Automatic Derivation of Statistical Data Analysis Algorithms: Planetary Nebulae and Beyond

Bernd Fischer*, Arsen Hajian†, Kevin Knuth** and Johann Schumann*

*RIACS / NASA Ames Research Center
†United States Naval Observatory
**NASA Ames Research Center

Abstract. AUTOBAYES is a fully automatic program synthesis system for the data analysis domain. Its input is a declarative problem description in form of a statistical model; its output is documented and optimized C/C++ code. The synthesis process relies on the combination of three key techniques. *Bayesian networks* are used as a compact internal representation mechanism which enables problem decompositions and guides the algorithm derivation. *Program schemas* are used as independently composable building blocks for the algorithm construction; they can encapsulate advanced algorithms and data structures. A *symbolic-algebraic system* is used to find closed-form solutions for problems and emerging subproblems.

In this paper, we describe the application of AUTOBAYES to the analysis of planetary nebulae images taken by the Hubble Space Telescope. We explain the system architecture, and present in detail the automatic derivation of the scientists' original analysis [1] as well as a refined analysis using clustering models. This study demonstrates that AUTOBAYES is now mature enough so that it can be applied to realistic scientific data analysis tasks.

INTRODUCTION

Planetary nebulae are remnants of dying stars. Scientists try to understand their physics by collecting and analyzing data, for example images taken by the Hubble Space Telescope (HST). The analysis follows a general pattern in science: formulate an initial understanding of the underlying physical processes as a statistical model, fit the model to the collected data, interpret the results, and refine the model as long as necessary. However, the large data volumes collected by modern instruments make computer support indispensable. Consequently, development and refinement of the necessary data analysis programs have become a bottleneck, and tapes of unanalyzed data have been sitting in warehouses for several years before the software was finally completed.

AUTOBAYES [2, 3] is a fully automatic program synthesis system for data analysis problems which increases the speed with which reliable data analysis software can be developed and thus eliminates this bottleneck. Its input is a declarative problem description in form of a statistical model; its output is documented and optimized C/C++ code. Externally, it thus looks like a compiler; internally, however, it is quite different. AUTOBAYES first derives a customized *algorithm* implementing the model and then transforms it into *code* implementing the algorithm. The algorithm derivation or *synthesis* process—which distinguishes AUTOBAYES from traditional compilers—relies on the intricate interplay of three key techniques. (*i*) AUTOBAYES uses *Bayesian networks*

(BNs) [4, 5] as a compact internal representation of the statistical models. BNs provide an efficient encoding of the joint probability distribution over all variables and thus allow to replace expensive probabilistic reasoning by faster graphical reasoning. In particular, they speed up the decomposition of a problem into statistically independent simpler subproblems. (*ii*) AUTOBAYES uses *program schemas* as the basic building blocks for the algorithm derivation. Schemas consist of a parameterized code fragment (i.e., template) and a set of constraints which are formulated as conditions on BNs. The templates can encapsulate advanced algorithms and data structures which lifts the abstraction level of the algorithm derivation. The constraints allow the network structure to guide the application of the schemas which prevents a combinatorial explosion of the search space. (*iii*) AUTOBAYES contains a specialized *symbolic-algebraic subsystem* which allows it to find closed-form solutions for many problems and emerging subproblems. The combination of these techniques results in fast turnaround times comparable to compilation times, supporting the iterative development style typical for the domain. AUTOBAYES thus enables the scientists to think and to program in models instead of code.

In this paper, we take one typical scientific data analysis problem—the analysis of planetary nebulae images taken by the HST—and show that AUTOBAYES can be used to automate the implementation of the necessary programs. We initially follow the analysis by Knuth and Hajian [1] and use AUTOBAYES to derive code for the published models. We show how concisely these models can be specified and how the code is derived; in particular, we show how the interaction between graphical reasoning, code instantiation, and symbolic computation is crucial for a fully automatic derivation. We then go beyond the original analysis and use AUTOBAYES to derive code for a simple mixture model which can be used as image segmentation procedure, automating a previously manual preprocessing step. Finally, we show how AUTOBAYES derives customized code for modifications of the standard model, which yield a more detailed analysis. The main contribution of this paper is to demonstrate that AUTOBAYES has reached a level of maturity which makes it applicable to realistic scientific data analysis applications.

PLANETARY NEBULAE

Stars with initial masses between roughly 0.8 and 8 solar masses turn into swollen red giants when they run out of hydrogen to support their primary fusion process. In a secondary fusion process, these giants then burn the helium produced by the hydrogen fusion, resulting in a carbon-oxygen core roughly the size of the earth. Eventually, the secondary fusion runs out of fuel as well and the red giants begin to collapse into extremely hot white dwarfs. During this collapse, most of the material is expelled, forming blown-out gaseous shells which are called planetary nebulae. The shells continue to expand and after 10,000 to 50,000 years their density becomes too small for the nebulae to be visible. Figure 1(1) shows an image of the planetary nebula IC 418.

Planetary nebulae occupy an important position in the stellar lifecycle and are the major sources of interstellar carbon and oxygen but their physics and dynamics are not yet well understood. The characterization and analysis of their properties is thus an important task in astronomy.

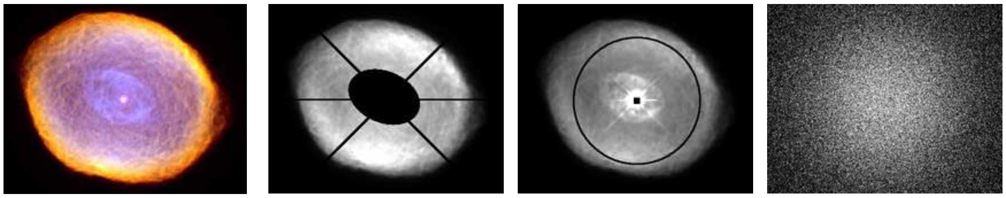

FIGURE 1. Planetary nebula IC 418 or Spirograph Nebula (1) Composite false-color image taken by the HST (Sahai et al., NASA and The Hubble Heritage Team). The different colors (resp. gray-scales) indicate the different chemicals prevalent in the different regions of the nebula; the origin of the visible texture is still unknown. The central white dwarf is discernible as a white dot in the center of the nebula. (2) Manually masked original image. (3) Original image with estimated Gaussian model parameters superimposed. (4) Sample data generated using estimated Gaussian model parameters.

AUTOBAYES

AUTOBAYES is implemented in SWI-Prolog [6] and currently comprises about 75,000 lines of documented code. Figure 2 shows the system architecture; in the following we explain the major components.

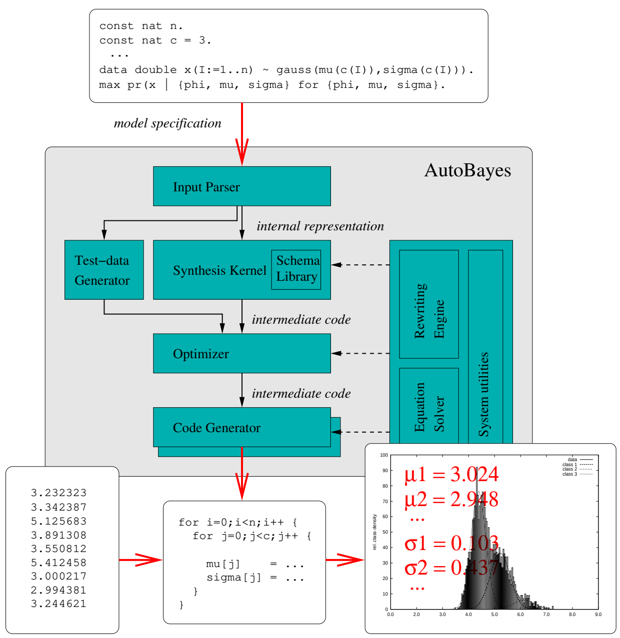

FIGURE 2. AUTOBAYES system architecture.

Statistical Models and Specification Language. A *statistical model* describes the expected properties of the data in a fully declarative fashion: for each problem variable of interest (i.e., observation or parameter), properties and dependencies are specified via probability distributions and constraints. Figure 3 shows how the Gaussian model discussed in more detail in the next section is represented in AUTOBAYES's specification language. Line 1 just identifies the model. Lines 2 and 4 introduce symbolic constants whose values are left unspecified but constrained by the `where`-clauses in lines 3 and 5, respectively. In general, constraints can be complex boolean formulae tying together

```
1  model gauss as '2D Gauss-Model for Nebulae Analysis'.
   % Image size
2  const nat nx as 'number of pixels, x-dimension'.
3    where 0 < nx.
4  const nat ny as 'number of pixels, y-dimension'.
5    where 0 < ny.
   % Center; assume center is on the image
6  double x0 as 'center position, x-dimension'.
7    where 1 =< x0 && x0 =< nx.
8  double y0 as 'center position, y-dimension'.
9    where 1 =< y0 && y0 =< ny.
   % Extent; assume full nebula is on the image
10 double r as 'radius of the nebula'.
11   where 0 < r && r < nx/2 && r < ny/2.
   % Intensity; upper bound determined by instrument
12 double i0 as 'overall intensity of the nebula'.
13   where 0 < i0 && i0 =< 255.
   % Noise; upper bound arbitrary, for initialization
14 double sigma as 'noise'.
15   where 0 < sigma && sigma < 100000.
   % Data and Distribution
16 data double pixel(1..nx, 1..ny) as 'image'.
17 pixel(I,J) ~ gauss(i0*exp(-((I-x0)**2+(J-y0)**2)/(2*r**2)),
                      sigma).
   % Task
18 max pr(pixel|{i0,x0,y0,r,sigma}) for {i0,x0,y0,r,sigma}.
```

FIGURE 3. Complete AUTOBAYES-specification for Gaussian model. Keywords have been underlined and line numbers have been added for reference; comments start with a % and extend to the end of the line.

multiple variables (cf. line 11). Lines 6–14 introduce the parameters, again constrained by `where`-clauses. Variables can be annotated with `as`-clauses; these textual annotations are propagated into the generated code to improve its legibility. Line 16 declares the observation (denoted by the `data`-modifier) as a matrix; its expected properties are specified in the distribution clause in line 17. Distributions can be both discrete (e.g., binomial) or continuous (e.g., Gaussian, Poisson, ...) but have to be univariate; support for multivariate distributions is currently under development. Distributions are chosen from a predefined list; adding more distributions is straightforward and requires only the density functions. The final line in the model is the task clause. It specifies the analysis problem the synthesized program has to solve. Since AUTOBAYES only supports *parameter learning* (i.e., the estimation of the parameter values best explaining the observed data, given a model) but not *structure learning* (i.e., the estimation of the best model itself), tasks have the form to maximize a conditional probability w.r.t. a set of goal variables. In this case, the task is a *maximum likelihood estimation* because all goal variables occur to the right of the conditioning bar in the conditional probability and there are no priors on the parameters. However, AUTOBAYES can also solve *maximum aposteriory estimation* problems.

Bayesian Networks. A *Bayesian network* (cf. Figure 4) is a directed, acyclic graph whose nodes represent random variables and whose edges define probabilistic dependencies between the random variables. AUTOBAYES uses a variant of *hybrid* BNs to represent the statistical models internally. Here, nodes can represent discrete as well as continuous random variables; these are rendered as boxes and ellipses, respectively. However, in Figure 4 all variables are continuous. Shaded nodes represent known variables, i.e., input data. Shaded boxes enclosing a set of nodes represent *plates* [4], i.e., collections of independent, co-indexed random variables. Distribution information for the random variables is attached to the respective nodes. Here, *pixel* is a $nx \times ny$ matrix of independent and identically distributed (*i.i.d.*) Gaussian random variables with observed values.

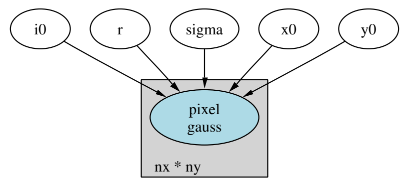

FIGURE 4. Bayesian net for Gaussian model, automatically extracted from the Gaussian model specification and drawn using the `dot` graph layouter.

Bayesian networks combine probability theory and graph theory. They are a common representation method in machine learning [4, 5] because they provide an efficient encoding of the joint probability distribution over all variables and thus allow to replace expensive probabilistic reasoning by faster graphical reasoning.

Schemas and Schema Library. Program synthesis from first principles is notoriously difficult to scale up (cf. [7, 8]). AUTOBAYES thus follows a schema-based approach. A *schema* consists of a parameterized code fragment (i.e., template) and a set of constraints. The parameters are instantiated by AUTOBAYES, either directly or by calling itself recursively with a modified problem. The constraints determine whether a schema is applicable and how the parameters can be instantiated. Constraints are formulated as conditions on the Bayesian network or directly on the specified model; they include the task clause as special case. This allows the network structure to guide the application of the schemas and thus to constrain combinatorial explosion of the search space, even if a large number of schemas is available. Schemas are implemented as Prolog-clauses and search control is thus simply relegated to the Prolog-interpreter: schemas are tried in their textual order. This simple approach has not caused problems so far, mainly because the domain admits a natural layering which can be used to organize the schema library. The top layer comprises network decomposition schemas which try to break down the network into independent subnets, based on independence theorems for Bayesian networks. These are domain-specific divide-and-conquer schemas: 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 larger programs by composition of different schemas. The next layer comprises more localized decomposition schemas which work on products of *i.i.d.* variables. Their application is also guided by the network structure but they require more substantial symbolic computations. The core layer of the library contains statistical algorithm schemas as for example *expectation maximization* (EM) [9, 10] and k-Means (i.e., nearest neighbor clustering); these generate the skeleton of the program. The final layer contains standard numeric optimization methods as for example the Nelder-Mead simplex method or different conjugate gradient methods. These are applied after the statistical problem has been transformed into an ordinary numeric optimization problem and AUTOBAYES failed to find a symbolic solution for the problem. Currently, the library comprises 28 top-level schemas, with a number of additional variations (e.g., different initializations).

Symbolic Subsystem. AUTOBAYES relies significantly on symbolic computations to support schema instantiation and code optimization. The core part of the symbolic subsystem implements symbolic-algebraic computations, similar to those in Mathematica [11]. It is based on the concept of term rewriting [12] and uses a small but reasonably efficient rewrite engine. Expression simplification and symbolic differentiation are implemented as sets of rewrite rules for this rewrite engine. The basic rules are straightforward; however, the presence of vectors and matrices introduce a few complications and require a careful formalization. In addition, AUTOBAYES contains a rewrite system which implements 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*. AUTOBAYES then uses a relatively simple symbolic equation solver built on top of these rewrite systems. It essentially handles only low-order polynomials (i.e., linear, quadratic, and simple cubic). However, it also shifts and normalizes exponents, recognizes multiple roots and bi-quadratic forms, and tries to find polynomial factors, and handles expressions in x and $(1-x)$ which are common in statistical applications.

A smaller part of the symbolic subsystem implements the graphical reasoning routines necessary for Bayesian networks with plates, for example, computing the parents, children, or Markov blanket [5] of a node.

Backend. The code constructed by schema instantiation and composition is represented in an imperative intermediate language. This is essentially a "sanitized" subset of C (e.g., no pointers), which is extended by a number of domain-specific constructs like vector/matrix operations, finite sums, and convergence-loops. Since straightforward schema application can produce suboptimal code, AUTOBAYES interleaves synthesis and optimization. Schemas can explicitely trigger aggressive large-scale optimizations like code motion, common sub-expression elimination, and memoization which can take advantage of information from the synthesis process. Traditional low-level optimizations like constant propagation or loop fusion, however, are left to the compiler. In a final step, AUTOBAYES translates the intermediate code into code tailored for a specific run-time environment. Currently, AUTOBAYES has code generators for the Octave and Matlab environments; it can also produce stand-alone C and Modula-2 code. Each code generator is implemented as a rewrite system which eliminates the intermediate language constructs not supported by the target environment; most rules are shared between the different code generators.

Test-Data Generation. Statistical models also allow to generate problem-specific test data (i.e., *sampling*), if the the edges of the extracted network are followed forward from the sources and random numbers are generated along the way. AUTOBAYES uses this interpretation to generate model-specific sampling programs which can be used to validate the models and solutions.

A HIERARCHICAL SET OF MODELS

Knuth and Hajian [1] present three models they used to analyze images of the planetary nebula IC418 (cf. Figure 1(1)). Each model estimates a parameter set which is then refined by the subsequent models. Their common idea is (*i*) that the light intensity which is expected at a given pixel position (x,y) on the image can be described by a function F of this position, the (unknown) nebula center (x_0,y_0), and some additional parameters, and (*ii*) that the measured intensities can be fitted against F using a simple mean square error minimization. The only difference between the models is the form of F.

Here we sketch how these models are represented in AUTOBAYES' specification language, and how the code is derived. In particular, we show how the interaction between graphical reasoning, code instantiation, and symbolic computation is crucial for a fully automatic derivation.

Gaussian Model

In the first model, F has the shape of a bell whose apex is at (x_0,y_0) which can be formalized by a two-dimensional Gaussian curve:

$$F(x,y) = i_0 \cdot e^{-\frac{(x_0-x)^2+(y_0-y)^2}{2r^2}} \tag{1}$$

The additional parameters i_0 and r capture the overall intensity and extent of the nebula (i.e., the height and diameter of the bell).

Model Specification. The AUTOBAYES specification shown in Figure 3 is a direct transcription of the underlying mathematics. The distribution clause for the image pixels in line 17 formalizes the idea that the expected value of the pixel (i,j) can be described by the function $F(x,y)$ from Equation (1) since the expected value of a Gaussian random variable is given by the mean of the distribution. The variance of the distribution represents the error of the fit. In combination with the Gaussian distribution, the task clause in line 18 thus specifies a mean square error minimization. The constraints formalize additional assumptions on the structure of the image or the output of the instrument (cf. line 13).

Mathematical Derivation. For the Gaussian model, the program derivation can neatly be separated into two phases such that the first phase is a purely mathematical derivation and the second phase only instantiates code templates; in general, however, this is not the case and symbolic computation and template instantiation are interleaved.

The first step is to unfold the entire *pixel*-matrix element by element, using a decomposition schema based on the conditionalized version of the general product rule for probabilities:

$$pr(\boldsymbol{pixel} \mid i_0,x_0,y_0,r,\sigma) = \prod_{i=1}^{nx} \prod_{j=1}^{ny} pr(pixel(i,j) \mid i_0,x_0,y_0,r,\sigma)$$

The precondition for this step is that the pixels are pairwise independent, given the remaining variables, i.e., that

$$pr(pixel(i,j) \mid pixel(i',j'),i_0,x_0,y_0,r,\sigma) = pr(pixel(i,j) \mid i_0,x_0,y_0,r,\sigma)$$

holds for all i, j, i', j' such that $i \neq i'$ or $j \neq j'$. Instead of proving this from first principles, using the distribution information given in the model specification, AUTOBAYES can easily check it on the Bayesian network: there is no edge going from the *pixel*-node into itself and, hence, by definition of Bayesian networks, the pixels are pairwise independent.

The probability is now elementary in the sense that on the left of the conditioning bar we have only a single variable $pixel(i, j)$ which depends exactly on all the variables to the right of the conditioning bar. Hence, the probability expression can be replaced by the distribution function. AUTOBAYES's domain theory contains rewrite rules for the most common probability density functions; additional density functions can easily be added. This rewrite yields the likelihood-function

$$\prod_{i=1}^{nx}\prod_{j=1}^{ny} \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{\left[pixel(i,j)-i_0\cdot e^{-\frac{(x_0-i)^2+(y_0-j)^2}{2r^2}}\right]^2}{2\sigma^2}}$$

which must be maximized w.r.t. goal variables i_o, x_0, y_0, r, and σ. In general it is easier to work with the log-likelihood function which yields the same solutions during maximization since the logarithm is strictly monotone. After simplification using the symbolic subsystem, AUTOBAYES thus derives the following log-likelihood function:

$$L = -nx \cdot ny \cdot \log(2\pi) - nx \cdot ny \cdot \log(\sigma) - \frac{1}{2\sigma^2} \cdot \sum_{i=1}^{nx}\sum_{j=1}^{ny}\left[pixel(i,j) - i_0 \cdot e^{-\frac{(i-x_0)^2+(j-y_0)^2}{2r^2}}\right]^2$$

A solution can now be attempted in two different ways, numerically or symbolically. By default, AUTOBAYES tries to find symbolic solutions first. In this case, it computes the partial differentials

$$\frac{\partial L}{\partial i_0} = \frac{1}{\sigma^2} \cdot \sum_{i=1}^{nx}\sum_{j=1}^{ny} pixel(i,j) \cdot e^{-\frac{(i-x_0)^2+(j-y_0)^2)}{2r^2}} - \frac{i_0}{\sigma^2} \cdot \sum_{i=1}^{nx}\sum_{j=1}^{ny} e^{-\frac{(i-x_0)^2+(j-y_0)^2)}{r^2}}$$

$$\frac{\partial L}{\partial \sigma} = \frac{1}{\sigma^3} \cdot \sum_{i=1}^{nx}\sum_{j=1}^{ny}\left[pixel(i,j) - i_0 \cdot e^{-\frac{(i-x_0)^2+(j-y_0)^2)}{2r^2}}\right]^2 - \frac{nx \cdot ny}{\sigma}$$

and solves the equations $\partial L/\partial i_0 \stackrel{!}{=} 0$ and $\partial L/\partial \sigma \stackrel{!}{=} 0$ which are essentially simple polynomials in i_0 and σ. These can easily be handled by the built-in equation solver; however, attempts to solve for the remaining three variables x_0, y_o, and r fail.

Code Derivation. At this point, the symbolic computations have been exhausted without leading to a complete symbolic (i.e., closed-form) solution. AUTOBAYES thus derives code for a numeric solution, incorporating the computed partial symbolic solution. This is done in two steps, which correspond to schemas in the schema library.

In the first step, AUTOBAYES converts the symbolic solutions into assignment statements. Then it identifies their order and position relative to the remaining code which is still to be synthesized. Since both solutions contain at least one (in fact all) of the remaining variables, they must follow that code: since the solution for σ contains i_0, its assignment must in turn follow that of i_0. AUTOBAYES then eliminates both of them from the formula by applying the substitution corresponding to the solution. Variables whose solutions do not depend on any unsolved variable can be considered as symbolic constants and need not be eliminated; their corresponding assignments must precede the missing code block. Since this reasoning is done on the (side-effect free) expression level, a dataflow analysis is not required.

In the second step, AUTOBAYES instantiates a numeric optimization routine, in this case the Fletcher-Reeves conjugate gradient method. The schema's template actually contains a wrapper to the implementation provided by the GNU Scientific Library (GSL) [13]. However, the schema itself is not just boilerplate code because it also constructs specific initialization code and a number of auxiliary functions to evaluate the goal function and the derivatives. AUTOBAYES contains different heuristics to derive initialization code from specification information; here, the initial values are taken as the midpoints of the specified ranges. AUTOBAYES also generates the auxiliary functions; since a straightforward translation from the goal expression would be prohibitively inefficient, AUTOBAYES aggressively optimizes the functions. The optimizations include common subexpression elimination, memoization, and code motion, and are applied both intra- and inter-procedural (although the latter restricted to the generated auxiliary functions). The optimizations can also take into account locally constant variables, since AUTOBAYES knows the set of goal variables. Again, a dataflow analysis is not required since the reasoning is done on the expression-level.

Program Results. We have applied the generated program to the manually masked image of IC418 shown in the second panel of Figure 1. The third panel shows the results. The program roughly approximates the center but its estimate of the overall extent is predictably off the mark.

Sigmoidal Models

The Gaussian model hard-codes a number of assumptions about the structure of the image, in particular that it is circular, with a pronounced intensity peak and a gradual intensity falloff at the edges. However, a quick look at Figure 1 shows that the image is clearly elliptic, with a broad intensity plateau and a pronounced falloff at the edges.

Simple Sigmoidal Model. Knuth and Hajian thus refine their initial model and replace the two-dimensional Gaussian by a two-dimensional sigmoidal function of the form

$$F(x,y) = i_0 \cdot \left[1 - \frac{1}{1+e^{-a\sqrt{r(x,y)}-1}}\right] \tag{2}$$

with the auxiliary function $r(x,y)$

$$r(x,y) = c_{xx}\cdot(x_0-x)^2 + 2c_{xy}\cdot(x_0-x)(y_0-y) + c_{yy}\cdot(y_0-y)^2$$

and constants c_{xx}, c_{yy}, and c_{xy}

$$c_{xx} = \frac{\cos^2\theta}{r_x^2} + \frac{\sin^2\theta}{r_x^2} \qquad c_{yy} = \frac{\sin^2\theta}{r_x^2} + \frac{\cos^2\theta}{r_x^2} \qquad c_{xy} = \frac{\sin\theta\cdot\cos\theta}{r_x^2\cdot r_y^2}$$

where r_x and r_y are the extent of the nebula along the major and minor axis, resp., θ is its orientation, and a the intensity falloff.

The specification for this modified model can easily be derived from the one for the Gaussian model shown in Figure 3, essentially by replacing the mean value in the distribution clause (cf. line 17) with the new version of F and adding declarations for the new model variables. The auxiliary constants and functions are represented as deterministic nodes in the Bayesian network, which are expanded like C-style macro definitions during the program definition. The program for this model is then derived using the same steps as before; the only difference is that the symbolic expressions become much more complicated.

Axis-aligned Sigmoidal Model. The derivation and resulting program can be simplified and sped up, if the nebula image is assumed to be axis-aligned. This can be modeled by changing the random variable θ into a constant with known value, i.e.,

```
const double theta := 0 as 'orientation'.
```

AUTOBAYES can then propagate this constant value already on the specification level and derive code from the simplified model.

Dual Sigmoidal Model. In a final refinement step, Knuth and Hajian try to estimate the thickness of the shell as well. Since projecting the three-dimensional ellipsoidal shell of gas onto a two-dimensional image produces an ellipsoidal blob surrounded by a ring of higher intensity, the image can be modeled as the difference of two sigmoidal functions with the same center and orientation but different extents, intensities, and falloffs. Re-using the auxiliary definitions from the simple sigmoidal model, this refinement can also be specified easily for AUTOBAYES.

IMAGE SEGMENTATION MODELS

In their original analysis, Knuth and Hajian manually masked the central star and the diffraction spikes (cf. Figure 1(2)). This prevents their analysis from misinterpreting the comparatively bright star as the center of the nebula. A simple segmentation of the image into different conceptual classes (e.g., central star, nebula, and background) can replace the generic mask. In this section we show how AUTOBAYES can be used to derive code for this and further refined segmentation models.

Segmentation via Clustering. The simplest segmentation model is the straightforward Gaussian mixture model shown Figure 5. At its core is the latent variable c (cf. lines 12–14) which represents the unknown class each pixel belongs to and which determines the mean and variance (cf. line 16); its values are independently drawn from a discrete distribution with relative frequencies φ, i.e., $pr(c_{ij} = k) = \varphi_k$. The constraint on the probability vector φ (cf. line 11) is required to make this model well-formed. Since we need the class-assignments for the segmentation, c is declared as `output`. Note that the model assumes that the number of classes is known.

AUTOBAYES solves this model by an application of the EM-schema which is triggered by the latent variable structure of the corresponding Bayesian network. After instantiation of the EM-schema, all emerging sub-problems (for the E- and M-step) can eventually be solved symbolically (for the detailed derivation cf. [2, 3]).

Figure 6 shows the result of segmenting the IC418 image according to the identified different classes. In particular, a segmentation into three classes already isolates the

```
model segment as 'Image segmentation via Clustering'.
... (see Figure 3) ...
% Class parameters and relative frequencies
const nat n_classes as 'number of classes'.
  where 0 < n_classes.
double mu(1..n_classes), sigma(1..n_classes).
  where 0<sigma(_).
double phi(1..n_classes).
  where sum(I:=1..n_classes, phi(I))=1.
% Classes and Distribution
output nat c(1..nx, 1..ny) as 'class'.
  where 1 =< c(_,_) && c(_,_) =< n_classes.
c(_,_) ~ discrete(phi).
% Data and Distribution
data double pixel(1..nx, 1..ny) as 'image'.
pixel(I,J) ~ gauss(mu(c(I,J)), sigma(c(I,J))).
% Task
max pr(pixel|{phi,mu,sigma}) for {phi,mu,sigma}.
```

FIGURE 5. AUTOBAYES-specification for image segmentation model.

central star, the nebula, and the background from each other. It can thus be used as a precise, image-specific mask (cf. Figures 1(2) and 6(1)). More classes reveal more details, e.g., identify the nebula's shell and separate the halo from the background, but too many classes bear the usual risk of overfitting the image.

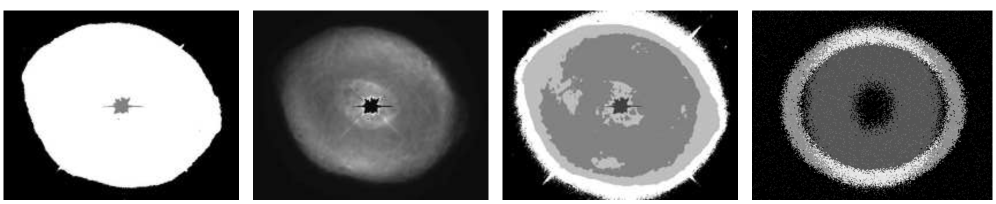

FIGURE 6. Segmentations of IC418 image: (1) three-class segmentation (2) ditto, white class used as mask load to original image (3) five-class segmentation, and (4) sample data from geometrically refined segmentation model; class 1 (white) corresponds to the hull, class 2 (gray) to the core, and class 3 (not shown) to the background.

Model modifications. Code implementing this standard model can be found in many libraries. AUTOBAYES, however, can also derive customized code for model modifications. For example, a uniform variance for all classes can be enforced by simply changing the distribution clause into

```
pixel(I,J) ~ gauss(mu(c(I,J)), sigma).
```

and adapting the declaration of σ accordingly. AUTOBAYES still solves this model by an application of the EM-schema as before, but the structure of the M-step changes. In the standard model, it contains a loop over all classes which computes the current estimates of μ_i and σ_i. In the modified model, only the μ_i are computed inside the loop, while the single σ is computed separately. This modification cannot be achieved by simple code parametrization—it requires an actual code adaptation.

Similarly, the standard model can be refined by adding priors (e.g., on the class means) to capture additional background knowledge. In AUTOBAYES, these priors are simply added to the specification as distribution clauses for the parameters. For example, a conjugate prior on each individual mean can be specified by

```
mu(I) ~ gauss(mu0(I), sqrt(sigma_sq(I)) * kappa0(I)).
```

AUTOBAYES again solves the model by an application of the EM-schema, and again the structure of the M-step changes, now to reflect the MAP estimate. More model modifications can be specified easily: a conjugate prior using the same parameters for all class means, non-conjugate priors, a mixture of different distributions (e.g., a single Cauchy and multiple Gaussians), and many more. In each case, AUTOBAYES derives an appropriate version of the EM-algorithm, instantiating and composing schemas as required. This makes it much more flexible than a simple code library.

Multivariate approximation. Images of planetary nebulae are taken at different wavelengths, usually in the infrared and visible bands. Such data sets can be analyzed by AUTOBAYES if the covariance matrix is (assumed to be) diagonal, even though AUTOBAYES cannot yet properly handle multivariate distributions. In the specification shown in Figure 5, only the declaration (line 15) and distribution (line 16) of the input data need to be modified as shown below. Again, AUTOBAYES generates an appropriately instantiated EM algorithm.

```
data double pixel(1..nc,1..nx, 1..ny) as 'data cube'.
pixel(C,I,J) ~ gauss(mu(C,c(I,J)), sigma(C,c(I,J))).
```

Segmentation with geometric information. A further refinement of the basic model can be obtained by adding spatial information, e.g., the geometric knowledge about the elliptic shape of the nebula. In AUTOBAYES, we can model such a refinement, for example, by making the mean values of the mixed Gaussian distributions depend on the location (i, j) of the pixel. Here, the pixels in each class are located on an elliptical ring around the center $x_0(c_{ij}), y_0(c_{ij})$ with radii $r_x(c_{ij})$ and $r_y(c_{ij})$, and a thickness $d(c_{ij})$. Each of these unknown parameters depend on the class c_{ij} of each pixel at position i, j. In the AUTOBAYES specification of Fig 5, we only need to replace the simple mean $\mu(c_{ij})$ of the Gaussian distribution (cf. line 16) by the expression

$$i_0(c_{ij}) \cdot e^{-\frac{(\sqrt{r(i,j)}-1)^2 \cdot (r_x(c_{ij})^2 + r_y(c_{ij})^2)}{4d(c_{ij})^2}} \tag{3}$$

where $r(i, j) = (i - r_x(c_{ij}))^2 / r_x(c_{ij})^2 + (j - r_y(c_{ij}))^2 / r_y(c_{ij})^2$. AUTOBAYES again synthesizes an EM-algorithm, this time with a numerical optimization routine (e.g., a Nelder-Mead simplex method) nested within the M-step. Figure 6(4) shows sampling data generated with the parameters estimated by this model.

EVALUATION

Table 1 summarizes the results; it shows that AUTOBAYES's specification language allows a compact problem representation: none of the models requires more than 35

TABLE 1. Summary of Results. For each model, the size of the specification and the generated program (*including* generated comments), and the synthesis time (measured on an unloaded 2GHz/4GB LinuxPC) are given. `-nosolve` and `-nolib` are AUTOBAYES command line options which suppress the application of the schemas using partial symbolic solutions and library components. The entries for `sigmoid-0`, `sigmoid-2`, and `segment-geom` refer to the `-nolib -nosolve` variants.

Model	|Spec|	|Code|	T_{synth}
`gauss`	18	1045	36.4s
`-nosolve`		703	2.4s
`-nolib`		764	4.9s
`-nolib -nosolve`		494	1.1s
`sigmoid`	28	-	-
`-nosolve`		12650	39m42.9s
`-nolib -nosolve`		872	3.2s
`sigmoid-0`	27	581	1.3s
`sigmoid-2`	35	1202	6.7s
`segment`	17	518	1.2s
`segment-multi`	18	602	1.4s
`segment-geom`	30	1765	40.6s

lines. The major difficulty in writing the specifications was to understand and then to express the core idea of the original models. After that, each specification took only a few minutes to write and in general one or two iterations to debug and complete (e.g., adding constraints).

The table also shows the overall feasibility of the approach. AUTOBAYES was able to derive code for each of the models; scale-up factors are generally around 1:30. Synthesis times are generally only a few seconds and comparable to compilation times of the derived code. However, for the `gauss` and `sigmoid` models, AUTOBAYES spends almost all time simplifying the partial differentials and then optimizing the auxiliary functions evaluating them; in the `sigmoid`-case, this even exhausts the available memory. With the command line options, AUTOBAYES can be forced to stay away from these expensive calculations and code is derived much faster, using the Nelder-Mead simplex method which requires no differentials.

Search space explosion is a common problem in program synthesis. In our case, it is mitigated by the deterministic nature of the symbolic-algebraic computations, the higher level of abstraction inherent to the schemas, and the inherent structure of the schema library. Still, search spaces are large. For the `gauss`-model, AUTOBAYES derives 224 programs, many of them equivalent, in 35 minutes total synthesis time. Parts of the search space can be pruned away manually by command line options like `-nosolve`, but more implicit control is required, especially when the schema library grows further.

The original Matlab code uses a gradient descent method; it computes the differentials in a single iteration over all pixels while the synthesized code iterates once for each differential but reuses memoized subexpressions. This decomposition requires domain knowledge not yet formalized for AUTOBAYES. For the `gauss`-model, the (interpreted)

Matlab-code is approx. 70 lines, mainly for the computation of the gradient. It requires on average 174 seconds to converge, while the synthesized C++-code only requires 11 seconds.

RELATED WORK

Scientific computing is a (relatively) popular application domain for program synthesis; due to its complexity, most related work also follows a schema-based approach. However, most work focuses on wrapping solvers for partial differential equations (PDEs). SciNapse [14] is a general "problem-solving environment" for PDEs; it supports code generation for a variety of features in the PDE-domain, as for example coordinate transformation or grid generation. Ellman *et al.* [15] describe a system to generate simulation programs from PDEs. Both systems use Mathematica as underlying symbolic engine. Ctadel [16] is PDE-based synthesis system for the weather forecasting domain. It is also implemented in SWI-Prolog and contains its own symbolic engine.

Other domains than PDEs have been tackled less often. Amphion [17] is a purely deductive synthesis system which has been used to synthesize astronomical software using library components. Amphion/NAV [18] applies the same technology to the state estimation domain. The scale-up difficulties encountered there led to a switch to a schema-based approach and the development of the AUTOFILTER-system [19] as a domain-specific extension of AUTOBAYES. It provides its own specification language and schemas but reuses the core system. Planware [20] deductively synthesizes high-performance schedulers. It uses concepts from higher-order logic and category theory to structure the domain theory and thus to reduce the required proof effort.

Schema-based program synthesis is also related to generative programming [21], since schemas (more precisely, the code fragments) correspond to *templates*. The major differences are that (*i*) the almost completely syntax-directed template instantiation is less powerful and less secure than schema instantiation and (*ii*) the users still have to write the core algorithm.

Code libraries are common in scientific computing and data analysis, but they lack the level of automation achievable by program synthesis. For example, the Bayes Net Toolbox [22] is a Matlab library which allows program development on the model level, but it does not derive algorithms or generate code. BUGS [23] is a statistical model interpreter based on Gibbs-sampling, a more general but less efficient Bayesian inference technique; it could be integrated into AUTOBAYES as an additional schema.

CONCLUSIONS

We presented AUTOBAYES, a fully automatic program synthesis system for the data analysis domain and demonstrated how it can be used to support a realistic scientific task, the analysis of planetary nebulae images taken by the HST. We specified the hierarchy of models presented in [1]. AUTOBAYES was able to automatically generate code for these models; in tests, the synthesized code gave the same results as the scientists' Matlab code. We also used AUTOBAYES to refine the analysis, combining cluster-based image segmentation with geometric constraints.

Key elements to achieve these results are the schema-based synthesis approach, an efficient problem representation via Bayesian networks, and a strong symbolic-algebraic subsystem. This combination is unique to AUTOBAYES and allows us to solve realistic data analysis problems. In this particular case study the recently completed integration of the GSL and the aggressive optimization of the goal expressions proved to be instrumental to derive efficient code.

However, AUTOBAYES must still be improved before it can be delivered to the working data analyst. In particular, the domain coverage must be extended and schemas must be added to enable solutions for new classes of models (e.g., multivariate distributions). Synthesis from large models requires support for hierarchical specifications and more powerful optimizations to generate faster code. Finally, numeric optimization schemas must be extended with numeric constraint handling to produce more versatile and robust code. Yet, AUTOBAYES already demonstrates that program synthesis technology has become a viable and versatile technology that enables scientists to think and to program in models rather than in code.

Availability. AUTOBAYES is not yet available publicly; for a web-based interface and other papers see `http://ase.arc.nasa.gov/autobayes`.

Acknowledgments. A. Hajian was supported for this work by NASA through grant numbers GO7501, GO8390, GO8773 from the Space Telescope Science Institute, which is operated by AURA, Inc. under NASA contract NAS5-26555. K. Knuth was supported by the NASA Ames DDF Grant 274-51-02-51, the NASA IDU/IS/CICT Program, and the NASA Aerospace Technology Enterprise. W. Buntine contributed substantially to the early development of AUTOBAYES.

REFERENCES

1. Knuth, K. H., and Hajian, A. R., "Hierarchies of Models: Toward Understanding of Planetary Nebulae," in *Proc. Bayesian Inference and Maximum Entropy Methods in Science and Engineering*, edited by C. Willimans, American Institute of Physics, 2002, pp. 92–103.
2. Fischer, B., and Schumann, J., *J. Functional Programming*, **13**, 483–508 (2003).
3. Gray, A., Fischer, B., Schumann, J., and Buntine, W., "Automatic Derivation of Statistical Algorithms: The EM Family and Beyond," in *Advances in Neural Information Processing Systems 15*, edited by L. Saul, MIT Press, 2002.
4. Buntine, W., *J. AI Research*, **2**, 159–225 (1994).
5. Pearl, J., *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann Publishers, 1988.
6. Wielemaker, J., *SWI-Prolog 3.1 Reference Manual, Updated for Version 3.1.0 July, 1998*, Amsterdam (1998).
7. Ayari, A., and Basin, D., *J. Symbolic Computation*, **31**, 487–520 (2001).
8. Manna, Z., and Waldinger, R., *The Deductive Foundations of Computer Programming*, Addison-Wesley, 1993.
9. Dempster, A. P., Laird, N. M., and Rubin, D. B., *J. of the Royal Statistical Society series B*, **39**, 1–38 (1977).
10. McLachlan, G., and Krishnan, T., *The EM Algorithm and Extensions*, Wiley Series in Probability and Statistics, John Wiley & Sons, 1997.
11. Wolfram, S., *The Mathematica Book*, Cambridge Univ. Press, UK, 1999, 4th edn.
12. Baader, F., and Nipkow, T., *Term Rewriting and all that*, Cambridge Univ. Press, 1998.

13. Galassi, M., Davies, J., Theiler, J., Gough, B., Jungman, G., Booth, M., and Rossi, F., Gnu Scientific Library - Reference Manual (2002).
14. Akers, R., Baffes, P., Kant, E., Randall, C., Steinberg, S., and Young, R., *IEEE Computational Science and Engineering*, **4**, 33–42 (1997).
15. Ellman, T., Deak, R., and Fotinatos, J., "Knowledge-Based Synthesis of Numerical Programs for Simulation of Rigid-Body Systems in Physics-Based Animation," in *Proc. 17th Intl. Conf. Automated Software Engineering*, edited by W. Emmerich and D. Wile, IEEE Comp. Soc. Press, 2002, pp. 93–104.
16. van Engelen, R. A., Wolters, L., and Cats, G., "Ctadel: A Generator of Efficient Code for PDE-based Scientific Applications," in *Proc. 9th Intl. Conf. on Supercomputing*, ACM Press, 1996, pp. 86–93.
17. Stickel, M., Waldinger, R., Lowry, M., Pressburger, T., and Underwood, I., "Deductive Composition of Astronomical Software from Subroutine Libraries," in *Proc. 12th Intl. Conf. Automated Deduction*, edited by A. Bundy, Springer, 1994, vol. 814 of *Lect. Notes Artificial Intelligence*, pp. 341–355.
18. Whittle, J., Van Baalen, J., Schumann, J., Robinson, P., Pressburger, T., Penix, J., Oh, P., Lowry, M., and Brat, G., "Amphion/NAV: Deductive Synthesis of State Estimation Software," in *Proc. 16th Intl. Conf. Automated Software Engineering*, edited by M. S. Feather and M. Goedicke, IEEE Comp. Soc. Press, 2001, pp. 395–399.
19. Whittle, J., and Schumann, J., Automating the implementation of Kalman-filter algorithms (2003), in review.
20. Blaine, L., Gilham, L.-M., Liu, J., Smith, D. R., and Westfold, S., "Planware – Domain-Specific Synthesis of High-Performance Schedulers," in *Proc. 13th Intl. Conf. Automated Software Engineering*, edited by D. F. Redmiles and B. Nuseibeh, IEEE Comp. Soc. Press, 1998, pp. 270–280.
21. Czarnecki, K., and Eisenecker, U. W., *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2002.
22. Murphy, K., *Computing Science and Statistics*, **33** (2001).
23. Thomas, A., Spiegelhalter, D. J., and Gilks, W. R., "BUGS: A program to perform Bayesian inference using Gibbs sampling," in *Bayesian Statistics 4*, edited by J. M. Bernardo, J. O. Berger, A. P. Dawid, and A. F. M. Smith, Oxford Univ. Press, 1992, pp. 837–842.