attributes. In the dependence graph, a connector only suggests a deterministic or stochastic dependence, but does not specify how the output attributes (or their pdf) are determined. The purpose of transformers is exactly that. For example, the user may have the knowledge that *demandAqty* is computed as follows.

```
DEFINE TRANSFORM
DemandA(real priceA, marketIndex): (real-pdf demandAqty)

{  if       priceA <= 0.79 {expectedDemAqty = 230000;}
   else if  priceA <= 0.99 {expectedDemAqty = 150000;}
   else if  priceA <= 1.29 {expectedDemAqty = 95000;}
   else if  price  <= 2.99 {expectedDemAqty = 5000;}
   else {expectedDemAqty = 0;}
demandAqty = expectedDemAqty + Gaussian(0,15000^2)
}
```

Note that the output *demandAqty* is a pdf, defined using the construct `Gaussian(0, 15000^2)` which is the a normally distributed random variable with expectation 0 and standard deviation 15000. The input values *priceA*, and *marketIndex* are real values. However, their type is polymorphic, and the input for each one or both of them can also be a pdf over reals. Semantically, variables whose domain type is pdf are treated as random variables with pdf defined by the transform.

In other words, a transformer takes the values under the attributes $A_1, \ldots, A_k$ and generates a PDF over the attributes $B_1, \ldots, B_m$, or a tuple over $B_1, \ldots, B_m$. A stochastic transformer can also be viewed as a density function over $B_1, \ldots, B_m$ conditioned on the variables $A_1, \ldots, A_k$, i.e., $pdf(B_1, \ldots, B_m | A_1, \ldots, A_k)$. This is similar to an s-tuple, but in an s-tuple, A_i $(1 \leq i \leq k)$ have fixed values. A deterministic transformer can also be viewed as a constraint over the variables $A_1, \ldots, A_k, B_1, \ldots, B_m$.

Transformers can be used to derive other transformers via decomposition and composition. For example, given transformers for *CostAggr* and *Supply*, they can be composed to form a transformer that defines the attribute *totalCost* from *manufCost*, *i1qty*, *i2qty*, *i3qty* and *inflationIndex*.

The connection between transformers and s-tuples can be described as follows. The s-tuple is said to be *consistent* with a given transformer if the transformer takes the input attribute values from the s-tuple and gives the value that's exactly as given in the s-tuple on the output attributes. This means that the s-tuple follows the "model" described by the transformer. This fact can useful when queries are processed on s-relations.

6. PREDICTION AND SIMULATION IN DG-SQL

DG-SQL is a superset of SQL over the data model introduced in the previous sections. To illustrate the use of deterministic or stochastic transformers in DG-SQL queries, consider an DG-SQL view definition to provide the view PredictedSales. It expresses, for each *marketID*, the expected demand for and revenue from the two ABC's products. This view definition assumes two input tables: (1) PlannedPricing, which includes, for every marketID, prices for products A and B, and the marketIndex; and (2) PlannedManuf, which includes, for every marketID, the quantities of products A and B to be manufactured (which may be decided by users after they look at a predicted demand), and the inflationIndex.

```
DEFINE VIEW PredictedSales
SELECT
   P.marketID, P.priceA, P.priceB, P.marketIndex,
   M.manufAqty, M.manufBqty,
   demandAqty = demandA(P.priceA,P.marketIndex).demAqty,
   demandBqty = demandB(P.priceB,P.marketIndex).demBqty,
   revenueA =
       revA(demandAqty, M.manufAqty, P.priceA).revenueA,
   revenueB =
       revB(demandBqty, M.manufBqty, P.priceB).revenueB
FROM PlannedPricing P, PlannedManufacturing M
WHERE P.marketID = M.marketID
```

Note that since demandA and demandB transformers define pdf's, demandAqty and demandBqty attributes are, intuitively, random variables. Because of that, application of the transformers revA and revB also produces random variable. Thus, all the new attributes produced are stochastic. The view for PredictedManufacturing&Procurement can be defined in a similar way.

A view for PredictedSummary can be defined as follows:

```
DEFINE VIEW PredictedSummary
SELECT S.marketID, S.priceA, S.priceB,
       M.manufAqty, M.manufBqty,
       totalRevenue = S.revenueA + S.revenueB,
       totalCost = M.manufCost + M.supplyCost,
       profit = totalRevenue - totalCost
FROM PredictedSales S,
     PredictedManufacturing&Procurement M
WHERE S.marketID = M.marketID
```

Note that totalRevenue, totalCost, and profit are probabilistic attributes, which correspond to random variables. Therefore, we can talk about probability of a logical formula expressed in terms of random variables, e.g., `Prob(profit < 0) >= 0.05`. Suppose now we would like to find all markets in which the likelihood of losing money (i.e., of having negative profit) is greater than or equal to 5%, but expected revenue is high, say $2M or more. This can be easily done using the PredictedSummary view:

```
SELECT P.marketID, P.priceA, P.priceB,
       P.manufAqty, P.manufBqty,
       expectedRevenue = expect(totalRevenue),
       expectedCost = expect(totalCost),
       expectedProfit = expect(profit)
FROM PredictedSummary P
WHERE
  expect(P.revenue)>= 2000000 and Prob(P.profit<0)>=0.05
```

Note that if we would like to explicitly represent (e.g., print out) a table, such as the result of this query, the attributes must be values, not random variables. In this query, we returned expected values in the SELECT clause. It is important to note, that while we use transformers defined separately, in DG-SQL queries, transformers can also be defined on the fly in queries. The system then will be able to extract the dependency structure and the transform definitions.

DG-SQL queries can also be used as simulation. For example, if we replace the *expect* function in the select clause, with the *random* selection function, we get:

```
SELECT P.marketID, P.priceA, P.priceB,
       P.manufAqty, P.manufBqty,
       randRevenue = random(totalRevenue),
       randCost = random(totalCost),
       randProfit = random(profit)
FROM PredictedSummary P
WHERE
  expect(P.revenue)>= 2000000 and Prob(P.profit<0)>=0.05
```

This query will return randomly selected revenue, cost and profit. The selection will be done using the joint conditional probability distribution of random variables *totalRevenue*, *totalCost* and *profit* which are defined through the transformers and views that we exemplified. Note, that the above query can be run multiple times, each time producing a new randomly generated table. These tables can be useful for statistical analysis when the distribution is complex.

7. DECISION OPTIMIZATION IN DG-SQL

DG-SQL can also be used to express decision optimization queries. To intuitively understand the semantics, consider a query similar to one we used before:

```
SELECT P.marketID, P.priceA, P.priceB,
       P.manufAqty, P.manufBqty,
       expectedRevenue = expect(totalRevenue),
       expectedCost = expect(totalCost),
       expectedProfit = expect(profit)
FROM PredictedSummary P
WHERE P.marketID<>’NYC’ and P.marketID <> ’Washington DC’
```

The result of this query is dependent on the selection of product prices and quantities of manufactured products in the PredictedSummary view. In particular the total profit in all markets returned by this query is dependent on product pricing and manufactured quantities.

Suppose now we would like to compute the same query, except we do not want to provide priceA, priceB, manufAqty, and manufBqty as input, but rather ask the system to select those values for us in an optimal way. For example, we would like pricing and quantities to be chosen so that expected total profit in all market will be maximized, and yet the expected revenue in each market returned by the query be $ 100,000 or higher and probability of loss less than 5%.

In DG-SQL, the above can be done as follows. Instead of providing the view PredictedSummary as input, consider the view PlannedSummary, in which priceA, priceB, manufAqty and manufBqty are designated ND-nulls, which is a symbol for a nondeterministic value, within a certain range.

```
DEFINE VIEW PlannedSummary AS
SELECT I.marketID, priceA, priceB, manufAqty, manufBqty,
       totalCost = manufCost + supplyCost,
       totalRevenue = revA + revB,
       profit = totalRevenue - totalCost
FROM Indicies I
LET    priceA = ND-null, priceB = ND-null,
 manufAqty = ND-null, manufBqty = NDnull,
 demandAqty = DemandA(priceA,I.marketIndex).demandAqty,
 demandBqty = DemandB(priceA,I.marketIndex).demandBqty,
 revA = RevA(priceA,demandAqty,manufAqty).revA,
 revB = RevB(priceB,demandBqty,manufBqty).revA,
 i1qty = Manuf(manufAqty,manufBqty).i1qty,
 i2qty = Manuf(manufAqty,manufBqty).i2qty,
 i3qty = Manuf(manufAqty,manufBqty).i3qty,
 manufCost = Manuf(manufAqty,manufBqty).manufCost,
 supplyCost=
   Supply(i1qty,i2qty,i3qty,I.inflationIndex).supplyCost
ASSERT 0.0 <= priceA <= 10.0 and
       0.0 <= priceB <= 10.0 and
       0.0 <= manufAqty <= 1000.0 and
       0.0 <= manufBqty <= 1000.0
```

Note the the input table Indices provides marketID and its marketIndex and inflationIndex as attributes. Intuitively, the ND-null values in the definition of this view indicate that the user is leaving flexibility, to be filled by the system in the selection of a value for priceA, priceB, manufAqty and manufBqty for every market, but the selection must satisfy a certain constraint in the ASSERT clause. Then the optimization query will be as follows:

```
SELECT P.marketID, P.priceA, P.priceB,
       P.manufAqty, P.manufBqty,
       expectedRevenue = expect(totalRevenue),
       expectedCost = expect(totalCost),
       expectedProfit = expect(profit)
MAXIMIZE SUM(expectedProfit)
FROM PredictedSummary P
WHERE  P.marketID <> 'NYC' and
       P.marketID <> 'Washington DC'
ASSERT expect(P.totalRevenue)>=100000 and
       Prob(P.profit<0)<=0.05
```

Intuitively, this query first finds an optimal instantiation of values into all ND-nulls, which appear in the input and intermediate tables constructed by the optimization query and the related views. It then executes the optimization query as a regular DG-SQL query, except ND-values are the optimal ones found. We formalize these notions and provide more explanation below. Generally, the optimization syntax of DG-SQL involves the following.

`Maximize/Minimize` clause needs two items, namely `aggr` and `attr`, where `aggr` is an aggregation function (e.g.,sum and avg), and `attr` is an attribute name that appears in the `Select` clause. Example: `Maximize sum(profit)`. Intuitively, this means that we want to choose the "parallel" s-relation that has the maximum sum of the profit values in the resulting tuples, among all the "parallel" s-relations.

`Assert` clause is syntactically the same as the `Having` clause. The condition in the `Assert` clause is applied to the (only) group of all the tuples. Intuitively, an assert clause is to choose the "parallel" s-relations that we want to consider in optimization. That is, a "parallel" s-relation is considered in the optimization only if it satisfies the `Assert` condition. In contrast, the `Having` clause is to choose the groups (resulting from a `Group by` clause) *one* "parallel" s-relation. Note even if no groups satisfy the condition in the `Having` clause, that particular "parallel" s-relation still exists (as the empty relation). An empty `Assert` clause means that all the "parallel" s-relations are considered in the optimization.

To precisely understand the optimization semantics in DG-SQL, note that a sequence of specific selections in nondeterministic choice statements corresponds to an *execution path*. We define a *feasible* execution path as one that satisfies (1) the range conditions in the choice statements, and (2) the assert-constraint statements. An *optimal* execution path is a *feasible* path that produces the optimal value in the Minimize/Maximize clause, among all *feasible* execution paths.

Given a DG-SQL query Q, input database I, and a `Minimize` or `Maximize` Expr clause, we denote by QE the set of all feasible query evaluations e. For a particular feasible query evaluation $e \in QE$, we denote by $Expr(e)$ the value of the `Expr` in the Minimize/Maximize clause for the evaluation e.

An *optimal* query evaluation is a solution to the optimization problem OP below:

$$Optimize \quad Expr(e) \quad s.t. \quad e \in QE$$

where *Optimize* stands for *Minimize* or *Maximize*, depending on whether Minimize or Maximize clause is used in the query.

Note that a solution to this problem may not be unique, as more than one feasible query evaluation $e \in QE$ may have the minimal/maximal $Expr(e)$. An optimal query evaluation e defines the values for each execution of a `choice` method.

The computation of the optimization DG-SQL query Q is the regular computation where the values for each nondeterministic parameter are those corresponding to an optimal query computation e.

8. LEARNING TRANSFORMERS IN DG-SQL

Up to now we assumed that a set of transformers (consistent with the dependency graph) is readily available for use. However, some transformers, such as *demandA*, *manuf*, and *supply* may not be known. However, the user may have partial knowledge about transformers. For example, for the *manuf* transformer, the user may not know the coefficients used, but may know a *template* of the transformer, i.e., the function defined by it up to some parameters, which are not known, but can be learned. Given a learning set for such transformer, which a relation with the same attributes (names and types) as the input and output variables of the transformer, below we show how a learning query can be imposed.

```
DEFINE TRANSFORM manuf(real manufAqty, manufBqty):
                (real i1Qty, i2Qty, i3Qty, manufCost) {
i1Qty = L-null*manufAqty + L-null*manufBqty +
        Gaussian(0, VAR);
i2Qty = L-null*manufAqty + L-null*manufBqty +
        Gaussian(0, VAR);
i3Qty = L-null*manufAqty + L-null*manufBqty +
        Gaussian(0, VAR);
manufCost = L-null*manufAqty+L-null*manufBqty +
        Gaussian(0, VAR)
}
LEARNING RELATION
SELECT M.manufAqty, M.manufBqty,
       M.i1qty, M.i2qty, M.i3qty, M.manufCost
MINIMIZE SUM
   (t.i1Qty - M.i1qty)^2 + (t.i2Qty - M.i2qty)^2 +
   (t.i3Qty - M.i3qty)^2  + (t.manufCost - M.manufCost)^2
FROM Manufacturing&Procurement M
LET t = manuf(M.manufAqty, M.manufBqty)
WHERE M.year >= 2001
```

Note that the learning is integrated syntactically with TRANSFORM definition statement. In this case, Standard Regression is a general learning algorithm understood by DGMS. The TRANSFORM definition above gives a parametric function f, where the parameters are the L-null values used in its definition. The use of standard regression means that the L-null parameters must be instantiated to minimize the sum of the squares of errors.

More generally, in DGMS, the learning processing is integrated into the query language via the notion of transformer. Learning is a process to acquire necessary transformers by using learning sets produced by queries from the database. In turn, transformers can be used to apply the knowledge in further queries. This process can be nested as necessary.

Furthermore, a transformer may need query the database for predicting for an instance. For example, in order to predict what items a customer is likely to pick up while visiting a store, the transformer may need to look at the items already in the basket. Hence, a transformer may not simply be a "static" mathematical formula. Rather, it may be a "live" database query. In other words, a learned transformer may be a query expression that performs the transformation, or prediction, by query the database.

9. DATA ACQUISITION

The general idea to tackle data acquisition problem is based on the following observation. It is neither feasible, nor necessary, to demand all possible predictions to be of high quality before any decision query is conceived and executed. A more economical approach is to pin-point down to where the problem might be in a decision making process, and ask the system to suggest ways to improve the prediction quality. In our case, the suggestion will be in terms of what kind of data should the system acquire. This falls into the domain of active learning, but with significant difference when used in decision guidance (see Related Work section).

This data acquisition requirement in a DGMS can be translated to the following two capability requirements. First, the system must be able to reason about the quality of prediction. Second, the system should be able to "trace back" to specific predictions whose quality has a impact on the confidence of the user decision.

In general, prediction qualities of transformers can be provided by using test data. Every data mining and machine learning task can usually be associated with a testing phase. For statistical learning, hypothesis testing may be useful to judge the quality of the learned model (e.g., see [30]). For general learning algorithms, cross testing methods have been used (e.g., see [21]).

We assume that prediction transformers will provide the quality measure as part of the output. For example, for probabilistic transformers that gives normal distributions, we may require confidence intervals for its expectation and its variance as two additional attributes. With these confidence intervals, in the query, it is possible to give confidence to a satisfied condition on the random variable.

When the the confidence of a query result is derived, the question then is to "trace back" the transformers whose improvement has the greatest impact on the quality of the results. This becomes an optimization problem. Indeed, if we assume that each transformer can have increased quality, but the quality is associated with a cost, we can then try to minimize the cost while increase the quality of the whole query result to a satisfactory level. An interesting future research question is how to automatically generate such an optimization query.

Note that the above strategy does not only provide the quality requirement of a transformer as a whole, but the quality of particular use of the transformer. Indeed, the transformer may be able to predict for all possible values in the input domain, but a particular decision problem does not actually use the transformer in all the possible values. Instead, the use of the transformer is only for a particular input values. In this case, we do not need to increase the quality of the transformer as a whole, but rather to increase the quality in a subdomain.

10. RELATED WORK

As mentioned above, significant advances have been made in the areas of operations research, mathematical and constraint programming, machine learning and data mining, and database systems. We emphasize that this paper is to introduce a seamless integration of different components for a decision-guidance management system. To the best of our knowledge, such an effort has never been reported in the literature. Nevertheless, our integration brings in new perspective to the various components of the system, and our solutions are innovative in many cases. Since each area has a vast literature, we only briefly discuss researches that are closely related to this paper, and point out the major differences.

Probabilistic data have attracted attention of researchers for a long time under the general area of uncertainty or imprecise data, e.g., [9, 3]. Data integration as well as integration of information retrieval with database system also brings interests in probabilistic data. Representative papers are [18, 15, 14, 13, 35, 29]. However, most papers in the literature deal with the situation that a tuple in a database table is labeled with a probability value, namely what's the probability that the tuple belongs to the table. In contrast, we are interested in the situation that an attribute is a (complete) probability distribution. There are also papers that treat the case of attributes with probability distribution but most concentrate on estimating the unknown attribute values, e.g., [23, 24, 11]. However, in all these works, the probabilistic information in explicit form. In contrast, in DG-SQL, stochastic attributes are defined via a flexible user-defined stochastic process, captures as transformers. Furthermore, contrary to existing work, DG-SQL stochastic attributes are used in prediction, simulation and decision optimization queries. This is an extension [6], which provided an efficient evaluation strategy for querying such relations.

The integration of databases with data mining has attracted researchers for long, e.g., [20, 10, 26, 38, 2, 1]. Most papers are dealing with using databases in managing the data mining aspects, e.g., pre-processing the data for data mining, specifying the kinds of rules to be mined, and filter out rules after mining. The use of data mining results in further querying in an iterative fashion has attracted less attention, but people started to attack the problem [2, 1] with a logic programming paradigm. One of the major concerns of a

DGMS is to use learned rules for prediction and decision optimization. A particularly interesting statistical learning method is the Probabilistic Relational Model [19], which learns a Bayesian network from a relational database. The user knowledge in terms of dependency structure mentioned earlier is inspired by this learning method. DG-SQL, however, in addition to the dependence structures, incorporates partial user knowledge, which can range from zero knowledge (and then it will be very similar to Bayesian network) to full knowledge (in which the output random variables are deterministically defined), or anything in between.

In terms of decision optimization, operations research has been a major field of mathematical sciences for a century, with mature theories and practical tools. Two related issues are critical for decision-guidance, namely decision problem modeling and efficient solvers. In a DGMS, we are mainly concerned with the decision problem modeling as efficient solvers are available to many mathematical programming problems. We believe innovation in the decision problem modeling is still needed, to bridge the gap between the decision problem domain and the mathematical constraint formulation. There are languages specialized in constraint modeling, such as [16, 5], and constraint programming languages, such as [22, 25, 8, 32, 37, 27]. However, these languages are focused on providing a programming environment for writing constraints or guiding search. A more intuitive approach is to use simulation languages, such as [12, 34], or even other high-level programming languages, like Java and SQL, to specify business processes while taking decision variables as possible inputs to the business process. The system should be able to translate the process written in these languages into mathematical programming formulation and provide decision optimization support. However, existing simulation languages and high-level languages do not support optimization. The closest language that provides a similar functionality but in a limited context is [17], since fundamentally it is still a language to describe equations. In contrast, we adopt the approach of [7] and allow users to write decision queries in a database language that are closer to a process simulation, and then let the system provide solutions to the so-described decision optimization problems with appropriate methods of solving the mathematical programming problem.

Data acquisition is also a vast research area, ranging from experiment design to active learning. Most of the work, however, is on producing better models. In the DGMS framework, better decision is our guiding principle, not better models, although they are closely related. A recent paper [28] showed that these two perspective makes a difference in terms of data acquisition. The authors introduced a concept of decision-centric data acquisition. The specific learning task dealt with in [28] is classification.

11. CONCLUSION AND FUTURE WORK

To the best of our knowledge, this is the first paper that introduced a unified data model and a query language to support decision-guidance, based on a unified domain knowledge representation. Many research questions remain open. Key challenges include the development of stochastic relational algebra and data structures to support efficient prediction query evaluation; developing and proving correctness of reduction procedures that map DG-SQL optimization and learning formulations to the standard mathematical programming; integrating constraint programming to guide combinatorial optimization cases; developing incremental query evaluation techniques that will include incremental learning; and implementing a proof-of-concept prototype.

12. REFERENCES

[1] Marcelo A. T. Aragão and Alvaro A. A. Fernandes. Combined inference databases, technical report (version 1.0.0). The University of Manchester, June 2004.

[2] Marcelo A. T. Aragão and Alvaro A. A. Fernandes. Seamlessly supporting combined knowledge discovery and query answering: A case study. In Einoshin Suzuki and Setsuo Arikawa, editors, *Discovery Science*, volume 3245 of *Lecture Notes in Computer Science*, pages 403–411. Springer, 2004.

[3] D. Barbara, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Transactions on Knowledge and Data Engineering*, 4(5):487–50, October 1992.

[4] G. Belvaux and L.A. Wolsey. BC-prod: A specialized branch-and-cut system for lot-sizing problems. *Management Science*, 46:724–738, 2000.

[5] Ronald F. Boisvert, Sally E. Howe, and David K. Kahaner. Gams: a framework for the management of scientific software. *ACM Trans. Math. Softw.*, 11(4):313–355, 1985.

[6] Alexander Brodsky, Carlotta Domeniconi, and David Etter. Regression databases: Probabilistic querying using sparse learning sets. In *Proc. the Fifth International Conference on Machine Learning and Applications (ICMLA'06)*, 2006.

[7] Alexander Brodsky and Hadon Nash. CoJava: Optimization modeling by nondeterministic simulation. In Frédéric Benhamou, editor, *CP*, volume 4204 of *Lecture Notes in Computer Science*, pages 91–106. Springer, 2006.

[8] Y. Caseau, F.X. Josset, and F. Laburthe. CLAIRE: combining sets, search and rules to better express algorithms. *Theory and Practice of Logic Programming*, 2(06):769–805, 2002.

[9] R. Cavallo and M. Pittarelli. The theory of probabilistic databases. In *Proc. Int'l. Conf. on Very Large Data Bases*, page 71, Brighton, England, August 1987.

[10] Surajit Chaudhuri. Data mining and database systems: Where is the intersection? *IEEE Data Eng. Bull*, 21(1):4–8, 1998.

[11] Reynold Cheng, Yuni Xia, Sunil Prabhakar, Rahul Shah, and Jeffrey Scott Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *VLDB*, pages 876–887. Morgan Kaufmann, 2004.

[12] Ole-Johan Dahl and Kristen Nygaard. Simula: an algol-based simulation language. *Commun. ACM*, 9(9):671–678, 1966.

[13] Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875. Morgan Kaufmann, 2004.

[14] Debabrata Dey and Sumit Sarkar. A probabilistic relational model and algebra. *ACM Transactions on Database Systems*, 21(3):339–369, September 1996.

[15] Thomas Eiter, Thomas Lukasiewicz, and Michael Walter. A data model and algebra for probabilistic complex values. *Ann. Math. Artif. Intell*, 33(2-4):205–252, 2001.

[16] R. Fourer, D. M. Gay, and B. W. Kernighan. A modeling language for mathematical programming. *Manage. Sci.*, 36(5):519–554, 1990.

[17] Peter Fritzson and Vadim Engelson. Modelica - a unified object-oriented language for system modelling and simulation. In *ECCOP '98: Proceedings of the 12th European Conference on Object-Oriented Programming*, pages 67–90, London, UK, 1998. Springer-Verlag.

[18] Fuhr, Norbert. A probabilistic relational model for the integration of IR and databases. In *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, DBMS/IR Integration, pages 309–317, 1993.

[19] Lise Getoor and John Grant. PRL: A probabilistic relational language. *Machine Learning*, 62(1-2):7–31, 2006.

[20] J. Han, Y. Fu, K. Koperski, W. Wang, and O. Zaiane. DMQL: A data mining query language for relational databases. In *1996 SIGMOD'96 Workshop. on Research Issues on Data Mining and Knowledge Discovery (DMKD'96)*, Montreal, Canada, 1996.

[21] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers, San Francisco, 2001.

[22] Pascal Van Hentenryck, Laurent Michel, Laurent Perron, and Jean-Charles Régin. Constraint programming in OPL. In Gopalan Nadathur, editor, *PPDP*, volume 1702 of *Lecture Notes in Computer Science*, pages 98–116. Springer, 1999.

[23] Wen-Chi Hon, Zhongyang Zhang, and Nong Zhou. Statistical inference of unknown attribute values in databases. In Bharat Bhargava, Timothy Finin, and Yelena Yesha, editors, *Proceedings of the 2nd International Conference on Information and Knowledge Management*, pages 21–30, New York, NY, USA, November 1993. ACM Press.

[24] Wen-Chi Hou and Zhongyang Zhang. Enhancing database correctness: a statistical approach. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 223–232, San Jose, California, 22–25 May 1995.

[25] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *POPL '87: Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 111–119, New York, NY, USA, 1987. ACM Press.

[26] Hasan M. Jamil. Ad hoc association rule mining as SQL3 queries. In Nick Cercone, Tsau Young Lin, and Xindong Wu, editors, *ICDM*, pages 609–612. IEEE Computer Society, 2001.

[27] Jean-Francois Puget and Michel Leconte. Beyond the glass box: Constraints as objects. In *International Logic Programming Symposium*, pages 513–527, 1995.

[28] Maytal Saar-Tsechansky and F. Provost. Active learning for decision-making. Working paper (CeDER-04-06, Stern School of Business, New York University), 2006.

[29] Anish Das Sarma, Omar Benjelloun, Alon Y. Halevy, and Jennifer Widom. Working models for uncertain data. In *ICDE*. IEEE Computer Society, 2006.

[30] Oleg Seleznjev and Bernhard Thalheim. Joining multiple random tables. http://www.matstat.umu.se/personal/Oleg/personal/JoinVariability2_1.pdf, 2006.

[31] David Simchi-Levi, Xin Chen, and Julien Bramel. *The Logic of Logistics: Theory, Algorithms and Applications for Logistics and Supply Chain Management.* Springer, Series in Operations Research and, 1997.

[32] G. Smolka, M. Henz, and J. Wurtz. Object-oriented concurrent constraint programming in Oz. *Principles and Practice of Constraint Programming*, pages 29–48, 1995.

[33] Sridhar Tayur, Ram Ganeshan, and Michael Magazine (Eds.). *Quantitative Models for Supply Chain Management.* Kluwer, International Series on Operations Research and Management Science, 1998.

[34] Timothy Thomasma and James Madsen. Object oriented programming languages for developing simulation-related software. In *WSC' 90: Proceedings of the 22nd conference on Winter simulation*, pages 482–485, Piscataway, NJ, USA, 1990. IEEE Press.

[35] Octavian Udrea, Yu Deng, Edward Hung, and V. S. Subrahmanian. Probabilistic ontologies and relational databases. In *OTM Conferences (1)*, volume 3760 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2005.

[36] S. Voss and D.L. Woodruff. *Introduction to Computational Optimization Models for Production Planning in a Supply Chain (2nd Edition).* Springer, 2006.

[37] M. Wallace, S. Novello, and J. Schimpf. ECLiPSe: A platform for constraint logic programming. *ICL Systems Journal*, 12(1):159–200, 1997.

[38] Haixun Wang and Carlo Zaniolo. ATLaS: A native extension of SQL for data mining. In Daniel Barbará and Chandrika Kamath, editors, *SDM*. SIAM, 2003.