Automatic Derivation of Statistical Algorithms: The EM Family and Beyond

Alexander G. Gray
Carnegie Mellon University
agray@cs.cmu.edu

Bernd Fischer and Johann Schumann
RIACS / NASA Ames
{fisch,schumann}@email.arc.nasa.gov

Wray Buntine
Helsinki Institute for IT
buntine@hiit.fi

Abstract

Machine learning has reached a point where most probabilistic methods can be understood as variations, extensions and combinations of a much smaller set of abstract themes, e.g., as different instances of the EM algorithm. This enables the systematic derivation of algorithms customized for different models. Here, we demonstrate the AUTOBAYES system which takes a high-level statistical model specification, uses powerful symbolic techniques based on *schema-based program synthesis* and *computer algebra* to derive an efficient specialized algorithm for learning that model, and generates executable code implementing that algorithm. This capability is far beyond that of code collections such as Matlab toolboxes or even tools for model-independent optimization such as BUGS for Gibbs sampling: complex new algorithms can be generated without new programming, algorithms can be highly specialized and tightly crafted for the exact structure of the model and data, and efficient and commented code can be generated for different languages or systems. We present automatically-derived algorithms ranging from closed-form solutions of Bayesian textbook problems to recently-proposed EM algorithms for clustering, regression, and a multinomial form of PCA.

1 Deriving Statistical Algorithms by Computer

Overview. We describe a symbolic program synthesis method which acts as a "statistical algorithm compiler:" it can compile a statistical model specification into a custom *algorithm design* and from that further down into a working program implementing the algorithm design. The system implementing this method, AUTOBAYES, can be loosely thought of as "part theorem prover, part Mathematica, part statistics/learning textbook, and part Numerical Recipes." It provides much more than a fixed code repository such as a Matlab toolbox, and allows the creation of efficient algorithms which have never before been implemented, or even written down. The system is intended to automate the more routine application of complex methods in novel contexts. For example, recent multinomial extensions to PCA [2, 4] can be derived in this way.

The algorithm design problem. Given a dataset and a task, creating a learning method can be characterized by two main questions: 1. What is the model? 2. What algorithm will optimize the model parameters? The *statistical algorithm* (i.e., a parameter optimization algorithm for the statistical model) can then be implemented manually. The system in this paper answers the algorithm question given that the user has chosen a model for the data,[1] and continues through to implementation. Performing this task at the state-of-the-art level requires an intertwined meld of probability theory, computational mathematics, and software engineering. However, a number of factors unite to allow us to solve the algorithm design problem computationally: 1. The existence of fundamental building blocks (e.g., standardized probability distributions, standard optimization procedures, and generic data structures). 2. The formalization of a common representation (i.e., graphical models [3, 14] and program *schemas* – to be described). 3. The formalization of schema *guards* (i.e., their known applicability constraints).[2]

The challenges of algorithm design. The design problem has an inherently combinatorial nature, since subparts of a function may be optimized recursively and in different ways. It also involves the use of new data structures or approximations to gain performance. As the research statistical algorithms advances, its creative focus should move beyond the ultimately mechanical aspects and towards extending the abstract applicability of already existing schemas (algorithmic principles like EM), improving schemas in ways that generalize across anything they can be applied to, and inventing radically new schemas.

2 Combining Schema-based Synthesis and Bayesian Networks

Computational logic and theorem proving. Our approach falls into a class of techniques known as computational logic. It has its basis in *logic programming*, as exemplified by the Prolog language (in which AUTOBAYES is implemented), a computationally efficient embodiment of the general idea of *theorem proving* within Horn clause logic. It begins with an initial goal and a set of initial assertions, or axioms, and adds new assertions, or theorems, by repeated application of the axioms, until the goal is proven. In our context, the goal is given by the statistical model; the derived algorithms are side effects of constructive theorems proving the existence of algorithms for the goal.

Computer algebra. The first core element which makes automatic algorithm derivation feasible is the fact that we can mechanize the required symbol manipulation, using computer algebra methods. General *symbolic differentiation* and *expression simplification* are capabilities fundamental to our approach. AUTOBAYES contains a computer algebra engine using term rewrite rules which are an efficient mechanism for substitution of equal quantities or expressions and thus well-suited for this task.[3]

Schema-based synthesis. While attractive from a first-principles point of view, the computational cost of full-blown theorem proving grinds simple tasks to a halt while elementary and intermediate facts are reinvented from scratch. To achieve the scale of deduction required by algorithm derivation, we thus utilize a *schema-based synthesis* technique

[1] Strictly, there is a third question: "3. What mechanism will be used to ensure generalization?" In AUTOBAYES generalization procedures such as cross-validation are implementable as ordinary schemas, and techniques such as likelihood penalization are implementable as additional maximization goals; here we focus on parameter optimization schemas. Learning methods outside the fixed-model format, such as structure learning and feature selection, can be addressed in system extensions using the described capabilities in an inner loop.

[2] Guards vary widely; for example, contrast Nead-Melder simplex or simulated annealing or Gibbs sampling (which require only numeric function evaluation), conjugate gradient (both Jacobian and Hessian), EM and its variational extension [7] (a latent-variable structure).

[3] Popular symbolic packages such as Mathematica contain known errors allowing unsound derivations; they also lack the support for reasoning with vector and matrix quantities.

which breaks away from strict theorem proving. Instead, we effectively encode high-level knowledge, such as the general EM strategy, as *schemas*, or auxiliary lemmas with explicitly specified preconditions. The second core element which makes automatic algorithm derivation feasible is the fact that we can use Bayesian networks to efficiently encode the preconditions of complex algorithms such as EM.

First-order logic representation of Bayesian networks. A theory of indexed Bayesian networks was developed in [8]; here indices are represented as Prolog variables and networks correspond to back-track free datalog programs, allowing the dependencies to be efficiently computed. We have extended these results to work with non-ground probability queries since we seek to determine probabilities over vectors and matrices. Tests for independence on these indexed Bayesian networks are easily developed in Lauritzen's framework which uses ancestral sets and set separation [10] and is more amenable to a theorem prover than the double negatives of the more widely known d-separation criteria. Given a Bayesian network, some probabilities can easily be extracted by enumerating the component probabilities at each node:

Lemma 1. *Let U, V be sets of variables over a Bayesian network with $U \cap V = \emptyset$. Then $V \cap \mathit{descendents}(U) = \emptyset$ and $\mathit{parents}(U) \subseteq V$ hold in the corresponding dependency graph iff the following probability statement holds:*

$$Pr(U \mid V) = Pr(U \mid \mathit{parents}(U)) = \prod_{u \in U} Pr(u \mid \mathit{parents}(u)).$$

Symbolic probabilistic inference. How can probabilities not satisfying these conditions be converted to symbolic expressions? While many general schemes for inference on networks exist, our principal hurdle is the need to perform this over symbolic expressions incorporating real and integer variables from disparate real or infinite-discrete distributions. For instance, we might wish to compute the full *maximum a posteriori* probability for the mean and variance vectors of a Gaussian mixture model under a Bayesian framework. While the sum-product framework of [9] is perhaps closer to our formulation, we have out of necessity developed another scheme that lets us extract probabilities on a large class of mixed discrete and real, potentially indexed variables, where no integrals are needed and all marginalization is done by summing out discrete variables. We give the non-indexed case below; this is readily extended to indexed variables (i.e., vectors).

Lemma 2. *$V \cap \mathit{descendents}(U) = \emptyset$ holds and $\mathit{ancestors}(V)$ is independent of U given V iff there exists a set of variables U' such that Lemma 1 holds if we replace U by $U \cup U'$. Moreover, the unique minimal set U' satisfying these conditions is given by $\mathit{ancestors}(U) \,/ (\mathit{ancestors}(V) \cup V)$.*

Lemma 3. *Let V' be a subset of $V / \mathit{descendents}(U)$ such that $\mathit{ancestors}(V')$ is independent of $(U \cup V)/(V' \cup \mathit{ancestors}(V'))$ given V'. Then Lemma 2 holds if we replace U by $U \cup V/V'$ and V by V'. Moreover, there is a unique maximal set V' satisfying these conditions.*

Lemma 2 lets us evaluate a probability by a summation:

$$Pr(U \mid V) = \sum_{u' \in \mathrm{Dom}(U')} Pr(U' = u', U \mid V)$$

while Lemma 3 lets us evaluate a probability by a summation and a ratio:

$$Pr(U \mid V) = \frac{Pr(U \cup V/V' \mid V')}{Pr(V/V' \mid V')}$$

Since the lemmas also show minimality of the sets U' and V/V', they also give the minimal conditions under which a probability can be evaluated by discrete summation without integration. These inference lemmas are operationalized as network decomposition schemas.

However, we usually attempt to decompose a probability into independent components before applying this schema.

3 The AutoBayes System — Implementation Outline

Levels of representation. Internally, our system uses three conceptually different levels of representation. *Probabilities* (including logarithmic and conditional probabilities) are the most abstract level. They are processed via methods for Bayesian network decomposition or matches with core algorithms such as EM. *Formulae* are introduced when probabilities of the form $Pr(U \mid \text{parents}(U))$ are detected, either in the initial network, or after the application of network decompositions. Atomic probabilities (i.e., U is a single variable) are directly replaced by formulae based on the given distribution and its parameters. General probabilities are decomposed into sums and products of the respective atomic probabilities. Formulae are ready for immediate optimization using symbolic or numeric methods but sometimes they can be decomposed further into independent subproblems. Finally, we use an imperative *intermediate code* as the lowest level to represent both program fragments within the schemas as well as the completely constructed programs. All transformations we apply operate on or between these levels.

Transformations for optimization. A number of different kinds of transformations are available. *Decomposition* of a problem into independent subproblems is always done. Decomposition of probabilities is driven by the Bayesian network; we have a separate system for handling decomposition of formulae. A formula can be decomposed along a loop, e.g. the problem "optimize $\vec{\theta}$ for $\prod_i f(\theta_i)$" is transformed into a for-loop over subproblems "optimize θ_i for $f(\theta_i)$." More commonly, "optimize θ, ϕ for $f(\theta) + g(\phi)$" is transformed into the two subprograms "optimize θ for $f(\theta)$" and "optimize ϕ for $g(\phi)$." The lemmas given earlier are applied to change the level of representation and thus for *simplification of probabilities*. Examples of general *expression simplification* include simplifying the log of a formula, moving a summation inwards, and so on. When necessary, *symbolic differentiation* is performed. In the initial specification or in intermediate representations, *likelihoods* (i.e. subexpressions of the form $log \prod_i Pr(x_i \mid \theta)$) are identified and simplified into linear expression with terms such as *mean*(x_i) and *mean*(x_i^2). The *statistical algorithm schemas* currently implemented include EM, k-means, and discrete model selection. Adding a Gibbs sampling schema would yield functionality comparable to that of BUGS [15]. Usually, the schemas require a particular form of the probabilities involved; they are thus tightly coupled to the decomposition and simplification transformations. For example, EM is a way of dealing with situation where Lemma 2 applies but where U' is indexed identically to the data.

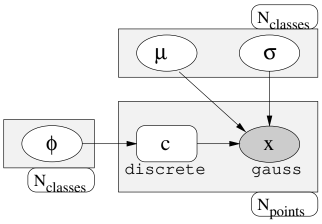
Code and test generation. From the intermediate code, code in a particular *target language* may be generated. Currently, AutoBayes can generate C++ and C which can be used in a stand-alone fashion or linked into Octave or Matlab (as a mex file). During this code-generation phase, most of the vector and matrix expressions are converted into for-loops, and various *code optimizations* are performed which are impossible for a standard compiler. Our tool does not only generate *efficient* code, but also highly readable, *documented* programs: model- and algorithm-specific comments are generated automatically during the synthesis phase. For most examples, roughly 30% of the produced lines are comments. These comments provide explanation of the algorithm's derivation. A generated HTML software design document with navigation capabilities facilitates code understanding and reading. AutoBayes also automatically generates a program for *sampling* from the specified model, so that closed-loop testing with synthetic data of the assumed distributions can be done. This can be done using simple forward sampling since test data needs to be generated along with the "true" model parameters to generate a full test case.

4 Example: Deriving the EM Algorithm for Gaussian Mixtures

1. User specifies model. First, the user specifies the model of interest in a high-level specification language[4] (as opposed to a *programming* language). Note the constraint that the sum of the class probabilities must equal one (line 8) along with others (lines 3 and 5) that make optimization of the model well-defined. Also note the ability to specify assumptions of the kind in line 6, which may be used by some algorithms. The last line specifies the goal inference task: maximize the conditional probability $\mathrm{pr}(\vec{x}|\{\vec{\phi}, \vec{\mu}, \vec{\sigma}\})$ with respect to the parameters $\vec{\phi}$, $\vec{\mu}$, and $\vec{\sigma}$. Note that moving the parameters across to the left of the conditioning bar converts this from a maximum likelihood to a *maximum a posteriori* problem.

```
 1 model mog as 'Mixture of Gaussians';

 2 const int n_points as 'nr. of data points'
 3   with 0 < n_points;
 4 const int n_classes := 3 as 'nr. classes'
 5   with 0 < n_classes
 6   with n_classes << n_points;

 7 double phi(1..n_classes) as 'weights'
 8   with 1 = sum(I := 1..n_classes, phi(I));
 9 double mu(1..n_classes);
 9 double sigma(1..n_classes);

10 int c(1..n_points) as 'class labels';
11 c ~ disc(vec(I := 1..n_classes, phi(I)));

12 data double x(1..n_points) as 'data';
13 x(I) ~ gauss(mu(c(I)), sigma(c(I)));

14 max pr(x|{phi,mu,sigma}) wrt {phi,mu,sigma};
```

2. System parses model to obtain underlying Bayes net. From the model, the underlying Bayesian network (dependency graph) of the model is derived and represented internally as a directed graph. For visualization, AUTOBAYES can also produce a graph drawing as shown in the figure.



3. System observes hidden-variable structure in Bayesian network. First the system attempts to decompose the optimization goal into independent parts, but finds that it cannot. However, it then finds that the probability in the initial optimization statement matches the conditions of Lemma 2.

4. System invokes abstract EM-family schema. This triggers the EM-schema, whose overall structure is shown. The syntactic structure of the current subproblem must match the first argument of the schema; if additional applicability constraints (not shown here) hold, this schema is executed. It constructs a piece of code which is returned in the variable C. This code fragment can contain itself calls to other schemas (denoted by $[\ldots]$) which return code for subproblems which then is inserted into the schema, such as `converging`, a generic convergence criterion here imposed over the variables $\vec{\mu}, \vec{\sigma}, \vec{\phi}$. The system identifies the discrete variable $\vec{c}$ as the single hidden variable, i.e. $W = \{\vec{c}\}$. For representation of the distribution of the hidden variable a matrix $\vec{q}$ is generated, where q_{ij} is the probability that the i-th point falls into the j-th class. AUTOBAYES then constructs the new distribution `c(I) ~ disc(vec(J := 1..n_classes, q(I, J))` which replaces the original distribution in the following recursive calls of AUTOBAYES.

schema(`max` $Pr(U\,|\,V)$`wrt` V, C) :−
 ...
 C = "[initialize W];
   `while`([`converging`(V)]){
    /* M-step */ [`max` $Pr(W, U\,|\,V)$ `wrt` V];
    /* E-step */ [calculate $Pr(W\,|\,U, V)$];
   }"

[4] In this figure, keywords have been underlined and line numbers have been added for reference in the text. The `as`-keyword allows annotations to variables which end up in the generated code's comments. In this example `n_classes` has been set to three (line 4), but `n_points` is left unspecified. The class variable and single data variable (feature) are vectors, which define them as independently and identically distributed.

$$
\begin{array}{l}
\texttt{while(converging(}\vec{\mu}, \vec{\sigma}, \vec{\phi}\texttt{))\{} \\
\quad \texttt{for } i = 1 : N \\
\qquad \texttt{for } j = 1 : C \\
\qquad\quad q_{ij} = Pr(x_i | c_i = j, \vec{\mu}, \vec{\sigma}); \\
\qquad \texttt{max } Pr(x_i, c_i | \mu_{c_i}, \sigma_{c_i}, \vec{\phi}) \texttt{ wrt } \{\vec{\mu}, \vec{\sigma}, \vec{\phi}\} \\
\texttt{\}}
\end{array}
$$

5. E-step: System performs marginalization. The freshly introduced distribution for c_i implies that c_i can be eliminated from the objective function by summing over $q_{i,*}$. This gives us the partial program shown in the internal pseudocode.

$$
\begin{array}{l}
\texttt{while(converging(}\vec{\mu}, \vec{\sigma}, \vec{\phi}\texttt{))\{} \\
\quad \texttt{for } i = 1 : N \\
\qquad \texttt{for } j = 1 : C \\
\qquad\quad q_{ij} = Pr(x_i | c_i = j, \vec{\mu}, \vec{\sigma}); \\
\qquad \texttt{for } j = 1 : C \\
\qquad\quad \texttt{max } \sum_{i=1}^{N} q_{ij} \log Pr(x_i | \mu_j, \sigma_j) \texttt{ wrt } \{\mu_j, \sigma_j\} \\
\qquad \texttt{max } \sum_{j=1}^{C} (\sum_{i=1}^{N} q_{ij}) \phi_j \texttt{ wrt } \{\vec{\phi}\} \\
\texttt{\}}
\end{array}
$$

6. M-step: System recursively decomposes optimization problem. AUTOBAYES is recursively called with the new goal $\texttt{max} \ \log Pr(\{\vec{c}, \vec{x}\} \mid \{\vec{\phi}, \vec{\mu}, \vec{\sigma}\}) \ \texttt{wrt} \ \{\vec{\phi}, \vec{\mu}, \vec{\sigma}\}$. Now, the Bayesian network decomposition schema applies with $U = \{\vec{c}, \vec{x}\}$, $V = \{\vec{\phi}, \vec{\mu}, \vec{\sigma}\}$, revealing that $\vec{\phi}$ is independent of $\vec{\sigma}, \vec{\mu}$, thus the optimization problem can be decomposed into two optimization subproblems: $\texttt{max} \ Pr(\vec{x} \mid \{\vec{c}, \vec{\mu}, \vec{\sigma}\}) \ \texttt{wrt} \ \{\vec{\mu}, \vec{\sigma}\}$ and $\texttt{max} \ Pr(\vec{c} | \vec{\phi}) \ \texttt{wrt} \ \{\vec{\phi}\}$.

7. System unrolls i.i.d. vectors. The first subgoal from the decomposition schema, $\texttt{max} \ Pr(\vec{x} \mid \{\vec{c}, \vec{\mu}, \vec{\sigma}\}) \texttt{wrt} \ \{\vec{\mu}, \vec{\sigma}\}$, can be unrolled over the independent and identically distributed vector $\vec{x}$ using an index decomposition schema which moves expressions out of loops (sums or products) when they are not dependent on the loop index. Since $\vec{c}$ and $\vec{x}$ are co-indexed, unrolling proceeds over both (also independent and identically distributed) vectors in parallel: $\texttt{max} \ \prod_{i=1}^{N} Pr(x_i \mid \{c_i, \vec{\mu}, \vec{\sigma}\}) \texttt{wrt} \ \{\vec{\mu}, \vec{\sigma}\}$.

8. System identifies and solves Gaussian elimination problem. The probability $Pr(x_i \mid \{c_i, \vec{\mu}, \vec{\sigma}\})$ is atomic because $parents(x_i) = \{c_i, \vec{\mu}, \vec{\sigma}\}$. It can thus be replaced by the appropriately instantiated Gaussian density function. Because the strictly monotone $\log(\cdot)$ function can first be applied to the objective function of the maximization, it becomes $\texttt{max} \ \sum_{i=1}^{N} \sum_{j=1}^{C} q_{ij} (-\frac{1}{2\sigma_j}(x_i - \mu_j)^2 - \log \sqrt{2\pi} - \log \sigma_j) \ \texttt{wrt} \ \{\vec{\mu}, \vec{\sigma}\}$. Another application of index decomposition allows solution for the two scalars μ_j and σ_j. Gaussian elimination is then used to solve this subproblem analytically, yielding the sequence of expressions $\mu_j = \sum_{i=1}^{N} q_{ij} x_i / \sum_{i=1}^{N} q_{ij}$ and $\sigma_j = \sum_{i=1}^{N} q_{ij} (x_i - \mu_j)^2 / \sum_{i=1}^{N} q_{ij}$.

9. System identifies and solves Lagrange multiplier problem. The second subgoal $\texttt{max} \ Pr(\vec{c} \mid \vec{\phi}) \ \texttt{wrt} \ \{\vec{\phi}\}$ can be unrolled over the i.i.d. vector $\vec{c}$ as before. The specification condition $\sum_{j=1}^{C} \phi_j = 1$ creates a constrained maximization problem in the vector $\vec{\phi}$ which is solved by an application of the Lagrange multiplier schema. This in turn results in two subproblems for a single instance ϕ_j and for the multiplier which are both solved symbolically. Thus, the usual EM algorithm for Gaussian mixtures is derived.

10. System checks and optimizes pseudocode. During the synthesis process, AUTOBAYES accumulates a number of constraints which have to hold to ensure proper operation of the code (e.g., absence of divide-by zero errors). Unless these constraints can be resolved against the model (e.g., $\sigma_i > 0$), AUTOBAYES automatically inserts a run-time check into the code. Before finally generating C/C++ code, the pseudocode is optimized using information from the specification (e.g., $\sum_{j}^{C} \phi_j = 1$) and the domain. Thus, deep optimizations beyond the capability of a regular compiler can be done.

11. System translates pseudocode to real code in desired language. Finally, AUTOBAYES converts the intermediate code into code of the desired target system. The source code contains thorough comments detailing the mathematics implemented. A regular compiler containing generic performance optimizations not repeated by AUTOBAYES turns the code into an executable program. A program for sampling from a mixture of Gaussians is also produced for testing purposes.

5 Range of Capabilities

In this section, we discuss 18 examples which have been successfully handled by AUTOBAYES, ranging from simple textbook examples to sophisticated EM models and recent multinomial versions of PCA. For each entry, the table below gives a brief description, the number of lines of the specification and synthesized C++ code (loc), and the runtime to generate the code (on a dual-2.2GHz Linux system). Correctness was checked for these examples using automatically-generated test data and hand-written implementations.

Bayesian textbook examples. Simple textbook examples, like Gaussian with simple prior B_1, Gaussian with inverse gamma prior B_2, or Gaussian with conjugate prior B_3 have closed-form solutions. The symbolic system of AUTOBAYES can actually find these solutions and thus generate short and efficient code. However, a slight relaxation of the prior on μ (Gaussian with semi-conjugate prior, B_4) requires an iterative numerical solver.

Gaussians in action. One applied example is a Gaussian change-detection model G_1. A slight extension of the model (toward several features) yields a Gaussian Bayes classifier model G_2. G_2 has been successfully tested on various standard benchmarks [1], e.g., the Abalone dataset. Currently, the number of expected classes has to be given in advance.

Mixture models and EM. A wide range of k-Gaussian mixture models can be handled by AUTOBAYES, ranging from the simple 1D (M_1) and 2D with diagonal covariance (M_2) to 1D models for multi-dimensional classes M_3 and with (conjugate) priors on mean M_4 or variance M_5. Using only a slight variation in the specification, the Gaussian distribution can be replaced by other distributions (e.g., exponentials, M_6, for failure analysis) or combinations (e.g.. Gaussian and Beta, M_7, or k-Cauchy and Poisson M_8). In the algorithm generated by M_7, the analytic subsolution for the Gaussian case is combined with the numerical solver. Finally, M_9 is a k_1-Gaussians and k_2-Gaussians two-level hierarchical mixture model which is solved by a nested instantiation of EM [16]: i.e. the M-step of the upper mixture's EM algorithm is a second EM algorithm nested inside the first.

Mixtures for Regression. We represented regression with Gaussian error and Legendre polynomials with full conjugate priors allowing smoothing (e.g., [11]). Two versions of this were then done: robust linear regression R_1 replaces the Gaussian error with a mixture of two Gaussians (one broad, one peaked) both centered at zero. Trajectory clustering R_2 replaces the single regression curve by a mixture of several curves [6]. In both cases an EM algorithm is correctly integrated with the exact regression solutions.

Principal Component Analysis. We represented the multinomial version of PCA called latent Dirichlet allocation [2]. AUTOBAYES currently lacks variational support, yet it manages to combine a k-means style outer loop on the component proportions with an EM-style inner loop on the hidden counts, producing the original algorithm of Hofmann, Lee and Seung, and others [4].

#	Description	loc	T_s	#	Description	loc	T_s
B_1	$\mu \sim \mathrm{N}(\mu_0, \tau_0^{0.5})$ σ^2	12/137	0.2	B_2	μ $\sigma^2 \sim \Gamma^{-1}(\frac{\delta_0}{2}+1, \frac{\delta_0}{2}\sigma_0^{.5})$	13/148	0.2
B_3	$\mu \sim \mathrm{N}(\mu_0, (\frac{\sigma^2}{\kappa_0})^{0.5})$ $\sigma^2 \sim \Gamma^{-1}(\frac{\delta_0}{2}+1, \frac{\delta_0}{2}\sigma_0^{.5})$	16/188	0.4	B_4	$\mu \sim \mathrm{N}(\mu_0, \tau_0)$ $\sigma^2 \sim \Gamma^{-1}(\frac{\delta_0}{2}+1, \frac{\delta_0}{2}\sigma_0^{.5})$	17/233	0.4
G_1	Gauss step-detect	19/662	2.0	G_2	Gauss Bayes Classify	58/1598	4.7
M_1	k-Gauss mix 1D	17/418	0.7	M_2	k-Gauss mix 2D, diag	22/599	1.2
M_3	–"–, multi-dim	24/900	1.1	M_4	–"– 1D, μ prior	25/456	1.0
M_5	–"–, σ prior	21/442	0.9	M_6	k-Exp mix	15/347	0.5
M_7	Gauss/Beta mix	22/834	1.7	M_8	k-Cauchy/Poisson mix	21/747	1.0
M_9	k_1, k_2-Gauss hierarch	29/1053	2.3				
R_1	rob. lin. regression	54/1877	14.5	P_1	PCA mult/w k-means	26/390	1.2
R_2	mixture regression	53/1282	9.8				

6 Conclusion

Beyond existing systems. Code libraries are common in statistics and learning (e.g. Matlab, S), but they lack the high level of automation achievable only by deep symbolic reasoning. The Bayes Net Toolbox [13] is a Matlab library which allows users to program in models but does not derive algorithms or generate code. The BUGSsystem [15] also allows users to program in models but is specialized for Gibbs sampling. The stochastic parametrized grammars of Mjolsness and Turmon [12] allow a concise model specification similar to AUTOBAYES's specification language, but are currently only a notational device similar to XML. The system of Ellman and Murata [5] also synthesizes programs from a high-level specification, but for the domain of differential equation modeling.

Benefits of automated algorithm and code generation. *Industrial-strength code.* Code generated by AUTOBAYES is efficient, validated, and commented. *Extreme applications.* Extremely complex or critical applications such as spacecraft challenge the reliability limits of human-developed software. Automatically generated software allows for pervasive condition checking and correctness-by-construction. *Fast prototyping and experimentation.* For both the data analyst and machine learning researcher, AUTOBAYES can function as a powerful experimental workbench. *New complex algorithms.* Even with only the few elements implemented so far, we showed that algorithms encroaching on research-level results [11, 6, 16, 4] can be automatically derived. As more distributions, optimization methods and generalized learning algorithms are added to the system, an exponentially-growing number of complex new algorithms become possible, including non-trivial variants which may challenge any single researcher's particular algorithm design expertise.

Future agenda. The ultimate goal is to give researchers the ability to experiment with the entire space of complex models and the state-of-the-art statistical algorithms needed for them, and to allow new algorithmic ideas, as they appear, to be implicitly generalized to every model and special case known to be applicable. We have already begun work on generalizing the EM schema to continuous hidden variables, as well as adding schemas for variational methods, fast *kd*-tree and N-body algorithms, MCMC, and temporal models.

References

[1] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[2] D. Blei, A.Y. Ng, and M. Jordan. Latent Dirichlet allocation. To appear in *NIPS*14*, 2002.

[3] W.L. Buntine. Operations for learning with graphical models. *JAIR*, 2:159–225, 1994.

[4] W.L. Buntine. Variational extensions to EM and multinomial PCA. To appear in *ECML 2002*.

[5] T. Ellman and T. Murata. Deductive synthesis of numerical simulation programs from networks of algebraic and ordinary differential equations. *Automated Software Engineering*, 5(3):291–319, 1998.

[6] G.S. Gaffney and P. Smyth. Trajectory clustering using mixtures of regression models. In *5th KDD*, pp. 63–72, 1999.

[7] Z. Ghahramani and M.J. Beal. Propagation algorithms for variational Bayesian learning. In *NIPS*12*, pp. 507–513, 2000.

[8] P. Haddawy and A.M. Frisch. Modal logics of higher-order probability. In *UAI 10*, 1994.

[9] F. R. Kschischang, B. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Trans. Inform. Theory*, 47(2):498–519, 2001.

[10] S.L. Lauritzen, A.P. Dawid, B.N. Larsen, and H.-G. Leimer. Independence properties of directed Markov fields. *Networks*, 20:491–505, 1990.

[11] D.J.C. Mackay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1991.

[12] E. Mjolsness and M. Turmon. Stochastic parameterized grammars for Bayesian model composition. In *NIPS*2000 Workshop on Software Support for Bayesian Analysis Systems*, Breckenridge, December 2000.

[13] K. Murphy. Bayes Net Toolbox 2.0 for Matlab 5, 2000. Internet Report.

[14] P. Smyth, D. Heckerman, and M. Jordan. Probabilistic independence networks for hidden Markov models. *Neural Computation*, 9(2):227–269, 1997.

[15] A. Thomas, D.J. Spiegelhalter, and W.R. Gilks. BUGS: A program to perform Bayesian inference using Gibbs sampling. In *Bayesian Statistics 4*, pp. 837–842, 1992.

[16] D.A. van Dyk. The nested EM algorithm. *Statistica Sinica*, 10:203-225, 2000.