Automated Synthesis of Statistical Data Analysis Programs

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

Abstract

Program synthesis is the systematic, usually automatic construction of correct and efficient executable code from declarative specifications. Its main advantage over other software development approaches is that it can make the overall process faster and more reliable, since validation and maintenance can be done on the specification level rather than the code level. In this paper we present AutoBayes, 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. We have applied AutoBayes to a number of advanced textbook examples, machine learning benchmarks, and NASA applications. These problems are taken from a variety of classes including classification, clustering and change-point detection.

1 Introduction

Statistical data analysis is a core activity in experimental sciences. It encompasses a wide variety of tasks, ranging from, e.g., a simple linear regression to fitting complex dynamical models. Developing statistical data analysis programs, however, is an arduous and error-prone process which requires profound expertise in different areas: statistics, numerics, software engineering, and of course the scientific application domain.

Automated program synthesis is a formal approach to software development. A synthesis system takes a high-level specification as its input and produces executeable code. On a very high level, a synthesis system has many similarities with a compiler. However, the conceptual gap between its input (i.e., the specification) and its output (i.e., the code) is substantially larger, because the generation process involves instantiation of algorithms, logic reasoning, and symbolic calculation. A synthesis system must thus encode considerable domain knowledge; during the synthesis process, the specification guides the application of this domain knowledge by the usually logic-based synthesis engine to construct the program.

We believe that statistical data analysis is a very promising domain for the application program synthesis, despite—or even because—the aforementioned difficulties. Statistics provides a unifying and concise domain-specific notation. Graphical models [3] provide a structuring mechanism which can be exploited during the syn-

thesis process, e.g., to decompose a problem into independent subproblems. Statistical algorithms like EM [4] are often applicable to a wide range of problems; their generic formulations allow a "plug'n'play"-style algorithm combination. Recently developed sophisticated data structures as for example *kd*-trees [15] offer orders-of-magnitude speed-up for certain problems but are rarely employed due to the increased programming complexity they cause. Finally, data analysis is characterized by an iterative development style: an initial model is hypothesized, implemented, evaluated on real data, and—if necessary—refined. However, each iteration typically involves substantial programming efforts as prototyping is often not sufficient to work with real data sets and even small model modifications may require radically dfferent algorithms. Program synthesis encapsulates many of the statistical, numerical, and software engineering aspects of each iteration and thus allows users to concentrate on their scientific application. Its fast turn-around times make model refinement and design-space exploration feasible.

In this paper, we briefly describe AUTOBAYES, an automatic program synthesis tool for data analysis programs. In the following section we will illustrate the specification language with an example (random walk) and will describe AUTOBAYES's system architecture. Section 3 contains an overview of typical problems solved by AUTOBAYES and Section 4 concludes. For further details on AUTOBAYES we refer to [6, 7].

2 System Architecture

2.1 An Example Specification

As a running example we use the model for a random walk (i.e., a simple noisy dynamical process) with a step, i.e., at some unknown point in time the rate changes quickly, e.g., after a failure of the measurement equipment. Figure 1 shows the specification in AUTOBAYES's input notation. The last two statements are the core of

```
model walk as 'Random Walk w/ Step'.
const nat n_points.
  where 3 < n_points.
const nat switch_point.
  where switch_point in 2..n_points-2.
double rate       as 'drift rate / unit'.
double fail_rate  as '-"- after step'.
double error      as 'drift error / unit'.
  where 0 < error.
data double x(0..n_points-1).
x(I) ~ gauss(cond(I < switch_point,
                  cond(I>0,x(I-1),0)+rate,
                  cond(I>0,x(I-1),0)+fail_rate),
             error).
max pr(x|{switch_point,rate,fail_rate,error})
for {switch_point,rate,fail_rate,error}.
```

Figure 1: AUTOBAYES specification of random walk with abrupt change in rate.

the specification; the remaining statements just declare the model variables and impose some additional constraints on them. The distribution statement `x(I) ~ ...` characterizes the observed data x: each observation x_i is normal (or Gaussian) distributed with a gradually changing mean and constant but unknown error. All mean value are expected to depend on the previous observation x_{i-1} and the unknown drift rates. Over time, the data points thus drift away from their origin, which is here modeled as zero. However, at some unknown index position *switch_point*, the change in the mean suddenly switches from *rate* to *fail_rate*. The last statement of the specification characterizes the data analysis task: find the values for the unknown parameters *switch_point*, *rate*, *fail_rate* and *error* which explain best the known data x, i.e.,

maximize the data probability under the parameter values and model distributions.

2.2 Architecture Overview

AUTOBAYES's overall system architecture is shown in Figure 2. In a first processing step, the given specification is parsed and converted into internal form and a Bayesian network representing the model is constructed. Then the *synthesis kernel* analyzes the network, tries to solve the given optimization task, and instantiates appropriate algorithm schemas which are given in a schema library. The schemas encode the domain knowledge. They contain rules to decompose the network into independent parts, rules to search symbolically for closed-form solutions, and algorithm skeletons which are instantiated during the synthesis process. These schemas are guarded by applicability conditions and may be called recursively. Thus, the synthesized program can be assembled from various (and different) parts and algorithms. The output of the synthesis kernel is a program in a procedural intermediate language. AUTOBAYES's backend takes this intermediate code, optimizes it and generates code for the chosen target system. Currently, we support Octave [13], Matlab, and stand-alone C, but only small parts of the code generator are system-specific; new target systems can thus be added easily.

Certification procedures for high-quality data analysis code often mandate manual code inspections. These require that the code is readable and well documented. Human understandability is a strong requirement, even for programs not subject to these procedures, as manual code modifications are often necessary, e.g., for performance tuning or system integration. However, existing program generators often produce

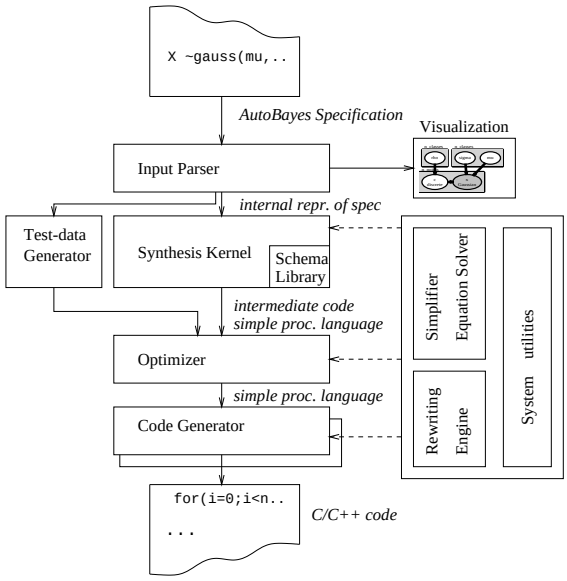

Figure 2: System architecture of AUTOBAYES.

code that is hard to read and understand. In order to overcome this problem, AUTOBAYES generates thoroughly documented code: approximately one third of the output lines are automatically generated comments. These comments contain explanations of the crucial "synthesis decisions", e.g., which algorithm schema has been used. Also, model assumptions and proof obligations that could not be discharged during the synthesis are laid out clearly. In future versions of AUTOBAYES we will extend this to produce detailed, standardized design documents along with the generated code. Furthermore, AUTOBAYES can generate code which generates artificial data for the model, e.g., for visualization, simulation, and testing purposes. The entire AUTOBAYES system has been implemented in SWI-Prolog and comprises about 31,000 lines of documented Prolog code.

3 Examples

We have applied the AUTOBAYES system to a number of different textbook and benchmark examples. The results of these experiments are shown in Table 1. For each problem, a short description of the task or the used priors is given. The next two columns give the size of the specification and the respective number of lines of generated Octave/C++ code, including the automatically generated comments and interface code to Octave. Finally, the synthesis time T_{syn} (i.e., AUTOBAYES's runtime) as well as the compilation time T_{com} for the GNU `g++` compiler (optimization level `-O2`) are given. All times are in seconds and have been obtained on a Sun Ultra 60 (400 Mhz) using the Unix `time` command.

The examples G_1 to G_4 describe different estimation problems for Gaussian distributions. Given a sample of n data points and various prior information (e.g., the variance of the distribution and an estimate of the mean value), the task is to estimate the remaining parameters of the distribution. For several of these textbook examples (G_1, G_2, G_3) closed-form solutions exist [8] and are found by AUTOBAYES, which demonstrates the capabilities of its symbolic system. It is interesting to note that G_4 produces roughly four times as much code as G_3 although the specifications are very similar—only the prior on σ is generalized. This code blow-up is caused by the fact that for G_4 no closed-form solution exists and thus an iterative numerical algorithm has to be used. We have also been able to synthesize code for a large number of mixture problems. M_1 is the standard Mixture of Gaussians problem with a known number of classes. M_2 is a variation of the Mixture of Gaussians problem for uncorrelated two-dimensional observations, and M_3 one for hidden variables composed from multiple independent dimensions. M_4 is a one-dimensional mixture problem on exponentially distributed data which comprises a simple model useful for failure analysis; M_5 is a disjoint mixture model, combining binomial and Poisson data.

#	Description (priors)	lines of spec/C++		$T_{syn}[s]+$ $T_{com}[s]$
G_1	$\mu \sim \mathrm{N}(\mu_0, \tau_0^{0.5}), \sigma^2$	12	99	$1.5 + 7.1$
G_2	μ, $\sigma^2 \sim \Gamma^{-1}(\delta_0/2 +1, \sigma_0^{0.5}\delta_0/2)$	13	99	$2.0 + 8.8$
G_3	$\mu \sim \mathrm{N}(\mu_0, \sqrt{\sigma^2/\kappa_0})$ $\sigma^2 \sim \langle$see $G_2\rangle$	17	126	$8.9 + 7.7$
G_4	$\mu \sim \mathrm{N}(\mu_0, \tau_0)$, $\sigma^2 \sim \langle$see $G_2\rangle$	17	478	$14.6 + 20.0$
M_1	1D Gaussian mix	16	389	$11.7 + 12.4$
M_2	2D Gaussian mix	22	536	$19.6 + 19.7$
M_3	1D Gaussian mix	24	519	$18.1 + 16.7$
M_4	exponential mix	15	321	$6.4 + 10.0$
M_5	disjoint mixture	21	425	$19.5 + 11.9$
M_6	1D mix, σ prior	20	401	$15.4 + 15.0$
M_7	1D mix, μ prior	24	424	$18.2 + 16.5$
A	Abalone	58	1310	$63.5 + 139.1$
W	Wines	91	2045	$86.0 + 124.0$
I	Iris	41	827	$14.0 + 34.9$
D	Digits	36	674	$10.1 + 23.8$
R	Rocks	83	1987	$64.2 + 132.5$
S_1	step detection	14	473	$9.8 + 10.5$
S_2	step detection σ	22	453	$13.7 + 21.0$
C	Coal Mining	12	445	$5.5 + 8.8$
W_1	random walk	11	122	$1.8 + 5.6$
W_2	-"- w/rate-step	21	1103	$23.4 + 82.4$
W_3	-"- w/error-step	21	517	$18.7 + 34.0$
JM	Jelinski/Moranda	15	557	$3.6 + 6.9$
SW	Schick/Wolverton	14	560	$3.8 + 8.0$

Table 1: List of examples

Both M_6 and M_7 are similar to M_1, but here the conjugate priors for μ, and σ, respectively, are assumed in addition. These examples

demonstrate AUTOBAYES's capability to synthesize code for classical maximum-likelihood problems (i.e., without priors) as well as for maximum aposteriori (i.e., Bayesian inference) problems.

The examples A, W, I, R are applications of mixture models to well-known benchmark examples mostly taken from UCI Machine Learning Repository [2]. A is the Abalone mussel classifier, W a classification of Italian wines, I the classical Iris-flower example, and D classification of handwritten digits using morphological features [11]. Finally, R is a classification of rocks according to spectral data [14].

All mixture problems are solved by different instantiations of the EM-schema; however, the different distributions give rise to different maximization problems in the M-step. An efficient implementation requires the symbolic solution of the emerging maximization problem. AUTOBAYES' symbolic system is already powerful enough to provide such solutions for the distributions from the exponential family, including the binomial, exponential (M_4), Gaussian, and Poisson distributions.

Step detection problems are concerned with estimation of the point in time at which parameters of a Gaussian (or exponential) process changes. Such a change can indicate a failure in the underlying physical process. S_1 and S_2 detect changes in the mean value and standard deviation of a Gaussian process, respectively. Step detection in exponential processes have been used for finding bursts in Gamma-ray data from the BATSE experiment [1] (S_3), or classical examples like coal-mining disasters or treatment of the haemolytic-uraemic syndrome [16] (S_4).

The ability to model (and analyze) random walks is important for many process- and time-series-oriented applications. W_1 is a simpler random-walk example where the rate and error are unknown. The model W_3 is similar to the running example W_2. However, in this model, there is an abrupt change in the error rather than in the rate. Statistical reliability models for software [5] try to estimate the reliability of code (e.g., number of still existing errors) based on observations (e.g., interfailure times). We applied AUTOBAYES to the well-known (basic) Jelinski-Moranda model [9] JW and to the Schick-Wolverton hazard rate model SW.

In general, these results are very encouraging as they indicate that AUTOBAYES can already be applied to realistic examples. Synthesis times are generally shorter than one or two minutes; they also compare well with the compile times for the synthesized code. In the cases where no closed-form solution exists, the scale-up factor (i.e., the ratio between specification size and code size) is generally around 1:20 which supports our claim that models are much more compact (and thus faster to develop) than programs.

We are currently testing AUTOBAYES in two larger case studies concerning data analysis tasks for finding extra-solar planets, either by measuring dips in the luminosity of stars [10], or by measuring Doppler effects [12], respectively. Both projects required substantial effort to manually set up data analysis programs. Our goal for the near future is to demonstrate AUTOBAYES's capability to handle major subproblems (e.g., the CCD-sensor registration problem) arising in these projects.

4 Conclusions

The ultimate goal of this project is to change the way programs for science data processing are developed at NASA: we want to allow the

end users (i.e., scientists and modelers) to program in abstract models without having to worry about efficiency, portability, and correctness of the ultimately running code. To this end, we are developing AUTOBAYES which takes a compact description of a statistical model and automatically produces high-quality code. This model compilation capability allows the fast exploration of different modeling assumptions and thus increases productivity of the data analysis and confidence in its validity. We are currently incorporating state-of-the-art data analysis algorithms (e.g., kd-trees) into AUTOBAYES and extending its built-in domain knowledge for time series, filtering, and image processing problems.

References

[1] `http://www.batse.msfc.nasa.gov`

[2] C.L. Blake and C.J. Merz. UCI Repository of Machine Learning Databases, 1998.

[3] W. Buntine. Operations for Learning with Graphical Models. *J. AI Research*, 2:159–225, 1994.

[4] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm (with discussion). *J. of the Royal Statistical Society series B*, 39:1–38, 1977.

[5] W. Farr. Software Reliability Modeling Survey. In M. R. Lyu (ed.), *Handbook of Software Reliability Engineering*, pp. 71–117. McGraw-Hill, 1995.

[6] B. Fischer and J. Schumann. AutoBayes: A System for Generating Data Analysis Programs from Statistical Models, 2001. Submitted. Preprint available at `http://ase.arc.nasa.gov/people/fischer/`.

[7] 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*, LNCS 1924, pp. 212–229, Springer, 2000.

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

[9] Z. Jelinski and P. B. Moranda. Software Reliability Research. In W. Freiberger (ed.), *Statistical Computer Performance Evaluation*, pp. 465–484. Academic Press, 1972.

[10] D. G. Koch, W. Borucki, E. Dunham, J. Jenkins, L. Webster, and F. Witteborn. CCD Photometry Tests for a Mission to Detect Earth-size Planets in the Extended Solar Neighborhood. In *Proceedings SPIE Conference on UV, Optical, and IR Space Telescopes and Instruments*, 2000.

[11] `http://www.ulb.ac.be/polytech/march/dm.html`

[12] G. W. Marcy and R. P. Butler. Extrasolar Planets Detected by the Doppler Technique. In *Proceedings of Workshop on Brown Dwarfs and Extrasolar Planets*, 1997.

[13] M. Murphy. Octave: A free, high-level Language for Mathematics. *Linux Journal*, 39, July 1997.

[14] L. Pedersen, M.D. Wagner, D. Apostolopoulos, and W.L. Whittaker. Autonomous Robotic Meteorite Identification in Antarctica. 2001 IEEE Int. Conf. on Robotics and Automation, pp. 4158-4165, IEEE, 2001.

[15] D. Pelleg and A. Moore. Accelerating exact k-Means Algorithms with Geometric Reasoning. In *Proc. 5th Intl. Conf. Knowledge Discovery and Data Mining*, pp. 277–281, ACM Press, 1999.

[16] R. W. West and R. T. Ogden. Continuous-time Estimation of a Change-point in a Poisson Process. *Journal of Statistical Computation and Simulation*, 56:293–302, 1997.