Relational Causal Discovery User's Guide

Relational Causal Discovery User's Guide

Published September 30, 2013

General inquiries regarding Relational Causal Discovery should be directed to:

Knowledge Discovery Laboratory
c/o Professor David Jensen, Director
School of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003-9264

or

kdlsupport@kdl.cs.umass.edu

Table of Contents

Chapter 1. Introduction

Relational Causal Discovery (RCD) [1] is a sound and complete algorithm for learning causal models from relational data. RCD employs a novel rule, called relational bivariate orientation, that can detect the orientation of a bivariate dependency with no assumptions on the underlying distribution. Combined with relational extensions to the rules utilized by the PC algorithm [3], RCD is provably sound and complete under the causal sufficiency assumption. Given a database and schema, RCD outputs a partially directed model that represents the equivalence class of statistically indistinguishable relational causal models. Please refer to the paper for additional details.

This initial release of RCD software provides instructions for installing the Relational Causal Discovery package and executing a single run that learns a model for a randomly generated schema, given an oracle or a relational database.

Chapter 2. Installation

Requirements

The Relational Causal Discovery package (RCD) is an experimental prototype and has not been designed to support multiple platforms or alternate versions of the required components. The instructions for installing and running RCD assume that you are working under Mac OS X 10.6 or later. Although RCD may run on other selected UNIX platforms, this has not been tested and is not supported.

The RCD package requires the components listed below. Later versions may also work, but have not been tested.

- Python 3.3
- Network X 1.7
- Numpy 1.7
- Mock 1.0.1
- rpy2 2.3

If you want to use something other than the in-memory data store, make sure you have one of the following supported database systems installed. As with the packages listed above, later versions have not been tested.

- Postgres 9.2
- SQLite 3.7
- MySQL 5.6

Installation

RCD is experimental code. It is important to download the required packages from the listed sources and to use the indicated versions. Other versions, including newer versions, have not been tested and may not work correctly.

To install the Relational Causal Discovery package:

1. Download and unpack the RCD package.
   a. Go to https://kdl.cs.umass.edu/rcd.
   b. Download the file `rcd-1.0.tgz`.
   c. Uncompress the distribution.

      ```
      > tar -xvzf rcd-1.0.tgz
      ```

      This creates the `rcd-1.0` directory.
   d. Set `$RCD_HOME` to be the directory containing the unpacked RCD files (`rcd-1.0`).
2. RCD requires Python 3.3. If you do not have Python 3.3 installed, you can download it from http://www.python.org/download/releases/3.3.0/.

   To reinforce the requirement to use Python 3.3, this document shows calls to execute Python scripts as **`python3.3`**.
3. Set `$PYTHONPATH` to include `$RCD_HOME/src`.

   ```
   > export PYTHONPATH="$RCD_HOME/src:$PYTHONPATH"
   ```
4. Download and install the required packages.
   - Network X 1.7 (http://networkx.lanl.gov/download.html)
   - NumPy 1.7 (http://www.numpy.org/)
   - Mock 1.0.1 (https://pypi.python.org/pypi/mock)

- rpy2 2.3 (https://pypi.python.org/pypi/rpy2/). You must have R version 2.8 or later to use rpy2; version 2.8 is recommended.

You can optionally test the installation by running the unit tests.

```
> cd $RCD_HOME/src/causality
> python3.3 -m unittest
```

You should see that 186 tests ran and 27 tests were skipped.

Chapter 3. Running the Relational Causal Discovery Algorithm

Scripting Overview

RCD scripts are Python scripts and require Python 3.3. To reinforce the requirement to use Python 3.3, this document shows calls to execute Python scripts as

```
> python3.3 script-file
```

where

- *`script-file`* is the name of the Python script file.

This chapter describes two simple scripts that us the RCD algorithm to learn a model. The first uses RCD to learn a model from a randomly generated schema and compares that model to an oracle. The second uses RCD to learn a model from data in a database. Source code files for the scripts discussed in this section are available in `$RCD_HOME/example`.

Oracle Example

The `runOracleRCD.py` script runs a single trial that learns a model on a randomly generated schema and compares that model to an oracle created from the same schema.

Import the needed packages from the Relational Causal Discovery package.

```
from causality.citest.CITest import Oracle
from causality.learning import ModelEvaluation
from causality.learning.RCD import RCD
from causality.modelspace import ModelGenerator
from causality.modelspace import SchemaGenerator
```

We use the Python `logging` module to print execution trace messages and output from the algorithm. Change `INFO` to `DEBUG` for more extensive logging comments.

```
import logging
logger = logging.getLogger(__name__)
logging.basicConfig(level=logging.INFO)
```

Set parameter values.

- `numEntities` is the number of entity types to include in the randomly generated schema
- `numRelationships` is the number of relationship types to be included in the randomly generated schema
- `numDependencies` is the number of dependencies randomly created in the true model
- `hopThreshold` specifies the maximum length of paths underlying these dependencies
- `maxNumParents` is the maximum number of causes for each attribute in the generated model

```
numEntities = 2
numRelationships = 1
numDependencies = 5
hopThreshold = 4
maxNumParents = rcdDepth = 3
```

Randomly generate a schema. For this example, the schema may not include cycles and may include a maximum of one relationship linking any pair of entities.

```
schema = SchemaGenerator.generateSchema(numEntities,
            numRelationships, allowCycles=False, oneRelationshipPerPair=True)
```

Print the generated schema.

```
logger.info(schema)
```

Generate the model from the schema.

```
model = ModelGenerator.generateModel(schema, hopThreshold,
                numDependencies, maxNumParents=maxNumParents)
```

Print the generated model.

```
logger.info('Model: %s', model.dependencies)
```

Set up the oracle using twice the hop threshold specified for running the RCD algorithm. The oracle uses relational d-separation [2] to answer conditional independence queries.

```
oracle = Oracle(model, 2*hopThreshold)
```

Run the RCD algorithm and collect statistics on the learned model.

First, create an RCD instance.

```
rcd = RCD(schema, oracle, hopThreshold, depth=rcdDepth)
```

Phase I creates the skeleton by identifying the undirected dependencies.

```
rcd.identifyUndirectedDependencies()
```

Phase II orients the edges.

```
rcd.orientDependencies()
```

Print recall and precision statistics for the unoriented skeleton and oriented model.

```
logger.info('Skeleton precision: %s',
         ModelEvaluation.skeletonPrecision(model, rcd.undirectedDependencies))
logger.info('Skeleton recall: %s',
         ModelEvaluation.skeletonRecall(model, rcd.undirectedDependencies))
logger.info('Oriented precision: %s',
         ModelEvaluation.orientedPrecision(model, rcd.orientedDependencies))
logger.info('Oriented recall: %s',
         ModelEvaluation.orientedRecall(model, rcd.orientedDependencies))
```

Running the oracle example

To execute the `runOracleRCD.py` script, change to the `$RCD_HOME` directory and run the Python script:

```
> cd $RCD_HOME
> python3.3 example/runOracleRCD.py
```

Understanding RCD Output

Relational paths and variables

The output from RCD uses the notation described below to represent relational paths, relational variables, and relational dependencies. See the figure in Chapter 4 for the schema underlying the following examples.

A *relational path* is an alternating sequence of entity and relationship classes that follow connected paths in the schema. For example:

`[Researcher]` is a singleton path.
`[Venue, Accepts, Paper]` denotes papers published by a specific venue, e.g., in a particular conference proceedings.

`[Researcher, Authors, Paper, Authors, Researcher]` denotes co-authors of the same paper.

A *relational variable* consists of a relational path and an attribute. Relational variables describe attributes of classes reached via a relational path. For example:

`[Venue, Accepts, Paper].topic` is the set of topics covered by papers published in a specific venue.

A *relational dependency* consists of a pair of relational variables with a common first item. It corresponds to a directed probabilistic dependence. For example:

`[Researcher, Authors, Paper].citation_count -> [Researcher].h_index` states that a researcher's h-index is dependent on the citation count for the papers he or she publishes.

See Maier, Marazopoulou, and Jensen [2] for more information on relational paths, variables, and dependencies.

Results from the oracle example

The output shown below represents one run of `runOracleRCD.py`. The trace has been annotated with descriptions of each section and reformatted for legibility. Because the schema and model are randomly generated for each run, the trace will also vary from run to run.

The generated schema.

```
INFO:__main__:<Schema
   [<Entity 'B': [<Attribute 'Y1' None>]>,
    <Entity 'A': [<Attribute 'X1' None>,
                  <Attribute 'X2' None>,
                  <Attribute 'X3' None>]>]:
   [<Relationship 'AB': {'B': 'one', 'A': 'many'}
                 [<Attribute 'XY1' None>]>]>
```

The generated model.

```
INFO:__main__:Model:
   [[A, AB].XY1 -> [A].X2, [A].X1 -> [A].X3,
    [AB, A].X3 -> [AB].XY1, [AB, B].Y1 -> [AB].XY1,
    [B, AB, A].X3 -> [B].Y1]
```

Phase I execution.

```
INFO:causality.learning.RCD:Phase I: identifying undirected dependencies
INFO:causality.learning.RCD:Number of potentialDeps 22
INFO:causality.learning.RCD:Conditioning set size 0
INFO:causality.learning.RCD:Conditioning set size 1
INFO:causality.learning.RCD:Conditioning set size 2
INFO:causality.learning.RCD:Conditioning set size 3
```

Undirected dependencies after Phase I.

```
INFO:causality.learning.RCD:Undirected dependencies:
   [[A].X1 -> [A].X3,
    [A].X3 -> [A].X1,
    [B, AB].XY1 -> [B].Y1,
    [A, AB].XY1 -> [A].X2,
    [A, AB].XY1 -> [A].X3,
    [AB, B].Y1 -> [AB].XY1,
    [AB, A].X2 -> [AB].XY1,
    [AB, A].X3 -> [AB].XY1,
    [B, AB, A].X3 -> [B].Y1,
    [A, AB, B].Y1 -> [A].X3]
```

Number of conditional independence tests run so far.

```
INFO:causality.learning.RCD:{
   'depth 0': 22,
   'depth 1': 39,
   'depth 2': 12,
   'depth 3': 3,
   'total': 76,
   'Phase I': 76,
   'Phase II': 0}
```

Phase II execution.

```
INFO:causality.learning.RCD:Phase II: orienting dependencies
INFO:causality.learning.EdgeOrientation:RBO Oriented edge:
   [B, AB, A].X3->[B].Y1
INFO:causality.learning.EdgeOrientation:RBO Oriented edge:
   [AB].XY1->[AB, A].X2
INFO:causality.learning.EdgeOrientation:RBO Oriented edge:
   [AB, A].X3->[AB].XY1
```

Final separating sets.

```
INFO:causality.learning.RCD:Separating sets:
  {([B, AB, A].X1, [B].Y1): {[B, AB, A].X3},
   ([B, AB, A].X2, [B].Y1): {[B, AB, A].X3, [B, AB].XY1},
   ([A].X2, [A].X3): [A, AB].XY1},
   ([AB].XY1, [AB, A, AB].XY1): {[AB, B].Y1, [AB, A].X3},
   ([A, AB].XY1, [A].X1): {[A].X3},
   ([A, AB, B].Y1, [A].X2): {[A, AB].XY1},
   ([A].X1, [A, AB, B].Y1): {[A].X3},
   ([B].Y1, [B, AB, A, AB].XY1): {[B, AB, A].X3},
   ([B].Y1, [B, AB, A].X1): {[B, AB, A].X3},
   ([B].Y1, [B, AB, A, AB, B].Y1): {[B, AB, A].X3},
   ([A].X2, [A].X1): {[A].X3},
   ([A].X1, [A].X2): {[A].X3},
   ([AB].XY1, [AB, A].X1): {[AB, A].X3},
   ([AB, A, AB, B].Y1, [AB].XY1): {[AB, B].Y1, [AB, A].X3},
   ([AB, A].X1, [AB].XY1): {[AB, A].X3},
   ([A].X3, [A].X2): {[A, AB].XY1},
   ([B, AB, A, AB].XY1, [B].Y1): {[B, AB, A].X3},
   ([B, AB, A, AB, B].Y1, [B].Y1): {[B, AB, A].X3},
   ([A].X1, [A, AB].XY1): {[A].X3},
   ([B].Y1, [B, AB, A].X2): {[B, AB, A].X3, [B, AB].XY1},
   ([AB, A, AB].XY1, [AB].XY1): {[AB, B].Y1, [AB, A].X3},
   ([A, AB, B].Y1, [A].X1): {[A].X3},
   ([AB].XY1, [AB, A, AB, B].Y1): {[AB, B].Y1, [AB, A].X3},
   ([A].X2, [A, AB, B].Y1): {[A, AB].XY1}}
```

Final oriented dependencies after Phase II. (Unoriented dependencies are presented in both directions.)

```
INFO:causality.learning.RCD:Oriented dependencies:
   {[B, AB, A].X3 -> [B].Y1,
    [A].X3 -> [A].X1,
    [AB, B].Y1 -> [AB].XY1,
    [AB, A].X3 -> [AB].XY1,
    [B, AB].XY1 -> [B].Y1,
    [A, AB].XY1 -> [A].X2,
    [A].X1 -> [A].X3}
```

Final number of conditional independence tests run.

```
INFO:causality.learning.RCD:
  {'depth 0': 22,
   'depth 1': 39,
   'depth 2': 12,
   'depth 3': 3,
   'total': 97,
   'Phase I': 76,
   'Phase II': 21}
```

Counts of how many times each edge orientation rule was applied.

```
INFO:causality.learning.RCD:{'CD': 0, 'KNC': 0, 'CA': 0, 'RBO': 3, 'MR3': 0}
```

Precision and recall statistics for the initial and oriented skeleton.

```
INFO:__main__:Skeleton precision: 1.0
INFO:__main__:Skeleton recall: 1.0
INFO:__main__:Oriented precision: 1.0
INFO:__main__:Oriented recall: 0.6
```

Database Example

The `runDatabaseRCD.py` script runs a single trial that learns a model on data stored in a Postgres database.

Start a local server running PostgreSQL 9.2 running on “localhost” and port 5432. You must load `example/run-test-data.sql` into the “test” database before running this script.

Import the needed packages from the Relational Causal Discovery package.

```
from causality.model.Schema import Schema
from causality.learning.RCD import RCD
from causality.citest.CITest import LinearCITest
from causality.datastore.PostgreSqlDataStore import PostgreSqlDataStore
```

We use the Python `logging` module to print execution trace messages and output from the algorithm. Change `INFO` to `DEBUG` for more extensive logging comments.

```
import logging
logger = logging.getLogger(__name__)
logging.basicConfig(level=logging.INFO)
```

Define and print the schema. See Chapter 4 for more information about defining schemas.

```
schema = Schema()
schema.addEntity('A')
schema.addAttribute('A', 'X')
schema.addAttribute('A', 'W')
schema.addEntity('B')
schema.addAttribute('B', 'Y')
schema.addRelationship('AB', ('A', Schema.ONE), ('B', Schema.MANY))
schema.addEntity('C')
schema.addAttribute('C', 'Z')
schema.addAttribute('C', 'V')
schema.addRelationship('BC', ('B', Schema.ONE), ('C', Schema.MANY))

logger.info(schema)
```

Load the data into a Python object. You must have loaded the data into your Postgres database before running this script.

```
dataStore = PostgreSqlDataStore(dbname='test', user='test', password='test',
                                host='localhost', port='5432')
```

Define the statistical test used for tests of conditional independence. `LinearCITest` uses linear regression.

```
linearCITest = LinearCITest(schema, dataStore)
```

Set parameter values: `hopThreshold` specifies the maximum length of paths underlying dependencies.

```
hopThreshold = 4
```

Run the RCD algorithm.

First, create an RCD instance. In this example, we use the default value for `depth`.

```
rcd = RCD(schema, linearCITest, hopThreshold)
```

Phase I creates the skeleton by identifying the undirected dependencies.

```
rcd.identifyUndirectedDependencies()
```

Phase II orients the edges.

```
rcd.orientDependencies()
```

Running the database example

To execute the `runDatabaseRCD.py` script, change to the `$RCD_HOME` directory and run the Python script:

```
> cd $RCD_HOME
> python3.3 example/runDatabaseRCD.py
```

See “Understanding RCD Output” for information on understanding the output for this run.

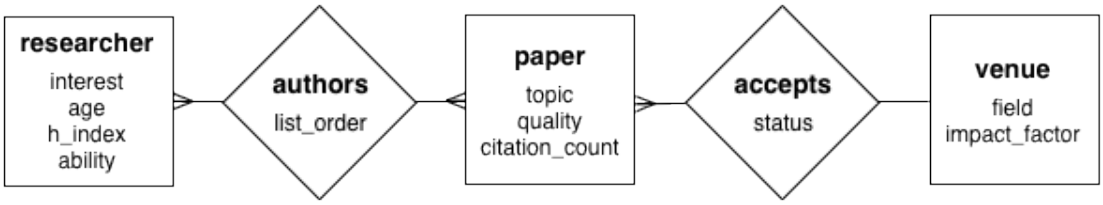
Chapter 4. Schemas and Databases

This chapter describes how to specify a data schema for RCD in Python. Schemas consist of entities, relationships, attributes on those entities and relationships, and any known dependencies. The final section describes constraints on relational databases used to hold data for RCD.

Defining Schemas

Schema

The code fragments in this chapter define the schema used for the example shown below:



The schema definition is stored in a `Schema` object.

```
schema = Schema()
```

Entities

The example schema contains three entities: researchers, papers, and venues. Each entity is defined by a line in the Python script of the form

`schema.entity(`*`label`*`)`

The entities in the example schema are defined by the following lines:

```
schema.addEntity('paper')
schema.addEntity('researcher')
schema.addEntity('venue')
```

Relationships

The example schema contains two relationships: authors and accepts. Each relationship is defined by a line in the Python file of the form

***`schema`*`.addRelationship(`*`label`*`, (`*`entity_1`*`, `*`cardinality`*`), (`*`entity_2`*`, `*`cardinality`*`))`**

where

- *`schema`* is the name of the `Schema` object
- *`label`* is the name of the relationship
- *`entity_1`* and *`entity_2`* are the names of the entities participating in this relationship
- *`cardinality`* is the cardinality of the relationship for that entity. Cardinality can take one of two values:

  ```
  Schema.ONE
  Schema.MANY
  ```

Relationships are bi-directional.

The relationships in the example schema are defined by the following lines:

```
schema.addRelationship('accepts', ('paper', Schema.MANY),
                                  ('venue', Schema.ONE))
```

```
schema.addRelationship('authors', ('researcher', Schema.MANY),
                                  ('paper', Schema.MANY))
```

These lines specify that accepts is a one-to-many relationship linking papers and venues (a single venue may accept multiple papers, but a paper can only be accepted by one venue), and authors is a many-to-many relationship linking researchers and papers (researchers can author multiple papers, and an individual paper may have multiple authors).

Attributes on entities

The example schema defines attributes for each entity, as shown in the illustration. Each entity attribute is defined by a line in the Python file of the form

***schema*.addAttribute(*entity*, *attr-name*)**

where

- *schema* is the name of the Schema object
- *entity* is the name of the entity possessing this attribute
- *attr-name*
  is the name of the attribute being defined

The entity attributes in the example schema are defined by the following lines:

```
schema.addAttribute('researcher', 'interest')
schema.addAttribute('researcher', 'age')
schema.addAttribute('researcher', 'h_index')
schema.addAttribute('researcher', 'ability')
schema.addAttribute('paper', 'topic')
schema.addAttribute('paper', 'quality')
schema.addAttribute('paper', 'citation_count')
schema.addAttribute('venue', 'field')
schema.addAttribute('venue', 'impact_factor')
```

Attributes on relationships

The RCD algorithm is able to use information from attributes on both entities and relationships. Each relationship attribute is defined by a line in the Python script of the form:

***schema*.addAttribute(*relationship*, *attr-name*)**

where

- *schema* is the name of the Schema object
- *relationship* is the name of the relationship posessing this attribute
- *attr-name* is the name of the attribute being defined

The relationship attributes in the example schema are defined by the following lines:

```
schema.addAttribute('authors', 'list_order')
schema.addAttribute('accepts', 'status')
```

Dependencies

You can represent known dependencies for a schema. Dependencies are stored in a list.

```
dependencies = []
```

Each dependency is defined by a line in the Python script of the form:

***dependencies*.append('*relVar1 -> relVar2*')**

where

- *dependencies* is the list containing the defined dependencies

- *relVar1* is the cause relationship variable
- *relVar2* is the effect relationship variable (that is, the attribute being caused)

The known dependencies for the example schema are defined by the following lines:

```
dependencies.append('[paper, authors, researcher].interest -> [paper].topic')
dependencies.append('[paper, authors, researcher].ability -> [paper].quality')
dependencies.append('[paper, authors, researcher].h_index ->
                     [paper].citation_count')
dependencies.append('[paper, accepts, venue].impact_factor ->
                     [paper].citation_count')
dependencies.append('[paper].quality -> [paper].citation_count')
dependencies.append('[researcher].age -> [researcher].ability')
dependencies.append('[researcher].ability -> [researcher].h_index')
```

Database Configuration

You must obey the following conventions when using a database to store data for RCD.

- There must be one table per entity or relationship.
- Every table must have an ID column named "`id`".
- Every attribute must have a column in the appropriate entity or relationship table. The column name must be the same as the attribute name.
- Every relationship table must include two foreign key columns identifying the linked entities. These columns must be named *`entity`*`_id` where *`entity`* is the name of one of the linked entities.

References

[1] M. Maier, K. Marazopoulou, D. Arbour, and D. Jensen. A sound and complete algorithm for learning causal models from relational data. *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*. 2013.

[2] M. Maier, K. Marazopoulou, and D. Jensen. Reasoning about Independence in Probabilistic Models of Relational Data. arXiv:1302.4381. 2013.

[3] P. Spirtes, C. Glymour, and R. Schienes. *Causation, Prediction, and Search*. MIT Press. 2000.