6.945 Project Proposal

Eyal Dechter, Matthew Johnson, Zenna Tavares

Revised April 1, 2013

1 Domain of interest: probabilistic programming

In recent years, multiple probabilistic programming languages have emerged with the goal of merging declarative programming with probabilistic modeling. These languages seek to build abstractions in part so that the task of probabilistic modeling can be separated from the zoo of inference algorithms available. Often these languages either focus on a particular inference algorithm or restricted set of probability distributions (such as STAN [4] or BUGS [3]) or employ exceedingly general inference algorithms so that any probabilistic model can be expressed at the expense of efficiency or even tractability (like Church [1] or BLOG [2]).

We aim to understand the design and implementation of probabilistic programming languages. In particular, we hope to identify abstractions that can allow a fully general probabilistic programming language to leverage special model structures for inference so that its capabilities can be easily extended. We'd also like to understand why no existing probabilistic programming languages seem to attempt such abstractions.

2 A plausible decomposition

1. **Modeling language**: means for expressing probability distributions and mapping those expressions into representations that enable analysis. One approach is to accept expressions that describe the generative process that gives rise to the data (as in Church), but there are other useful distributions to include that are not easily expressed as generative models (such as constraints or Markov random fields).

2. **Analysis**: mechanisms for recognizing known tractable structure when present and choosing good inference algorithms to apply. This process is necessarily heuristic and should be made extensible so that new structures or algorithms can be added.

3. **Inference engines**: implementation of inference algorithms that can operate on the modeling language abstraction. The set of algorithms must also be extensible.

3 Plan for implementation

Probabilistic programming is not well understood and is an area of active research, so it is probable that our understanding of the tasks involved in our project and even our project goals themselves may change as we build the system. Therefore, these are our current best guesses at a plan for implementing something reasonable.

We will jointly design and implement the base system, consisting of a basic representation of models and a generic inference algorithm (such as the Metropolis-Hastings approach in Church). With that common base, here is an assignment of parts:

- **Zenna** will focus at first on improving model representations and methods for automatically identifying tractable classes, either analytically or by inspecting program traces.

- **Eyal** will focus at first on representations of and algorithms for discrete models. Some tractable classes include discrete graphical models with small treewidth, and algorithms include many sampling methods, (loopy) belief propagation, and the junction tree algorithm.
- **Matt** will focus at first on model manipulation and inference for linear Gaussian models. Linear Gaussian inference is extremely powerful, mapping to linear algebraic operations, and can be extended to some nonlinear systems (common in robotics) by applying techniques like automatic differentiation.

References

[1] Noah Goodman, Vikash Mansinghka, Daniel Roy, Keith Bonawitz, and Daniel Tarlow. Church: a language for generative models. *arXiv preprint arXiv:1206.3255*, 2012.

[2] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L Ong, and Andrey Kolobov. 1 blog: Probabilistic models with unknown objects. *Introduction to statistical relational learning*, page 373, 2007.

[3] David Spiegelhalter, Andrew Thomas, Nicky Best, and Wally Gilks. Bugs 0.5: Bayesian inference using gibbs sampling manual (version ii). *MRC Biostatistics Unit, Institute of Public Health, Cambridge, UK*, 1996.

[4] Stan Development Team. Stan: A c++ library for probability and sampling, version 1.1, 2013.