Applying AutoBayes to the Analysis of Planetary Nebulae Images

(Extended Abstract)

Bernd Fischer Johann Schumann

RIACS / NASA Ames Research Center

E-mail: {fisch,schumann}@email.arc.nasa.gov

Abstract

We take a typical scientific data analysis task, the analysis of planetary nebulae images taken by the Hubble Space Telescope, and describe how program synthesis can be used to generate the necessary analysis programs from high-level models. We describe the AUTOBAYES *synthesis system, discuss its fully declarative specification language, and present the automatic program derivation starting with the scientists' original analysis [7].*

1 Introduction

Planetary nebulae are remnants of dying stars. Scientists try to understand them by collecting and analyzing data, for example images taken by the Hubble Space Telescope (HST). The analysis follows a general pattern in science: formulate the initial understanding of the underlying physical processes as a model, fit the model to the collected data, interpret the results, and refine the model as long as necessary. Since the underlying processes and the data collection are both fraught with uncertainty and noise, statistical models are used.

In most disciplines, 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. AUTOBAYES is a fully automatic program synthesis system for data analysis problems which is intended to increase the speed with which reliable data analysis software can be developed. Its input is a declarative problem description in form of a statistical model; its output is documented and optimized C/C++ code. Its schema-based approach allows the use of advanced algorithms and data structures and yields fast turnaround times comparable to compilation times, supporting the iterative development style typical for the domain. AUTOBAYES thus enables the scientists to think and program in models instead of code.

In this paper, we take the analysis of planetary nebulae images taken by the HST and show that and how AUTOBAYES can be used to automate the implementation of the necessary analysis programs. We follow the approach described in [7] and use AUTOBAYES to derive code for the published models. The main contribution of this paper is to demonstrate that AUTOBAYES has reached a level of maturity which makes it applicable to real-life problems.

2 Planetary Nebulae

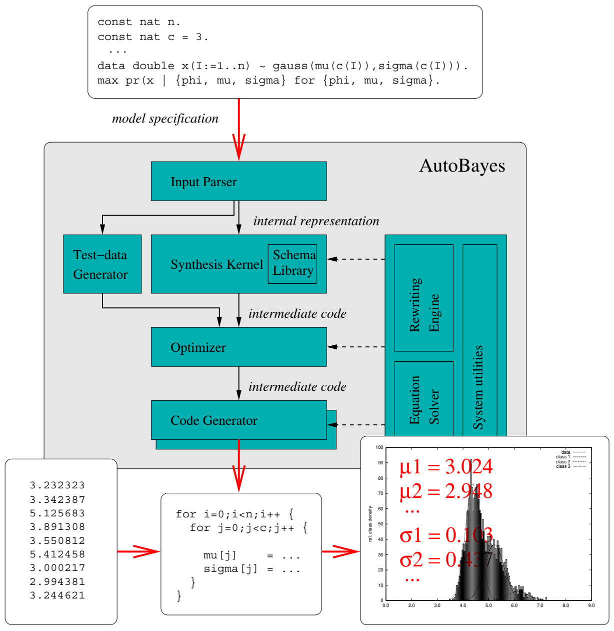
Stars with initial masses between roughly 0.8 and 8 solar masses turn into red giants when they run out of hydrogen to support their primary fusion process. In a secondary fusion process, they then burn the helium produced by the hydrogen fusion, resulting in a carbon-oxygen core roughly the size of 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 into a variety of shapes and after 10,000 to 50,000 years their density becomes too small for the nebulae to be visible. Planetary nebulae occupy an important position in the stellar life-cycle 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.

3 AutoBayes

AUTOBAYES [5, 6] is a fully automatic program synthesis system for data analysis problems.[1] It is implemented in SWI-Prolog and currently comprises about 64,000 lines of documented code. Figure 1 shows the system architecture; in the following we explain the major components.

Statistical Models and Specification Language. A *statistical model* describes the expected properties of the data in a fully declarative fashion: for each problem variable,

[1]http://ase.arc.nasa.gov/autobayes


```
const nat n.
const nat c = 3.
  ...
data double x(I:=1..n) ~ gauss(mu(c(I)),sigma(c(I))).
max pr(x | {phi, mu, sigma} for {phi, mu, sigma}.
```

```
3.232323
3.342387
5.125683
3.891308
3.550812
5.412458
3.000217
2.994381
3.244621
```

```
for i=0;i<n;i++ {
  for j=0;j<c;j++ {

    mu[j]    = ...
    sigma[j] = ...
  }
}
```

Figure 1. AUTOBAYES system architecture.

```
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 2. Specification for Gaussian model.

properties and dependencies are specified via probability distributions and constraints. Figure 2 shows how a model (see Section 4.1 for more details) is specified in AUTOBAYES.[2] 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; constraints can also be complex boolean formulae tying together multiple variables (cf. line 11). Lines 6–15 introduce the model 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. In general, a distribution clause is of the form $x \sim D(\vec{y})$ where x is a variable or vector/matrix element, D is a distribution, and $\vec{y}$ are the distribution's parameters. Distributions are chosen from a predefined list but adding more distributions is straightforward. The final line in the model is the *task* clause. It specifies the analysis problem the synthesized program has to solve, i.e., maximizing the conditional probability w.r.t. the set of goal variables.

Bayesian Networks. AUTOBAYES uses *Bayesian networks* to represent the statistical models internally. They are directed, acyclic graphs where nodes represent random variables and edges define probabilistic dependencies between them. This yields an efficient encoding of the joint probability distribution over all variables and allows to replace expensive probabilistic reasoning by faster graphical reasoning. Bayesian networks are thus a common representation method in machine learning [3, 9].

Schemas and Schema Library. Purely deductive program synthesis is notoriously difficult to scale up. 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 specified model or on the Bayesian network. This allows the network structure to guide the application of the schemas and thus to constrain the search space. Schemas can in principle be understood as conditional rewrite rules on partially instantiated programs where the only redexes are maximization tasks. They 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 automat-

[2] Keywords have been underlined and line numbers have been added for reference; comments start with a % and extend to the end of the line.

ically synthesize larger programs by composition of different schemas. The next layer comprises more localized decomposition schemas which work on products of independent random variables. Their application is also guided by the network structure but they require more symbolic computations. The core layer of the library contains statistical algorithm schemas as for example *expectation maximization* (EM) [8] and k-Means; 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 plus some additional variants (e.g., different initializations).

Symbolic Subsystem. AUTOBAYES relies on symbolic computations to support schema instantiation and code optimization. The core of the symbolic subsystem is a small rewrite engine which supports associative-commutative operators and explicit contexts. AUTOBAYES thus allows contextual rules as for example $x/x \rightarrow_{C \vdash x \neq 0} 1$ where $\rightarrow_{C \vdash x \neq 0}$ means "rewrites to, provided $x \neq 0$ can be proven from the current context C." The contexts are managed almost transparently by the rewrite engine; rewrite systems only need to contain non-congruent propagation rules which modify the contexts under which immediate subterms are rewritten, e.g., $p \,?\, s : t \overset{\sim}{\rightarrow}_C (p\downarrow_C) \,?\, (s\downarrow_{C \wedge p}) : (t\downarrow_{C \wedge \neg p})$ for C-style conditionals. Here, $\overset{\sim}{\rightarrow}_C$ and $\downarrow_C$ denote context propagation from and normal form computation under the context C.

Expression simplification and symbolic differentiation, similar to those in Mathematica, 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 i and j are usually unknown at synthesis time, the partial derivative $\partial x_i / \partial x_j$ can only be rewritten into $i = j \,?\, 1 : 0$. Some rules require explicit meta-programming, e.g., when bound variables are involved. Abstract interpretation is used to efficiently evaluate frequently occurring range constraints such as $x > 0$ or $x \neq 0$. AUTOBAYES 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*. In total, the symbolic subsystem contains 365 rewrite rules.

For equation solving, AUTOBAYES essentially relies on a simple low-order polynomial (i.e., linear, quadratic, and simple cubic) symbolic solver built on top of the core system. However, the solver also shifts and normalizes exponents, recognizes multiple roots and bi-quadratic forms, 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 for Bayesian networks, for example computing the parents, children, or Markov blanket [9] of a node.

Intermediate Code. The code fragments in AUTOBAYES' schemas are written in an imperative intermediate language. This is essentially a "sanitized" variant of C (i.e., no pointers, side effects in expressions etc.); however, it also contains a number of domain-specific constructs like vector/matrix operations, finite sums, and convergence-loops.

Optimization. Straightforward schema instantiation and composition produces suboptimal code; worse, many of the suboptimalities cannot be removed completely using a separate, after-the-fact optimization phase. AUTOBAYES thus interleaves synthesis and optimization. Schemas can explicitly trigger large-scale optimizations which take into account information from the synthesis process. For example, all numeric routines restructure the goal expression using code motion, common sub-expression elimination, and memoization; since the schemas know the goal variables, no dataflow analysis is required to identify invariant sub-expressions, and code can be moved around aggressively, even across procedure borders.

Code Generation. In a final step, AUTOBAYES translates the optimized 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 standalone C and Modula-2 code.

Each code generator employs one rewrite system to eliminate the constructs of the intermediate language which are not supported by the target environment ("desugaring") and a second rewrite system to clean up the desugared code; most rules are shared between the different code generators.

4 Models and Code Derivation

Knuth and Hajian [7] present three models they used to analyze images of the planetary nebula IC418 (cf. Figure 3). 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, symbolic computation, and code instantiation is crucial for a fully automatic derivation.

4.1 Gaussian Model

In the first model, F has the shape of a bell whose apex is at (x_0, y_0). This 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 2 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); remember that the expected value of a Gaussian random variable is given by the mean (i.e., first parameter) of the distribution. The standard deviation (i.e., second parameter) 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 models here the program derivation can be separated such that a purely mathematical derivation is followed by a pure instantiation of code templates; in general, however, symbolic computation and template instantiation are interleaved.

The first step unfolds the *pixel*-matrix element-wise, using a decomposition schema based on the conditionalized version of the general product rule for probabilities:

$$\begin{aligned} &pr(\textbf{\textit{pixel}} \mid i_0, x_0, y_0, r, \sigma) \\ &= \textstyle\prod_{i=1}^{nx} \prod_{j=1}^{ny} pr(pixel(i,j) \mid i_0, x_0, y_0, r, \sigma) \end{aligned}$$

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

$$\begin{aligned} &pr(pixel(i,j) \mid pixel(i',j'), i_0, x_0, y_0, r, \sigma) \\ &= pr(pixel(i,j) \mid i_0, x_0, y_0, r, \sigma) \end{aligned}$$

holds for all i, j, i', j' with $i \neq i'$ or $j \neq j'$. AUTOBAYES can easily check this on the Bayesian network: since no edge connects the *pixel*-node to itself, the pixels are pairwise independent by the definition of Bayesian networks

The probability is now a *likelihood*, i.e., the single variable $pixel(i, j)$ on the left of the conditioning bar depends exactly on all the variables on the right. Hence, it can be replaced by the distribution function. AUTOBAYES's domain theory contains rewrite rules for the most common distributions; additional distributions 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, AUTOBAYES thus derives the following log-likelihood function:

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

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

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

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

Code Instantiation. At this point, the symbolic computations have been exhausted without leading to a complete symbolic solution. AUTOBAYES thus derives code for a numeric solution, incorporating the computed partial symbolic solution. This is done in two steps, which both correspond to schemas in the schema library.

In the first step, AUTOBAYES converts the symbolic solutions into assignment statements and identifies their order and position relative to the remaining code which is still to be synthesized. Since both solutions contain at least one 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 substitutes the solution into the formula and thus eliminates these variables. Variables whose solutions do not depend on any unsolved variable are 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 wraps an implementation provided by the GNU Scientific Library (GSL); this includes specific initialization code and auxiliary functions to evaluate the goal function and the derivatives. AUTOBAYES uses heuristics to derive initialization code from specification information, e.g., the midpoints of the specified ranges. It also generates the auxiliary functions; since a straightforward translation of the goal expression would be prohibitively inefficient, it aggressively optimizes the functions. The optimizations include common subexpression elimination, memoization, and code motion, and are applied both intra- and inter-procedural. They 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.

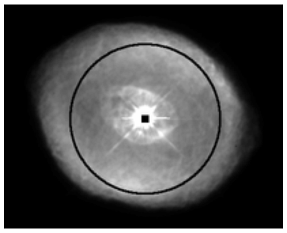


Figure 3. IC418 image with results of Gaussian analysis superimposed.

Program Results. We applied the generated program to the image of IC418 shown in Figure 3, where the results are superimposed. The program correctly approximates the center but its estimate of the overall extent is off the mark.

4.2 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 3 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

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

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}

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

where r_x and r_y are the extent of the nebula along its axes, 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 2, 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.

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

Table 1. Summary of Results

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 a known value of zero. 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.

5 Evaluation

Table 1 summarizes the synthesis results; for each model it lists the size of the specification and the generated program including generated comments, and the synthesis time measured on a 2GHz/4GB LinuxPC. `-nosolve` and `-nolib` are AUTOBAYES command line options which suppress the application of the schemas using partial symbolic solutions and library components as described above.

The table shows that the specification language allows a compact problem representation; none of the models requires more than 35 lines. The major difficulty in writing the specifications was to understand and then to express the core idea of the scientists' models, which was not completely clear from the original paper. After that, each specification took only a few minutes to write and one or two iterations to debug and complete (e.g., adding constraints).

The table also shows the overall feasibility of the approach. AUTOBAYES is 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, AUTOBAYES spends most of the 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 away from these expensive calculations and code is derived much faster, using the Nelder-Mead simplex method which requires no differentials.[3]

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 scientists' original Matlab code uses a simple 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. Both versions give the same results.

6 Related Work

Scientific computing is a popular application domain for program synthesis; however, most work focuses on wrapping solvers for ordinary (ODEs) and partial differential equations (PDEs). SciNapse [1] is a "problem-solving environment" for PDEs; it supports code generation for a variety of features in the PDE-domain, e.g., coordinate transformation and grid generation. Ellman *et al.* [4] describe a system to generate simulation programs from ODEs. Both systems also follow a schema-based approach.

Other domains have been tackled less often. Amphion [10] and Amphion/NAV [12] are purely deductive synthesis systems for the astronomical calculation and state estimation domains. The scale-up difficulties encountered there led to a switch to a schema-based approach and the development of AUTOFILTER [11] as a domain-specific extension of AUTOBAYES. It provides its own specification language and schemas but reuses the core system. Planware [2] 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.

[3]The table entries for `sigmoid-0` and `sigmoid-2` thus refer to the `-nolib -nosolve` variants.

7 Conclusions

We successfully applied AUTOBAYES to the analysis of planetary nebulae images taken by the HST. We specified the models presented in [7], from which AUTOBAYES was able to automatically generate code; in tests, this code gave the same results as the scientists' Matlab code.

The crucial success factors 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.

AUTOBAYES is an ongoing development effort; recent improvements include the integration of the GSL and the aggressive optimization of goal expressions. Both were instrumental in this case study to derive efficient code. Current work focuses on adding schemas that enable solutions for new classes of models, e.g., covariate data. Similarly, the numeric optimization schemas are extended with numeric constraint handling to produce more robust code.

References

[1] R.L. Akers *et al.* SciNapse: A problem-solving environment for partial differential equations. *IEEE Comp.Sci.Eng.*, 4(3):33–42, 1997.

[2] L. Blaine *et al.* Planware – domain-specific synthesis of high-performance schedulers. In *ASE-13*, pp. 270–280. IEEE, 1998.

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

[4] T. Ellman, R. Deak, and J. Fotinatos. Knowledge-based synthesis of numerical programs for simulation of rigid-body systems in physics-based animation. In *ASE-17*, pp. 93–104. IEEE, 2002.

[5] B. Fischer and J. Schumann. AutoBayes: A system for generating data analysis programs from statistical models. *JFP*, 13(3):483–508, 2003.

[6] A.G. Gray, B. Fischer, J. Schumann, and W. Buntine. Automatic derivation of statistical algorithms: The EM family and beyond. In *NIPS-15*. MIT Press, 2002.

[7] K.H. Knuth and A.R. Hajian. Hierarchies of models: Toward understanding of planetary nebulae. In *Bayesian Inference and Maximum Entropy Methods in Science and Engineering 22*, pp. 92–103. 2002.

[8] G. McLachlan and T. Krishnan. *The EM Algorithm and Extensions*. Wiley, 1997.

[9] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[10] M. Stickel *et al.* Deductive composition of astronomical software from subroutine libraries. In *CADE-12*, *LNAI 804*, pp. 341–355. Springer, 1994.

[11] J. Whittle and J. Schumann. Automating the implementation of Kalman-filter algorithms. In review.

[12] J. Whittle *et al.* Amphion/NAV: Deductive synthesis of state estimation software. In *ASE-16*, pp. 395–399. IEEE, 2001.