Scalable Test Data Generation from Multidimensional Models

Emina Torlak
U.C. Berkeley*
emina@eecs.berkeley.edu

ABSTRACT

Multidimensional data models form the core of modern decision support software. The need for this kind of software is significant, and it continues to grow with the size and variety of datasets being collected today. Yet real multidimensional instances are often unavailable for testing and benchmarking, and existing data generators can only produce a limited class of such structures. In this paper, we present a new framework for scalable generation of test data from a rich class of multidimensional models. The framework provides a small, expressive language for specifying such models, and a novel solver for generating sample data from them. While the satisfiability problem for the language is NP-hard, we identify a polynomially solvable fragment that captures most practical modeling patterns. Given a model and, optionally, a statistical specification of the desired test dataset, the solver detects and instantiates a maximal subset of the model within this fragment, generating data that exhibits the desired statistical properties. We use our framework to generate a variety of high-quality test datasets from real industrial models, which cannot be correctly instantiated by existing data generators, or as effectively solved by general-purpose constraint solvers.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging—*Testing tools*; D.2.4 [**Software Engineering**]: Software / Program Verification—*Formal methods*

Keywords

Test data generation, multidimensional models, specification, constraint solving

1. INTRODUCTION

Background and Motivation. OLAP (Online Analytical Processing) technologies provide interactive decision support in many domains, enabling users to obtain abstract yet intuitive views of very large datasets, and to pose "what-if" queries about the impact of hypothetical view changes on the underlying data. As such, they have become indispensable tools for experts in many fields (*e.g.*, business, medicine, education, research, and government [31]), and the demand for them is growing. With the rapid increase in the variety and volume of data being collected, there is significant interest in extending decision support facilities to both non-relational datasets (such as streams [14], sequences [29], text [27], and networks [45]) and extremely large relational datasets [11, 6] that exceed storage capacities of traditional data warehouses.

A key feature—and attraction—of OLAP is the use of *multidimensional data models* [19, 30, 31, 40], which enable both the interactivity and intuitiveness of OLAP tools. These models structure data in a multidimensional analysis space, called the *data cube*. Dimensions themselves are graphs, and their structure defines the levels of detail at which data can be summarized for viewing and manipulation. For example, Fig. 1c shows a toy person dimension, whose structure specifies that data about individuals—such as the fitness data in Fig. 1a—can be summarized by gender and location, which is given at the level of cities, states or provinces, and countries.

Queries on a multidimensional dataset compute projections, or views, of the underlying data cube. OLAP-style tools exploit this to achieve interactive response times, by answering queries about less detailed views of the data (*e.g.*, at the level of countries) using more detailed, precomputed views (*e.g.*, at the level of cities) [39]. But the application of this core technique is complicated by several hard problems, such as: (1) ensuring that all dimensional structures are summarizable [32], which is a pre-condition for obtaining correct results from precomputed views, and (2) selecting an optimal set of views to precompute for a given analysis space. Solving the first problem for all but simplest structures requires intricate dimension transformations [31, 39]. The second problem is not only NP-complete [15] but it is also inapproximable if $P \neq NP$ [23]. It is usually addressed with sophisticated view selection heuristics (*e.g.*, [15, 1]).

Like most software, OLAP-style tools are validated through testing. However, OLAP systems need large data sets with special characteristics—evaluating view selection heuristics at scale, for example, requires a variety of large data sets with summarizable dimensions that have different (graph) structures and statistical properties. Yet real, structurally rich datasets are hard to come by [38] due to privacy and proprietary concerns, and existing database synthesizers [2, 4, 5, 17, 18, 28, 41] are of limited help. They can generate simple tree-shaped dimensions but not (valid) DAGs such as person, since, as we will show, the data generation problem for these structures is itself NP-hard.

*This research was conducted at LogicBlox, Inc.

SIGSOFT/FSE'12, November 10-18, 2012, Cary, NC, USA.

Approach. In this paper, we propose a new model-based approach to synthesizing a rich class of multidimensional structures, suitable for systematic testing and evaluation of decision support software. Our approach is implemented in an industrial-strength testing framework, called TestBlox, which provides a small, declarative modeling language, and a scalable solver for generating sample data from TestBlox specifications. The language is complete [2], in the sense that it can express every multidimensional structure, and it captures common features of existing multidimensional models [19, 30, 31, 40].

The solver takes as input a multidimensional model and, optionally, a set of statistical constraints on the *shape* (*i.e.*, distribution properties) and *scale* (*i.e.*, size) of the desired instance of that model. The model consists of a schema, given as a DAG over abstract categories (*e.g.*, Fig. 1c), together with a set of integrity constraints. Its instances, if any, are DAGs that conform to the schema and that satisfy the integrity constraints. While deciding the satisfiability of an arbitrary TestBlox model is NP-hard, most of the models observed in practice fall within a fragment of the modeling language that can be instantiated in polynomial time. The TestBlox solver identifies and solves constraints in this fragment, generating a sample instance that satisfies a maximal (polynomially solvable) subset of the input model, and that exhibits the desired statistical properties (when possible). Instance generation is parallelized across all model components, including those related by global invariants.

Usage Scenarios. Scalable, model-based synthesis of dimensional structures, as provided by TestBlox, is helpful in many validation and benchmarking scenarios:

Testing and performance tuning of decision support applications. In practice, domain-specific decision support tools are usually built on top of a generic OLAP engine. The *facts* to be analyzed are stored as tuples in a relational database. Each fact associates a point in a structured analysis space with a set of numerical values, or *measures*. For example, Fig. 1 illustrates a toy fitness survey application, where the analysis space consists of two dimensional structures, person × education, and the measures specify the height, weight and BMI (body mass index) for a set of people. The user of the application can view and manipulate the given fitness facts with respect to the analysis space by posing queries such as "If the average BMI for people of all education levels in each country were to decrease by 2 points, what would be the corresponding decrease in the average weight of college educated males?" (See Fig. 1b for the answer.)

As in the case of traditional database applications, the OLAP developer uses integrity constraints on fact and dimension schemas to express preconditions on the inputs accepted by the application. To test the application, however, he may need instances that are not only valid but that also have specific statistical properties—for example, a small dimensional instance with uniformly distributed edges for functional testing, and many large instances with various edge distributions for performance tuning. TestBlox can help in either case, by quickly synthesizing valid dimensional structures of various shapes and sizes, given only the data model and relevant statistical constraints.[1] In fact, even if the developer had access to a real, large end-user dataset (which is rare in practice), he would still need synthetic data to expose corner cases, since sampling a single real dataset would produce a biased test suite of similarly shaped inputs.

Benchmarking of decision support systems. Many common modeling patterns lead to non-summarizable dimensional structures [30, 31, 40], which interfere with basic OLAP optimizations and have to be specially handled either at the engine or the application level. The TestBlox modeling language can express these patterns (Sec. 2), and it can be used, together with statistical constraints, to precisely describe a wide variety of non-summarizable dimensional structures of different shapes and sizes. An application developer can use the data generated from such specifications to benchmark competing OLAP engines, in order to determine which one offers the best trade-off between performance and native support for the kind of structures that arise in the targeted application domain.

Validation of new decision support facilities. In addition to generating non-summarizable structures, TestBlox can also synthesize any desired summarizable structure. We expect that such structures would be particularly useful for validation of new kinds of decision support facilities [6, 11, 14, 27, 29, 45], as they usually assume summarizable analysis spaces. For example, one could apply random testing to validate a new cloud-based OLAP engine [11, 6] against a standard relational one [34], using a DAG generator [8] to synthesize random dimensional schemas, and TestBlox to synthesize random summarizable instances of those schemas.

Contributions. To the best of our knowledge, TestBlox is the first framework to tackle the problem of generating large-scale, realistic instances of rich multidimensional models. Its key enabling features are also the novel contributions of this paper:

- A small but expressive modeling language, with a practical fragment that can be detected and decided in polynomial time (relative to model size).
- A solver for the polynomial fragment, with algorithms for computing data generation parameters that yield instances of desired shape and scale.
- A data generation architecture that produces instances in polynomial time (relative to instance size) and in parallel, even in the presence of global constraints.

The modeling language and statistical (or, *soft*) constraints are introduced in Sec. 2, followed by a discussion of hardness results and the definition of the polynomially solvable fragment in Sec. 3. The solver and the architecture are presented next (Sec. 4), as is an evaluation of the framework on three real, industrial models (Sec. 5). A discussion of related work (Sec. 6) and future directions (Sec. 7) conclude the paper.

2. MULTIDIMENSIONAL MODELING

To illustrate the multidimensional analysis problem, consider the toy application scenario in Fig. 1. The application provides multidimensional analysis for facts collected via an online fitness survey. The goal of the survey is to investigate the relationship between individuals' education level and fitness, as (roughly) measured by the body mass index. The end user is a public health policy analyst. She wants to use

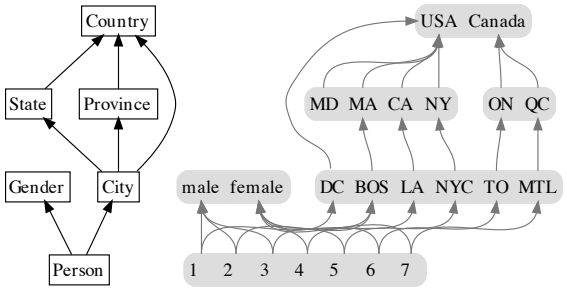
[1] TestBlox will also generate a valid fact database with desired statistical properties, but relational data generation is not unique to our system (see, *e.g.*, [2, 4, 5, 17, 18, 28, 41]).

person	education	height (m)	weight (kg)	BMI
1	H.S.D.	1.85	90	26
2	B.Sc.	1.70	83	29
3	M.Sc.	1.80	85	26
4	H.S.D.	1.64	75	26
5	B.A.	1.55	60	23
6	M.Eng.	1.72	65	22
7	Ph.D.	1.59	61	24

(a) fitness facts

	male	female
high school	7.0	5.8
college	6.4	4.7
graduate	6.6	4.9

(b) A "what-if" view of fitness facts: decrease in average weight (in kg) after a 2 point decrease in the average BMI



(c) person schema (left) and instance (right)

Figure 1: Data for a toy fitness survey application

the application to view, for example, the effect of a 2 point decrease in the average BMI of the entire North American population on the average weight of males and females with different education levels (Fig. 1b).

Based on these analysis requirements, a developer may design the application to provide an analysis space with two dimensions: person and education. The education dimension (not shown) classifies academic degrees into "high school," "college" and "graduate" in the usual way. The person dimension (Fig. 1c) has a more interesting structure, classifying individuals according to gender and location. In the rest of this section, we use the person dimension to introduce the TestBlox modeling language, as well as the statistical constraints that the developer may provide to our solver in order to obtain test instances of the person model.

2.1 Dimension Model

Like the standard relational data model, a *dimension model* consists of a schema and a set of integrity constraints. A *dimension schema* represents the upper bound on the structure of valid *dimension instances*; the person schema, for example, admits many instances, including the one shown in Fig. 1c. Some instances that conform to the schema, however, violate domain-specific requirements, such as Gender having exactly two members. *Dimension (integrity) constraints* encode these additional requirements, restricting the set of instances that are valid under the model as a whole. TestBlox schema and constraints form a complete [2] multidimensional modeling language that naturally captures common modeling patterns.

Dimension Schema and Instances. A dimension schema is traditionally represented as a DAG (directed acyclic graph) over *category names* or, simply, categories. Each category represents an abstract analysis concept. Schema edges define abstraction relations between categories, mapping more detailed categories, such as City, to less detailed (*i.e.*, more abstract) ones, such as State.

Definition 1 (Dimension Schema). *A* dimension schema *is a directed acyclic graph* $D = (C, \prec)$. *Schema vertices, called* categories*, are drawn from an infinite set of category names* $\mathbf{C}$. *The expression* $c_i \prec c_j$ *denotes the edge* $\langle c_i, c_j \rangle$ *in the schema graph. We say that* c_i *is a* parent *of* c_j*, and that* c_j *is a* child *of* c_i. *The symbol* $\preceq$ *stands for the reflexive transitive closure of* $\prec$.

(C1)	$\lvert\text{Gender}\rvert \leq 2$	*cardinality*
(C2)	$\lvert\text{Gender}\rvert \geq 2$	
(C3)	$\lvert\text{Country}\rvert \leq 196$	
(C4)	$\text{Person} \rightarrow \text{Gender} \vee \text{City}$	*rollup*
(C5)	$\text{City} \rightarrow \text{State} \odot \text{Province} \odot \text{Country}$	*exclusive rollup*
(C6)	$\text{Person} \leftarrow \text{Gender}$	*drilldown*
(C7)	$\text{State} \odot \text{Province} \leftarrow \text{Country}$	*exclusive drilldown*

Figure 2: Dimension constraints for the person schema

A dimension schema describes a family of dimension instances, which are themselves DAGs over *member names* (or members) associated with each category. Each schema edge $c_i \prec c_j$ represents a set of instance edges between the members of categories c_i and c_j. The reflexive transitive closure of an instance graph, called the *rollup relation*, is constrained to be functional, or *strict* [19, 22, 25], between every pair of categories. In other words, a member of any category (such as Person) may be abstracted by—or, equivalently, may *rollup to*—at most one member of another category (such as Country).

Definition 2 (Dimension Instance). *A* dimension instance *is a triple* $d = (D, m, <)$, *where* $D = (C, \prec)$ *is a dimension schema;* m *is a function from each category* $c \in C$ *to a non-empty, finite set of* members*; and* $<$ *relates members* x_i *and* x_j *only if* $x_i \in m(c_i), x_j \in m(c_j)$ *and* $c_i \prec c_j$. *The member set of each category is drawn from an infinite set of member names* $\mathbf{M}$*, and member sets of different categories are disjoint. The* rollup relation $\leq$*, defined as the reflexive transitive closure of* $<$*, is functional between every pair of categories. That is, for all categories* $c_i \preceq c_j$*, and for every member* $x_i \in m(c_i)$*, there is at most one member* $x_j \in m(c_j)$ *such that* $x_i \leq x_j$.

Dimension Constraints. In addition to a schema, a dimension model may also specify a set of integrity constraints that instances have to satisfy. These include *cardinality*, *(exclusive) rollup*, and *(exclusive) drilldown* constraints. Cardinality constraints bound the size of a category's member set in every instance, while (exclusive) rollup and drilldown constraints specify which $<$ edges must exist between adjacent categories and how they are structured.

Figure 2 shows examples of each kind of constraint, applied to the person schema. The cardinality constraints C1-C3 require each instance to have exactly two genders and no more than the (current) total number of countries in the world. The rollup constraint C4 forces every person to rollup to a member of *some* specified child category, expressing the expectation that the toy survey results may be incomplete but will always include some gender or location information about every person. The exclusive rollup constraint C5, on the other hand, ensures that each city rolls-up (via $<$) to a member of *exactly one* specified child category, expressing the domain knowledge that North American cities belong to either a state, or a province, or a country. These two rollup patterns specify *(non-)overlapping specialization* [30, 31].

Drilldown constraints, such as C6 and C7, provide complementary functionality to rollup constraints, and help express *(non-)overlapping generalization* [25, 30, 31]. According to C6 and C7, each gender must have members of *some* specified parent category (in this case, only person) rolling-up to it, and each country must have members of *exactly one* specified parent category rolling-up to it (via $\prec$). Every person instance therefore contains at least one person of each gender, and all countries contain either states or provinces, but not both. Note that C7 does not prevent countries from having cities as children, along with either states or provinces. All sample constraints are satisfied by the instance in Fig. 1c.

DEFINITION 3 (DIMENSION CONSTRAINT). *A constraint ϕ is a* dimension constraint *on a schema $D = (C, \prec)$ iff it takes one of the following forms:*

(1) $|c| \leq k$ or $|c| \geq k$, where $c \in C$ and $k \geq 1$;

(2) $c \to \bigvee_{i\geq 1} c_i$ or $c \to \bigodot_{i>1} c_i$, where $c \prec c_i$ for all i;

(3) $\bigvee_{i\geq 1} c_i \leftarrow c$ or $\bigodot_{i>1} c_i \leftarrow c$, where $c_i \prec c$ for all i.

A dimension instance $d = (D, m, <)$ satisfies *ϕ, written $d \models \phi$, iff it fulfills the conditions given below, or their natural extension to omitted constraints:*[2]

(1) $d \models |c| \leq k$ iff $|m(c)| \leq k$;

(2) $d \models c \to \bigvee C'$ iff $\forall x \in m(c), \exists c_i \in C', \exists y \in m(c_i), x < y$;

(3) $d \models c \to \bigodot C'$ iff $\forall x \in m(c), \exists! c_i \in C', \exists y \in m(c_i), x < y$.

Dimension Model. The person schema and our sample dimension constraints collectively form a dimension model. To simplify reasoning about dimension models, we require that any included cardinality constraints be consistent and minimal. In particular, we disallow models with constraints that conflict with one another, such as $|c| \leq k$ and $|c| \geq k+1$, or that are mutually redundant, such as $|c| \leq k$ and $|c| \leq k+1$. We also require that no two constraints involve the same schema edge unless one is (exclusive) rollup and the other (exclusive) drilldown. The person model, for example, may not include another rollup constraint on the City $\prec$ State edge, but it could include a drilldown constraint on the same edge. The TESTBLOX solver automatically rejects models with constraints that violate these requirements.

DEFINITION 4 (DIMENSION MODEL). *A* dimension model *is a tuple $\mathcal{D} = (D, \Phi)$, where $D = (C, \prec)$ is a schema, and Φ is a set of dimension constraints on D. Each schema edge $c_i \prec c_j$ participates in at most two constraints in Φ, one (exclusive) rollup and one (exclusive) drilldown; and each category $c \in C$ is constrained by at most two non-conflicting cardinality constraints in Φ, one lower bound ($\geq$) and one upper bound ($\leq$). A dimension instance $d = (D, m, <)$* satisfies *the model $\mathcal{D}$, written $d \models \mathcal{D}$, iff it satisfies every constraint in Φ. That is, $d \models \phi$ for all $\phi \in \Phi$.*

Expressiveness. Despite its compactness, the TESTBLOX modeling language is *complete* [2] in the sense that it can be used to fully describe any dimension instance (by, *e.g.*, introducing a category for each member, constraining that category's cardinality to be exactly 1, and introducing a rollup constraint on each edge between singleton categories with adjacent members). Our language is also practical in that it naturally captures important families of instances—for example, all summarizable [30, 31, 32, 40] instances of a given schema. An instance of a schema is summarizable, and thus directly amenable to efficient multidimensional analysis, if its rollup relation is strict, total and surjective between every pair of categories related by the schema. We can describe all such instances of a schema using a model that constrains every schema edge $c_i \prec c_j$ with $c_i \to c_j$ and $c_i \leftarrow c_j$.

[2]Recall that '∃!' stands for 'there exists exactly one.'

```
(S1) Person { size = 1000 }                                   scale
(S2) Person → City {                                          shape
       source = .85, target = .7,
       distribution = normal(30, 5) }
```

Figure 3: Sample soft constraints for the toy retail application

2.2 Soft Constraints

While an application or a technique needs to work on all valid instances of a multidimensional model, this set is, in most cases, too large to be tested exhaustively. As a result, testing efforts generally focus on a subset of instances that are representative, or interesting, in some way. To guide TESTBLOX toward generation of such instances, we provide a set of *soft constraints*, which describe the desired scale and distribution characteristics of generated data. We also define two simple metrics to quantify the degree to which an instance satisfies a given set of soft constraints. The TESTBLOX solver optimizes the value of these metrics when computing data generation parameters.

Figure 3 shows a sample of supported soft constraints, applied to the person schema. A *scale* constraint specifies the preferred size of a category; according to S1, for example, Person should contain roughly 1000 members. A *shape* constraint characterizes the rollup relation between adjacent categories using three parameters: the percentage of members in each category that participate in a rollup edge, and the distribution of rollup edges between participating members. For example, S2 describes a Person to City relation that maps 85% of people to 70% of the cities according to a normal distribution, where each participating city has, on average, 30 participating people rolling-up to it, with a standard deviation of 5.

For ease of exposition, we define soft constraints on a dimension schema using functions over categories and schema edges, as follows.

DEFINITION 5 (SOFT CONSTRAINTS). Soft constraints *f on a dimension schema $D = (C, \prec)$ are expressed with four functions. These are $n : C \to \mathbb{N}$; $s, t : C \times C \to [0..1]$; and $p : C \times C \to \mathbb{P}$, where $\mathbb{P}$ is the set of all probability distributions. The meaning of $f = (n, s, t, p)$ is defined in terms of a* desired *instance $d = (D, m, <)$ as follows. For all categories $c \in C$, $n(c) \approx |m(c)|$. For all schema edges $e = c_i \prec c_j$, $s(e) * |m(c_i)| \approx |\operatorname{dom}(<_e)|$ and $t(e) * |m(c_j)| \approx |\operatorname{ran}(<_e)|$, where $<_e$ is the rollup mapping from $m(c_i)$ to $m(c_j)$, and $\operatorname{dom}(<_e)$ and $\operatorname{ran}(<_e)$ are its domain and range. The number of members of c_i that rollup to a member of c_j varies according to p.*

Because soft constraints may not be precisely satisfiable, especially in the presence of dimension constraints, we define two metrics to quantify the conformance of an instance to the scaling and participation requirements encoded by n, s, and t. *Scale fidelity* measures the squared distance between the

desired and actual sizes of category member sets, and *shape fidelity* measures the squared distance between the desired and actual participations of category members in the rollup relation. Measuring, and optimizing, the conformance of instances to desired distributions is an area for future work.

DEFINITION 6 (FIDELITY). *Given a dimension instance $d = (D, m, <)$ and soft constraints $f = (n, s, t, p)$ on D, we define the* scale *and* shape fidelity *of d with respect to f as $\alpha_f(d) = \sum_{c \in C}(n(c) - |m(c)|)^2$ and $\beta_f(d) = \sum_{e=c_i \prec c_j}(s(e) * |m(c_i)| - |\operatorname{dom}(<_e)|)^2 + (t(e) * |m(c_j)| - |\operatorname{ran}(<_e)|)^2$. We say that d has* higher scale fidelity *than d' with respect to a model $\mathcal{D} = (D, \Phi)$ and soft constraints f iff $d \models \mathcal{D}$, $d' \models \mathcal{D}$, and $\alpha_f(d) \leq \alpha_f(d')$. Similarly, $d \models \mathcal{D}$ is said to have* higher shape fidelity *than $d' \models \mathcal{D}$ iff $\beta_f(d) \leq \beta_f(d')$.*

3. HARDNESS OF DATA GENERATION

To instantiate a multidimensional model, we first need to decide whether its constituent dimension models are satisfiable—*i.e.*, whether or not they each have an instance. This decision problem turns out to be NP-hard, and as a result, we cannot efficiently generate arbitrary dimension instances. Most practical models, however, tend to exhibit *regularity*, which TESTBLOX exploits. Such models, characterized by *tieredness* (which restricts the use of cardinality constraints) and *weak exclusivity* (which restricts the use of exclusive rollup and drilldown), can be satisfied in polynomial time and space.

The hardness of the dimension satisfiability problem, stated below, can be proved by reduction from 3SAT.[3] We do note that, perhaps surprisingly, the proof does not use exclusivity constraints—even a simpler version of the decision problem, with no exclusive rollup or drilldown constraints, is hard.

THEOREM 1. *Deciding the satisfiability of dimension models $\mathcal{D} = (D, \Phi)$ is NP-hard, even when Φ is limited to containing only cardinality, rollup and drilldown constraints.*

The key to ensuring tractability of exclusivity-free models is a simple and natural restriction on the use of cardinality constraints. The restriction, called tieredness, requires that the upper bound on the cardinality of every category be no smaller than the lower bound on the cardinality of each of its descendants. In other words, the model must admit a *hierarchical instance* in which every category has at least as many members as its more abstract descendants in the dimension schema. This is, in fact, an implicit assumption in nearly all dimensions observed in practice—more detailed categories rollup to less detailed ones with fewer members.

DEFINITION 7 (TIEREDNESS). *A dimension model $\mathcal{D} = (D, \Phi)$ over a schema $D = (C, \prec)$ is* tiered *iff $h_\Phi(c_i) \geq l_\Phi(c_j)$ for all $c_i \preceq c_j$, where h_Φ and l_Φ are defined as follows:*

$$h_\Phi(c) = \begin{cases} k & \exists k, |c| \leq k \in \Phi \\ \infty & \text{otherwise; and,} \end{cases} \quad l_\Phi(c) = \begin{cases} k & \exists k, |c| \geq k \in \Phi \\ 1 & \text{otherwise.} \end{cases}$$

DEFINITION 8 (HIERARCHY). *A dimension instance $d = (D, m, <)$ is* hierarchical *iff $|m(c_i)| \geq |m(c_j)|$ for all $c_i \prec c_j$.*

THEOREM 2. *A tiered dimension model $\mathcal{D}$ with no exclusive rollup or drilldown constraints is always satisfiable, and it has hierarchical instances that can be encoded (via closures) in polynomial time and space with respect to the size of $\mathcal{D}$.*

[3]All proofs can be found in the full version of the paper [43].

While tiered, exclusivity-free models are easily satisfied, the problem becomes intractable if we allow unrestricted use of either exclusive rollups or exclusive drilldowns. To maintain tractability, as well as practical expressiveness, we limit the use of these constraints so that a schema edge involved in an exclusive constraint is otherwise unconstrained—a condition called weak exclusivity. Weakly exclusive models can capture non-overlapping generalization and specialization, which is the main use case for exclusivity constraints, but the generalized or specialized rollup relation cannot also be explicitly constrained to relate all members of involved categories. For the purposes of data generation, the latter can be ameliorated by the use of soft constraints.

THEOREM 3. *Deciding the satisfiability of tiered dimension models $\mathcal{D}$, with either exclusive rollups or exclusive drilldowns, is NP-hard.*

DEFINITION 9 (WEAK EXCLUSIVITY). *A dimension model $\mathcal{D} = (D, \Phi)$ is* weakly exclusive *iff, for each exclusive rollup and drilldown constraint $\phi \in \Phi$, there is no other $\phi' \in \Phi$ that involves a schema edge constrained by ϕ.*

Tiered, weakly exclusive models are said to be regular. Many models exhibit regularity—for example, a model that captures all summarizable instances of a schema is regular. Another example of a regular model is the person model from our toy survey application, which is not summarizable. Regular models always admit at least one polynomially encodable hierarchical instance. As we will see in the next section, these instance encodings can be executed in parallel to efficiently generate test data of desired scale and shape.

DEFINITION 10 (REGULARITY). *A dimension model $\mathcal{D} = (D, \Phi)$ is regular iff it is tiered and weakly exclusive.*

THEOREM 4. *A regular dimension model $\mathcal{D}$ is always satisfiable, and it has hierarchical instances that can be encoded (via closures) in polynomial time and space with respect to the size of $\mathcal{D}$.*

4. GENERATING DATA FOR MAXIMAL REGULAR FRAGMENTS OF MODELS

Given a dimension model with soft constraints, TESTBLOX first *solves* the model with respect to the soft constraints, producing a set of closures that encode the desired instance: one closure for each edge in the model's schema. This *solution* is then *instantiated* by invoking the closures to generate an explicit representation of the instance (as a set of text files, one for each schema edge). Despite having to collectively satisfy the strictness requirement on dimension instances (Def. 2), the closures are constructed in such a way that they can be called in any order. As a result, instance generation can be fully parallelized, with each schema edge instantiated by a separate process, independently of all other edges.

Once a model is solved, the remainder of the data generation process is straightforward. This section therefore focuses on the four steps taken to solve a dimension model $\mathcal{D} = (D, \Phi)$ with respect to soft constraints f:

(1) find $(D, T \cup W)$, a maximal regular fragment of $\mathcal{D}$, since we know that such a fragment is efficiently solvable;

(2) solve the cardinality constraints $T \subseteq \Phi$ to obtain a cardinality $card(c)$ for each category, such that the solution is hierarchical, with maximal scale fidelity α_f;

MAX-REGULAR-FRAGMENT(D, Φ)

1 $T, W \leftarrow \{\}$
2 **for** $\phi_c \in \Phi$ **such that** *isCardinality*(ϕ_c) **do**
3 **if** *isTiered*$(D, T \cup \{\phi_c\})$ **then**
4 $T \leftarrow T \cup \{\phi_c\}$
5 **for** $\phi_e \in \Phi$ **such that** $\neg$ *isCardinality*(ϕ_e) **do**
6 **if** *isWeaklyExclusive*$(D, W \cup \{\phi_e\})$ **then**
7 $W \leftarrow W \cup \{\phi_e\}$
8 **return** $(D, T \cup W)$

Figure 4: Algorithm for computing a maximal regular fragment of a dimension model $\mathcal{D} = (D, \Phi)$

(3) solve the constraints $W \subseteq \Phi$ on schema edges, using shape fidelity β_f as a guide, to obtain rollup parameters $dom(e)$ and $ran(e)$ for each edge; and,

(4) use the rollup parameters to construct individual rollup closures for all schema edges.

4.1 Finding a Maximal Regular Fragment

The first step to efficient generation of dimension instances is to check whether the input model $\mathcal{D} = (D, \Phi)$ is regular—and therefore solvable in polynomial time. If so, we proceed to the next step using $\mathcal{D}$. Otherwise, we proceed using a (locally) maximal fragment of $\mathcal{D}$ that is regular, discarding the constraints outside of the fragment with a warning.

Figure 4 shows a greedy algorithm for detecting and enforcing regularity. The algorithm simply builds a regular model $(D, T \cup W)$ out of the schema D, a maximal subset $T \subseteq \Phi$ of the cardinality constraints, and a maximal subset $W \subseteq \Phi$ of the constraints on schema edges. The set T is maximal (lines 2-4) in that it cannot be augmented with any omitted cardinality constraints without violating the tieredness condition—*i.e.*, $(D, T \cup \{\phi_c\})$ is not tiered for any cardinality constraint $\phi_c \in \Phi \setminus T$. Similarly, W is maximal (lines 5-7) in that $(D, W \cup \{\phi_e\})$ violates weak exclusivity for any omitted edge constraint $\phi_e \in \Phi \setminus W$. As a result, $(D, T \cup W)$ is a maximal regular fragment of $\mathcal{D}$, which may be $\mathcal{D}$ itself (*e.g.*, the person model, Figs. 1c and 2).

The auxiliary function *isCardinality* tests whether its argument is a cardinality constraint. The functions *isTiered* and *isWeaklyExclusive* return true only when applied to a tiered or a weakly exclusive model, respectively. Both can be implemented directly from Defs. 7 and 9 with worst case running time of $O(|D|)$, where $|D|$ is the number of nodes and edges in the schema graph.[4] The fragment computation algorithm as a whole is therefore quadratic in $|D|$.

4.2 Solving Cardinality Constraints

Once we have identified a regular fragment $\mathcal{D}_r = (D, T \cup W)$ of our input model, the next step is to compute the number of members that each category $c \in C$ will have in the generated instance. These values, denoted by $card : C \to \mathbb{N}$, need to satisfy two sets of constraints: the cardinality constraints T, and the hierarchy constraints $card(c_i) \geq card(c_j)$ for each $c_i \prec c_j$ in D (Def. 8). Semantically, *card* is required only to satisfy T. We impose the hierarchy constraints for efficiency: by Thm. 4, the regularity of $\mathcal{D}_r$ ensures the satisfiability of both sets of constraints, while also enabling efficient encoding of the rollup relation.

[4]Recall from Def. 4 that Φ contains at most $2 * |D|$ constraints.

The cardinality and hierarchy constraints collectively form a system of linear inequalities over integer variables, as shown in Fig. 5. The variables represent category sizes: each v_i encodes the size of a category $c_i \in C$ in the generated instance. Inequalities 2 and 3 bound the sizes of categories according to the cardinality constraints in T. In particular, the functions l_T and h_T (Def. 7) map each c_i to its lower and upper bound specified by T, if any, or to the implicit lower bound of 1 (Def. 2) and upper bound of int_∞ (*e.g.*, $2^{31} - 1$) otherwise. The remaining inequalities (4) are induced by the hierarchy constraints. Figure 6 shows the corresponding inequalities for the person model.

$$\text{MINIMIZE:} \quad \sum_{c_i \in C} (n(c_i) - v_i)^2 \tag{1}$$

$$\text{SUBJECT TO:} \quad v_i \geq l_T(c_i) \quad \text{for all } c_i \in C \tag{2}$$

$$v_i \leq h_T(c_i) \quad \text{for all } c_i \in C \tag{3}$$

$$v_i \geq v_j \quad \text{for all } c_i \prec c_j \in D \tag{4}$$

Figure 5: Quadratic program for a model $(D, T \cup W)$, with $D = (C, \prec)$, and soft constraints $f = (n, s, t, p)$ over D

(2)	(3)	(4)
$v_{\mathsf{Person}} \geq 1$	$v_{\mathsf{Person}} \leq \mathsf{int}_\infty$	$v_{\mathsf{Person}} \geq v_{\mathsf{Gender}}$
$v_{\mathsf{Gender}} \geq 2$	$v_{\mathsf{Gender}} \leq 2$	$v_{\mathsf{Person}} \geq v_{\mathsf{City}}$
$v_{\mathsf{City}} \geq 1$	$v_{\mathsf{City}} \leq \mathsf{int}_\infty$	$v_{\mathsf{City}} \geq v_{\mathsf{State}}$
$v_{\mathsf{State}} \geq 1$	$v_{\mathsf{State}} \leq \mathsf{int}_\infty$	$v_{\mathsf{City}} \geq v_{\mathsf{Province}}$
$v_{\mathsf{Province}} \geq 1$	$v_{\mathsf{Province}} \leq \mathsf{int}_\infty$	$v_{\mathsf{City}} \geq v_{\mathsf{Country}}$
$v_{\mathsf{Country}} \geq 1$	$v_{\mathsf{Country}} \leq 196$	$v_{\mathsf{State}} \geq v_{\mathsf{Country}}$
		$v_{\mathsf{Province}} \geq v_{\mathsf{Country}}$

Figure 6: System of inequalities for the customer model

It is easy to see that the linear system Fig. 6 is satisfiable in polynomial time. For example, we could assign v_{Person} and v_{Gender} to 2, and all other variables to 1. More generally, we can set each v_i to $\max_{c_i \preceq c_j} l_T(c_j)$, which is the maximum lower bound over all categories reachable from c_i (including c_i itself) in the schema graph. Due to the tieredness of T (Def. 7), a solution of this form would be valid for every system 2-4 derived from a regular model $\mathcal{D}_r$. It is unlikely, however, that such a solution would have high scale fidelity (Def. 6) with respect to the provided soft constraints f. We therefore solve the system 2-4 by forming an *integer quadratic program* (IQP) that minimizes the solution with respect to the scale metric α_f (Fig. 5, line 1).

In general, solving IQPs is intractable [16]. But our problem has a special structure: we can show that the matrix corresponding to the system (2)-(4) is totally unimodular, and that the matrix corresponding to the quadratic term of the optimization function (1) is principally unimodular [12]. This allows us to relax the IQP to a *quadratic program* (QP) over real variables, and still obtain an integral solution [26]. Moreover, since the optimization matrix is also positive semi-definite, an optimal solution to the relaxed problem can be obtained in polynomial time [35]. We use CPLEX [20] to solve the relaxed problem and populate *card*.

4.3 Solving Edge Constraints

Given *card*, we next compute the *domain* and *range* sizes of the rollup functions $<_{ij}$ that will be generated for the schema edges $c_i \prec c_j$. The computed sizes are stored in

```
STRENGTHEN(D, W)
1  D_s, W_s ← D, {}
2  for φ_s ∈ SELECT(W) do
3    D_s, φ_s ← RESTRICT(D_s, φ_s)
4    W_s ← W_s ∪ {φ_s}
5  return (D_s, W_s)
```

```
RESTRICT(D_s = (C_s, ≺_s), φ_s)
1  E_s ← constrainedEdges(φ_s)
2  ⟨c_i, c_j⟩ ← chooseAny(E_s)
3  if isExclusive(φ_s)
4    then E_omit ← E_s \ {⟨c_i, c_j⟩}
5    else E_omit ← {}
6  φ'_s ← constrain(c_i, op(φ_s), c_j)
7  D'_s ← (C_s, ≺_s \ E_omit)
8  return (D'_s, φ'_s)
```

Figure 7: Strengthening algorithm for (D, W)

c	$card(c)$
Person	32
Gender	2
City	16
State	8
Province	8
Country	4

e	$dom(e)$	$ran(e)$
e0: Person ≺ Gender	24	2
e1: Person ≺ City	24	16
e2: City ≺ State	8	4
e3: City ≺ Province	0	0
e4: City ≺ Country	8	2
e5: State ≺ Country	8	4
e6: Province ≺ Country	0	0

Figure 8: Sample solutions for the person model

$dom : C \times C \to \mathbb{N}$ and $ran : C \times C \to \mathbb{N}$, respectively, and they satisfy a system of integer (in)equalities arising from two sources: the (exclusive) rollup and drilldown constraints in W, and the strictness constraints on the rollup relation as a whole (Def. 2). As before, the (in)equalities solved in this step are stronger than those implied by W and strictness alone, in order to enable efficient encoding of rollup closures in the final step. To simplify the presentation, we show only how to strengthen a constraint in W with respect to the schema D, and how the strengthened problem translates into integer inequalities. The algorithms for selecting which constraints to strengthen, and for solving the inequalities, are described only briefly, in terms of their inputs and outputs.

To determine *dom* and *ran*, we begin by transforming the schema graph D and a subset of the constraints W, with the help of the procedure STRENGTHEN (Fig. 7). Its key step is the application of the function RESTRICT, which takes two inputs: a schema D_s, and a constraint ϕ_s over two or more edges in D_s. Given these inputs, RESTRICT outputs a schema $D'_s \subseteq D_s$ and a constraint $\phi'_s \subset \phi_s$, which admit fewer instances than (D_s, ϕ_s). If ϕ_s is a non-exclusive constraint (lines 5-8), such as $c_i \to c_j \vee \cdots \vee c_k$, the output is D_s itself, together with a single-edge constraint $\phi'_s \subset \phi_s$, such as $c_i \to c_j$. For exclusive constraints (line 4), the output is ϕ'_s and a schema $D'_s \subset D_s$, which omits the same edges from D_s as ϕ'_s does from ϕ_s. The auxiliary functions are self-explanatory, and all run in constant time.

It follows easily from Defs. 3-4 that the RESTRICT transformation simply reduces the set of satisfying instances. We can also prove (from Def. 9 and Thm. 4) that applying STRENGTHEN to D and all of W will not eliminate all instances of $\mathcal{D}_r$. In most cases, however, TESTBLOX selects only a small subset of W to strengthen. It would not, for example, strengthen any constraints in the person model. In general, we strengthen all multi-edge constraints except the following: (1) an exclusive rollup constraint over edges that form a cut-set of D (such as C5); and, (2) any constraint over cut-edges [10] found in D_s after the exclusively constrained cut-sets are removed (*e.g.*, C4 and C7).

Using a strengthened schema and constraints (D_s, W_s), we obtain the system of (in)equalities in Fig. 9. The integer

$$d_{ij} \geq 0 \qquad r_{ij} \geq 0 \qquad \text{for all } c_i \prec c_j \in D_s \quad (5)$$
$$d_{ij} \leq card(c_i) \qquad r_{ij} \leq card(c_j) \qquad \text{for all } c_i \prec c_j \in D_s \quad (6)$$
$$d_{ij} \geq r_{ij} \qquad \mathrm{sgn}(d_{ij}) = \mathrm{sgn}(r_{ij}) \qquad \text{for all } c_i \prec c_j \in D_s \quad (7)$$
$$\sum d_{ij} \geq card(c_i) \qquad \text{for all } c_i \to \bigvee c_j \in W_s \quad (8)$$
$$\sum d_{ij} = card(c_i) \qquad \text{for all } c_i \to \bigodot c_j \in W_s \quad (9)$$
$$\sum r_{ij} \geq card(c_j) \qquad \text{for all } \bigvee c_i \leftarrow c_j \in W_s \quad (10)$$
$$\sum r_{ij} = card(c_j) \qquad \text{for all } \bigodot c_i \leftarrow c_j \in W_s \quad (11)$$

Figure 9: Domain and range inequalities for (D_s, W_s)

variables d_{ij} and r_{ij} represent the size of the domain and range of $<_{ij}$, respectively. These variables are non-negative and bounded above by the corresponding category sizes (5-6). Due to the strictness requirement, each $<_{ij}$ must be a function, and, consequently, its domain must be at least as large as its range (7). Moreover, the range must contain some element whenever the domain does (*i.e.*, d_{ij} and r_{ij} must both be 0 or both be positive), since we cannot generate a function from a non-empty domain to an empty range. The rest of the system (8-11) reflects the meaning of (exclusive) rollup and drilldown constraints. For example, an exclusive rollup constraint $c_i \to c_j \odot c_k$ constrains the domains of the functions $<_{ij}$ and $<_{ik}$ to partition the members of c_i. As a result, the sizes of the domains (9) are related by the equality $d_{ij} + d_{ik} = card(c_i)$.

The system in Fig. 9 is always satisfiable in polynomial time, due to the regularity of $(D_s, T \cup W_s)$. TESTBLOX solves it with a simple greedy algorithm, using the shape fidelity metric β_{W_s} to guide the assignment of values to variables. We are unaware of results that would suggest the structure of this problem admits an optimal solution in polynomial time. After computing a greedy solution to the problem, TESTBLOX invokes CPLEX to search for a better one, within a user-specified time bound. The functions *dom* and *ran* are populated using the best available solution.

4.4 Constructing Instance Closures

Given the solutions from the previous steps, as well as the strengthened model $(D_s, T \cup W_s)$, we use the procedure GENERATE, shown in Fig. 10, to construct a rollup closure for each schema edge $e \in D_s$. For simplicity, the version of GENERATE presented in this section omits the handling of soft distribution constraints specified by p (Def. 5). TESTBLOX uses a more sophisticated version that takes p into account, guaranteeing conformance to distributions provided for edges that are exclusively constrained by W_s, and for all cut-edges in D_s. The implemented algorithm would faithfully handle distributions on all edges of the person model, for example.

Details aside, both the shown and implemented algorithms follow the same basic approach. They generate rollup mappings from c_i to c_j with the aid of three lazy streams: X, which produces a permutation of the domain set (line 1); Y, which produces a permutation of the range set (line 2); and Z, which produces the number of domain members that map to each range member (line 3). The main loop (lines 6-9) prints z values drawn from X for each value drawn from Y, where z itself is obtained from the stream Z. The function PERMUTATION-STREAM implements Gray's permutation generator [13], which (deterministically) maps a pos-

```
GENERATE(e = c_i ≺ c_j, D_s, W_s, card, dom, ran)
1  X ← PERMUTATION-STREAM(card(c_i))
2  Y ← PERMUTATION-STREAM(card(c_j))
3  Z ← PARTITION-STREAM(e, D_s, card, dom, ran)
4  x_o, y_o ← OFFSETS(e, W_s, card, dom, ran)
5  ADVANCE-STREAMS(X, Y, Z, x_o, y_o, dom(e), ran(e))
6  for (r ← ran(e); r > 0; r ← r − 1) do
7    y ← next(Y)
8    for (z ← next(Z); z > 0; z ← z − 1) do
9      print next(X), y

PARTITION-STREAM(e = c_i ≺ c_j, D_s, card, dom, ran)
1  (C_bcc, E_bcc) ← BICONNECTED-COMPONENT(D_s, e)
2  if |E_bcc| = 1 then
3    return SUM-STREAM(ran(e), dom(e))
4  a ← SORT([card(c) for c ∈ C_bcc])
5  m ← BINARY-SEARCH(a, card(c_j))
6  n ← BINARY-SEARCH(a, card(c_i))
7  Z ← SUM-STREAM(a[n], a[n])
8  for (k ← n; k > m; k ← k − 1) do
9    Z ← COMPOSE(SUM-STREAM(a[k − 1], a[k]), Z)
10 return Z
```

Figure 10: Generating roll-ups for $c_i \prec c_j$ in $(D_s, T \cup W_s)$

itive number n to a permutation of the set $\{0, \ldots, n-1\}$. Note that category members are referred to by their indices, $0, \ldots, card(c) - 1$.

We use the OFFSETS function (line 4) to indirectly coordinate instantiation of edges that are constrained by W_s. To illustrate, consider the sample solution for the person model in Fig. 8. According to C4, the functions $<_{e0}$ and $<_{e1}$ must collectively map all 32 members of Person. Line 1 yields the same permutation X of the Person set for both e0 and e1. The OFFSETS function returns $x_o = 0$ as the domain offset of e0, and $x_o = card(\text{Person}) - dom(\text{e1}) = 32 - 24 = 8$ as the domain offset of e1. Using these offsets (line 5), e0 maps the first $dom(\text{e0})$ members of X, which we denote by $X[0:24]$, while e1 skips the first 8 members, mapping $X[8:32]$ instead. Consequently, all members of Person, $X[0:32]$, are mapped by at least one of the functions. OFFSETS can be straightforwardly implemented with $O(|D_s|)$ worst case time.

The PARTITION-STREAM procedure fulfills the global strictness requirement on the rollup relation (Def. 2). It works by maintaining strictness within each biconnected component of the schema graph, obtained by viewing the schema as undirected. Recall that a biconnected component of an undirected graph is a maximal subgraph with no articulation vertices. An articulation vertex is a node that cannot be removed without disconnecting the rest of the graph. For example, the undirected version of the person schema (Fig. 1c) has two articulation vertices, Person and City, and three biconnected components: {e0}, {e1}, and {e2, e3, e4, e5, e6}. It is not hard to see that any rollup relation $\leq$ for the person graph is strict if and only if it is strict within each component.

PARTITION-STREAM computes Z streams for all edges within a biconnected component (line 1) via a deterministic sequence (lines 4-6) of stream compositions (lines 7-9), based on the cardinalities of the categories within the component. The function SUM-STREAM(k, n) returns a stream of k positive values that add up to n, denoted by Z_n^k. COMPOSE takes two streams, Z_n^k and Z_s^n, and composes them via addition to produce a stream Z_s^k with k values that sum up to s.

To see how this construction process enforces strictness, consider the component {e2, e3, e4, e5, e6} in Fig. 1c. Line 4 constructs the array $a = [4, 8, 16]$ for all five edges. The array a governs the construction of the resulting Z streams so that, for example,

$$\text{PARTITION-STREAM}(\text{e2}, \ldots) = Z_{16}^8,$$
$$\text{PARTITION-STREAM}(\text{e5}, \ldots) = Z_8^4, \text{and}$$
$$\text{PARTITION-STREAM}(\text{e4}, \ldots) = \text{COMPOSE}(Z_8^4, Z_{16}^8).$$

The above means that both the rollup function $<_{e4}$, and the composition of $<_{e2}$ and $<_{e5}$, are contained in the same total, surjective function from City to Country. Consequently, every city must map to the same country via all paths in a customer instance that go directly through $<_{e4}$, or indirectly, through $<_{e2}$ and $<_{e5}$. A similar intuitive argument can be made for all pairs of paths in a biconnected graph.

5. EVALUATION

To evaluate TESTBLOX, we conducted a set of experiments on three multidimensional data models developed at a retail consulting company. All three models are used in real decision-support applications. For confidentiality reasons, we refer to the applications as A1, A2, and A3. Two of these, A1 and A3, are deployed at major retail chains; A2 is a demo application. Figure 11 shows the anonymized dimension schemas for each application's analysis space. Every graph corresponds to one dimension model, for a total of 10 models.

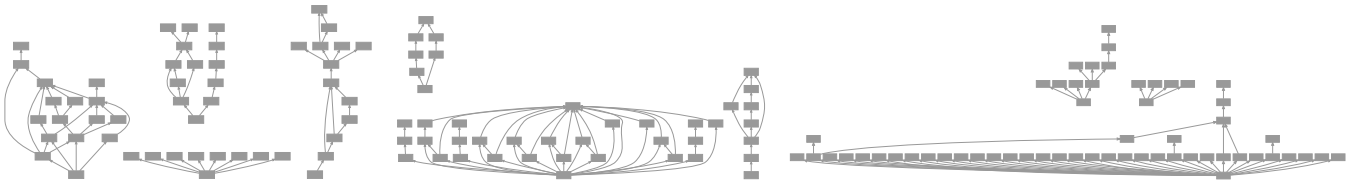
The experiments were designed to evaluate the scalability of TESTBLOX, and the quality of the generated data. The experimental setup is described in Sec. 5.1, and the results in Sec. 5.2. All experiments were conducted on an ordinary laptop with a two core 2.8 GHz processor and 8 GB of RAM. TESTBLOX is implemented in Java, using CPLEX [20] for optimization, and the Apache Commons Mathematics library [7] for distribution sampling.

5.1 Experimental Setup

Our applications' data models were developed using a graphical modeling tool, with support for specifying schema graphs but no notion of dimension constraints. As a result, any assumptions about the expected shape of the dimension data are hard-coded in the application logic, and difficult to recover. We therefore evaluate TESTBLOX against each application's schema using a randomly generated dimension model (Table 1a), and four sets of randomly generated soft constraints. The soft constraint sets describe instances of varying sizes, ranging from .25 to 1 GB of data (Table 1b).

The generated dimension models are intended to simulate a worst-case usage scenario, in which every element of its schema graph participates in as many constraints as allowed by Def. 10. We place random lower and upper bounds on the cardinality of each category, and ensure that each edge is involved in at least one (exclusive) rollup or drilldown constraint. The cardinality constraints are tiered, and the exclusivity constraints are placed only on edges leaving the sources, or entering the sinks, of biconnected components. The placement of exclusivity constraints approximates the use of generalization and specialization patterns [30, 31, 32].

The generated soft constraints are similarly intended to simulate a realistic but challenging usage scenario. We place



(A1) (A2) (A3)

Figure 11: Dimension schemas for three retail applications

	A1	A2	A3
categories	51	42	57
edges	62	63	56
cardinalities	102	84	114
rollups	46	42	23
exc. rollups	4	3	1
drilldowns	44	29	53
exc. drilldowns	2	6	0

(a) Number of categories, edges and dimension constraints

	A1	A2	A3
Soft 1	.253	.252	.249
Soft 2	.501	.496	.498
Soft 3	.750	.751	.748
Soft 4	1.03	1.02	1.06

(b) Instance size, in GB, for each soft constraint set

Table 1: Summary of experimental parameters

a random scale constraint on each category, ensuring only that the requested size is within the generated cardinality bounds. Every edge is also constrained using a shape constraint with random source and target participations, as well as a random distribution function. The participations are chosen from between .85 and 1, while the distribution is either uniform, normal, exponential, or zipfian. We attempt to choose the defining parameters of a distribution so that they are compatible with other constraints, since TestBlox ignores distributions that are unlikely to produce values within solution parameters. For example, it would not try to create rollup mappings for members of a category c using a uniform distribution over the interval $(card(c), \infty)$.

5.2 Results and Discussion

Efficiency. To evaluate the scalability and parallelism of data generation, we executed TestBlox four times on the models and soft constraint sets for each application, increasing the number of instance generation threads from 1 to 4. No special load balancing strategy was used; the generator relies on the balancing provided by the standard Java concurrency library. The timing results are grouped by application and shown in Fig. 12. As evident from the graphs, the time taken to generate data increases roughly linearly with instance size and decreases linearly with the number of threads. Solving time is negligible in all cases, with the entire process dominated by instance generation. Generating the largest instances, with 1 GB of data each, takes 40 seconds or less for all benchmarks.

Data Quality. In addition to execution time, we also tracked two sets of indicators of data quality for each generated instance:

(I1) the ratio between the desired and actual size of each category; and,

(I2) the ratios between the desired and actual source participation, target participation, and distribution mean for each rollup function.

The results are summarized in Figure 13 for the largest instance of every benchmark. Smaller instances exhibit similar patterns.

Since the dimension solving algorithm finds an optimal solution to cardinality constraints, the I1 values for all three applications remain very close to 1. Only two categories overall had ratios that were within 2 percent of 1 rather than 1 exactly. The I2 values are less consistent for two reasons. First, the solution to edge constraints is not guaranteed to be optimal, and second, we can only guarantee conformance to specified distributions for certain kinds of schema edges, as described in Sec. 4.4. For the remaining edges, distribution constraints are sacrificed to enforce strictness. Consequently, A3, which has the simplest structure, also has indicators for 95% of its edges within 20 percent of 1. But even the indicators for the more complex structures remain largely within 20 percent of the ideal—85% of all A1 edges have ratios between .8 and 1.2, while the same is true for 73% percent of A2 edges.

State-of-the-Art. These results are a significant improvement on the general constraint-solving approach, which is currently the only way to generate complex multidimensional structures. As discussed in Sec. 6, specialized database generators [2, 4, 5, 17, 18, 28, 41] can be used to instantiate tree-shaped schemas, but not general schema DAGs, such as those in Fig. 11. We could not find publicly available prototypes of these tools for comparison on tree-shaped schemas.

For a rough comparison to general constraint solving, we manually translated the person model to Alloy [21] and instantiated it using the Alloy4 solver, which is based on SAT. The largest instance we could obtain in 60 seconds consists of 0.0003 GB of data, corresponding to a dimension graph with about 1650 edges. This instance satisfies the person model, but its shape is otherwise arbitrary. Like most constraint solvers, Alloy provides no support for statistical constraints. TestBlox, in contrast, took 14 seconds (using 4 threads) to produce 1 GB of data for A1, corresponding to three dimension graphs with a total of 14,084,547 edges. The generated dimensions additionally exhibit desired statistical properties.

6. RELATED WORK

Database Generation. A variety of techniques have been developed for generating synthetic databases with specific characteristics. Many of them [5, 13, 17, 18, 41] focus on scalable, parallel generation of large databases, subject to user-specified column distributions and intra-table correlations. A classic use case for such tools is the generation of realistic data for TPC benchmarks [44]. Other approaches, such as [2, 3, 4, 9, 28, 36], are designed to produce smaller volumes of data, subject to richer constraints. Most of them address the following problem—given a query, a schema,

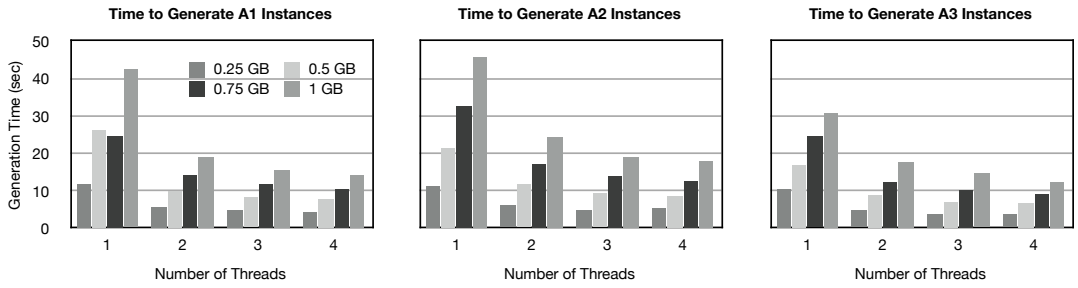


Figure 12: Time to generate .25 to 1 GB of data for each benchmark, using 1 to 4 threads

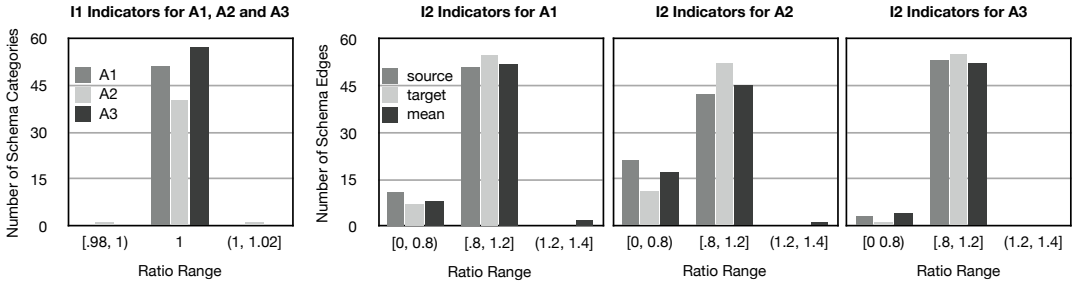


Figure 13: Quality of 1 GB instances, given as the number of categories and edges with I1 and I2 values in the specified ranges

and a specification of the desired output, produce an input database that conforms to the schema, and that will elicit an output with the specified characteristics. The specification may take the form of a table [3], a set of relation cardinalities [2, 4, 28], or a set of coverage rules [9].

While many of the existing tools are quite general, with some being based on general-purpose constraint solvers (*e.g.*, [4, 9, 28]), none are designed to work on multidimensional models with rich constraints. Most could be used to generate instances of chain and tree-like dimension schemas, but not of general schema DAGs. For practical use, the latter requires enforcement of global strictness constraints, as well as constraints implied by common modeling patterns. TestBlox is, to our knowledge, the first tool with this capability.

Test Data Generation. In addition to database generators, there are also many tools for producing other kinds of test data from models (*e.g.*, [33, 42]) and from programs (*e.g.*, [24, 37]). Of these, the ORM generation tool by Smaragdakis *et al.* [42] and McGill *et al.* [33] is most closely related to TestBlox. Given a specification in the Object-Role Modeling (ORM) language, which is NP-hard to satisfy, the tool identifies a subset of the input that is within a commonly used, polynomial fragment of the language. It then checks whether the identified subset of the model is satisfiable, and, if it is, generates an instance of that subset. The ORM fragment recognized by the tool cannot express global strictness constraints over relations, making it unsuitable for specifying (and instantiating) multidimensional models. The tool itself differs from TestBlox in that it is designed to produce any satisfying instance; it does not have a notion of a preferred instance, described by soft constraints.

Multidimensional Modeling. The TestBlox modeling language was inspired by much of the prior work on multidimensional modeling [19, 22, 25, 30, 31, 40]. Schemas and instances, as defined in Sec. 2, capture all dimension shapes expressible in existing multidimensional models, with the exception of instances with non-strict rollup relations [30, 31, 40]. We omit this shape since it is not supported in practice [30]. But it could be easily integrated into our framework, by removing the strictness requirement from Def. 2 and treating it as an explicit integrity constraint on (a subset of) schema edges.

Like the model by Hurtado *et al.* [19], TestBlox supports the notion of a dimension constraint. The focus of the two languages is different, however. The TestBlox language is designed for easy encoding of common modeling patterns, and for efficient generation of test data. The language in [19] is designed for reasoning about a generalized notion of summarizability. It supports a much richer superset of our rollup constraints, but it cannot express either cardinality or drilldown constraints. It would be interesting to extend TestBlox to handle a larger subset of the rollup constraints proposed in [19].

7. CONCLUSION

TestBlox is a new framework for modeling and scalable generation of complex multidimensional structures. The first of its kind, the framework provides a small but expressive modeling language, and a solver for generating data from specified models. While the problem of generating data from TestBlox models is NP-hard in general, we have identified a practical fragment of the language that can be solved in polynomial time. Our solution approach is modular, enabling parallel data generation in the presence of global constraints. We have also shown how to solve constraints in this fragment so that the resulting instance exhibits preferred scale and shape. Plans for future work include enriching the language, extending its polynomial fragment, and investigating ways to incorporate reasoning about distributions into the solver.

Acknowledgement. We thank Rastislav Bodik and the anonymous reviewers of this paper for their valuable comments and feedback.

8. REFERENCES

[1] K. Aouiche and J. Darmont. Data mining-based materialized view and index selection in data warehouses. *J. Intell. Inf. Syst.*, 33(1):65–93, Aug. 2009.

[2] A. Arasu, R. Kaushik, and J. Li. Data generation using declarative constraints. In *SIGMOD*, 2011.

[3] C. Binnig, D. Kossmann, and E. Lo. Reverse query processing. In *ICDE*, 2007.

[4] C. Binnig, D. Kossmann, E. Lo, and M. T. Özsu. QAGen: generating query-aware test databases. In *SIGMOD*, 2007.

[5] N. Bruno and S. Chaudhuri. Flexible database generators. In *VLDB*, 2005.

[6] Y. Cao, C. Chen, F. Guo, D. Jiang, Y. Lin, B. C. Ooi, H. T. Vo, S. Wu, and Q. Xu. ES^2: A cloud data storage system for supporting both OLTP and OLAP. In *ICDE*, 2011.

[7] Commons Math: The Apache Commons Mathematics Library. commons.apache.org/math, 2011.

[8] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J.-M. Vincent, and F. Wagner. Random graph generation for scheduling simulations. In *SIMUTools*, 2010.

[9] C. de la Riva, M. J. Suárez-Cabal, and J. Tuya. Constraint-based test database generation for SQL queries. In *AST*, 2010.

[10] R. Diestel. *Graph Theory*. Springer, 2005.

[11] L. D'Orazio and S. Bimonte. Multidimensional arrays for warehousing data on clouds. In *Globe*, 2010.

[12] J. F. Geelen. *Matchings, Matroids and Unimodular Matrices*. PhD thesis, University of Waterloo, 1995.

[13] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P. J. Weinberger. Quickly generating billion-record synthetic databases. *SIGMOD Rec.*, 23, 1994.

[14] J. Han, Y. Chen, G. Dong, J. Pei, B. W. Wah, J. Wang, and Y. D. Cai. Stream cube: An architecture for multi-dimensional analysis of data streams. *Distrib. Parallel Databases*, 18(2):173–197, Sept. 2005.

[15] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD*, 1996.

[16] R. Hemmecke, M. Köppe, J. Lee, and R. Weismantel. Nonlinear integer programming. In *50 Years of Integer Programming 1958–2008: The Early Years and State-of-the-Art Surveys*. Springer-Verlag, 2009.

[17] J. E. Hoag and C. W. Thompson. A parallel general-purpose synthetic data generator. *SIGMOD Rec.*, 36(1), 2007.

[18] K. Houkjaer, K. Torp, and R. Wind. Simple and realistic data generation. In *VLDB*, 2006.

[19] C. A. Hurtado, C. Gutierrez, and A. O. Mendelzon. Capturing summarizability with integrity constraints in OLAP. *ACM Trans. Database Syst.*, 30(3), 2005.

[20] IBM ILOG CPLEX Optimizer. www.ilog.com, 2011.

[21] D. Jackson. *Software Abstractions: Logic, Language, and Analysis*. MIT Press, 2006.

[22] H. V. Jagadish, L. V. S. Lakshmanan, and D. Srivastava. What can hierarchies do for data warehouses? In *VLDB*, 1999.

[23] H. Karloff and M. Mihail. On the complexity of the view-selection problem. In *PODS*, 1999.

[24] S. Khurshid and D. Marinov. TestEra: Specification -based testing of Java programs using SAT. *ASE'00*, 11(4), 2004.

[25] J. Lechtenbörger and G. Vossen. Multidimensional normal forms for data warehouse design. *Inf. Syst.*, 28(5), 2003.

[26] Z. Li, S. Zhang, Y. Wang, X.-S. Zhang, and L. Chen. Alignment of molecular networks by integer quadratic programming. *Bioinformatics*, 23(13), 2007.

[27] C. X. Lin, B. Ding, J. Han, F. Zhu, and B. Zhao. Text cube: Computing IR measures for multidimensional text database analysis. In *ICDM*, 2008.

[28] E. Lo, N. Cheng, and W.-K. Hon. Generating databases for query workloads. *Proc. VLDB Endow.*, 3(1-2), 2010.

[29] E. Lo, B. Kao, W.-S. Ho, S. D. Lee, C. K. Chui, and D. W. Cheung. OLAP on sequence data. In *SIGMOD*, 2008.

[30] E. Malinowski and E. Zimányi. Hierarchies in a multidimensional model: from conceptual modeling to logical representation. *Data Knowl. Eng.*, 59(2), 2006.

[31] S. Mansmann and M. H. Scholl. Empowering the OLAP technology to support complex dimension hierarchies. *IJDWM*, 3(4), 2007.

[32] J.-N. Mazón, J. Lechtenbörger, and J. Trujillo. A survey on summarizability issues in multidimensional modeling. *Data Knowl. Eng.*, 68(12), 2009.

[33] M. J. McGill, L. K. Dillon, and R. Stirewalt. Scalable analysis of conceptual data models. In *ISSTA*, 2011.

[34] K. Morfonios, S. Konakas, Y. Ioannidis, and N. Kotsis. ROLAP implementations of the data cube. *ACM Comput. Surv.*, 39(4), 2007.

[35] K. G. Murty. *Linear Complementarity, Linear and Nonlinear Programming*. Heldermann Verlag, 1988.

[36] A. Neufeld, G. Moerkotte, and P. C. Lockemann. Generating consistent test data: restricting the search space by a generator formula. *VLDB J*, 1993.

[37] C. Olston, S. Chopra, and U. Srivastava. Generating example data for dataflow programs. In *SIGMOD*, 2009.

[38] T. B. Pedersen, J. Gu, A. Shoshani, and C. S. Jensen. Object-extended OLAP querying. *Data Knowl. Eng.*, 68(5), 2009.

[39] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. Extending practical pre-aggregation in on-line analytical processing. In *VLDB*, 1999.

[40] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. A foundation for capturing and querying complex multidimensional data. *Inf. Syst.*, 26(5), 2001.

[41] T. Rabl, M. Frank, H. M. Sergieh, and H. Kosch. A data generator for cloud-scale benchmarking. In *TPCTC*, 2010.

[42] Y. Smaragdakis, C. Csallner, and R. Subramanian. Scalable satisfiability checking and test data generation from modeling diagrams. *ASE*, 16(1), 2009.

[43] E. Torlak. Scalable test data generation from multidimensional models. Technical Report UCB/EECS-2012-177, EECS Department, University of California, Berkeley, July 2012.

[44] TPC benchmarks. www.tpc.org.

[45] P. Zhao, X. Li, D. Xin, and J. Han. Graph cube: on warehousing and OLAP multidimensional networks. In *SIGMOD*, 2011.