The AutoBayes Program Synthesis System — System Description —

Bernd Fischer[†], Thomas Pressburger[‡], Grigore Rosu[†], and Johann Schumann[†]

[†]RIACS / [‡]Code IC, NASA Ames Research Center, M/S 269-2
Moffett Field, CA 94035 USA,
{fisch,ttp,grosu,schumann}@ptolemy.arc.nasa.gov

1 Introduction

AutoBayes is a fully automatic program synthesis system for the statistical data analysis domain. Its input is a concise description of a data analysis problem in the form of a statistical model; its output is optimized and fully documented C/C++ code which can be linked dynamically into the Matlab and Octave environments. AutoBayes synthesizes code by a schema-guided deductive process. Schemas (i.e., code templates with associated semantic constraints) are applied to the original problem and recursively to emerging subproblems. AutoBayes complements this approach by symbolic computation to derive closed-form solutions whenever possible. In this paper, we concentrate on the interaction between the symbolic computations and the deductive synthesis process; a detailed description of AutoBayes can be found in [FSP00,FS01].

A statistical model specifies for each problem variable (i.e., data or parameter) its properties and dependencies in the form of a probability distribution. A typical data analysis task is to estimate the best possible parameter values from the given observations or measurements. The following example models normal-distributed data but takes prior information (e.g., from previous experiments) on the data's mean value and variance into account.

```
 1 model normal as 'Normal model with conjugate priors'.
 2 const double kappa_0, mu_0.
 3   where 0 < kappa_0.
 4 double mu ~ gauss(mu_0, sqrt(sigma_sq/kappa_0)).
 5 const double sigma_0_sq, delta_0.
 6   where 0 < sigma_0_sq and 0 < delta_0.
 7 double sigma_sq ~ invgamma(delta_0/2+1, sigma_0_sq*(delta_0/2)).
 8 const nat n_points.
 9   where 0 < n_points.
10 data double x(0..n_points-1) ~ gauss(mu, sqrt(sigma_sq)).
11 max pr({x, mu, sigma_sq}) for {mu, sigma_sq}.
```

Here, lines 8–10 describe the data properties: `x` is a vector of `n_points` real-valued observations that are independently drawn from a normal or Gaussian distribution with unknown mean `mu` and variance `sqrt(sigma_sq)`. Lines 2–4 specify the prior information on `mu`, which is itself drawn from a normal distribution. This prior summarizes a number of previous experiments, where `mu` turned out to be `mu_0` on average. Similarly, lines 5–7 specify the prior on `sigma_sq`. Lines 2, 5, and 8 declare constants to represent the model parameters; lines 3, 6, and 9 state constraints on their allowed values. Finally, line 11 comprises the proper analysis task: given the data `x`, find the values for `mu` and `sigma_sq` which maximize the joint probability `pr({x, mu, sigma_sq})`—in other words, find the values for `mu` and `sigma_sq` which explain the observed data `x` in the statistically best possible way.

Graphical models [Bun94] are a uniform framework in which many typical data analysis problems, e.g., data compression [Fre98] or image restoration [Kok98], can be formulated as similar *parameter learning* problems. AutoBayes uses graphical models to represent models internally and to guide the decomposition of the statistical learning task into simpler, independent subtasks.

2 System Architecture

Program genreation proceeds in a number of distinct stages that are reflected in AutoBayes's system architecture. In a preprocessing step, the given specification is parsed and converted into the internal graphical model form. The *synthesis kernel* then analyzes the model and tries to solve the given optimization task. It instantiates appropriate algorithm schemas which are given in a library and produces a procedural program in AutoBayes's intermediate language. This code is optimized and finally converted into the language of the target system. The synthesized code is fully documented; assumptions and proof obligations which have not been discharged during synthesis are laid out clearly in the documentation or are converted into runtime assertions.

The entire system is implemented in SWI-Prolog [Wie98] and comprises about 24,000 lines of documented code. SWI-Prolog proved to be a very stable and efficient development platform with reasonably good debugging facilities.

Synthesis Kernel. Synthesis is performed by exhaustive, layered application of *schemas*. A schema consists of a program fragment with open slots and a set of applicability conditions. The slots are filled in with code pieces by the synthesis kernel. The conditions constrain how the slots can be filled; they must be discharged (i.e., proven to hold in the given model) before the schema can be applied. Conditions can also be described by specific network patterns; checking then proceeds efficiently by pattern matching. This allows the network structure to guide the application of the schemas and thus to prevent combinatorial explosion of the search space, even if a large number of schemas are applicable.

AutoBayes currently comprises four different layers of schemas; schemas can easily be added without restructuring the system. Network decomposition schemas try to break down the network into independent subnets, based on independence theorems for graphical models. The emerging subnets are fed back into the synthesis process and the resulting programs are composed to achieve a program for the original problem. AutoBayes is thus able to automatically synthesize large programs by composition of different schemas. Formula and vector decomposition schemas work on products of conditional probability distributions. The application of these schemas is also guided by the network structure but they require more substantial symbolic computations. The skeleton of the synthesized code is generated by the application of statistical algorithm schemas. AutoBayes currently implements two such schemas, the EM-algorithm and k-Means (i.e., nearest neighbor clustering). After this last network-oriented layer, the statistical problem has been transformed into an ordinary optimization problem. If AutoBayes cannot find a symbolic solution for this problem, it applies standard numeric optimization methods. AutoBayes currently provides schemas for the Newton-Raphson and Nelder-Mead simplex algorithms. These schemas are instantiated with the function to be optimized. In contrast to using a library function, this open approach allows further symbolic simplifications and optimizations.

Symbolic Subsystem. The main task of this subsystem is to find symbolic solutions to optimization problems. This daunting task, however, is simplified substantially by the relatively uniform structure of the optimization problems which allows implementing powerful heuristics.

At the core of the symbolic subsystem is a small but reasonably efficient AC-rewrite engine implemented in Prolog. Since a rewrite system for this engine is implemented naturally as a Prolog-predicate, conditional rewriting comes "for free." Moreover, the rule clauses can access explicit assumptions; hence, AutoBayes allows conditional rules as for example $x/x \rightarrow_{\models\, x\neq 0} 1$ where $\rightarrow_{\models\, x\neq o}$ means "rewrites to, provided $x \neq 0$ can be proven from the current assumptions." The assumptions are managed almost transparently by the rewrite engine; the rewrite system only needs to contain the non-congruent propagation rules which modify the assumptions under which subterms are rewritten, e.g., $\texttt{if}\ p\ \texttt{then}\ s\ \texttt{else}\ t\ \texttt{fi} \rightarrow_{\models A} \texttt{if}\ p\downarrow_{\models A}\ \texttt{then}\ s\downarrow_{\models A\wedge p}\ \texttt{else}\ t\downarrow_{\models A\wedge\neg p}\ \texttt{fi}$ where $t\downarrow_{\models A}$ is the normal form of t under the assumptions A.

Expression simplification and symbolic differentiation are implemented on top of the rewrite engine. The basic rules are straightforward; however, vectors and matrices introduce the usual aliasing problems and require careful formalizations. For example, as the index values $\boldsymbol{i}$ and $\boldsymbol{j}$ are usually unknown at synthesis time, the partial derivative $\partial x_{\boldsymbol{i}}/\partial x_{\boldsymbol{j}}$ can only be rewritten into $\texttt{if}\ \boldsymbol{i}=\boldsymbol{j}\ \texttt{then}\ 1\ \texttt{else}\ 0\ \texttt{fi}$. More advanced rules, however, require explicit meta-programming, especially when bound variables are involved.

Abstract interpretation is used as an efficient mechanism to evaluate range constraints such as $x > 0$ or $x \neq 0$ which occur in the conditions of many rewrite rules. AutoBayes implements as a rewrite system a domain-specific refinement of the standard sign abstraction where numbers are not only abstracted into *pos* and *neg* but also into *small* (i.e., $|x| < 1$) and *large*.

It then turns out out that a relatively simple solver built on top of this core system is already sufficient. AutoBayes thus essentially relies on a low-order polynomial (i.e., linear, quadratic, and simple cubic) symbolic solver. However, it also shifts and normalizes exponents, recognizes multiple roots and bi-quadratic forms, and tries to find polynomial factors. It also handles expressions in x and $(1 - x)$ which are common in Bernoulli models.

3 Experimental Results

We have applied AutoBayes to a number of advanced textbook examples, machine learning benchmarks, and NASA applications. Table 1 summarizes the results. *cfs* indicates whether a closed-form solution exists and, if so, whether it was found by AutoBayes. The remaining columns give the size of the specification and the respective number of lines of generated Octave/C++ code, as well as synthesis and compilation (`g++ -O2`) times on a Sun Ultra 60.

#	Description (priors)	cfs	lines of spec	lines of C++	$T_{synth}[s]+$ $T_{compile}$[s]
N_1	$\mu \sim N(\mu_0, \tau_0^{0.5}), \sigma^2$	Y/Y	8	99	$1.5 + 7.1$
N_2	μ, $\sigma^2 \sim \Gamma^{-1}(\delta_0/2+1, \sigma_0^{0.5}\delta_0/2)$	Y/Y	9	99	$2.0 + 8.8$
N_3	$\mu \sim N(\mu_0, (\sigma^2/\kappa_0)^{0.5})$, $\sigma^2 \sim \Gamma^{-1}(\delta_0/2+1, \sigma_0^{0.5}\delta_0/2)$	Y/Y	12	126	$8.9 + 7.7$
N_4	$\mu \sim N(\mu_0, \tau_0)$, $\sigma^2 \sim \Gamma^{-1}(\delta_0/2+1, \sigma_0^{0.5}\delta_0/2)$	N/N	12	478	$14.6 + 20.0$
M_1	1D Gaussian mixture	N/N	16	389	$11.7 + 12.4$
M_2	2D Gaussian mixture (x, y uncorrelated)	N/N	22	536	$19.6 + 19.7$
M_3	1D Gaussian mixture (multi-dim. classes)	N/N	24	519	$18.1 + 16.7$
M_4	exponential mixture (simple failure analysis)	N/N	15	321	$6.4 + 10.0$
M_5	disjoint mixture (binomial + Poisson)	N/N	21	425	$19.5 + 11.9$
SD	step detection	N/N	14	1206	78.0 + 49.4
AB	Abalone classifier	N/N	58	1310	63.5 + 139.1
GR	γ-ray burst analysis	N/N	12	475	3.9 + 9.5

Table 1. List of examples

The examples N_1 to N_4 describe different estimation problems for normal distributions where N_3 is the example in Section 1. N_1 and N_2 are simpler versions where prior information is specified only for the mean or the variance of the data. These are advanced textbook examples [GC+95] and AutoBayes finds (almost) exactly the closed form textbook solutions. N_4 slightly generalizes the form of the prior for μ. However, this seemingly small modification renders the symbolic problem unsolvable. AutoBayes thus generates executable code by instantiation of an iterative numeric solver. This example shows that a purely symbolic system is not sufficient in practice.

The remaining examples all require the application of iterative algorithms. The examples M_1 to M_5 are all solved via the EM algorithm schema but each example induces a different symbolic maximization problem. However, after symbolic differentiation, these subproblems are reduced to essentially linear or quadratic equations which are easily solved by AutoBayes. *SD* is a simple time series model to detect a change of means in a Gaussian process; *AB* is a classifier for abalone mussels (`http://www.ics.uci.edu/~mlearn/MLRepository.html`). Finally, *GR* is a model to detect γ-rays bursts from the BATSE radio source (`http://cossc.gsfc.nasa.gov/batse`).

4 Conclusions

The tight combination of schema-guided synthesis, deduction, and symbolic computation in AutoBayes is essential to generate efficient code. Symbolic computation is used for simplification and for finding symbolic solutions if they exist. However, we can only synthesize a correct program from a specification when we can rely on the soundness of the symbolic machinery. This in particular means that all transformations have to be performed with respect to the proper assumptions, like an expression being non-zero. Transformations can also give rise to new proof obligations, e.g., showing that a possible solution is the minimum and not just a saddle point. AutoBayes keeps track of all assumptions and either discharges them during synthesis or generates assertions to be checked during runtime. The importance for symbolic calculation under assumptions and the possible unsoundness of a commercial symbolic algebra system like Mathematica led us to develop our own symbolic subsystem on top of Prolog.

Although we have been able to synthesize code for various non-trivial textbook examples, AutoBayes's code generating capabilities for a variety of statistical models need to be extended substantially. Besides adding further algorithm schemas for statistical computations and for general numerical optimization, improvement of the symbolic subsystem is of major importance. The power and generality of the equation solver will need to be enhanced. Furthermore, for marginalization in statistical models, symbolic handling of (relatively) simple integrals is important. Each enhancement in the symbolic subsystem will lead to improvement of the synthesized code as more subtasks can be solved in closed form rather than being approximated by (slower) numerical algorithms. In all cases, AutoBayes ensures correctness of the synthesized code with respect to the specification by generating the appropriate runtime assertions and documentation.

References

[Bun94] W. L. Buntine. "Operations for learning with graphical models". *J. AI Research*, **2**:159–225, 1994.

[Fre98] B. J. Frey. *Graphical Models for Machine Learning and Digital Communication*. MIT Press, Cambridge, MA, 1998.

[FS01] B. Fischer and J. Schumann. AutoBayes: A System for Generating Data Analysis Programs from Statistical Models, 2001. submitted for publication.

[FSP00] B. Fischer, J. Schumann, and T. Pressburger. "Generating Data Analysis Programs from Statistical Models (Position Paper)". In W. Taha, (ed.), *Proc. Intl. Workshop Semantics Applications, and Implementation of Program Generation*, *Lect. Notes Comp. Sci.* **1924**, pp. 212–229, Montreal, Canada, September 2000. Springer.

[GC+95] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis*. Texts in Statistical Science. Chapman & Hall, 1995.

[Kok98] A. C. Kokaram. *Motion Picture Restoration*. Springer, Berlin, 1998.

[Wie98] J. Wielemaker. *SWI-Prolog 3.1 Reference Manual, Updated for Version 3.1.0 July, 1998*. Amsterdam, 1998.