Figaro

Quick Start Guide

Version 3.1

charles river analytics

Copyright

Charles River Analytics Inc.
625 Mt. Auburn Street
Cambridge, MA 02138

Examples and Data

Trademarks and Patents

Document Information

Figaro Quick Start Guide, for Figaro Version 3.1
Last updated January 7, 2016

Table of Contents

ABOUT THIS GUIDE

This Quick Start Guide provides detailed steps so you can quickly create models with Figaro. It assumes you are already familiar with the Scala programming language. If you are an advanced user, or a programmer planning to incorporate Figaro into your application, please see *Practical Probabilistic Programming* or the *Figaro Tutorial*.

Key topics include:

- Related documents
- Typographic conventions
- Feedback and technical support

Related documents

In addition to this guide, you can find more information that will help you better understand and use Figaro in the following documents:

- Figaro Release Notes
- Practical Probabilistic Programming, by Avi Pfeffer
  http://www.manning.com/pfeffer
- Figaro Tutorial
- Charles River Analytics website
  https://www.cra.com/Figaro
- Scala documentation
  http://www.scala-lang.org/documentation/
- SBT documentation
  http://www.scala-sbt.org/documentation

Typographic conventions

Specific conventions are used in this guide to convey additional information about a subject:

Style	Description	Example
`Code`	Code style is used for text that is used literally, appearing exactly as shown. This includes command names, path and file names, and system information.	`E:\Figaro\setup.exe`
`Italic code`	Italic code style is used for names of variables that you must provide. For example, you need to supply a value for <*`your_file`*> in the path name example to the right.	`C:\Figaro\data\<`*`your_file`*`>`

Note Notes highlight information, provide supplementary information, offer time-saving or easier ways to perform the same task, or explain how to prevent errors or data loss. Be sure to read this information carefully.

Feedback and technical support

We appreciate your comments about this guide. Please contact us with your comments, questions, and requests for technical support.

By mail:	Charles River Analytics Attn: Figaro Technical Documentation 625 Mount Auburn Street, Cambridge, MA 02138
By email:	figaro@cra.com

1 GETTING STARTED

Figaro is a probabilistic programming language that helps you quickly develop probabilistic models. (Figaro models are data structures within the Scala programming language.) With Figaro, you can both develop models and select a reasoning algorithm that draws useful conclusions when you run your program. Figaro provides a library of models, elements, and reasoning algorithms.

This chapter includes the following key topics:

- Overview
- Installing the Simple Build Tool (SBT)
- Downloading the FigaroWork project

Overview

We strongly recommend that you begin by using Simple Build Tool (SBT) to compile and run your Figaro projects. We have provided the FigaroWork project as an initial example for you to explore. When you use the SBT FigaroWork project, it downloads the latest versions of Scala and Figaro files for you. Figaro does not require a particular platform; it works on Windows, Mac, and Linux.

Creating your models in the FigaroWork source directory and using SBT to compile and run your code is the quickest and easiest way to get started using Figaro. For other methods of using Figaro, please see *Practical Probabilistic Programming* or the *Figaro Tutorial*.

To get started using Figaro

1. Install the Simple Build Tool (SBT).

   For more information, see *Installing the Simple Build Tool* on page 2.

2. Download the FigaroWork project.

   For more information, see *Downloading the FigaroWork project* on page 3.

3. Create a simple probabilistic model and use a reasoning algorithm to query the model.

   For more information, see *Tutorial: Hello World* on page 5.

4. Explore the Burglary example provided in the Figaro download.

   For more information, see *Tutorial: Burglary Example* on page 11.

   Once you follow these steps, you should be ready to use *Practical Probabilistic Programming* or the *Figaro Tutorial* to combine Figaro model elements and reasoning algorithms to meet your probabilistic programming needs.

Installing the Simple Build Tool (SBT)

Figaro uses the Simple Build Tool (SBT) to manage builds. Figaro 3.1 requires SBT version 0.13.6 or later. Documentation for SBT is available at http://www.scala-sbt-org/0.13/tutorial/index.html.

To install the Simple Build Tool

1. Download the version of the SBT installer you need from http://www.scala-sbt.org/download.html.

2. Open the installer and follow the instructions to install SBT.

3. Open a command prompt.

4. Add the `sbt\bin` directory to your system `Path` variable using the appropriate method for your operating system.

5. Run SBT to download the necessary files from the internet and test the installation. Enter:

   ```
   sbt
   ```

   You can also double-click the `sbt.bat` file in Windows to display the SBT console window.

   SBT will provide information messages as it downloads files, then it will display the SBT prompt.

6 Exit SBT. Enter:

```
exit
```

To verify the Simple Build Tool installation

1 Create a `HiWorld` directory.

2 Create a `HiWorld.scala` file that contains the following:

```
object Hi {
  def main(args: Array[String]) = println("Hi!")
}
```

3 Open a command prompt.

4 Change directory to the HiWorld directory. For example, enter:

```
cd C:\Figaro\HiWorld
```

5 Start SBT. Enter:

```
sbt
```

SBT will provide information messages as it sets the project to the current directory. The command prompt will look like:

```
>
```

6 Run the HiWorld project. Enter:

```
run
```

SBT will provide information messages as it updates files, resolves sources, and compiles the `HiWorld.scala` file.

You will see a result similar to:

```
[info] ...
[info] Running Hi
Hi!
[success] Total time: 10 s, completed Sept 26, 2015 11:40:06 AM
```

7 Exit SBT. Enter:

```
exit
```

Downloading the FigaroWork project

We have provided the FigaroWork project as an initial example for you to explore. When you use the SBT FigaroWork project, it downloads the correct versions of Scala and Figaro files for you. Creating your models in the FigaroWork source directory and using SBT to compile and run them is the quickest and easiest way to get started using Figaro.

The instructions in this guide are optimized for Windows, but the commands are similar for Mac and Linux.

To download the Figaro Work project

1 Navigate to http://www.cra.com/figaro.

2 Click the Figaro Work link in the Figaro Work section to download the `FigaroWork.zip` file.

3 Extract the files to a folder on your local computer.

This will create a top-level `FigaroWork` directory, with two subdirectories: `FigaroWork` and `target`.

To verify your environment is configured properly

1 Open a command prompt.

2 Change directory to the `FigaroWork/FigaroWork` directory.

3 Test SBT on the FigaroWork project. For example, on Windows, enter:

```
sbt "runMain Test"
```

SBT will provide information messages as it updates files, resolves sources, and compiles the `Test.scala` file included in the FigaroWork download. This process may take a few minutes.

This command is different if you are using a Mac or Linux operating system.

4 Check that the following output is created:

```
[info] Running Test
1.0
[success] Total time: 200 s, completed Sept 26, 2015 12:40:06 AM
```

If the output is not created, check that you ran SBT from the correct FigaroWork directory.

Downloading Figaro

We recommend that you download the FigaroWork project when you are initially exploring Figaro. (For instructions, see *Downloading the FigaroWork project* on page 3.) When you are ready to explore Figaro further, you can download Scala and the Figaro JAR and run Figaro from the Scala prompt. You can also integrate Figaro into an existing project with Maven, Ivy, or SBT.

To download Figaro

1 Navigate to http://www.cra.com/figaro from a browser.

2 Click the Figaro zip file in the Download Figaro section.

3 Extract the files from the zip file.

The zip file contains the compiled Figaro code as a JAR file, examples, documentation, Scaladoc, and source code.

4 Add the directory that contains the Figaro JAR file to your system `Path` variable using the appropriate method for your operating system. For example, if you extracted the Figaro files to a Figaro folder on your C drive, add:

```
C:\Figaro\figaro_3.3.0.0
```

2 TUTORIAL: HELLO WORLD

This tutorial walks you through the three basic steps involved in probabilistic programming with Figaro. You will use the Scala interactive console, which reads one line of Scala code at a time and interprets it, to:

- Create a very simple probabilistic model—one that produces the string "Hello world!" with a probability of 1.0.
- Select and run a probabilistic reasoning algorithm.
- Query the model using your selected algorithm to produce a result.

To successfully follow this tutorial, you must first follow the instructions in *Installing the Simple Build Tool* and *Downloading the FigaroWork project*.

This tutorial includes the following key topics:

- Creating a model
- Instantiating a reasoning algorithm
- Querying the model using a reasoning algorithm
- Producing results from queries
- Modifying the model

Creating a model

1 Create a `HelloWorldTest.scala` file in the `FigaroWork/src/main/scala` directory.

SBT requires that your source code be placed in this directory.

If you cannot find this directory, follow the instructions in *Installing the Simple Build Tool* and *Downloading the FigaroWork project*.

2 Edit the file to load the portion of the Figaro package that allows you to create models. Enter:

```
import com.cra.figaro.language_
```

The _ at the end of this line imports all the classes in the `figaro.language` package. It is equivalent to Java's *.

3 Create an object.

```
object HelloWorldTest{
   def main(args: Array[String]) {
   }
}
```

4 Create a probabilistic model after the definition of main. Enter:

```
val helloWorldElement = Constant("Hello world!")
```

We are creating an instance of Figaro's `Constant` element with the field `HelloWorldElement`. (In Scala, a field is similar to an immutable variable.) This element produces a value with the probability of 1.0. When `helloWorldElement` is queried, it produces the string `"Hello world!"` with a probability of 1.0.

Your file now looks like this:

```
import com.cra.figaro.language.exit_
object HelloWorldTest{
   def main(args: Array[String]) {
     val helloWorldElement = Constant("Hello world!")
 }
}
```

Instantiating a reasoning algorithm

5 Determine which algorithm you want to use. Figaro includes the following algorithms:

- Exact inference using variable elimination – Expand the universe to include all elements generated in any possible world, convert each element into a factor, and apply variable elimination to all the factors with the `VariableElimination` object
- Approximate inference using belief propagation – Expand the universe to include all elements generated in any possible world, convert each element into a factor, create a factor graph from the factors, pass messages between factor and variable nodes, and answer queries on the targets using the posterior distributions computed at each variable node with the `BeliefPropagation` object
- Importance sampling – Compute the probability of evidence in a universe with a simple forward sampling approach with the `Importance` object. When this algorithm encounters a constraint, it multiplies the weight of the sample by the value of the constraint. Use this algorithm when

expanding a model produces an infinite number of elements or if you have atomic continuous models.

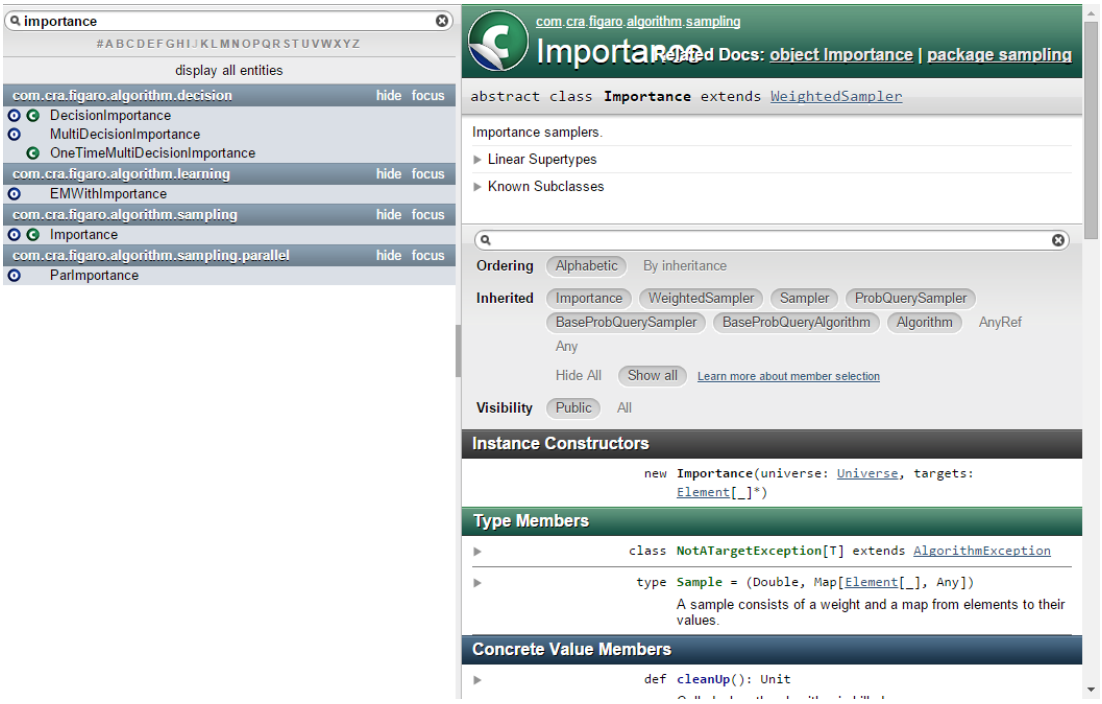
- Metropolis-Hastings Markov chain Monte Carlo – Define proposal distributions, where a proposal includes new randomnesses for any number of elements with the `ProposalScheme` object. Pass your proposal scheme to the `MetropolisHastings` object to propose a new state at each step of the algorithm, and either accept or reject the proposal.
- Probability of evidence – Compute the probability of all the evidence (not the conditional probability of evidence) in steps so you can include or exclude conditions and constraints with the `ProbEvidenceSampler` object
- Most probable explanation (MPE) – Compute the most likely values of elements given the available evidence using variable elimination or simulated annealing with the `MPEVariableElimination` or `MetropolisHastingsAnnealer` objects
- Particle filtering – Reason about dynamic models using an initial model, transition model, and number of particles the algorithm should produce at each step with the `ParticleFilter` object
- Parameter learning – Learn model parameters from data using expectation maximization with the `ExpectationMaximization` object

Figaro provides both one-time (the algorithm is run once) and anytime (the algorithm is run until stopped) versions of some of these algorithms. In addition to the built-in algorithms, Figaro provides a number of tools for creating your own reasoning algorithms. For more information about Figaro's reasoning algorithms, see *Practical Probabilistic Programming* or the *Figaro Tutorial*.

For this tutorial, we select the importance sampling algorithm, which we can use to determine the probability that the model produces a certain result.

6 Load the portion of the Figaro package that contains the definition of the sampling algorithm.

To determine which package you need to import, open http://www.cra.com/figaro, scroll to the Download Figaro section, and click the link to the documentation of the Figaro library interface.



You can review this javadoc or search for the a specific Figaro element to find the correct package.

After the first import statement in the `HelloWorldTest.scala` file, enter:

```
import com.cra.figaro.algorithm.sampling._
```

7 Instantiate the sampling algorithm. Enter:

```
val sampleHelloWorld = Importance(1000, helloWorldElement)
```

We are creating an instance of an importance sampler with Figaro's `Importance` algorithm using the field `sampleHelloWorld`. This algorithm uses a simple forward sampling approach. The first argument to this algorithm is the number of samples it will take. The second argument is the element in the model that will be queried. When this algorithm is run, it will sample the `helloWorldElement` 1000 times.

Your file now looks like this:

```
import com.cra.figaro.language._
import com.cra.figaro.algorithm.sampling._

object HelloWorldTest{
   def main(args: Array[String]) {
      val helloWorldElement = Constant("Hello world!")
      val sampleHelloWorld = Importance(1000, HelloWorld)
   }
}
```

Querying the model using a reasoning algorithm

8 Add code that will start the algorithm when you run the program. Enter:

```
sampleHelloWorld.start()
```

9 Write a query on the algorithm for the probability that the model will output a certain string. For example, enter:

```
sampleHelloWorld.probability(helloWorldElement, "Hello world!")
sampleHelloWorld.probability(helloWorldElement, "Goodbye world!")
```

These lines use the `probability` method of Figaro's `Importance` algorithm to calculate the probability of the `helloWorldElement` outputting the specified string after sampling it 1000 times.

10 Output the results. Edit the lines you created in the last step:

```
println("Probability of Hello world:")
println(sampleHelloWorld.probability(helloWorldElement, "Hello
world!"))
println("Probability of Goodbye world:")
println(sampleHelloWorld.probability(helloWorldElement, "Goodbye
world!"))
```

Your file now looks like this:

```
import com.cra.figaro.language._
import com.cra.figaro.algorithm.sampling._

object HelloWorldTest{
  def main(args: Array[String]){
    val helloWorldElement = Constant("Hello world!")
    val sampleHelloWorld = Importance(1000, helloWorldElement)

    sampleHelloWorld.start()
```

```
    println("Probability of Hello world:")
    println(sampleHelloWorld.probability(helloWorldElement, "Hello
      world!"))
    println("Probability of Goodbye world:")
    println(sampleHelloWorld.probability(helloWorldElement, "Goodbye
      world!"))
  }
}
```

Producing results from queries

11 Open a command prompt.

12 Change directory to the `FigaroWork/FigaroWork` directory. For example, enter:

```
cd c:\Program Files (x86)\sbt\FigaroWork
```

SBT requires that you run your source code from the top-level FigaroWork directory (the same directory that contains the `project`, `src`, and `target` directories).

13 Enter the following to run your project:

```
sbt "runMain HelloWorldTest"
```

Figaro executes the program, running the reasoning algorithm and querying the model defined within the program. Running the program instantiates an Importance sampler, which takes 1000 samples from the model and saves each sample. Then it computes the probability of each string you queried within that result set.

14 Your `HelloWorldTest` project should produce the following output:

```
[info] Running HelloWorldTest
Probability of Hello world:
1.0
Probability of Goodbye world:
0.0
```

That is, `helloWorldElement` produced the string `"Hello world!"` 1000 times and never produced the string `"Goodbye world!"`

Modifying the model

15 Modify the model to be slightly more complex using Figaro's `Select` element and `VariableElimination` algorithm.

```
val helloWorldElement = Select(0.8->"Hello world",0.2->"Goodbye
world"
```

This element produces a set of values with defined probabilities. Now, when the `helloWorldElemenet` is queried, it produces the string `"Hello world!"` with a probability of 0.8, and produces the string `"Goodbye world!"` with a probability of 0.2.

The `VariableElimination` algorithm will provide an exact probability, instead of the approximate probability provided by the Importance sampling algorithm.

Your file now looks like this (changes are shown in bold):

```
import com.cra.figaro.language._
import com.cra.figaro.algorithm.sampling._

object HelloWorldTest{
  def main(args: Array[String]){
    val helloWorldElement = Select(0.8->"Hello world",0.2->"Goodbye
      world")
    val sampleHelloWorld = VariableElimination(HelloWorld)

    sampleHelloWorld.start()

    println("Probability of Hello world:")
    println(sampleHelloWorld.probability(helloWorldElement, "Hello
      world!"))
    println("Probability of Goodbye world:")
    println(sampleHelloWorld.probability(helloWorldElement,
      "Goodbye world!"))
  }
}
```

If you cannot find this file, follow the instructions in *Tutorial: Hello World*.

16 Run your project:

```
sbt "runMain HelloWorldTest"
```

Figaro executes the program, running the reasoning algorithm and querying the model defined within the program. Running the program instantiates an instance of the VariableElimination algorithm which answers the query by marginalizing out non-query variables.

17 Your `HelloWorldTest` project should produce the following output:

```
[info] Running HelloWorldTest
Probability of Hello world:
0.8
Probability of Goodbye world:
0.2
```

18 Exit SBT. Enter:

```
exit
```

3 TUTORIAL: BURGLARY EXAMPLE

This tutorial reviews a Bayesian network implemented with Figaro. This model is provided as part of the Figaro download. It models the likelihood that a burglar set off a burglar alarm as opposed to an earthquake. In this tutorial, you will:

- Create a Bayesian network to model the scenario.
- Create a variable elimination algorithm to measure probabilities.
- Query the model using your algorithm to determine whether the alarm was triggered by a burglar.
- Explore other example projects provided in the Figaro download.

To successfully follow this tutorial, you must first follow the instructions in *Installing the Simple Build Tool* and *Downloading the FigaroWork project*.

This tutorial includes the following topics:

- Downloading the Figaro examples
- Creating a Bayesian network model
- Creating a reasoning algorithm
- Starting the algorithm and using it to query the model
- Running the Burglary example
- Exploring additional examples

Downloading the Figaro examples

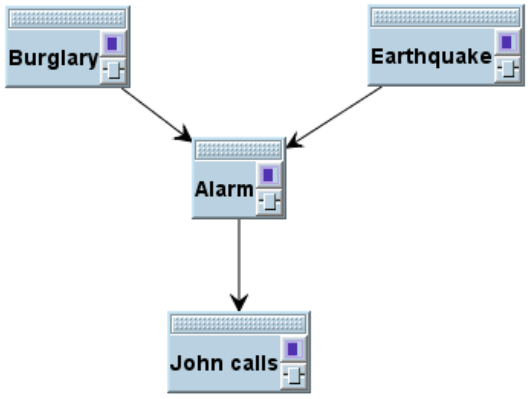
1 Download and extract the Figaro JAR. For instructions, see *Downloading Figaro* on page 4.

You can review the code of the examples by using a zip utility to open the `figaroexamples_<version>-sources.jar` file. For a description of the examples, see *Exploring additional examples* on page 15.

Creating a Bayesian network model

For this example, we want to model the following scenario: Your neighbor, John, is calling you. He usually calls you when your burglar alarm is going off. Sometimes, the alarm goes off because of a burglar, and sometimes an earthquake sets it off.

The Bayesian network for this scenario looks like this:



2 Create a `BurglaryExample.scala` file in the `FigaroWork/src/main/scala` directory.

If you cannot find this directory, follow the instructions in *Installing the Simple Build Tool* and *Downloading the FigaroWork project*.

3 Begin by loading the Figaro example classes and loading the portion of the Figaro package that allows you to use the VariableElimination algorithm to calculate conditional probabilities given evidence, create models, and easily create conditional probability distributions. Enter:

```
import com.cra.figaro.algorithm.factored._
import com.cra.figaro.language._
import com.cra.figaro.library.compound._
```

4 Create an object.

```
object Burglary{
   def main(args: Array[String]) {
   }
}
```

5 Define the nodes of the Bayesian network.

```
val burglary = Flip(0.01)

val earthquake = Flip(0.0001)
```

```
val alarm = CPD(burglary, earthquake,
  (false, false) -> Flip(0.001),
  (false, true) -> Flip(0.1),
  (true, false) -> Flip(0.9),
  (true, true) -> Flip(0.99))

val johnCalls = CPD(alarm,
  false -> Flip(0.01),
  true -> Flip(0.7))
```

Here, we define the parent nodes, `burglary` and `earthquake`, using Figaro's `Flip` element. Burglary is a probabilistic element that is true with probability .01 and false with probability .99. (That is, burglaries occur 1% of the time.) Earthquake is a probabilistic element that is true with probability .0001 and false with probability .9999. (That is, earthquakes occur .01% of the time.)

Next, we define the child node, alarm. This node has a conditional probability distribution, defined by Figaro's `CPD` element, which describes its dependency on the values of its parents. For example, the likelihood that the alarm will be triggered by only an earthquake is 10%, while the likelihood that it will be triggered by only a burglary is 90%.The likelihood that it will go off in the absence of either event is .1%, while if both events occur, the probability that the alarm will go off is 99%.

Finally, we define a child node of alarm. This node is called johnCalls. The conditional probability distribution for this node tells us that if the alarm is not going off, the probability that John will call is very low, and if the alarm is going off, he will probably call.

Your file now looks like this:

```
import com.cra.figaro.algorithm.factored._
import com.cra.figaro.language._
import com.cra.figaro.library.compound._

object Burglary {
  Universe.createNew()

  private val burglary = Flip(0.01)

  private val earthquake = Flip(0.0001)

  private val alarm = CPD(burglary, earthquake,
    (false, false) -> Flip(0.001),
    (false, true) -> Flip(0.1),
    (true, false) -> Flip(0.9),
    (true, true) -> Flip(0.99))

  private val johnCalls = CPD(alarm,
    false -> Flip(0.01),
    true -> Flip(0.7))

  def main(args: Array[String]) {
  }
}
```

Creating a reasoning algorithm

6 Post evidence. Enter:

```
johnCalls.observe(true)
val alg = VariableElimination(burglary, earthquake)
```

Given the evidence that John is calling, we calculate the likelihood that there is a burglary and the likelihood that there is an earthquake using Figaro's `VariableElimination` algorithm. This algorithm calculates the conditional probability of the targets, given the evidence. It converts each element into a factor, then applies variable elimination to all the factors.

Starting the algorithm and using it to query the model

7 Start the algorithm. Enter:

```
alg.start()
```

8 Query the algorithm for the probability that the alarm will be triggered by a burglary. Enter:

```
alg.probability(burglary, true)
```

These lines use the `probability` method of Figaro's `VariableElimination` algorithm to instantiate the algorithm, run inference, and return the probability that the burglary node is in the true state.

9 Output the results. Edit the lines you created in the last step:

```
println("Probability of burglary: " + alg.probability(burglary,
true))
```

10 Terminate the algorithm to free up the memory used for the results:

```
alg.kill
```

Your file now looks like this:

```
import com.cra.figaro.algorithm.factored._
import com.cra.figaro.language._
import com.cra.figaro.library.compound._

object Burglary {
  Universe.createNew()

  private val burglary = Flip(0.01)

  private val earthquake = Flip(0.0001)

  private val alarm = CPD(burglary, earthquake,
    (false, false) -> Flip(0.001),
    (false, true) -> Flip(0.1),
    (true, false) -> Flip(0.9),
    (true, true) -> Flip(0.99))

  private val johnCalls = CPD(alarm,
    false -> Flip(0.01),
    true -> Flip(0.7))

  def main(args: Array[String]) {
    johnCalls.observe(true)
    val alg = VariableElimination(burglary, earthquake)
```

```
    alg.start()
    println("Probability of burglary: " + alg.probability(burglary,
true))
    alg.kill
  }
}
```

Running the Burglary example

11 Enter the following to run the Burglary example:

```
sbt "runMain Burglary"
```

Figaro instantiates the initial state then calculates the exact probability of the target elements (burglary and earthquake), given the evidence that John called.

12 Your `Burglary` project should produce the following output:

```
[info] Running Burglary
Probability of burglary: 0.3733781172643905
```

Exploring additional examples

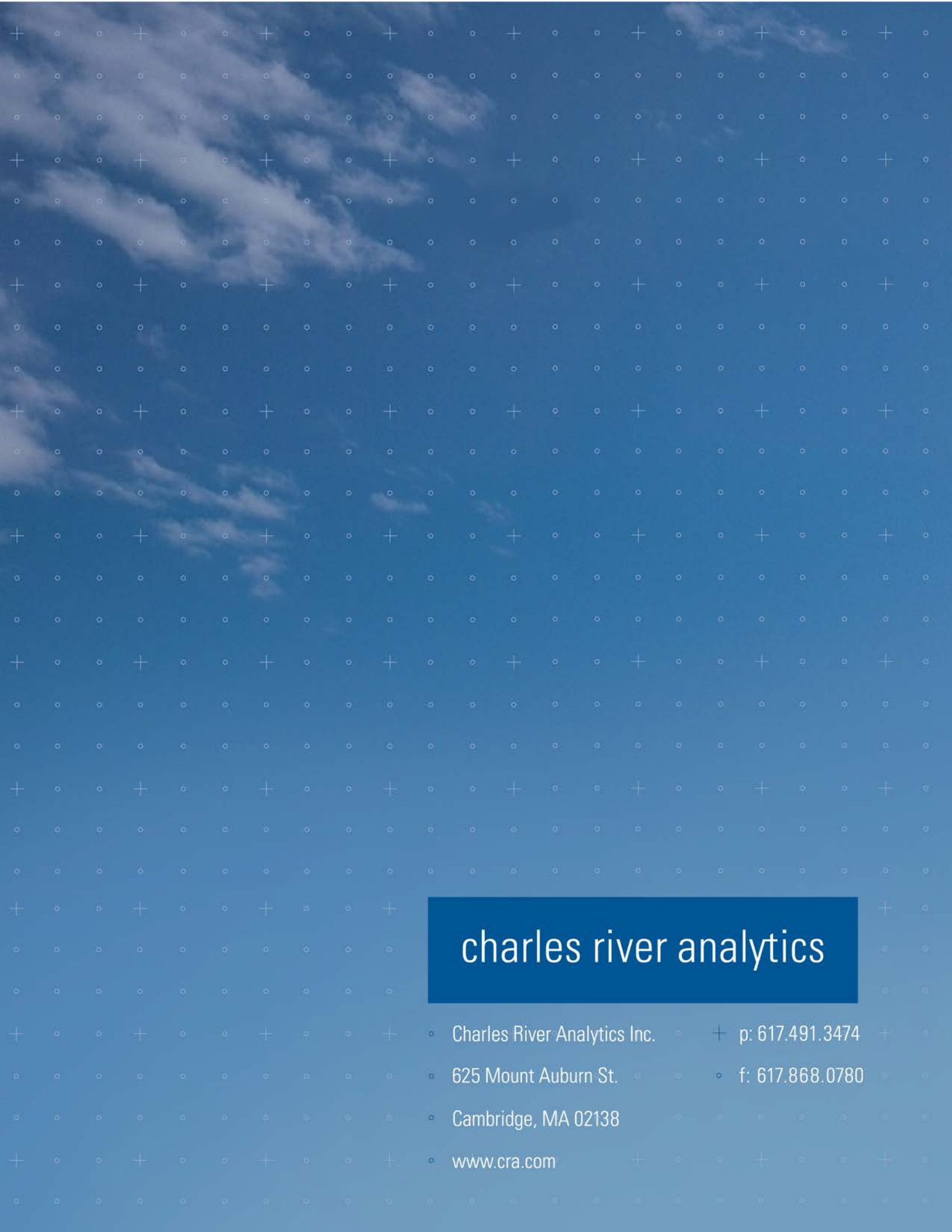
We included a number of example models within the `figaroexamples.jar` file. Many of the examples are described in detail in the *Figaro Tutorial*. They include:

- AnnealingSmokers – Models the likelihood that someone will smoke, based on their friends' smoking habits, using a simulated annealing algorithm.
- Burglary – Models the likelihood that a burglar alarm is due to a burglary using a Bayesian network.
- CarAndEngine – Models the speed of a car based on the power of its engine using a probabilistic relational model (PRM).
- FairDice – Learns the fairness of each die rolled for a data set using Dirichlet parameters.
- Firms – Models firms bidding for a contract and the likelihood that one will be selected as the winner using constraints.
- Hierarchy – Models vehicles and their attributes using a class hierarchy.
- LazyList – Models the likelihood that an infinite list of symbols contains a particular symbol using lazy variable elimination.
- MultiDecision – Models the decision by an entrepreneur to found a company using a multi-decision influence diagram and a Metropolis-Hastings decision algorithm.
- MultiValuedReferenceUncertainty – Models the sum over a container of integers using multi-valued references and aggregates.
- MutableMovie – Models the quality of a movie based on the skill of its actors using a non-functional style of programming, mutable variables, and Figaro collections.
- OpenUniverse – Models the attribution of observations to an unknown number of sources using a Metropolis-Hastings algorithm.
- OpenUniverseLearning – Learns the existence and observation of aircraft using Beta parameters and an Expectation-Maximization algorithm.

- SimpleLearning – Learns the parameters in a set of random data using Beta parameters and an Expectation-Maximization algorithm.
- SimpleMovie – Models the likelihood of an actor receiving an award for their appearance in a movie using a Metropolis-Hastings Markov chain Monte Carlo algorithm.
- Smokers – Models the likelihood that someone will smoke based on their friends' smoking habits using a Markov network/Markov random field.
- Sources – Models the distance between a point and its source using dependent universes.
- ValveReliability – Models the current state of a valve system using a dynamic Bayesian network and the factor frontier algorithm.

To run the Figaro examples

Copy the `figaroexamples.jar` file into the `FigaroWork/lib` directory. (You may need to create this directory.) Then, you can run any of the examples using the SBT `runMain` command.



charles river analytics

Charles River Analytics Inc.
625 Mount Auburn St.
Cambridge, MA 02138
www.cra.com

p: 617.491.3474
f: 617.868.0780