ELSEVIER

Advances in Engineering Software 33 (2002) 27–36

ADVANCES IN ENGINEERING SOFTWARE

www.elsevier.com/locate/advengsoft

GAUSS: an online algorithm selection system for numerical quadrature

Naren Ramakrishnan[a,*], John R. Rice[b], Elias N. Houstis[b]

[a]*Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061, USA*
[b]*Department of Computer Sciences, Purdue University, West Lafayette, IN 47907, USA*

Accepted 31 August 2001

Abstract

We describe the design and implementation of GAUSS — an online algorithm selection system for numerical quadrature. Given a quadrature problem and performance constraints on its solution, GAUSS selects the best (or nearly best) algorithm. GAUSS uses inductive logic programming to generalize a database of performance data; this produces high-level rules that correlate problem features with algorithm performance. Such rules then serve as the basis for recommending algorithms for new problem instances. GAUSS functions *online* (new data and information can be incrementally incorporated) and can also provide phenomenological *explanations* of algorithm recommendations. © 2002 Elsevier Science Ltd. All rights reserved.

Keywords: Algorith selection; Numerical quadrature; Performance evaluation

1. Introduction

The subject of this paper is GAUSS — an automatic algorithm recommender system for numerical quadrature. Given a quadrature problem and performance constraints on its solution, GAUSS selects the best (or nearly best) algorithm. The approach we take is to organize a database of test problems and quadrature algorithms, and to accumulate performance data for the given population. This database of performance data is then mined (generalized) to arrive at high-level rules that can form the basis for a recommendation (for future problems).

GAUSS demonstrates a novel approach to the design and implementation of algorithm recommender systems. Traditionally, recommender systems generalize performance data *offline* and have been restrictive in their coverage of the application domain (e.g. the use of weak learning methods such as neural network training). GAUSS however uses relational descriptions for domain modeling and interacts dynamically with its environment to gather the data needed to mine for recommendations. It is thus an *online* recommender system. New information about problems and algorithms can be easily incorporated without having to retrain on the old data. In addition, the end-user of the recommender system can query the system for the basis of the recommendations (relational rules that correlate the effect of problem features on algorithm performance).

Two ideas are critical in our presentation below:

- GAUSS *does not* evaluate integrals (symbolically or numerically). In addition, it does not require a symbolic form of the integrand, though it can make use of symbolic information if available. The goal of recommender systems is to help novice users or to be used in a completely automatic environment (such as problem solving environments [10]).
- Systems like GAUSS typically have two phases of development and operation. The first is the collection and generalization of performance information (to yield rules, decision procedures, etc.) while the second is the actual process of algorithm recommendation (and incorporating any feedback/changes). The first phase is time consuming but needs to be done only once. The second phase is comparatively fast, inexpensive, and can be performed as many times as desired.

1.1. Reader's guide

Section 2 provides a quick overview of the application domain considered in this study and identifies the difficulties in algorithm recommendation. The design of the GAUSS system is detailed in Section 3. Complexity and various practical implementation considerations are described here. Section 4 presents a careful experimental evaluation

* Corresponding author.
E-mail addresses: naren@cs.vt.edu (N. Ramakrishnan), jrr@cs.purdue.edu (J.R. Rice), enh@cs.purdue.edu (E.N. Houstis).

of the system. Section 5 identifies several important future research directions.

2. Numerical quadrature

The algorithm recommendation problem addressed by GAUSS is to:

Select an algorithm to evaluate $I = \int_a^b f(x)\,\mathrm{d}x$
so that relative error $\epsilon_r < \theta$ and N is minimized

where θ is an user-specified error requirement and N is the number of times *f(x)* is evaluated in $[a, b]$ to yield the desired accuracy. Most quadrature algorithms evaluate I as [6]:

$$\int_a^b f(x)\,\mathrm{d}x \approx w_1 f(x_1) + w_2 f(x_2) + \cdots + w_n f(x_n),$$

$$-\infty \le a \le b \le +\infty$$

We choose ϵ_r and N as performance criteria because:

- For most software implementations of integration routines, an absolute error ϵ_a and a relative error ϵ_r are input. For the integral $I = \int_a^b f(x)\,\mathrm{d}x$, these routines compute $\{R_n, E_n\}$ where R_n is the estimate of the integral using n values of *f(x)* while E_n is the relevant error estimate for R_n. Typically automatic quadrature routines terminate when the error condition

  $$|R_n - I| \le E_n \le \max(\epsilon_a, \epsilon_r |R_n|)$$

  is satisfied. In most of the literature (on performance evaluation of numerical integration software; see, for example, [7]) and in a majority of the implementations, the routines are made to impose a strictly relative accuracy by setting $\epsilon_a = 0$. Thus, ϵ_r is chosen as the main accuracy criterion in this study.
- The time required (excluding the time to evaluate $f(x_i)$, by a numerical quadrature rule varies quite widely from one implementation to another, even for the same generic technique (method) with the same number of nodes. Moreover, most efficient quadrature routines are of the adaptive nature so that the weights (w_i) and nodes (x_i) are chosen dynamically during the computation. Finally, in most applications the computing time is dominated by the time to do function evaluations. Thus, a more uniform metric is the number of function evaluations (N) required to evaluate an integral.

2.1. Difficulties in algorithm selection

Any introductory text on integral calculus provides readymade 'recipes' for tackling difficult integrands, and for transforming them to more tractable and traditional forms. In other words, it is easier to reformulate (change) the problem so that a general purpose algorithm will not be inefficient [1]. A simple example can be obtained from the integral

$$I = \int_0^{\pi/2} \frac{\cos x}{\sqrt{x}}\,\mathrm{d}x,$$

which has a non-polynomial behavior, whereas most quadrature algorithms assume that the curve to be integrated can be approximated (piecewise, if necessary) accurately by a polynomial of a certain degree, such as a straight line, parabola, or cubic (a simple example is the composite trapezoidal rule). Thus, general purpose algorithms encounter difficulties near the origin when integrating I, because at the lower limit, the value of the integrand approaches infinity. The traditional solution in textbooks is to make the substitution

$$x = t^2, \qquad \mathrm{d}x = 2t\,\mathrm{d}t,$$

which removes the singularity and produces the more well-behaved

$$I' = 2\int_0^{\sqrt{\pi/2}} \cos(t^2)\,\mathrm{d}t$$

which is amenable to even the most naive algorithms. The current lack of algorithm selection systems [1] forces one to think of altering troublesome problems in this manner rather than choosing from the multitude of algorithms that can handle integrands with special properties such as vertical and/or horizontal asymptotes, removable singularities, and unimportant factors. While not every such problematic integrand can be altered to remove the trouble, the above example reveals that the way a problem instance is (re)presented is critical for the success of an algorithm selection methodology.

Even if such representational features are taken into account, more difficulties arise because the overall factors influencing the applicability (or lack thereof) of an algorithm in a certain context are not very well understood. The way problem features affect methods is complex and algorithm selection might depend in an unstable way on the features actually used. While a simple structure might exist, the feature coordinate system might not reflect the simplicity properly. Consider the case when a particular feature system imposes the following methodology:

Method 1 is best if $x^2 + y^2 \le 1$
Method 2 is best otherwise.

If we now choose new coordinates (x', y') such that

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 1.0001 \\ 1 & 1.0000 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

then Method 1 is best in a very long, very thin area (in (x', y')

coordinates). If the following features are now chosen,

$$f1 = x' \qquad f2 = y' \qquad f3 = x' + y' \sin x' + z1$$

$$f4 = x' - y' + 1/x' + e^{y'} + z2$$

where $z1$ and $z2$ are irrelevant, or random, it will then take a lot of data and effort to recover the original simple selection methodology. Thus, the methodology might depend in an unstable manner on the features actually used $(f1, f2, f3, f4)$ and no reasonable amount of brute force computing can provide a robust selection methodology in such a situation. Moreover, the relative performances of various algorithms should be correlated with problem features when determining the selection methodology. Any solution approach must be able to reason about issues in performance evaluation using relevant *symbolic* information from problem features (such as the presence or absence of singularities).

2.2. Why attribute-value approaches fail

The term *attribute-value techniques* is used to encompass a wide variety of machine learning approaches such as decision tree induction, association rules, nearest neighbor classification, neural networks, fuzzy logic, and other approaches that reason with uncertainty. The common factor among these varied approaches is that they can be viewed as working at the level of propositional logic. While these schemes are advantageous for their relative simplicity, efficiency, ability to handle outliers (and noise), and support for incremental training, we contend that they are ineffective for the domain considered in this paper. In particular, since propositional logic is more restrictive than first-order predicate logic, attribute-value techniques are restricted to inducing non-relational generalizations from data. An example follows.

Consider the artificial data in Fig. 1 about an algorithm and its effect on some problems. Notice that some problems are variants of others. For instance, they could be parameterizations of some generic problem, restatements of a problem with a different constraint, etc. An attribute-value learning scheme would try to associate the ranking of the algorithm with feature information, and as is obvious from Fig. 1, will be in error by at least one entry. (This rule corresponds to rating $m1$ as the best method for all problems.) This is because such schemes attempt to discover a pattern involving a single relation in a relational database. However, a completely accurate (and more expressive) scheme is given by:

```
best_method (m1, X):- feature (X, g).
best_method (m1, X):- feature (Y, g),
variant (X, Y).
second_method (m1, X):- feature (X, f).
```

where the first rule indicates that method m1 is best for problem X if it has feature 'g', and so on.[1] Notice that these rules involve relations from more than one table in the database and are ordered so that the first applicable rule is 'fired.' This is a powerful representational facility that can be utilized effectively for algorithm selection.

The second and more serious drawback[2] to attribute-value mechanisms is that they solve the algorithm selection problem by determining a universal function approximator from the problem space to the algorithm space without taking into account the domain-specific background knowledge. This might lead to predictions that are sometimes invalid [23]. In most cases, the reason for this behavior is that ordinal variables are compared and assigned measures of cardinality, which leads to unstable models of representation and prediction. Research in reasoning and representation [14] has shown that assigning ordinal properties to cardinal variables preserves strictly monotonic transformations while the reverse can be dangerous. For an overview of relational approaches in comparison to attribute-value techniques, we refer the interested reader to [5].

This problem is further complicated by the multitude of adaptive algorithms applicable to certain problems. One study [20] concludes that there are between 1 and 10 million adaptive quadrature algorithms that are potentially interesting and significantly different from one another! This staggering number arises from the possible ways of permuting the choice of rules, processor components, error bounds, data structures, etc.

Problem	Feature Information	Method $m1$ (Rank)
$p1$	g	1
$p2$	f	1
$p3$	f	2
$p4$	f	1
$p5$	g	1
$p6$	g	1

Problem	Variant
$p1$	$p2$
$p2$	$p1$
$p4$	$p5$
$p5$	$p4$

Fig. 1. A relational database with two tables containing performance information about six problems and one method. A '1' in an entry indicates that method m1 is best for that particular problem while a '2' indicates that it is second best. Some of the problems are reformulations of others, which are also reflected in the database. Notice also that the second table models a symmetric relationship — every tuple (a, b) in the table implies that (b, a) is also present. Such a relationship is an inverse of itself.

3. The design of GAUSS

We outline the design of the GAUSS system with an example. Consider the effect of the QUADPACK routine QNG [18] on the integral (Fig. 2, left)[3]

$$\int_0^1 x^{1/2} \log(x) \mathrm{d}x = -\frac{4}{9}$$

[1] It should be mentioned that using predicates such as `best_method`, `second_method` is not advisable as addition of new algorithms 'throws' off the ranking. We provide a more stable encoding later on in the paper.

[2] The first drawback can be alleviated by performing a relational join of the two tables to construct a new bigger table but this destroys the natural connection that exists between two different entities in a database.

[3] Notice that one of the factors is not defined at one endpoint of the integration interval. For routines such as QNG, the function value in such cases has to be set equal to the limit value of the function, or equated to zero if this limit does not exist or is infinite.

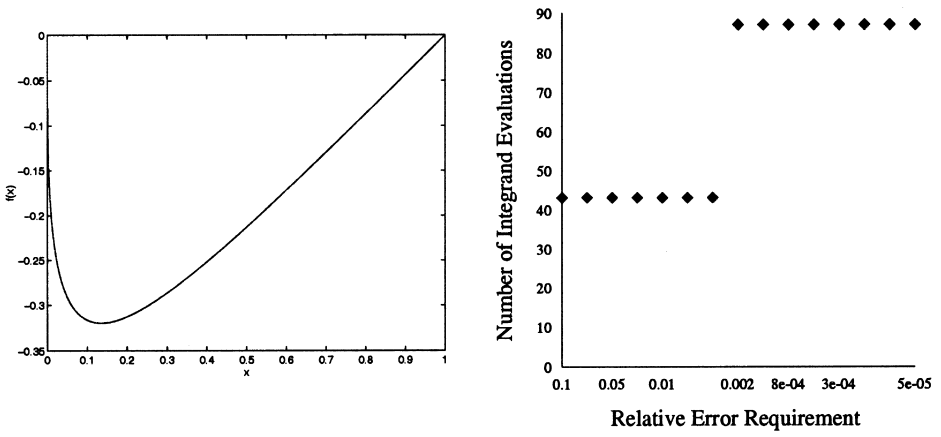

Fig. 2. (left) Sample integral problem and (right) Performance of QNG for various error constraints.

When the relative error requirement is reduced from 0.1 down to 5×10^{-5}, the number of function evaluations (nfe) varies as shown in (Fig. 2, right). Notice the abrupt change in the graph at $\epsilon_r = 0.002$. This is due to the non-adaptive (weights and nodes are not dynamically selected during the integration) and automatic (ability to guarantee a particular accuracy) nature of the QNG routine. QNG is based on a sequence of rules with increasing degrees of algebraic precision. In addition, QNG breaks down at $\epsilon_r = 1 \times 10^{-5}$, with an error code of 1 indicating that the maximum number of function evaluations (87) has been reached (in other words, the integral is probably too difficult to be computed within the number of steps allowed). For an adaptive algorithm, this error code would imply that the limit on the number of integer subdivisions has been achieved, which has been set a priori within the integrator.

The result(s) of performance evaluation and the properties of the sample problem are encoded as function-free first-order logic formulas. For example,

```
nfe (P1, QNG, 0.1,43).
nfe (P1, QNG, 0.05,43).
...
nfe (P1, QNG, 0.002,87).
breakdown (P1, QNG, 1E-05).
errorcode (P1, QNG, 1E-05,1).
range (P1, 'finite').
factors (P1, 'log').
```

models the information in (Fig. 2, right). This is repeated for executions of various algorithms on a selected benchmark of test problems (discussed in detail later). Additional intensional rules produce consequent predicates that determine which routine performed best for each sample problem, example:

```
qvalue (P1, QAGS, 1).
accuracy (P1, 0.1).
```

The qvalue predicate indicates some figure of merit for a given algorithm on a given problem (in this case, QAGS on P1). It could denote the relative ranking of the algorithm for the given problem or some other computed function of performance data. A particular definition of the Q-value we used is given later in the paper. In the above example, the Q-value of algorithm QAGS for problem P1 with error requirement $\epsilon_r = 0.1$ is 1 (meaning that it is best). Notice that this representation also captures the various error codes possible from the routines, they could mean occurrence of excessive round off error, too small an interval to be subdivided (by an adaptive routine), difficulties encountered in integrand behavior, divergent (or slowly convergent) results, or a limiting number of cycles obtained. The modeling of error codes aids in the generalization of results across the feature space of problem instances.

The next step in GAUSS is to generalize from performance data encoded in this manner. Generalization is benefited if we provide 'wrong' instances of performance data (so that the system can address the extent to which a particular performance information is indicative of a larger set of problems). It is also aided by the provision of any background knowledge we might have about the problem domain (such as the fact that some algorithms are not applicable for all integrands).

Given this information, GAUSS attempts to construct a predicate logic formula so that all the positive examples can be logically derived from the background knowledge and no negative example can be logically derived. The advantages of such an inductive logic programming system (ILP) lie in the generality of representation of background knowledge. The ILP system used by GAUSS is PROGOL [16], that inverts the resolution operator used in first-order predicate logic. An example rule induced by GAUSS is given by:

```
qvalue (X, QAGS, 1):- accuracy(X, Acc),
sing(X), endptsing(X), noderivsing(X),
range(X, finite).
```

which indicates that QAGS is appropriate for any integrand with any error criterion if the function has end point singularities. Detailed accuracy results and interpretations of the induction process are provided later in the paper.

3.1. Controlling the complexity

The above process of relational induction is computationally expensive in the general case. In fact, for the general setting of first-order logic, the problem is undecidable.

A first restriction to function-free horn clauses results in decidability (this is the form of programs used in the programming language PROLOG). Decidability here is with respect to induction; for deduction, first-order logic is semi-decidable. However, ILP is often prohibitively expensive and the standard practice is to restrict the hypothesis space to a proper subset of first-order predicate logic. Most commercial systems (like Golem and PROGOL) further require that background knowledge be ground, meaning that only base facts can be provided as opposed to intensional information (rules). This renders the overall complexity polynomial in the space of the database but pseudo-polynomial (and sometimes exponential) in the space of the modifiable parameters that describe the length of the rules and the types of variables allowed in the rules. GAUSS makes use of several techniques to control the complexity:

- The use of domain specific restrictions for the management of the recommendation spaces; GAUSS makes use of syntactic and semantic restrictions on the nature of the induced rules. An example of a syntactic restriction is that both the properties of a problem as well as the accuracy constraints need to be taken into account in the construction of a rule. Another example of a syntactic restriction is that one algorithm can obtain only one Q-value for a given problem. An example of a semantic restriction is consistency checks between algorithms and their inputs. For example, routine QDAWO requires that its input be in a certain form, so inducing a more general rule will not be fruitful.
- We incorporate a generality ordering to guide the induction of rules. This ordering is used to prune the search space for generating plausible hypotheses and to aid in abduction (which is the process of constructing a rule that needs to be justified further).
- Since the software architecture of the recommender system is augmented with a natural database query interface, we utilize this aspect to provide meta-level patterns for rule generation. Consider the meta rule
  ```
  qvalue (X,Y,#Z):- f1(X,_), f2(X,_),
  f3(X,_), f4(X,_), f5(X,_)
  ```
  which indicates that the second parameter Y should not occur in the antecedent part of a rule, nor should the parameter Z from the consequent. These correspond to the algorithm and its Q-value for a given set of problems. This rule ensures that the antecedents are based purely on features (f1, f2, f3,…) of the problem X. The # before the third parameter indicates that this symbol can only be instantiated with a constant value, meaning that we are interested in rules that help us to compute Q-values, not generalize across the space of Q-values.
- In our implementation of the software system for GAUSS, we utilize the Postgres database management system [21] which is an extension of a relational database management system that handles complex objects and rules. In addition, it provides a storage manager which supports transaction processing by archiving the history of a simulation into a spooling mechanism. This facility is central to the efficient organization of performance data and helps in integration where a simple query-memoization technique can be used to provide positive and negative instances for modeling.

3.2. Extending GAUSS for online recommendation

We now address the aspect of computation of Q-values, incremental learning of rules, and scaling up the methodology to domains of greater complexity.

3.2.1. Relational reinforcement

Consider the case when a new algorithm is added to the suite of quadrature routines. How can the generalization process be updated to take this into account? This issue of a system having to continually interact with its environment to obtain relevant data and to update the basis for recommendations (the rule bank) dynamically is termed *reinforcement learning* [13] in the AI literature.

The standard model of reinforcement supposes a discrete set of environment states S (which could provide information about the problem, its constraints, the error codes observed etc.), a discrete set of actions A (in this case, recommendations) that affect the environment, and a set of reinforcement signals (in this case, whether the particular algorithm indeed achieved the desired constraints or what, if anything, went wrong). On each step of interaction, GAUSS receives as input some indication of the current state (such as problem features) and it chooses an action (a recommendation) to generate as output. This recommendation changes the state of the environment (leading to an overflow in computation, for example) and the value of this state transition is communicated to GAUSS through the reinforcement signal. GAUSS then chooses recommendations that will tend to increase the long-run sum of values of the reinforcement signal. Thus, GAUSS can learn to do this over time, guided by a variety of learning algorithms available in the literature; the learning model, in our case, is the rule-bank already detailed in Section 3.1. Since we use relational learning for the rules and reinforcement from the environment, this paradigm is known as *relational reinforcement learning*, first introduced in Ref. [9].

There are two main themes of reinforcement learning and the GAUSS system utilizes the Q-learning paradigm for model-free learning. The goal of this approach is to abstract the utility of taking actions (algorithms) in specific states (problem instances). Relating utilities to (state, action) pairs is a more natural model for the numerical quadrature application because of a lack of states. The online algorithm is presented in Fig. 3. The basic idea of the algorithm is to learn a bank of rules for algorithm selection. This set of rules is induced via the TILDE-RT logical regression tree algorithm [8]. The logical tree — called a Q-tree — can then be considered to be an abstraction of the bank of rules that describe the Q-values. Initially, all algorithms are assigned a Q-value of 0 for all the problem instances. As each experiment unfolds, the Q-value for a particular (algorithm, problem) combination is updated and the regression tree is refined to reflect this fact. The induction process described in Section 2.2 guarantees generalization with respect to the problem space. There is also a discount parameter γ that models the relative importance of future experiments with respect to earlier ones. The penalty ratio k models the effect of negative feedback from bad algorithm selections. The theory of machine learning [15] shows that this algorithm converges to the optimal Q-values in the limit. In other words, the performance of GAUSS improves over time.

An alternative to reinforcement learning is presented in Ref. [17] wherein the author visualizes a space of possible algorithms and algorithm selection is then viewed as performing an optimization within this space without learning an evaluation function directly. In our work, we explicitly learn an evaluation function (the Q-value) for assessing the efficacies of algorithms.

```
Initialize the bank of rules (Q-Tree) to be empty

For each $a, p$ do
  Initialize the Q-values $Q(p, a) = 0$

While true
  For each $p_i$ from the problem database
    Set state $s$ = "unsolved"
    Select an algorithm $a_j$ using the bank of rules
    Run it for problem $p_i$ and examine the result
    If (performance constraints satisfied) set reward $r_i = 1$
      else if (no error codes) set reward $r_i = 0$
      else set reward $r_i = -k * c$ where $c$ is the error code

    If (problem solved), set state $s$ = "solved"
      else set $s$ = "unsolved"

    Generate example $(p_i, a_j, q_{ij})$
      where $q_{ij} = r_i + \gamma * max_{a'} Q(p_i, a')$ if $s$ = "unsolved"
      where $q_{ij} = r_i$ if $s$ = "solved"

    Update the Q-Tree using this example
```

Fig. 3. The GAUSS relational Q-learning algorithm, abstracted from Ref. [9]. (*a*,*p*) refers to an (algorithm, problem) combination. γ is the discount factor for reinforcement learning and k is a penalty ratio for negative selections. In the test study described in Section 4, we utilized the QUADPACK assignment of error codes to conditions as our primary means of modeling the variable c.

3.2.2. Incrementality

Incremental behavior is naturally achieved by the reinforcement learning algorithm presented above. Notice that there are only two possible outcomes of an experiment — 'solved' (with or without satisfying the requested performance criteria) and 'unsolved' (with appropriate error conditions). This simplifies the mechanics of incrementality further into a simple update for the Q-values. In a more complicated scenario, the step in Fig. 3 for generating the exemplar will involve a for-loop for accommodating multiple states in the solution path.

4. Experimental studies

In this section, we outline the experimental framework for the evaluation of GAUSS. We have utilized 124 routines from the GAMS cross-index [4], where the category H2a is for the evaluation of one dimensional integrals. In the experiments described below, methods for principal value integrals, interval analysis techniques, parallel methods of numerical integration, Monte Carlo methods and number theoretic methods are excluded. The integrand is also assumed to be either (i) a function available in symbolic form and for which it is possible to write a FORTRAN/C subroutine/function or (ii) provided as a finite number of data values $\{x_i, y_i\}$. The latter case is particularly important because it precludes any attempt to symbolically determine the features of functions. Since most quadrature routines are available in FORTRAN/C format and are run multiple times during performance evaluation with similar values for their parameters, we resort to *partial evaluation* [2,12], a program transformation technique that uses static knowledge of a program's input to optimize the run-time over a series of executions. Partial evaluation is a well-known technique for compiling scientific code for several specialized applications [3] and produced speedups ranging from 5 to 18.5 for the experiments conducted in this study. An example of partial evaluation is shown in Fig. 4. Since quadrature algorithms are heavily parameterized, partial evaluation can help take advantage of the problem-solving context involved in collecting performance data.

4.1. The quadrature routines

Our collection of integration routines consists of two main flavors — (i) Type 1: 104 routines that integrate functions defined in a symbolic form, and (ii) Type 2: 20 routines that integrate from points sampled in the domain of integration. A more detailed description of the Type 1 routines is given in Fig. 5.

The libraries from which the routines are obtained are

```
int pow(int base, int exponent) {      int pow2(int base) {
  int prod = 1;                          return (base * base)
  for (int i=0;i<exponent;i++)         }
    prod = prod * base;
  return (prod);
}
```

Fig. 4. Illustration of the partial evaluation technique. A general purpose power function written in C (left) and its specialized version (with exponent statically set to 2) to handle squares (right). Such specializations are performed automatically by partial evaluators such as C-Mix.

QUADPACK, NAG, IMSL, PORT, SLATEC, JCAM and the collected algorithms of the ACM (TOMS). The automatic quadrature routines are those in which the user specifies the required accuracy and the algorithm attempts to achieve it. The non-automatic ones, on the other hand, use a preset number of nodes and so cannot guarantee user accuracy constraints. Most of the Type 1 routines are variations on a single family of algorithms. For example, DQAG and QAG are both automatic adaptive integrators and handle many non-smooth integrands using Gauss–Kronrod formulas. The former produces double precision results while QAG's outputs are in single precision. Also, both of these routines dynamically select among other routines considered in this study, such as QK15, QK21, QK31, QK41, etc. which are themselves non-automatic Kronrod routines with a varying number of nodes. In other words, most of the automatic routines are actually 'polyalgorithms' based on non-automatic routines. While most of the routines were declared as general purpose modules, some of them require a special integrand such as weight functions, oscillatory or singular integrands. For example, QAWO is an automatic adaptive integrator for integrands with oscillatory (sine or cosine) functions and QAWS is one for functions with explicit algebraic and/or logarithmic endpoint singularities. There are other routines that use transformations, Newton–Cotes quadrature, Clenshaw–Curtis quadrature, monotone stable formulas, Patterson's quadrature formulas and differential equation solvers. It can be seen from Fig. 5 that the number of automatic algorithms far outnumber the non-automatic routines. Also, there are more algorithms available for integration over finite intervals than for infinite intervals.

Of the 20 Type 2 routines, 14 are automatic routines and 6 are non-automatic routines. Most of these routines evaluate the integral by approximating the data points by representations such as piecewise polynomials, overlapping parabolas, cubic splines, Hermite functions and B-splines. The quality of the answer is therefore dependent on the efficiency of the approximation technique.

4.2. Test problems

We have used a wide variety of integrands, most of them with special properties. The total number of integrands used in this study was 286. The integrals were selected so that they exhibit interesting or common features such as smoothness (or its absence), singularities, discontinuities, peaks, and oscillation. Some of the functions were selected so that they satisfy the special considerations on which some algorithms are designed. For example, routine QDAWO requires that its argument contain a sine or a cosine. Most of the functions are parameterized which generates families of integrands with similar features and characteristics, this aids in the generalization of the system. Examples of

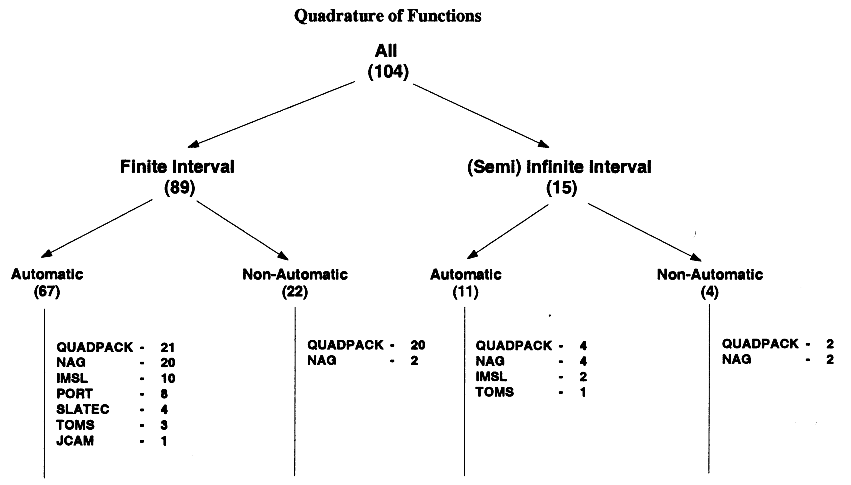

Fig. 5. The taxonomy of the Type 1 routines in GAUSS. The numbers in parentheses denote the number of routines below each node in the tree. QUADPACK, NAG, etc. denote the libraries from which the modules are obtained.

Lyness's Integral

$$I(\lambda) = \int_1^2 \frac{0.1}{(1-\lambda)^2+0.01} dx$$

Piessens' integrals

$$\int_0^1 x^\alpha log(\tfrac{1}{x}) dx = \frac{1}{1+\alpha^2}$$
$$\int_0^\infty \frac{x^{\alpha-1}}{(1+10x)^2} dx = \frac{(1-\alpha)\pi}{10^\alpha sin(\pi\alpha)}$$
$$\int_0^\infty x^2 e^{-2^{-\alpha} x} dx = 2^{3\alpha+1}$$
$$\int_0^1 \mid x - 1/3 \mid^\alpha dx = \frac{(2/3)^{(\alpha+1)}+(1/3)^{(\alpha+1)}}{\alpha+1}$$

A problem with one peak is

$$\int_1^2 \frac{10^\alpha}{(x-\lambda)^2+10^{2\alpha}} dx$$

while one with three such peaks is

$$\int_1^2 \sum_{i=1}^3 \frac{10^\alpha}{(x-\lambda_i)^2+10^{2\alpha}} dx$$

Fig. 6. Examples of test integrands.

integrands are given in Fig. 6. For each routine and each applicable integrand, experiments were conducted with varying requirements on the relative error accuracy ϵ_r. The number of accuracy levels was 10 and the strictest error requirement used was 10^{-8}. The testbed is described in more detail in Ref. [19].

4.3. Cost of GAUSS

One important aspect of the evaluation involves the (automatic) determination of features necessary to drive algorithm recommendation. Some of the more useful features that are specially relevant to this problem domain (and which affect the applicability of algorithms) are: whether the integrand can be expressed as $w(x)f(x)$ with several desirable features (such as $w(x)$ being one of several weight functions and f smooth on $[a,b]$), whether the integrand is smooth in $[a,b]$, whether we know the location of the singularities of f, whether we know the location (if any) of singularities of f', whether f has end-point singularities, whether f exhibits an oscillatory behavior of non-specific type, and whether the range of integration $[a,b]$ is finite, semi-infinite or infinite. We have explored the determination of such features by symbolic analysis (using the transformation rules interface to packages like *Mathematica*), by pattern recognition techniques (e.g. using invariant moments to characterize oscillatory behavior etc.), or using contextual information about the problem. One of the goals of our study is to see if such information is useful, and if it is worthwhile to explore efficient ways to obtain this information automatically.

4.4. User interface to GAUSS

There are two interfaces to GAUSS: that of the knowledge engineer and that of the end-user. The knowledge engineer interface is modeled after the PROOGOL execution environment and is the facility by which performance data and background knowledge are modeled in logic form. This stage also addresses the induction of rules and their subsequent incorporation (back) into the knowledge base. The end-user interface is one where the rules are fired, once the user specifies the features of his numerical integration problem. After each recommendation, the user can inspect the rules to obtain phenomenological explanations for why a particular algorithm was selected. More detailed designs for these interfaces are presented in Ref. [11].

4.5. Results

The rules obtained in the GAUSS system correspond to accepted general knowledge about numerical integration routines. It was observed, for example, that the adaptive algorithms use fewer function evaluations to achieve high-accuracy results than their non-adaptive counterparts; conversely, they use more evaluations to meet low-accuracy constraints. A high accuracy adaptive algorithm was recommended for an oscillating integrand. This could possibly be due to the fact that in an oscillating function, subdivisions are spread over the entire domain of integration and hence a smaller number of subdivisions are required to achieve a fairly high degree of accuracy. Conversely, integrands with singularities or peaks are more amenable to low and medium accuracy adaptive routines. There are many more such observations, and we have reproduced only the most interesting here. Finally, GAUSS has helped identify 'redundant' algorithms, i.e. algorithms which perform almost exactly the same for the test functions considered in this work. For example, the rules selecting the algorithms DPCHIA, DCSITG, and DCSPQU contained the same antecedents. In other words, the conditions under which these algorithms would be recommended are identical. All of these are Type 2 routines-DPCHIA evaluates the given integral using piecewise cubic Hermite functions, DCSITG evaluates the integral of a cubic spline and DCSPQU also uses spline interpolation. In addition, these routines yield the best overall performance for problems of Type 2. One possibility for this result is that quadrature algorithms using fitted functions often utilize similar mechanisms for obtaining data-dependent break points (for use in Hermite cubics or as knots in spline-based methods).

All selections made by GAUSS are *valid* (a selection is considered *invalid* if the method is inappropriate for the given problem or if any of the parameters do not apply correctly to the method). In prior research, accuracy of algorithm selection was measured as the fraction of the valid selections that are also correct (a correct selection is one where the selected method and parameters does result in solutions satisfying the requested criteria) [23]. For a discount factor $\gamma = 0.9$ and the penalty ratio equal to 10, GAUSS selected the best algorithm for 87% of the test problems, selected the second best algorithm for 7% of the cases, and was in error for only 3% of the problems (the remaining 3% account for the cases where the system did not make the selection of either the first or second best algorithm).

The design of GAUSS enables both input and output generalization. Input generalization refers to the ability to generalize to problems not previously encountered by the system. This facility is achieved by using relational descriptions of problem features as the primary scheme of representation. Output generalization refers to the ability to generalize to new algorithms not previously considered. This facility is attained as follows: Consider that a bank of rules has been constructed with a certain set of algorithms. When a new algorithm is added, we utilize the old rule bank to bootstrap the Q-learning algorithm. To understand how the bootstrapping process works, it is useful to think of the bank of rules mined as a compaction of performance information. When new information is obtained, the bank of rules guides the incorporation of new performance data using the Q-tree (which provides a structural organization of all the rules). This serves two useful purposes: (i) reduce the number of states to be explored by the algorithm and (ii) faster convergence to the optimal Q-tree. Furthermore, since the description language is sufficiently rich to abstract away the specific details of the problems and algorithms, our approach is inherently scalable.

5. Extensions

There are several possible extensions to the work presented here:

- The representational formalism can be enhanced to include so-called description logics or terminological logics that provide more functionality than the logic formalism employed here. In particular, this will permit constructive induction where rules of the form ‘If two or more of these features are present, then use algorithm X’ can be induced. The scheme presented in this paper is a proper subset of this language.
- The study can be extended to allow different algorithms applied to different sub-intervals in the domain of integration. The penalty ratio, whose effect is to provide negative feedback about the efficacy of an algorithm, aids in this design. It also allows for fine-tuning the Q-learning step with respect to a certain set of states (and problem features). For example, if a given integrand is singular in a particular region but oscillatory in another, the penalty ratio can be used to overcome the drawbacks of selecting one single algorithm for the entire domain of integration.
- Online recommendation of algorithms can be enhanced to (i) first model the dynamic selection of nodes by a general purpose adaptive code such as QAGS [18], which can then (ii) provide insight into the relevant features affecting (subsequent) algorithm selection.

6. Concluding remarks

The eventual success and acceptance of algorithm recommender systems rely on the expressiveness of their representation(s) and their ability to reason efficiently (and accurately) with such representations. We have shown how GAUSS achieves these objectives by (i) a direct control over the data-generation process, (ii) a rich representation language for recommendation rules and domain-specific context information, and (iii) the use of ILP for generalization. The online capability of systems like GAUSS can also help in realizing adaptivity, exploratory behavior, and control systems in next generation scientific software, as envisioned in Ref. [22].

Acknowledgements

We acknowledge suggestions from three anonymous reviewers whose comments greatly improved the presentation and readability of this paper. This research was supported in part by the National Science Foundation (CCR-9311486; EIA-9984317), the Defense Advanced Research Projects Agency through the Army Research Office (DAAH04-94-G-0010), Intel Corporation, and the Purdue Research Foundation.

References

[1] Acton FS. Real computing made REAL. Princeton, NJ: Princeton University Press, 1996.
[2] Baier R, Gluck R, Zochling R. Partial evaluation of numerical programs in FORTRAN, Proceedings of the ACM SIGPLAN workshop on partial evaluation and semantics-based program manipulation. New York: ACM Press, 1994. p. 119–32.
[3] Berlin A, Weise D. Compiling scientific code using partial evaluation. IEEE Comput 1990;23(12):25–37.
[4] Ronald F. Boisvert. The NIST guide to available mathematical software. URL: http://gams.nist.gov, 1996.
[5] Bratko I, Muggleton S. Applications of inductive logic programming. Commun ACM 1995;38(11):65–70.
[6] Davis PJ, Rabinowitz P. Methods of numerical integration. 2nd ed. Orlando, FL: Academic Press, 1984.
[7] de Boor C. CADRE: an algorithm for numerical quadrature. In: Rice

JR, editor. Mathematical software. New York: Academic Press, 1971. p. 417–49.

[8] de Raedt L, Blockeel H. Using logical decision trees for clustering, Proceedings of Seventh International workshop on inductive logic programming. New York: Springer, 1997. p. 133–41.

[9] Dzeroski S, Blockeel H, de Raedt L. Relational reinforcement learning, Proceedings of ILP'98. New York: Springer, 1998. p. 11–22.

[10] Gallopoulos E, Houstis EN, Rice JR. Computer as thinker/doer: problem-solving environments for computational science. IEEE Comput Sci Engng 1994;1(2):11–23.

[11] Houstis EN, Catlin AC, Rice JR, Verykios VS, Ramakrishnan N, Houstis CE. PYTHIA-II: a knowledge/database system for managing performance data and recommending scientific software. ACM Trans Math Software 2000;26(2):227–53.

[12] Jones ND. An introduction to partial evaluation. ACM Comput Survey 1996;28:480–503.

[13] Kaelbling LP, Littman ML, Moore AW. Reinforcement learning, a survey. J Artif Intell Res 1996;4:237–85.

[14] Kalagnanam J, Simon HA, Iwasaki Y. The mathematical bases of qualitative reasoning. IEEE Expert 1991;11(4):11–19.

[15] Mitchell TM. Machine learning. New York: McGraw-Hill, 1997.

[16] Muggleton S. Inverse entailment and PROGOL. New Generation Comput 1995;13:245–86.

[17] Nazareth JL. Multialgorithms for parallel computing. Proceedings of the Pacific Numerical Analysis Seminar, 1997.

[18] Piessens R, de Doncker-Kapenga E, Uberhuber CW, Kahaner DK. Quadpack. New York: Springer, 1983.

[19] Ramakrishnan N. Recommender systems for problem solving environments. PhD Thesis, Department of Computer Sciences, Purdue University, 1997.

[20] Rice JR. A metaalgorithm for adaptive quadrature. J ACM 1973;22:61–82.

[21] Stonebraker M, Rowe LA. The design of POSTGRES. SIGMOD Record 1986;15:340–55.

[22] Trefethen LN. Predictions for scientific computing fifty years from now. Technical Report NA-98/12, Oxford University Computing Laboratory, 1998.

[23] Weerawarana S, Houstis EN, Rice JR, Joshi A, Houstis CE. PYTHIA: a knowledge based system to select scientific algorithms. ACM Trans Math Software 1996;22:447–68.