Figaro 2.0 Release Notes

About Figaro

Reasoning under uncertainty requires taking what you know and inferring what you don't know, when what you know doesn't tell you for sure what you don't know. A well established approach for reasoning under uncertainty is probabilistic reasoning. Typically, you create a probabilistic model over all the variables you're interested in, observe the values of some variables, and query others. There is a huge variety of probabilistic models, and new ones are being developed constantly. Figaro is designed to help build and reason with the wide range of probabilistic models.

Developing a new probabilistic model normally requires developing a representation for the model and a reasoning algorithm that can draw useful conclusions from evidence, and in many cases also an algorithm to learn aspects of the model from data. These can be challenging tasks, making probabilistic reasoning require significant effort and expertise. Furthermore, most probabilistic reasoning tools are standalone and difficult to integrate into larger programs.

Figaro is a probabilistic programming language that helps address both these issues. Figaro makes it possible to express probabilistic models using the power of programming languages, giving the modeler the expressive tools to create all sorts of models. Figaro comes with a number of built-in reasoning algorithms that can be applied automatically to new models. In addition, Figaro models are data structures in the Scala programming language, which is interoperable with Java, and can be constructed, manipulated, and used directly within any Scala or Java program.

Figaro is extremely expressive. It can represent a wide variety of models, including:

- directed and undirected models
- models in which conditions and constraints are expressed by arbitrary Scala functions
- models involving inter-related objects
- open universe models in which we don't know what or how many objects exist
- models involving discrete and continuous elements
- models in which the elements are rich data structures such as trees
- models with structured decisions
- models with unknown parameters

Figaro provides a rich library of constructs to build these models, and provides ways to extend this library to create your own model elements.

Figaro's library of reasoning algorithms is also extensible. Current built-in algorithms include:

- Exact inference using variable elimination
- Importance sampling
- Metropolis-Hastings, with an expressive language to define proposal distributions
- Support computation

- Most probable explanation (MPE) using variable elimination or simulated annealing
- Probability of evidence using importance sampling
- Particle Filtering
- Parameter learning using expectation maximization

Figaro provides both regular (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.

Figaro is free and is released under an open-source license (see license file). The public code repository for Figaro can also be found at https://github.com/p2t2

New in Version 2.0

Requirements

- Scala 2.10.x is now required to run Figaro.

New Features

- Figaro now has the ability to represent decisions and contains several algorithms for computing optimal decisions in a model.
- Learning the parameters of elements based on observed evidence has been added to Figaro.
- Expectation maximization algorithm added to Figaro to support parameter learning
- Simulated annealing has been added to compute the most likely state of a model.
- Added basic reflection capability for Figaro elements

Improvements and Changes

- Probability of evidence can now make a distinction between conditions and constraints on elements and additional named evidence applied
- Particle Filtering
    - Now takes an initial universe and a transition function from a universe to universe instead of a stream of universes
    - Improved memory management enables old universes to be cleaned up after the process has moved on to later time steps
- Universe changes
    - Universe now collects conditioned and constrained elements so algorithms do not need to check all elements for satisfaction
    - Calling used and usedBy on elements now computes the element dependencies as needed, but caches the result until the model changes, resulting in more reliable operation and less space usage
    - Removed context control from Universe and moved to the elements themselves

    - Added registration of algorithms on universes so that algorithms can be killed when universe is cleaned up
    - Added the ability to generate a value for all elements in a universe in order
- Chain changes
    - There are no longer two implementations of Chain. Every chain is instantiated with a fixed cache size. Caching chains use a cache of 1000 and non-caching use a cache of 1
    - The type of chain (caching/non caching) is now automatically determined when instantiating from the object Chain
    - Chains now control clearing elements created in their context
    - Two parent chains now changed from nested chains to creating a tuple of the parents into a single parent chain
- Names and references
    - getElement in ElementCollection now changed to getElementByReference, to make clear that the reference is resolved to get a particular reference
    - Reference elements and aggregate elements now work with Variable Elimination
    - Improvements supporting reasoning about objects in class hierarchies
        - Multiple elements with the same name are now allowed. Getting the element by reference returns the most recent one
        - NamedEvidence is applied to all current and future elements with the name
- MakeList
    - Does not inherit from Chain anymore
    - Now works in Variable Elimination
- Setting the seed of the Figaro random number generator now creates deterministic inference
- Elements whose values have been explicitly set or observed do not have their values regenerated

Bug Fixes

- Fixed a bug that prevented factors from taking into account latest evidence when a new factored algorithm is created
- Fixed bug where importance sampling was sampling from inactive elements
- Fixed Metropolis-Hastings bug where it would not run when there were no stochastic elements in the initial state
- Fixed Metropolis-Hastings disable initialization during test mode
- Fixed Metropolis-Hastings bug that was erroneously recording state in non-satisfied conditions
- Fixed bug in CPD where temporary parent tuple not created on the same universe

Examples

- Examples now kill their algorithms when they are done
- Examples added for decisions, parameter learning and hierarchical reasoning