Towards Rigorous Evaluation of Data Integration Systems

–

It's All About the Tools

Boris Glavic

P. Arocena, R. Ciucanu, G. Mecca, R. J. Miller, P. Papotti, D. Santoro

IIT
University of Toronto
Université Blaise Pascal
Università della Basilicata
Arizona State University
Outline

1) Empirical Evaluation of Integration Systems
2) iBench
3) BART
4) Success Stories
5) Demo
6) Conclusions and Future Work
Overview

• Challenges of evaluating integration systems
  – Diversity of tasks
    • Various types of metadata used by integration tasks
  – Quality is as important as performance
    • Often requires “gold standard” solution
• Goal: make empirical evaluations …
  • … more robust, repeatable, shareable, and broad
  • … less painful and time-consuming
• This talk:
  – iBench – a flexible metadata generator
  – BART – generating data quality errors
Overview

• Challenges of evaluating integration systems
  – Diversity of tasks
  • Various types of metadata used by integration tasks

Patterson [CACM 2012]
“When a field has good benchmarks, we settle debates and the field makes rapid progress.”

– iBench – a flexible metadata generator
– BART – generating data quality errors
Many integration tasks work with metadata:

- **Data Exchange**
  - *Input*: Schemas, Constraints, (Source Instance), Mappings
  - *Output*: Executable Transformations, (Target Instance)

- **Schema Mapping Generation**
  - *Input*: Schemas, Constraints, Instance Data, Correspondences
  - *Output*: Mappings, Transformations

- **Schema Matching**
  - *Input*: Schemas, (Instance Data), (Constraints)
  - *Output*: Correspondences

- **Constraint-based Data Cleaning**
  - *Input*: Instance Data, Constraints
  - *Output*: Instance Data

- **Constraint Discovery**
  - *Input*: Schemas, Instance Data
  - *Output*: Constraints

- **Virtual Data Integration**
  - *Input*: Schemas, Instance Data, Mappings, Queries
  - *Output*: Rewritten Queries, Certain Query Results

- ... and many others (e.g., Mapping Operators, Schema Evolution, ...)

QDB 2016 - Towards Rigorous Evaluation of Data Integration Systems
Integration Tasks

Many integration tasks work with metadata:

- **Data Exchange**
  - *Input*: Schemas, Constraints, (Source Instance), Mappings
  - *Output*: Executable Transformations, (Target Instance)

- **Schema Mapping Generation**
  - *Input*: Schemas, Constraints, Instance Data, Correspondences
  - *Output*: Mappings, Transformations

- **Schema Matching**
  - *Input*: Schemas, (Instance Data), (Constraints)
  - *Output*: Correspondences

- **Constraint-based Data Cleaning**
  - *Input*: Instance Data, Constraints
  - *Output*: Instance Data

- **Constraint Discovery**
  - *Input*: Schemas, Instance Data
  - *Output*: Constraints

- **Virtual Data Integration**
  - *Input*: Schemas, Instance Data, Mappings, Queries
  - *Output*: Rewritten Queries, Certain Query Results

- **... and many others** (e.g., Mapping Operators, Schema Evolution, ...)

**Inputs/Outputs**

**Metadata**: Schemas, Constraints, Correspondences, Mappings

**Data**: Source Instance, Target Instance
State-of-the-art

- How are integration systems typically evaluated?
- **Small real-world integration scenarios**
  - **Advantages:**
    - Realistic ;-)  
  - **Disadvantages:**
    - Not possible to scale (schema-size, data-size, …)  
    - Not possible to vary parameters (e.g., mapping complexity)  
- **Ad-hoc synthetic scenarios**
  - **Advantages:**
    - Can influence scale and characteristics  
  - **Disadvantages:**
    - Often not very realistic metadata  
    - Diversity requires huge effort
Requirements

• **We need tools to generate inputs/outputs**
  – **Scalability**
    • Generate large integration scenarios efficiently
    • Requires low user effort
  – **Control over metadata and data characteristics**
    • Size
    • Structure
    • ...
  – **Generate inputs as well as gold standard outputs**
  – **Promote reproducibility**
    • Enable other researchers to regenerate metadata to repeat an experiment
    • Support researchers in understanding the generated metadata/data
    • Enable researchers to reuse generated integration scenarios
Related Work

• **STBenchmark** [Alexe et al. PVLDB ‘08]
  – Pioneered the **primitive** approach:
    • Generate metadata by combining typical micro scenarios

• **Data generators**
  – PDGF, Myriad
  – Data generators are not enough
Outline

1) Empirical Evaluation of Integration Systems
2) iBench
3) BART
4) Success Stories
5) Demo
6) Conclusions and Future Work
iBench Overview

• **iBench** is a metadata and data generator

• **Generates synthetic integration scenarios**
  
  – Metadata
  
  • Schemas
  
  • Constraints
  
  • Mappings
  
  • Correspondences

  – Data

• “Realistic” metadata
Integration Scenarios

• Integration Scenario

  \[ M = (S, T, \Sigma_S, \Sigma_T, \Sigma, I, J, \square) \]
Integration Scenarios

- **Integration Scenario**
  - $M = (S,T, \Sigma_S, \Sigma_T, \Sigma, I, J, \begin{array}{c} \square \, \square \end{array})$
  - Source schema $S$ with instance $I$
  - Target schema $T$ with instance $J$
  - Source constraints $\Sigma_S$ and target constraints $\Sigma_T$
    - Instance $I$ fulfills $\Sigma_S$ and instance $J$ fulfills $\Sigma_T$
  - Schema mapping $\Sigma$
    - Instances $(I,J)$ fulfill $\Sigma$
  - Transformations $\begin{array}{c} \square \, \square \end{array}$
iBench Input/Output

• **Inputs - Configuration**
  – **Scenario parameters** $\Pi$ (min/max constraints)
    • Number of source relations
    • Number of attributes of target relations
    • ...
  – **Primitive parameters**
    • Template micro-scenarios that are instantiated to create part of the output

• **Output**
  – A integration scenario $M$ that fulfills the constraints of specified in the configuration
    • XML file with metadata
    • CSV files for data
Example - MD Task

• Input

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Source</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number Relations</td>
<td>2-4</td>
<td>1-3</td>
</tr>
<tr>
<td>Number Attributes</td>
<td>2-10</td>
<td>2-10</td>
</tr>
<tr>
<td>Number of Join Attr</td>
<td>1-2</td>
<td>1-2</td>
</tr>
<tr>
<td>Number of Existentials</td>
<td></td>
<td>0-3</td>
</tr>
</tbody>
</table>

• Example solution (mappings)
  - \( S_1(A, B, C), S_2(C, D, E) \rightarrow T(A, E) \)
  - \( S_3(A, B, C, D), S_4(E, A, B) \rightarrow \exists X, Y, Z \ T_1(A, X, X), \quad T_2(A, Y, C), T_3(C, B, Y, Z) \)
Example - MD Task

• **Input**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Source</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number Relations</td>
<td>2-4</td>
<td>1-3</td>
</tr>
<tr>
<td>Number Attributes</td>
<td>2-10</td>
<td>2-10</td>
</tr>
<tr>
<td>Number of Join Attr</td>
<td>1-2</td>
<td>1-2</td>
</tr>
<tr>
<td>Number of Existentials</td>
<td>0-3</td>
<td></td>
</tr>
</tbody>
</table>

• **Example solution (mappings)**

- $S_1(A,B,C), S_2(C,D,E) \rightarrow T(A,E)$
- $S_3(A,B,C,D), S_4(E,A,B) \rightarrow \exists X, Y, Z \ T_1(A,X,X), T_2(A,Y,C), T_3(C,B,Y,Z)$

• **Limited usefulness in practice**
  – Can we generate “realistic” scenarios?
Mapping Primitives

- **Mapping Primitives**
  - Template micro-scenarios that encode a typical schema mapping/evolution operations
    - Vertical partitioning a source relation
  - Used as building blocks for generating scenarios

- **Comprehensive Set of Primitives**
  - Schema Evolution Primitives
    - Mapping Adaptation [Yu, Popa VLDB05]
    - Mapping Composition [Bernstein et al. VLDBJ08]
  - Schema Mapping Primitives
    - *STBenchmark* [Alexe, Tan, Velegrakis PVLDB08]
      - First to propose parameterized primitives
Scenario Primitives

Example Mapping Primitives

**Vertical Partition**
- Works
  - empld
  - ename
  - dept
  - manager

- Emp
  - empld
  - ename
  - WID

- Dept
  - dept
  - manager
  - WID

**Horizontal Partition**
- Dept
  - dname
  - addr
  - f1
  - Dept1
  - dname
  - addr
  - f2
  - Dept2
  - dname
  - addr

**Surrogate Key Invention**
- City
  - name
  - mayor

- City
  - name
  - mayor
  - ID

- f1/f2 predicates on addr

- Parameterize primitives
  - Number of relations for partitioning
  - Number of attributes for invention
  - ...

QDB 2016 - Towards Rigorous Evaluation of Data Integration Systems
Integration Scenario Generation

• **Approach**
  – Start with empty integration scenario
  – Repeatedly add instances of primitives according to specs
  – If necessary add additional random mappings and schema elements
Primitive Generation

• Example Configuration
  – I want 1 copy and 1 vertical partitioning
Primitive Generation

• Example Configuration
  – I want 1 copy and 1 vertical partitioning

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cust</td>
<td>Customer</td>
</tr>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>Addr</td>
<td>Addr</td>
</tr>
</tbody>
</table>
Primitive Generation

• Example Configuration
  – I want 1 copy and 1 vertical partitioning

Source
  Cust
    Name
    Addr

Emp
    Name
    Company

Target
  Customer
    Name
    Addr
    Loyalty

  Person
    Id
    Name

  WorksAt
    EmpRec
    Firm
    Id

QDB 2016 - Towards Rigorous Evaluation of Data Integration Systems
Sharing Schema Elements

• Sharing across primitives
  – Primitives cover many patterns that occur in the real world
  – however in the real world these primitives do not occur in isolation
• Enable primitives to share parts of the schema
  – Scenario parameters: source reuse, target reuse
  – Probabilistically determine whether to reuse previously generated relations
Sharing Schema Elements

• Example

Source
Cust
Name
Addr
Emp
Name
Company
Executive
Name
Position

Target
Customer
Name
Addr
Loyalty
Person
Id
Name
WorksAt
EmpRec
Firm
Id
User-defined Primitives

• Large number of integration scenarios have been shared by the community
  – Amalgam Test Suite (Bibliographic Schemas)
    • Four schemas - 12 possible mapping scenarios
  – Bio schemas originally used in Clio
    • Genomics Unified Schema GUS and BioSQL
  – Many others (see Bogdan Alexe’s archive)

• User defined primitive (UDP)
  – User encodes scenario as iBench XML file
  – Such scenarios can then be declared as UDPs
    • Can be instantiated just like any build-in primitive
Outline

1) Empirical Evaluation of Integration Systems
2) iBench
3) BART
4) Success Stories
5) Demo
6) Conclusions and Future Work
Motivation

• Evaluating constraint-based data cleaning algorithms
  – Need dirty data (and gold standard)
  – Algorithms are sensitive to type of errors

• Need a tool that
  – Given a clean DB and set of constraints
  – Introduces errors that are detectable by the constraints
  – Provides control over how hard the errors are to repair (repairability)
Overview

- **Benchmarking Algorithms for data Repairing and Translation**
Overview

- **Benchmarking Algorithms for data Repairing and Translation**
  - open-source error-generation system with a high level of control over the errors
Overview

• **Benchmarking Algorithms for data Repairing and Translation**
  – open-source error-generation system with an high level of control over the errors

• **Input:** a clean database wrt
  a set of data-quality rules
  and a set of configuration parameters
Overview

• **Benchmarking Algorithms for data Repairing and Translation**
  – open-source error-generation system with an high level of control over the errors

• **Input**: a clean database wrt a set of data-quality rules and a set of configuration parameters

• **Output**: a dirty database (using a set of **cell changes**) and an estimate of how hard it will be to restore the original values
Overview

- **Benchmarking Algorithms for data Repairing and Translation**
  - open-source error-generation system with an high level of control over the errors

- **Input**: a clean database wrt a set of data-quality rules and a set of configuration parameters

- **Output**: a dirty database (using a set of cell changes) and an estimate of how hard it will be to restore the original values
Overview

• **Benchmarking Algorithms for data Repairing and Translation**
  – open-source error-generation system with an high level of control over the errors

• **Input**: a clean database wrt a set of data-quality rules and a set of configuration parameters

• **Output**: a dirty database (using a set of **cell changes**) and an estimate of how hard it will be to restore the original values
Overview

- **Benchmarking Algorithms for data Repairing and Translation**
  - open-source error-generation system with an high level of control over the errors

- **Input**: a clean database wrt a set of data-quality rules and a set of configuration parameters

- **Output**: a dirty database (using a set of **cell changes**) and an estimate of how hard it will be to restore the original values
Creating Violations

• Constraint language: denial constraints
  – Subsumes FDs, CFDs, editing rules, ...
• Update values of a cell to create a violation of a constraint
  – \( t_2.\text{Team} = \text{‘Juventus’} \)

\[
dc: \neg( \text{Player}(n, s, t, st, g), \text{Player}(n’, s’, t’, st’, g’), t=t’, st \neq st’ )
\]
Challenges

• Error generation is an NP-complete problem
  – in the size of the DB

• How to identify cells to change efficiently?

• How to avoid interactions among introduced
  constraint violations?
Error Generation

• Our approach
  – Sound, but not complete
  – Avoid interactions among cell changes
    • Once we decide on a cell change to introduce a violation we exclude other cells involved in the violation from future changes
  – Vio-Gen queries
    • Derived from detection queries for denial constraints
    • Find cell to update such that the update is guaranteed to introduce a violation
    • Tuples that are almost in violation

\[ dq: \text{Player}(n, s, t, st, g), \text{Player}(n', s', t', st', g'), t=t', st \neq st' \]
\[ vg: \text{Player}(n, s, t, st, g), \text{Player}(n', s', t', st', g'), t=t', \text{st} = \text{st'} \]
Outline

1) Empirical Evaluation of Integration Systems
2) iBench
3) BART
4) Success Stories
5) Demo
6) Conclusions and Future Work
Success Stories

• iBench has already been applied successfully by several diverse integration projects
• We have used iBench numerous times for our own evaluations
  – Our initial motivation for building iBench stemmed from our own evaluation needs
Value Invention

• **Translate mappings**
  – from expressive, less well-behaved language (SO tgds)
  – into less expressive, more well-behaved language (st-tgds)

• **Input**: schemas, integrity constraints, mappings
• **Output**: translated mappings (if possible)
• **Evaluation Goal**: how often do we succeed
• **Why iBench**: need a large number of diverse mappings to get meaningful results
• **Evaluation Approach**: generated 12.5 million integration scenarios based on randomly generated configuration file
• **Vagabond**
  – Finding explanations for data exchange errors
    • User marks attribute values in generated data as incorrect
    • System enumerates and ranks potential causes
• **Input**: schemas, integrity constraints, mappings, schema matches, data, **errors**
• **Output**: enumeration of causes or incremental ranking
• **Evaluation Goal**: evaluate scalability, quality
• **Why iBench**:
  • Control characteristics for scalability evaluation
  • Scale real-world examples
Mapping Discovery

- Learning mappings between schemas using statistical techniques
- **Input**: schemas, data, constraints
- **Output**: mappings

– **University of California, Santa-Cruz**
  - Lise Getoor, Alex Memory
  - Reneé Miller
  - [https://linqs.soe.ucsc.edu/people](https://linqs.soe.ucsc.edu/people)
And more ...

- **Functional Dependencies Unleashed for Scalable Data Exchange**
  - Used iBench to compare a new chase-based data exchange algorithm to SQL-based exchange algorithm of ++Spicy

- **Approximation Algorithms for Schema-Mapping Discovery from Data**
  - [ten Cate, Kolaitis, Qian, Tan AMW 2015]
  - Approximate the Gottlob-Senellart notion
  - Kun Qian currently using iBench to evaluate effectiveness of approximation

- **Comparative Evaluation of Chase engines**
  - [Università della Basilicata, University of Oxford]
  - Using iBench to generate schemas, constraints
Outline

1) Empirical Evaluation of Integration Systems
2) iBench
3) BART
4) Success Stories
5) Demo
6) Conclusions and Future Work
Outline

1) Empirical Evaluation of Integration Systems
2) iBench
3) BART
4) Success Stories
5) Demo
6) Conclusions and Future Work
Conclusions

• Empirical Evaluations of Integration Systems
  – Need automated tools for robust, scalable, broad, repeatable evaluations

• BART
  – Controlled error generation
  – Detectable errors, measure repairability

• iBench
  – Comprehensive metadata generator
  – Produces inputs and outputs (gold standards) for a variety of integration tasks
Future Work

• **Data quality measures**
  – Implement complex quality measures

• **iBench**
  – More control over data generation
  – Orchestrating multiple mappings
    • Sequential: e.g., schema evolution
    • Parallel: e.g., virtual integration

• **BART**
  – Support combined mapping/cleaning scenarios
  – How to efficiently generate clean data (without having to run full cleaning algorithm)
  – Similarity measure for instances with labelled nulls/variables
Questions?

• **iBench**
  Webpage: [http://dblab.cs.toronto.edu/project/iBench/](http://dblab.cs.toronto.edu/project/iBench/)
  Code: [https://bitbucket.org/ibencher/ibench/](https://bitbucket.org/ibencher/ibench/)
  Public Scenario Repo: [https://bitbucket.org/ibencher/ibenchconfigurationsandscenarios](https://bitbucket.org/ibencher/ibenchconfigurationsandscenarios)

• **BART**
  Webpage: [http://www.db.unibas.it/projects/bart/](http://www.db.unibas.it/projects/bart/)
  Code: [https://github.com/dbunibas/BART](https://github.com/dbunibas/BART)
Questions?

"FOR A FAIR SELECTION EVERYBODY HAS TO TAKE THE SAME EXAM: PLEASE CLIMB THAT TREE"