1995

PYTHIA: A Knowledge Based System for Intelligent Scientific Computing

Sanjiva Weerawarana

Elias N. Houstis
Purdue University, enh@cs.purdue.edu

John R. Rice
Purdue University, jrr@cs.purdue.edu

Anupam Joshi

Catherine E. Houstis

Report Number:
95-044

PYTHIA: A Knowledge Based System for Intelligent Scientific Computing

S. Weerawarana, Elias N. Houstis
John R. Rice, Anupam Joshi, and
Catherine E. Houstis
Computer Sciences Department
Purdue University
West Lafayette, IN 47907

CSD-TR-95-044
June, 1995

PYTHIA: A Knowledge Based System for Intelligent Scientific Computing

Sanjiva Weerawarana, Elias N. Houstis, John R. Rice, Anupam Joshi
Purdue University
and
Catherine E. Houstis
University of Crete

Categories and Subject Descriptors: I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems; G.1.8 [**Numerical Analysis**]: Partial Differential Equations

General Terms: Knowledge Based Systems

Additional Key Words and Phrases: Computational Intelligence, Knowledge Based Systems, Partial Differential Equations, Performance Evaluation, Problem Solving Environments

1. ABSTRACT

Domain specific Problem Solving Environments (PSEs) are the key new ingredients that will aid in the widespread use of Computational Science & Engineering (CS&E) systems. Each PSE consists of a well defined library that supports the numerical and symbolic solution of certain mathematical model(s) characterizing a specific discipline, together with an easy to use software environment. This environment should ideally interact with the user in a language "natural" to the associated discipline, and provide a high level abstraction of the underlying, computationally complex, model. However, it appears that almost all extant PSEs assume that the user is familiar with the specific functionality/applicability of the PSE. Their primary design objective is to support some form of high level programming with predefined state-of-the-art algorithmic infrastructure. As the functionality of these systems increases, the user is expected to make complex decisions in the parametric space of the algorithmic infrastructure supported by the PSE. In this paper we describe a knowledge based system, PYTHIA, to automate this decision making process and aid in providing a high level abstraction to the user. Specifically, PYTHIA addresses the problem of (parameter, algorithm) pair selection within a scientific computing domain assuming some minimum user specified computational objectives and some characteristics of the given problem. PYTHIA's framework and methodology is general and applicable to any class of scientific problems and

Work supported in part by AFOSR award 91–F49620, NSF awards CCR 86–19817, CCR 92-02536, and ASC 9404859.

Authors' addresses: S. Weerawarana, E.N. Houstis, J.R. Rice and A. Joshi: Department of Computer Sciences, Purdue University, West Lafayette, IN 47907, USA; C.E. Houstis: Department of Computer Science, University of Crete, Heraklion, Greece.

solvers. For the purpose of initial experiments and validation, we have selected to develop and apply PYTHIA in the context of the Parallel ELLPACK PSE [Houstis et al. 1990]. In the Parallel ELLPACK environment there are many alternatives for the numerical solution of elliptic partial differential equations (PDEs). Moreover, these solvers normally require a number of parameters that the user must specify in order to obtain a solution within a specified error level while satisfying certain resource (e.g., memory and time) constraints. PYTHIA attempts to match the characteristics of the given problem with those of some PDE in an existing problem population. Then, it uses performance profiles of the various solvers applied to a population of *a priori* determined elliptic PDE problems to predict the appropriate method given user specified error and solution time bounds. The profiles are automatically generated out of known sets of performance data for each module of the Parallel ELLPACK library.

2. INTRODUCTION

A Problem Solving Environment (PSE) is a computer system that provides all the computational facilities necessary to solve a target class of problems [Gallopoulos et al. 1994]. These features include advanced solution methods, automatic or semiautomatic selection of solution methods, and ways to easily incorporate novel solution methods. Moreover, PSEs use the language of the target class of problems and provide a "natural" user interface, so users can run them without specialized knowledge of the underlying computer hardware or software. By exploiting modern technologies such as interactive color graphics, powerful processors, and networks of specialized services, PSEs can track extended problem–solving tasks and allow users to review them easily. Overall, they create a framework that is all things to all people: they solve simple or complex problems, support rapid prototyping or detailed analysis, and can be used in introductory education or at the frontiers of science. PSEs will play a vital role in making HPC based scientific computing systems accessible to application scientists who are not experts in computer science [Gallopoulos et al. 1994].

In order to develop PSEs that are truly easy to use they need to provide the user with a high level abstraction of the complexity of the underlying computational facilities. The user cannot, and should not, be expected to be well versed in selecting appropriate numerical, symbolic and parallel systems, along with their associated parameters, that are needed to solve a problem. Various existing techniques from artificial and computational intelligence have to be used, and new ones developed, for these decisions to be automated. These include techniques to select appropriate numerical solution methods given a problem, to select appropriate machines and configuration on which to solve the problem, to adaptively refine meshes on which the problem is solved, to dynamically distribute the problem and solve it cooperatively, *inter alia*. Techniques to combine natural language interpretation with visual recognition, to automatically analyze properties of computed solutions, to integrate disparate solution strategies and platforms using agents will also have to be developed. Our group is working on these problems under the PYTHIA project.

In this paper, we address the issue of intelligence within the specific class of scientific computing applications that can be described by a mathematical model involving partial differential equations (PDEs) defined on general geometric regions.

The design objectives and architecture of such PSEs are described in [Weerawarana 1994; Weerawarana et al. 1994; Houstis et al. 1990]. The goal of these PSEs is to assist the user to carry out the numerical solution of these models and visualize their solutions. Depending on the mathematical characteristics of the PDE models, there are "thousands" of numerical methods to apply, since very often there are several choices of parameters or methods at each of the several phases of the solution. On the other hand, the numerical solution must satisfy several objectives, primarily involving error and hardware resource requirements. It is unrealistic to expect that engineers and scientists will or should have the deep expertise to make "intelligent" selections of methods, their parameters and computational resources that will satisfy their computational objectives. Thus, we propose to build a computational advisor or expert system to make the "right", or at least "good", selections. Unfortunately, the knowledge about PDE solvers is vast. Fortunately, it is classified according to some general characteristics of the mathematical models.

The main objective of this work is to develop a paradigm for selecting pairs (method, machine) and their parameters for the numerical solution of PDE models. For verification and demonstration purposes, we have selected the ELLPACK PDE library [Rice and Boisvert 1985] whose modules can be combined to define a large number of PDE solvers. Without loss of generality, the methodology is implemented for PDE problems solvable by the ELLPACK solvers. The applicability of these solvers is restricted to single equation PDE boundary value problems of the form

$$\begin{aligned} Lu &= f \text{ on } \Omega, \\ Bu &= g \text{ on } \partial\Omega \end{aligned} \tag{1}$$

where L is a second order linear elliptic operator, B is a differential operator involving up to first order partial derivatives of u, and Ω is a bounded open region in 2– or 3–dimensional space. The PYTHIA expert system is based on a database (DB) of performance data for each PDE solver using a set $\mathcal{H}$ of hardware platforms and for an *a priori* defined parameterized population $\mathcal{P}$ of elliptic problems of type (1). The knowledge base (KB) of PYTHIA consists of *a priori* defined rules and facts together with *a posteriori* ones derived from the known data (DB, $\mathcal{H}$, $\mathcal{P}$) using the exemplar–based knowledge acquisition and learning approach [Bareiss 1986]. PYTHIA's inference strategy consists of matching the characteristics of the given PDE problem with those of one PDE problem P_0 in $\mathcal{P}$ or a subclass $\mathcal{S} \in \mathcal{P}$. Then, the known performance data for P_0 or $\mathcal{S}$ and the set of applicable solvers (U_a) are used to identify one (or several) solver(s) and machine pair(s) that satisfies the user's performance objectives (e.g., 2% percent error and turn around time of less than 20 minutes).

This paper is organized as follows: Section 3 discusses some related work in solving the algorithm selection problem in the domain of PDE solvers. Section 4 presents an overview of the computational paradigm of PYTHIA. The PYTHIA knowledge bases are described in Section 5. The inference algorithm implemented to produce advice is given in Section 6. Section 7 discusses the methodology used to handle the uncertainty present in the knowledge available to PYTHIA. Section 8 discusses an alternate approach to identifying problem classes using backpropagation–based neural networks. The overall PYTHIA architecture and its implementation is discussed in Section 9. Finally, in Section 10 we present a performance evaluation

of PYTHIA from several points of view.

3. RELATED WORK

There have been several attempts at developing expert systems for assisting in various aspects of the PDE solution process. In [Rice 1976], Rice describes an abstract model for the algorithm selection problem, which is the problem of determining a selection (or mapping) from the problem space (or its more accessible counterpart, the feature space) to the algorithm space. The "best" selection is then the mapping that is better (in the sense of having a better performance indicator in the performance measure space) than other possible mappings. Using this abstract model Rice describes an experimental methodology [Rice 1979] for applying the abstract model in the performance evaluation of numerical software.

In [Moore et al. 1990] the authors describe a strategy for the automatic solution of PDEs at a different level. They are concerned with the problem of determining (automatically) a geometry discretization that leads to a solution guaranteed to be within a prescribed accuracy. The refinement process is guided by various "refinement indicators" and refinement is affected by one of three mesh enrichment strategies. At the other end of the PDE solution process, expert systems can be used to guide the internals of a linear system solver. For example, the expert system described in [König and Ullrich 1990] applies self-validating methods in an economical manner to systems of linear equations. Other expert systems that assist in the selection of an appropriate linear equation solver for a particular matrix are also currently being developed.

In [Dyksen and Gritter 1989; Dyksen and Gritter 1992], Dyksen and Gritter describe an expert system for selecting solution methods for elliptic PDE problems based on problem characteristics. Problem characteristics are determined by textual parsing or with user interaction and are used to select applicable solvers and to select the best solver. This work differs significantly from ours; our work is based on using performance data for relevant problems as the algorithm selection methodology whereas Dyksen and Gritter's use rules based solely on problem characteristics. We argue that using problem characteristics solely is not sufficient because the characterization of a problem includes many symbolically and *a priori* immeasurable quantities (such as the degree of variation present in the solution of some problem), and also because in practice software performance depends not only on the algorithms used, but very much on the efficient implementation of those algorithms as well.

[Kamel et al. 1993] describes an expert system called ODEXPERT for selecting numerical solvers for initial value ordinary differential equation (ODE) systems. ODEXPERT uses textual parsing to determine some properties of the ODEs and performs some automatic tests (e.g., a stiffness test) to determine others. Once all the properties are known, it uses information from its knowledge base about available ODE solution methods (represented as a set of rules) to recommend a certain method. After a method has been determined, it selects a particular implementation of that method based on other criteria and then generates source code (FORTRAN) that the user can use. If necessary, symbolic differentiation is used to generate code for the Jacobian as well.

PDE problem characteristics			
Operator	Boundary Conditions	Functions	Domain
Poisson	Dirichlet	Smooth	Rectangle
Laplace	Neumann	Oscillatory	Square
Helmholtz	Mixed	Discontinuous	Non–rectangular
Self–adjoint	Homogeneous	Wave Front	Convex
Homogeneous	Linear	Singular	Nonconvex
Separable	Nonlinear	Peak	Geometric singularities
u_x, u_y, u_z			Union of domains
u_{xy}, u_{xz}, u_{yz}			Reentrant corners
Constant coefficients			

Table I. Examples of PDE problem characteristics for elliptic operators.

4. AN OVERVIEW OF PYTHIA

In this section we present a formal definition of the selection problem that PYTHIA attempts to solve and the reasoning approach or computational intelligence paradigm utilized.

4.1 The PDE Solver Selection Problem

In a PSE for PDE based applications the user starts by defining the components of the corresponding mathematical problem. In the context of //ELLPACK this consists of specifying the model (1) and its domains Ω in some high level representation. In an ideal scenario the system extracts the type of the differential operator, boundary conditions and the mathematical behavior of its input functions (i.e., coefficients and right sides of the equations) and its output, the exact solution. Table I lists examples of characteristics for each component of the PDE problem. Each of components of the PDE problem (1) can have several characteristics so that the number of combinations is very large, over 2,500 in Table 1. Still more characteristics are used in [Rice and Boisvert 1985]. In the first implementation of PYTHIA much of this information is assumed to be provided by the user. The automatic generation of most of this information is still a research issue that we hope to address in the future. Moreover, we assume that the user requests a solution that is within some error bound and produced within some time interval.

Two of the important design objectives of a PSE for PDE applications are first, to provide advice in selecting the mathematical model that "best" describes the application and, second, to select a (method, machine) pair to numerically solve the PDEs. In this work we only address the second objective. Specifically, PYTHIA attempts to determine an optimal, or at least good, strategy for solving a PDE problem of type (1) under *accuracy* and *time* constraints imposed by the user. From among the applicable methods in library of PDE solvers, PYTHIA must identify pairs (grid, method) and (machine, configuration) that minimize the computation cost, in other words, it determines the four "variables" so that

$$\rho(grid,\ method,\ configuration,\ machine)$$

is minimized subject to the constraints $\|u - u_h\| \leq \epsilon,\ t \leq \mathcal{T}$. Here u_h denotes the approximation of the exact solution u, ϵ and $\mathcal{T}$ are the specified accuracy and time constraints, respectively, and ρ is the computational cost measured, say, in some monetary terms. In this work we assume a fixed sequential machine with

known configuration bounds. The case of heterogeneous hardware platforms will be addressed later.

4.2 PDE Problem Characteristics

It is clear from the previous discussion that the goal of finding the "best" (method, machine) pair for a given PDE problem and computational objective depends on the PDE problem's characteristics. This information is quantified in the PDE problem *characteristic vector*

$$\underline{p} = \left(\overline{O}, \overline{BC}, \overline{F}, \overline{D}\right)$$

where each of the subvectors consists of the, *a priori* defined, characteristics of each PDE problem component, see Table I. The value (α) of each entry of the subvectors ranges in the interval [0,1] and indicates the presence level of each characteristic (e.g., $\alpha = 0$ means pure absence and $\alpha = 1$ pure presence). For type characteristics (for example, Laplace and Dirichlet), we assign binary values to the related entries of the characteristic vector. The type characteristics can be easily extracted by parsing the representation of (1) and can be done within the PSE environment (e.g., //ELLPACK). The identification of the characteristics of the mathematical functions is a much harder problem. Depending on their assumed representation, this information can be extracted by symbolic, numeric and imagistic (computational vision) techniques. Developing such techniques is a challenging research problem that we are addressing in ongoing work. For the work presented here, it is expected that the user provides some of this information in a question and answer session. The exact codification of the PDE problem characteristics vector in PYTHIA is given in Section 5.1.

4.3 Population of Elliptic PDEs

The first step of PYTHIA's reasoning approach is the identification of an *a priori* defined PDE problem or class of problems whose characteristic vector is "close" to that of the problem specified by the user. The success of this approach thus relies greatly on having available a reasonably large population of various PDE problems about which some performance information is known. For the class of linear second order elliptic PDEs we are currently using the population created in [Rice et al. 1981] for use in the evaluation of numerical methods and software for solving PDEs. It consists of fifty-six linear, two-dimensional elliptic PDEs defined on rectangular domains. Forty-two of the problems are parametrized which leads to an actual problem space of more than two-hundred and fifty PDEs. Many of the PDEs were artificially created so as to exhibit various characteristics of interest; the others are taken from "real world" problems in various ways. The population has been structured by introducing measures of complexity of the operator, boundary conditions, solution and problem. Figure 1 shows an example problem from the PDE population. The actual implementation and use of this population is reported in [Boisvert et al. 1979].

The population also contains information about the type characteristics of the problems. This information is encoded as a bit-string and is accessible via the ELLPACK Performance Evaluation System (PES) [Boisvert et al. 1979]. These properties are transferred to the PYTHIA characteristic vector representation by

PROBLEM #28	$(w\,u_x)_x + (w\,u_y)_y = 1,$ where $w = \begin{cases} \alpha, & \text{if } 0 \le x, y \le 1 \\ 1, & \text{otherwise.} \end{cases}$
DOMAIN	$[-1,1] \times [-1,1]$
BC	$u = 0$
TRUE	unknown
OPERATOR	Self-adjoint, discontinuous coefficients
RIGHT SIDE	Constant
BOUNDARY CONDITIONS	Dirichlet, homogeneous
SOLUTION	Approximate solutions given for $\alpha = 1, 10, 100$. Strong wave fronts for $\alpha \gg 1$.
PARAMETER	α adjusts size of discontinuity in operator coefficients which introduces large, sharp jumps in solution.

Fig. 1. A problem from the PDE population.

an automated procedure explained in Section 9. Characteristics not identified in the ELLPACK PES are identified by the characteristic extraction component of PYTHIA.

PYTHIA's reasoning approach requires the identification of previously seen problems similar to the present one. One possible implementation of this is to follow a pure exemplar based approach. In other words, the characteristics of the problem are compared to all the problems in the database, and the one closest to it (according to some similarity metric) is identified. This performance data of this problem can then be used to predict the method that should be used to solve the new problem. One clearly identifiable drawback of such an approach is the amount of time needed to find the match to the new problem, specially on large databases. Moreover, it is possible that the database contains no problems close enough to the new one.

Another possible approach is to follow a class based system. All the problems in the database can be grouped according to some criterion into classes, and a method that works best (on average) for each member of the class can be declared as suitable for all members. Thus, the new problem need be matched only against a few classes to see the one to which it belongs. This approach is problematic too, since it assumes that given a class, a single method will work "well enough" for all its members.

In the present implementation of PYTHIA, we use a hybrid of these approaches. The search for similar problems is reduced by first comparing the given problem characteristic vector $\underline{p}$ with vectors that represent some common classes of PDEs. The closest exemplar within this class is then used to make predictions about the new problem. We define $\underline{c}$, the characteristic vector of a class c, as the element–by–element average of the characteristic vectors $\underline{p}$ of all class members. That is, the i–th element of the characteristic vector of problem class c is defined to be:

$$c_i = \frac{1}{|c|} \sum_{p \in c} p_i.$$

Class identification and performance evaluation of PDE solvers is part of the knowledge acquisition for PYTHIA. The class c may be determined by certain simple characteristics (e.g., Helmhotz equation) or empirically by identifying all

those problems $p \in P$ whose p_i components range in certain subintervals of [0,1]. Once a class is identified the following information is generated and stored in the KB: i) the list of member problems, ii) the average characteristic vector, and iii) applicable solvers. The representation of problem features as a vector allows us to identify similar problems or classes (a single problem can be viewed as a class with one element) by measuring the distance between the two characteristics vectors in some norm.

Our hybrid solution, while quite efficient, is dependent on how well classes are defined. If the classes are not defined in a robust fashion, then it is possible to have a situation where a new problem beongs to one class, and its closest exemplar (globally) belongs to another. This can lead to erroneous predictions. We are working on methods to improve class definitions, using neuro–fuzzy techniques [Joshi et al. 1994; Ramakrishnan et al. 1995] to automatically infer class boundaries from selected examples.

4.4 Performance Knowledge for Elliptic PDE Solvers

For PYTHIA to infer the "best" solution path for a given PDE problem, it needs to know the ranking of the applicable methods within the class of problems that have characteristics similar to the user's problem. This ordering (ranking) is a function of the user's computational objectives (accuracy and time levels). The generation of this ordering is done by applying statistical techniques to the performance curves or profiles such as *accuracy vs. degrees of freedom (dofs)*, [1] *dofs vs. time*, and *accuracy vs. time* for each (method, problem) pair. These profiles are generated by linear least squares approximations to raw performance data. These three types of performance profiles are generated by the performance evaluation system (PES) system [Boisvert et al. 1979]. For example, Figure 2 shows the raw performance data and Figure 3 displays the *dofs vs. time* profiles for seven //ELLPACK solvers on a problem from the PDE population $\mathcal{P}$. Finally, in Figure 4 we present a ranking of the methods in Figure 3 with respect to *dofs vs. time* profiles for a certain class and for the accuracy level of 0.05%.

5. THE PYTHIA KNOWLEDGE BASES

The PYTHIA knowledge bases consist of 1) *a priori* rules specified by a (human) expert, 2) facts, and 3) rules generated from the performance knowledge for the PDE solvers considered.

5.1 Facts

The knowledge about each problem in the population $\mathcal{P}$, the various identified subclasses and the solvers is referred to as *facts* of the PYTHIA knowledge base.

5.2 Solver Facts

Recall that many solvers are created by composing various modules in the //ELLPACK library so we have facts about their compatibility. In addition, we have facts

[1] The number of degrees of freedom that are used in the discretized equation is the number of linear equations that are present in the system of equations produced by the operator discretizer. For first order operator discretization schemes this is approximately the same as the number of points in the grid used to discretize the domain.

Problem 10
Parameter Set 2 ($a = 50.0, b = 0.5$)
Method b4p2unix/1/14/46/

nx	ny	h	$nunk$	$rerr$	$rerrmax$	$errl2$	$resmax$
5	5	$0.25E+00$	9	$0.78E+00$	$0.80E+00$	$0.16E+00$	$0.15E+02$
9	9	$0.12E+00$	49	$0.23E+00$	$0.24E+00$	$0.16E-01$	$0.84E+01$
17	17	$0.62E-01$	225	$0.53E-01$	$0.58E-01$	$0.61E-02$	$0.57E+01$
33	33	$0.31E-01$	961	$0.13E-01$	$0.97E-02$	$0.31E-02$	$0.30E+01$
65	65	$0.16E-01$	3969	$0.32E-02$	$0.24E-02$	$0.16E-02$	$0.15E+01$

$resmxr$	$resl2$	$solmax$	nit	mem	tt	$t1$	$t2$	$t3$
$0.26E+04$	$0.12E+02$	$0.28E+00$	0	387	0.02	0.00	0.02	0.00
$0.32E+03$	$0.12E+02$	$0.81E-01$	0	2079	0.00	0.00	0.00	0.00
$0.22E+02$	$0.13E+02$	$0.66E-01$	0	14391	0.15	0.03	0.00	0.12
$0.15E+02$	$0.14E+02$	$0.63E-01$	0	106983	1.55	0.12	0.00	1.43
$0.26E+02$	$0.14E+02$	$0.63E-01$	0	822087	18.73	0.50	0.02	18.22

Fig. 2: A sample of raw performance data generated by the performance analyzer. These numbers are for solving problem # 10 with parameter set # 2 ($a = 50.0$, and $b = 0.5$) using the "method" b4p2unix/1/14/46 with 5×5, 9×9, 17×17, 33×33, and 65×65 grids. The method string encodes the machine, operating system and solver (here 5–point star discretization, as–is indexing, and band Gaussian elimination).

about the types of PDE problems to which solvers can be applied. This information allows PYTHIA to automatically form legal PDE solvers out of software parts. In the case of elliptic solvers, we have four types of modules [Rice and Boisvert 1985]: discretization, indexing, solution and triple.

Problem Facts. This information is the characteristic vector of each PDE problem. For example, the vector

$$\Big(\underbrace{2}_{1}, \underbrace{0,0,0,1,0,0,0,0,1,0,1,0,0}_{2}, \underbrace{0.0,0.0}_{3}\Big)$$

represents the type characteristics of the PDE operator of the problem specified in Figure 1.

The first number indicates that the operator is two dimensional. The second set of numbers indicate that the operator is not Poisson, not Laplace, not Helmholtz, is self–adjoint, does not have constant coefficients, does not have single derivatives, does not have mixed derivatives, is not homogeneous, is linear, is not nonlinear, is elliptic, is not parabolic and is not hyperbolic, respectively. The last two numbers are the subjective measures of the smoothness and local variation properties of the operator. In this study, we have assumed only linear PDE problems and solvers. Usually solving nonlinear problems is reduced symbolically to solving a sequence of linear ones [Weerawarana et al. 1992]. In these cases, PYTHIA can be applied to select the appropriate solver for the linearized PDE problem.

Class Facts. This information consists of the set of member identifiers and the average characteristic vector of the class as defined in Section 4.3. This vector

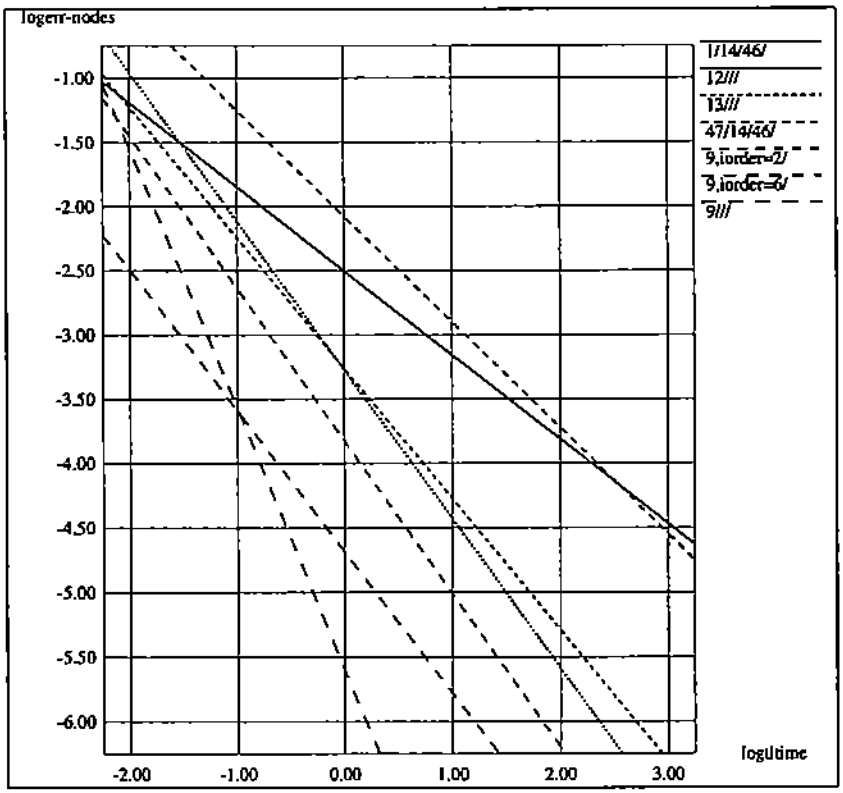

Fig. 3: Performance profiles generated using the knowledge engineering environment of PYTHIA for a single PDE problem. These plots show the (smoothed) relationship between solution error (measured at the nodes of the grid) and the computing time for various solvers. The strings at the upper right are coded identifications of 7 solvers.

method	rank	min.	1st quart	median	3rd quart	max.
b4p2unix/1/14/46/	5.36	2.2E+01	5.1E+01	1.5E+02	2.1E+02	1.7E+03
b4p2unix/12///	5.86	2.0E+01	5.7E+01	2.1E+02	1.7E+03	1.0E+30
b4p2unix/13///	4.27	1.2E+01	4.4E+01	6.0E+01	1.5E+02	1.0E+30
b4p2unix/47/14/46/	1.91	2.4E+00	8.4E+00	2.0E+01	3.2E+01	1.0E+30
b4p2unix/9,iorder=2///	4.86	2.0E+01	5.0E+01	1.4E+02	1.8E+02	1.2E+03
b4p2unix/9,iorder=6///	2.68	4.9E+00	1.1E+01	1.8E+01	7.5E+01	1.0E+30
b4p2unix/9///	3.05	7.3E+00	1.3E+01	2.7E+01	3.6E+01	1.6E+02

Fig. 4: A raw ranking table generated by the stochastic analyzer. This is for a 0.05% error level with the *dofs vs. time* as the performance criteria. The strings for method encode both the machine and solver used. The method 47/14/46 (Hermite Collocation discretizer and Band Gaussian Elimination solver) is the fastest (lowest rank) while method 12///(Dyakanov CG triple module) is the slowest for the specified accuracy level.

represents the centroid of all the member problems.

5.3 Rules

This information is characterized by some level of uncertainty or degree of confidence and it is usually derived by some approximate analysis [Houstis et al. 1988] or provided by an expert. In order to control the level of uncertainty, PYTHIA also develops rules for specified classes of problems and applicable methods from the performance knowledge acquired through actual runs of methods for each member of the class. This approach has the advantage that the learning phase is implemented by updating the performance knowledge either by adding more problems or by including more methods. This section describes the derivation procedure of PYTHIA rules and their representation. Rule defined *a priori* (such as provided by experts) are included using the same representation.

Problem Rules. This information is comprised of the performance profiles for each (method, problem) pair. As we have seen, these are straight line approximations to data points whose coordinates are the values of two performance indicators. The pair of indicators considered in each case is referred to as the performance criteria. The general form of this type of rule is:

For method m on problem p and performance criteria i,
the performance profile is (slope, intercept)

The performance criteria used so far include (accuracy, time) and (accuracy, dofs). In these two cases, the generated rules are of the form:

For method m on problem p, accuracy = f(n)
For method m on problem p, accuracy = g(t)

where the functions f and g are the linear profiles extracted from the experimental data for the (p, m) pair. Figures 5 and 6 show examples of such rules. The knowledge base contains additional information, for example, (memory, defs).

Class Rules. This information provides the average performance of the method for a class of problems. These rules are automatically generated by PYTHIA by applying a stochastic methodology on problem rules or method profiles. To increase the degree of confidence for these rules one has to use classes with larger numbers of members and consider multiple performance criteria. The rule is encoded in the form of a step function where each step is defined by a preset level of (accuracy, time) computational constraints. The general form of this rule is:

For class c and performance criteria i,
the method ranked n-th at error level e is m with confidence c_1 *and* c_2

The same performance criteria as with problem rules are used here. The numbers are generated by the PYTHIA stochastic analyzer and indicate the confidence in the ranking. The first number, c_1, indicates the probability that the method with rank k is in fact better than the method with rank $(k+1)$ and c_2 indicates the probability that the method with rank k is better than the method rank n, where n is the total number of methods applicable to that class of problems. Figure 7

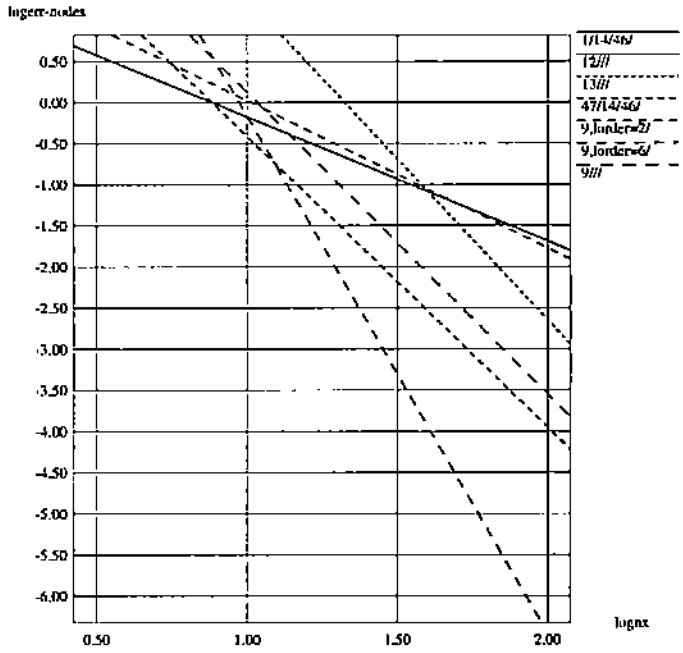

Fig. 5: Knowledge base rules showing relative error as a function of the number of nodes (dofs = nx) in the discretized domain. The solvers are identified (in coded form) at the upper right.

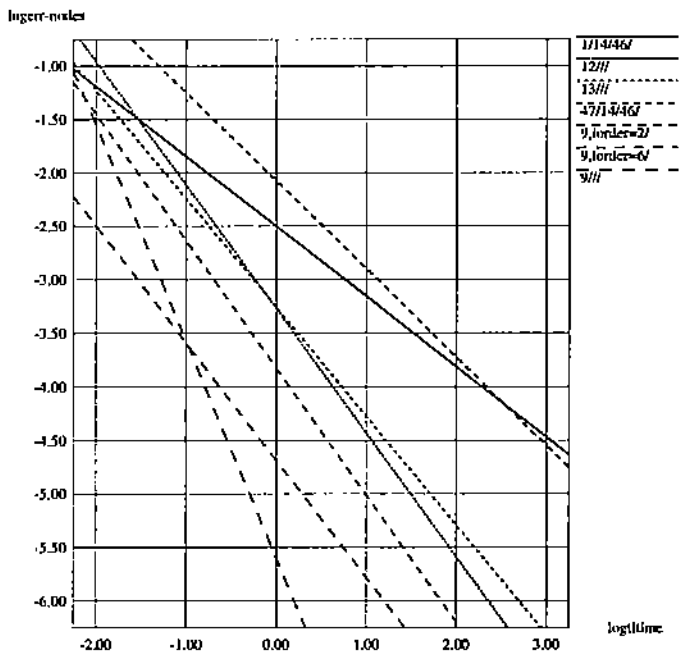

Fig. 6: Knowledge base rules showing relative error as a function of the time taken to achieve that error. The solvers are identified (in coded form) at the upper right.

```
(class-perfdata (name HR.3-10) (rank 1) (method "47/14/46")
   (criteria nx) (errlevel 5pc) (conf1 0.80) (conf2 1.00))

(class-perfdata (name HR.3-10) (rank 2) (method "9,iorder=6//")
   (criteria nx) (errlevel 5pc) (conf1 0.80) (conf2 1.00))

(class-perfdata (name HR.3-10) (rank 3) (method "9//")
   (criteria nx) (errlevel 5pc) (conf1 0.80) (conf2 0.99))
```

Fig. 7: Some class rules that rank the performance of various methods for problem class "HR.3–10" based on the number of nodes (NX) in the discretized domain to achieve a 5% relative error level.

indicates an example of a class rules for a class of problems from the population in [Rice et al. 1981].

User–Specified Rules. This information consists of rules about the performance of methods for various classes of problems. They are products of *a priori* and a posteriori analyses or observations made by experts. In some instances these rules have general acceptance while in many others these rules are subject to differences of opinion. We plan to incorporate rules of this type that are the results of some a posteriori analysis [Houstis and Rice 1982; Dyksen et al. 1988].

6. THE INFERENCE ALGORITHM

There are two significant parts of PYTHIA: the learning process and the selection task or the inference algorithm. In this section we address the latter, the learning or training process is discussed in Section 9. Specifically, we describe the process of combining the facts and rules in the knowledge base to provide a selection (grid, method) pair for the user's problem subject to specified constraints. The input to this phase of the reasoning process includes i) *problem features in the form of a characteristic vector*, ii) *user's computational objectives* (accuracy in % and time constraint), iii) *relative emphasis of achieving the subobjectives of (ii)* (e.g., 70% emphasis on achieving the accuracy requirement and 30% emphasis on achieving the time requirement), iv) *weights for the criteria used in the rules*, and v) *the norm for computing the distance* $d(p, \mathcal{S})$ *and* $d(p, q)$ *with* $p, q \in \mathcal{P}$ *and* $\mathcal{S} \subset \mathcal{P}$.

The inferencing logic in PYTHIA is organized as a Bayesian belief propagation network [Pearl 1988]. A detailed discussion of the techniques used for handling uncertainty is given in the next section. This section describes the overall algorithm without explaining in detail how to compute the various numbers involved.

The goal of the inference algorithm is the determination of the best matching class, and the closest exemplar to be used as a basis for predictions (Figure 8). Once the matching class and exemplar are found then information about their best solvers is extracted from the knowledge base.

The notation $P(\cdot)$ is used to indicate the probability of some event occuring. However, since we are interested in only comparing the relative probabilities of various events, we do not normalize the numbers to be in the range 0..1. This avoids unnecessary computations and make the process faster (see [Pearl 1988]).

```
Compute P(C = c) and sort
Select best class c^0 in the P(C = c) ordering
  Compute P(M = m|C = c^0) using given rule weights and sort
  Compute P(P = p|C = c^0) and sort
  Select best method m^0 in the P(M = m|C = c^0) ordering
    Select closest problem p^0 ∈ c^0 in the P(P = p|C = c^0) ordering
      Check satisfiability of users's objectives with (c^0, m^0, p^0)
      if satisfiable, make predictions and quit else, backtrack
    If not possible to predict with m^0, then backtrack
  End
  If not possible to predict with c^0, then backtrack
End
```

Fig. 8. The PYTHIA inference algorithm.

$P(C = c)$ means the probability that the closest matching class is c. The best class c^0 is defined to be the class appearing first in the $P(C = c)$ ordering sorted in decreasing order. $P(M = m|C = c)$ means the probability that the best method from the set of methods applicable to class c is m and similarly, $P(P = p|C = c)$ means the probability that the best matching exemplar from class c is p. Thus, once these probabilities are calculated using Bayesian belief propagation rules, the algorithm basically applies a backtracking search to find the tuple (c^0, m^0, p^0) so that c^0 is the closest class, m^0 is the best applicable method and p^0 is the closest exemplar.

Once (c^0, m^0, p^0) is known, we need to provide information about the number of dofs that should be used to satisfy the relative error requirement (i.e., how to discretize the domain) and to estimate the time of the computation. From the performance profile rules, we get profiles $f(n)$ and $g(t)$ for solving problem p^0 in class c^0 using method m^0, where

—n is the number of nodes in the grid and $f(n)$ is the error in the solution computed using n nodes, and

—t is execution time and $g(t)$ is the error in the solution when the solution is computed in t time.

Since the amount of relative error in the computed solution and the amount of time taken to compute the solution are obviously coupled, we may or may not be able to satisfy the user's computational objectives. Hence, we may have to compromise. Consider Figures 9 and 10 which show an example $f(n)$ and $g(t)$. Suppose that the user requests that the error be less than e, the time be less than t and that the weight on the error request being satisfied is α. Assume that:

$$e_1 = g(t)$$
$$t_1 = g^{-1}(e)$$

Then, we have two cases:

—Case 1: $e_1 \leq e$

If $e_1 \leq e$, then $t_1 \leq t$. Thus, we can satisfy the conditions that the time be less

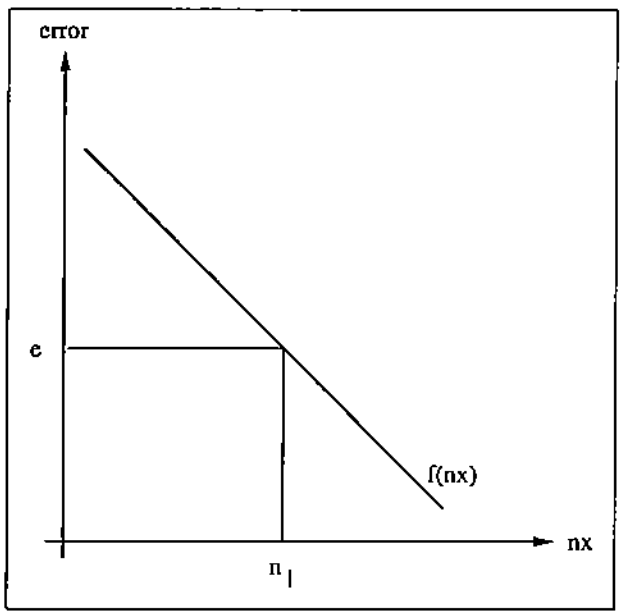

Fig. 9: A performance profile for solution method m on problem p that relates the grid size n_1 to the error e achieved.

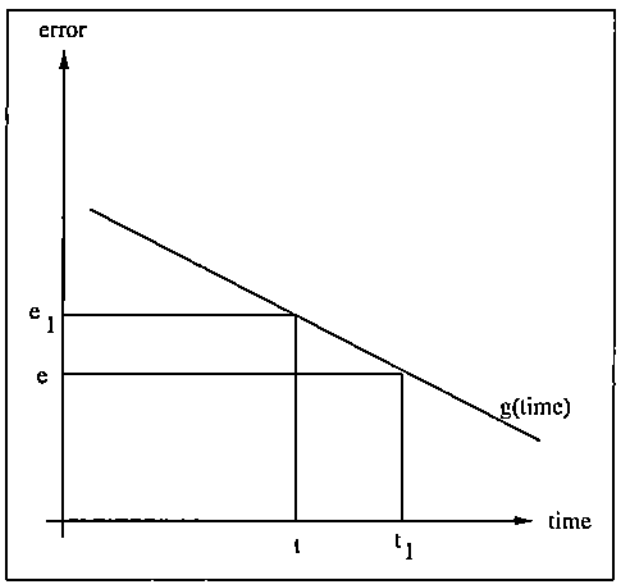

Fig. 10: A performance profile for solution method m on problem p that relates the grid size (n_1 from the previous figure) to the time t_1 taken to solve the problem. The pair (t, e_1) illustrates Case 1 in the text where the requested time t is smaller than the time required to meet the accuracy $= e$ request.

than t and that the error be less than e. The selection proposed is: *Use $f^{-1}(e)$ nodes to get a solution with a relative error of at most e in at most $g^{-1}(e)$ time.*

—Case 2: $e_1 > e$

In this case, it is not possible to satisfy the conditions that the solution time be less than t and that the error be less than e. Hence, we find an intermediate point between (t, e_1) and (t_1, e) taking into account the user–specified weight for the relative importance of time and error. Hence, let

$$(t^*, e^*) = \alpha(t_1, e) + (1 - \alpha)(t, e_1).$$

Thus, the selection proposed is: *Use $f^{-1}(e^*)$ nodes to get a solution with a relative error of at most e^* in at most t^* time.* An appropriate message is also provided.

7. DEALING WITH UNCERTAINTY

As explained earlier, PYTHIA's inference strategy is based on experimentally gathered performance data. Yet, while this information is a continuous quantity, we only have "discrete snapshots" of this information. This is to say, we have performance data for specific values of the many parameters involved. Thus, when we make decisions based on this data, a certain amount of uncertainty is inherent. In this section, we describe how we deal with this problem by combining the various snapshots of information under user guidance.

PYTHIA's inference algorithm needs $P(C = c)$, $P(M = m|C = c)$ and $P(P = p|C = c)$, where $P(x)$ is the probability of the event x happening. In this section, we describe how the various facts and rules are used to compute these probabilities.

7.1 Evidence Nodes $F_1, \ldots, F_n$: Problem Features

Each of these evidence nodes represents one problem characteristic or feature with belief in a feature is represented as a probability. Boolean valued features use the discrete probabilities 0.0 and 1.0. For example, if an operator is known to be homogeneous, then the feature indicating homogeneity is assigned a probability of 1.0, representing complete certainty in the homogeneity of the operator. If it is known to be non–homogeneous, then the feature is assigned a probability of 0.0. However, if neither case is known, then a probability of 0.5 is assigned to represent ignorance or equal belief in all the possibilities.

7.2 Node C: Finding Closest Classes

The computation performed at node C is the calculation of probability distribution for all the classes, $c \in C$. That is, we want to compute $P(C = c|F_1 = f_1, \ldots, F_n = f_n)$.

Let $\underline{c}$ be the characteristic vector of class c. Compute $d(c) = \|\underline{c} - \underline{u}\|$, where $\underline{u} = (f_1, f_2, \ldots f_n)$. We formulate the marginal probability $P(C = c)$ as

$$P(C = c) = \alpha \left(1 - \frac{d(c)}{\|\underline{d}\|_\infty}\right),$$

where $\underline{d}$ is the vector of all $d(c)$ for $c \in C$, $\|\cdot\|$ is an appropriate norm and α is chosen so that $\sum_{c \in C} P(C = c) = 1.0$. This calculation therefore assigns the

greatest probability to the class that is closest to the user's problem, as measured by the vector norm $\|\cdot\|$.

This formulation captures two significant aspects of the class selection. Computing d measures how close the new problem is to a given class. $\|\underline{d}\|_\infty$ measures how far away the farthest class is. Thus $P(C = c)$ attains high values only when the problem is close to class c and the other classes are relatively far away. In other situations, this formulation leads to a lower marginal probability. Clearly, this formulation is arbitrary, and other similar formulations can be made for the marginal probability. We feel that this formulation is a good one for the reasons outlined above.

7.3 Node P: Finding Closest Problems

The computation performed at Node P is the calculation of probability distribution for all the problems in the PDE population. That is, we want to compute $P(P = p|C = c, F_1 = f_1, \ldots, F_n = f_n)$.

Let $\underline{p}$ be the characteristic vector of problem p. Compute $d(p) = \|\underline{p} - \underline{u}\|$. Then, the marginal probability $P(P = p|C = c)$ is given by

$$P(P = p|C = c) = \begin{cases} \alpha \left(1 - \frac{d(p)}{\|\underline{d}\|_\infty}\right) & \text{if } p \in c \\ 0 & \text{otherwise} \end{cases},$$

where $\underline{d}$ is the vector of all $d(p)$ for $p \in P$ and α is chosen so that $\sum_{p \in P} P(P = p|C = c) = 1.0$. This calculation therefore assigns the greatest probability to those problems in each class c that are closest to the user's problem; that is, it identifies the most interesting exemplar from a class. α is scaling factor chosen to make the computed quantities lie in the zero to one range. Since our decisions are based on the magnitudes of the quantities concerned, scaling by a constant factor does not affect the choice in any manner. In our further computations, we shall leave out this scaling factor without loss of generality.

7.4 Evidence Node I: Performance Criteria Weights

This allows the user to specify the amount of weight to be placed on each performance criteria when selecting the solution method. Valid performance criteria are error and time. To specify that a weight of α is to be placed on the rules based on the "error" criterion, one gives $P(I = error) = \alpha$ and $P(I = time) = 1 - \alpha$.

7.5 Evidence Node R: Method Rank Weight

This allows the user to specify the amount of weight to be placed on performance profiles of each rank when selecting the solution method. Valid ranks are 1, ..., n, where n is the number of comparative rankings available. To specify that only those rules related to methods that ranked first in solving a particular class of problems are to be considered, one gives $P(R = 1) = 1$, $P(R = 2) = 0$, ..., $P(R = n) = 0$. Similarly, for equal weight on each rule irrespective of the rank, one gives $P(R = i) = \frac{1}{n}$, for $i = 1, \ldots, n$.

7.6 Evidence Node E: Error Level Weights

This allows the user to specify the amount of weight to be placed on performance data at each error level when selecting the solution method. Valid error levels are

$e_1, \ldots, e_n$, where n is the number of error levels at which performance rules are available. To specify that only the rules relating to the first error level are to be considered, one gives $P(E = e_1) = 1$, $P(E = e_2) = 0, \ldots, P(E = e_n) = 0$. Similarly, for equal weight on each error level, one gives $P(E = e_i) = \frac{1}{n}$, for $i = 1, \ldots, n$.

7.7 Node M: Determining the Best Method

Here, we want to compute the belief in method $m \in M$ being the method that best satisfies all given conditions. That is, we want to compute: $P(M = m|C = c, I = i, R = r, E = e)$.

The knowledge base has facts that give for each class, the r–ranked method for a fixed performance criteria and error level along with two confidence numbers (say γ_1 and γ_2, respectively). The first number indicates the confidence in the fact that this method is ranked above its immediate successor, while the second number indicates the confidence in ranking this method above the method ranked last. Hence, we let

$$P(M = m|C = c, I = i, R = r, E = e) = \gamma_1(c, i, r, e) \times \gamma_2(c, i, r, e).$$

This captures the confidence in the ranking of this method in a global sense. However, for a given method m and class c, it is possible that there is no rule in the database that relates m and c. In this case, we let $P(M = m|C = c, \ldots) = 0$. (This is reasonable as such a rule not appearing in the database means either that the method is not applicable to that class of problems or that it performs very poorly in that class.)

The output of this node is the marginal probability of M. This is computed as follows:

$$P(M = m) = \sum_{c \in C, i \in I, r \in R, e \in E} P(C = c)\, P(I = i)\, P(R = r)\, P(E = e)\, \gamma_1(c, i, r, e)\, \gamma_2(c, i, r, e)$$

We also need $P(M = m|C = c)$. This is computed as follows:

$$P(M = m|C = c) = \sum_{i \in I, r \in R, e \in E} P(I = i)\, P(R = r)\, P(E = e)\, \gamma_1(c, i, r, e)\, \gamma_2(c, i, r, e)$$

7.8 Node O: Generating Output

The function of this node is to generate the final output; i.e., recommend a certain method as the best method to use to satisfy the given conditions and also indicate what size grid to use to achieve the specified error level within the given time bound. The reasoning used for this computation has been described in Section 6

8. DETERMINING CLASSES USING NEURAL NETWORKS

We have defined classes in a deterministic way and computed a characteristic centroid for each class. Then, a new problem is classified by computing its distance (using the characteristic vector norm) to these centroids. Note that there are some important classes where there is an *a priori* defined way to obtain class memberships. The probabalistic approach presented above, or the approach we shall now present exclude such classes.

The Neural Network approach is to replace the characteristic vector norm computation. It allows much greater flexibility in the shape of classes than the norm of vectors. Note that the classes in PYTHIA are not disjoint, so the problem is as follows: We have a characteristic vector u of length n for the problem and a membership vector v of length m for the classes. v_i is 0 or 1 depending on whether membership in class i is absent or present. The problem then is to map the vector u into the vector v.

The neural network has an input layer (where u is given), an output layer (where v is produced), and one or more hidden layers. Each layer consists of many nodes (neurons) connected to nodes in the other layers. Each of the connections has a weight associated with it. At each node a simple processing of inputs produces the output. This processing is a function of the weights of the connections incident at the node. It is not difficult to see that even with very simple processing, a neural network with hidden layers and moderate connectivity can represent extremely complex relationships between the inputs and output [Siu et al. 1995]. Let o_j be the inputs to a node i, that is, outputs from nodes at the preceeding layer connected to it. The output o_i of node i is computed as:

$$s = \sum_j w_{ji} o_j,$$

$$o_i = \frac{1}{1 + e^{-s}},$$

where w_{ji} are the weights of the neural network. The precise relationship represented by a net is encoded in the weight of the connections. In our case the objective is to determine the weights that will map the characteristic vector of the problem into its membership vector.

Many schemes like Backpropagation [Rumelhart and McClelland 1986], Hopfield Networks [Hopfield and Tank 1986], Elastic Networks [Durbin and Willshaw 1987], Adaptive Resonance [Carpenter and Grossberg 1988], LMS [Widrow and Winter 1988] etc. have been proposed to compute the weights given samples of the relationship (input/output pairs) for different kinds of neural networks. Here, we use backpropagation with momentum. In this scheme, one makes a guess (usually small random values) at the weights, and then compares the actual output with the desired output for some known correct pairs of input output vectors. These are called training samples. The observed differences are used for a gradient correction to the weights designed to reduce the error. The process is continued until the error is sufficiently small. This process is often called training or learning and the set of correct input–output pairs is called the training set.

This approach differs from the probabalistic approach outlined earlier in that during the training phase, there is no uncertainity in the output. The network is trained using samples where the mapping from the characteristic vector to he membership vector is known. When the trained network is used to predict the membership for some previously unknown problem the output at each node will not always be binary. The amount of deviation observed from the perfect binary case is a measure of how far away from the class decision boundary the new vector lies. The reader is referred to [Joshi et al. 1994] for details.

9. ARCHITECTURE AND IMPLEMENTATION

This section discusses the software framework that is required to realize the knowledge acquisition and inference processes described earlier. The PYTHIA system is organized in the form of two components, the *knowledge acquisition* environment where one generates and records various information about problems and problem classes, and the *consultation* environment where users query the system for advice on solution schemes and their parameters. These components are interfaced with the //ELLPACK system so that PYTHIA can leverage from //ELLPACK's components (such as the symbolic analysis facilities) and so that users can use PYTHIA in a seamless fashion.

We use the CLIPS [Giarratano 1991] expert system development shell to implement most of the inferencing logic in PYTHIA. CLIPS (C Language Integrated Production System) was originally only a forward chaining rule language based on the Rete algorithm (hence the production system part of the CLIPS acronym) [Cooper and Wogrin 1988]. However, it now supports two more programming paradigms: procedural programming and object–oriented programming. The object–oriented programming language provided within CLIPS is called the CLIPS Object–Oriented Language (COOL). CLIPS comes in three forms: A system with a regular dumb–terminal interface, a system with a graphical user interface (for X Windows, MS–DOS, and MacIntoshes), or as an embeddable kernel system. The last form is extremely useful as it allows one to seamlessly integrate CLIPS into other programs without forcing users to learn about CLIPS. In fact, the CLIPS interfaces are all built by developing an interface around the CLIPS kernel. We used the X interface during the development of the CLIPS component of PYTHIA and then the embeddable version when integrating it into the //ELLPACK system.

The neural network simulations in PYTHIA are implemented using SNNS (Stuttgart Neural Network Simulator) [Zell et al. 1993], a freely available simulator. In addition to the user level interface, SNNS also has the very convenient feature of being embeddable in another system. The network configurations and input patterns for SNNS training and consultation are automatically generated from the CLIPS knowledge bases by the PYTHIA kernel. Output patterns are interpreted and transformed back into CLIPS also by the kernel.

9.1 The Knowledge Engineering Environment

The PYTHIA knowledge engineering environment supports the development and maintenance of the PYTHIA knowledge bases described in Section 5. The knowledge engineering environment is implemented using a variety of languages and systems including CLIPS, FORTRAN, C, Perl, and shell scripts.

The ELLPACK problem database (from the ELLPACK Performance Evaluation System) that we use consists of a set of structured files that are accessed using various manipulation programs. For performance data generation, we have developed convenient tools that allow the user to select a problem from the database, generate multiple test cases by specifying various solution methods and parameters (such as the number of grid lines), execute test cases, and finally extract relevant and interesting performance data. Once the performance data is available, it is run through a statistical analyzer whose task is to produce both linear least squares approxima-

tions to the performance data and a comparative ranking of the performance of all the methods applied to all the PDEs used in the test. The data produced by the statistical analyzer is automatically translated into CLIPS facts for later use in the consultation environment.

The problems can optionally be run through the problem characteristic extraction tools that we have developed for MAXIMA[2] in order to determine their characteristics as required by PYTHIA. Once all the necessary characteristics are known (either by automatic extraction or by user specification), they are also automatically translated into CLIPS objects for later use. Problem features are represented by instances of a "Problem" class that we have defined using CLIPS' object-oriented capabilities. The characteristic vector of a problem is computed by one of the methods defined for the "Problem" class by encoding all the properties in the form of a vector.

Problem classes (i.e., a collection of problems identified by some common properties) are represented in CLIPS as instances of a "Problem-Class" class that we have defined. A problem class object consists of a list of member problems (i.e., the names of instances of "Problem" objects) and a method that computes the characteristic vector of the entire class by averaging the sum of the characteristic vectors of all the class members.

Once all the problems and problem classes are defined, we generate some facts and rules for later use in the consultation environment: Facts describing problems, facts describing classes, rules describing the performance of various methods on problems, and, finally, rules describing the performance of various methods on classes of problems. All these facts and rules are implemented as ordered facts in CLIPS. These four knowledge bases are the basis on which the consultation environment makes its selections.

9.2 The Consultation Environment

The task of the PYTHIA consultation environment is to use the information generated by the knowledge engineering environment as a basis for making selections for users' about what method and parameters to use to solve a particular problem within specified performance constraints. The consultation environment is also implemented using a variety of languages and systems: CLIPS, C, Tcl, Perl, and shell scripts. The software architecture of the PYTHIA consultation environment is shown in Figure 11.

All the knowledge bases are loaded into PYTHIA when the consultation environment starts up. Let us assume that the user has specified a PDE problem in the //ELLPACK problem solving environment and wishes to get some advice on what method to use and what the associated parameters are. At this point, //ELLPACK attempts to determine all automatically determinable features or characteristics of the problem by sending the problem description to the feature analyzer in MAXIMA. //ELLPACK then sends the problem description and all the known features to PYTHIA using the communication facilities of //ELLPACK. PYTHIA allows the user to specify some performance objectives and also what weight is to be placed

[2]MAXIMA is the Austin Kyoto Common Lisp version of the well known computer algebra system MACSYMA.

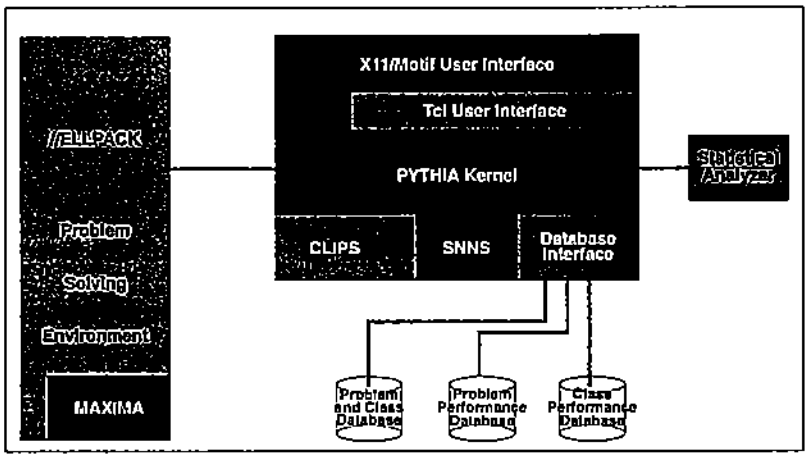

Fig. 11. The software architecture of the PYTHIA consultation environment.

on achieving each objective. Any unknown problem features are requested from the user at this time as well.

Once all this information is available, it is put into CLIPS' working memory via instance creations (for problem features) and fact assertions (for the performance objectives and rule weights). Firing CLIPS then executes the inference engine to compute a selection (a prediction as to what the best solver is). The selection and prediction data (the proposed method and its parameters as well as information on what accuracy is to be expected and how long the computation will take) is then returned to the problem solving environment. //ELLPACK examines the data and incorporates the corresponding modules into the PDE solution algorithm being developed.

10. PERFORMANCE EVALUATION OF PYTHIA

In this section we evaluate PYTHIA in terms of the "accuracy" of its solutions and the cost of its overhead. We evaluate the class selection process (as implemented in the neural network approach), the correctness of the selected method and parameters for various classes (in terms of the appropriateness of the suggested method as well as the correctness of the performance predictions) and finally the overhead of PYTHIA itself. We chose not to apply the symbolic analysis of [Dyksen and Gritter 1992] to eliminate inapplicable methods for a particular PDE problem. This is because we wanted to see how often our proposed methodology would select inapplicable methods. The production version of PYTHIA, of course, uses this symbolic filter to eliminate inapplicable methods. All the performance tests use the ELLPACK PDE population [Rice et al. 1981].

10.1 Performance of Class Selection Using Neural Networks

To study the effectiveness of the class identification process using neural networks, we define the following non–exclusive classes (the number in parenthesis indicates the number of problems that belong to that class):

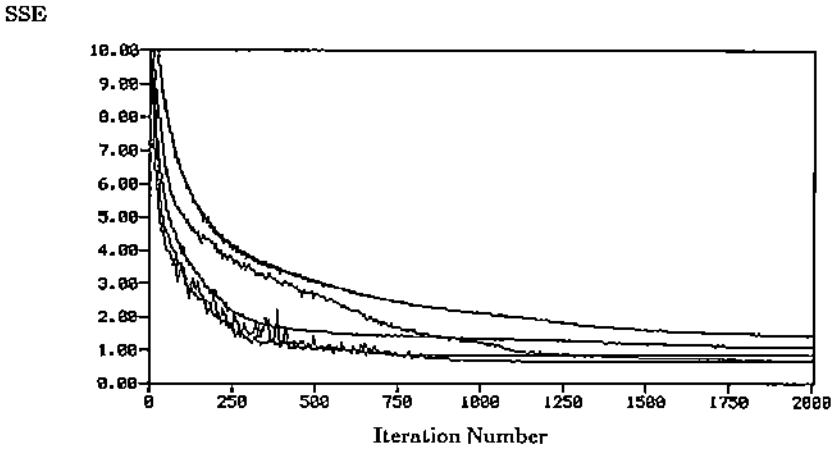

Fig. 12: Plot of sum of squared error (SSE) vs. iteration number for the first training algorithm.

(1) SOLUTION–SINGULAR: Problems whose solution has at least one singularity (6).

(2) SOLUTION–ANALYTIC: Problems whose solution is analytic (35).

(3) SOLUTION–OSCILLATORY: Problems whose solution oscillates (34).

(4) SOLUTION–BOUNDARY–LAYER: Problems whose solutions depict a boundary layer (32).

(5) BOUNDARY–CONDITIONS–MIXED: Problems with mixed boundary conditions (74).

The input characteristic vector has 32 components (m = 32) and the membership vector has 5 components (k = 5). We arbitrarily choose to use one hidden layer with 10 nodes and use a completely connected neural net. There are thus $32*10+10*5 = 370$ connections and weights for this test case.

There are a total of 167 problems in the population that belong to at least one of these classes. We split this data into two parts– one part contains two thirds of the exemplars and is used to train the network. The other part is used to test the performance of the trained network. All the simulations were performed using the Stuttgart Neural Network Simulator [Zell et al. 1993], a very useful public domain simulator with a convenient graphical interface.

The first training algorithm used is a "vanilla" backpropagation routine. The *learning rate* is an important free parameter here. It is the value by which the gradient term is multiplied before being used to update the weights of the network. In Figure 12, we show the Sum of Squared Error (SSE) for the network as it changes with the training epochs (iterations). As expected, the best performance is obtained by some reasonably small value for the learning rate parameter, in this case 0.2. Using smaller values also leads to the same SSE value, but the learning takes much longer. Using larger values give oscillatory behavior, and the algorithm may get trapped in local minimum with a larger SSE value, as can be seen in the graph. Figure 13 concentrates on the early stages of iteration using various learning rate parameters to illustrate the behavior of the learning scheme there.

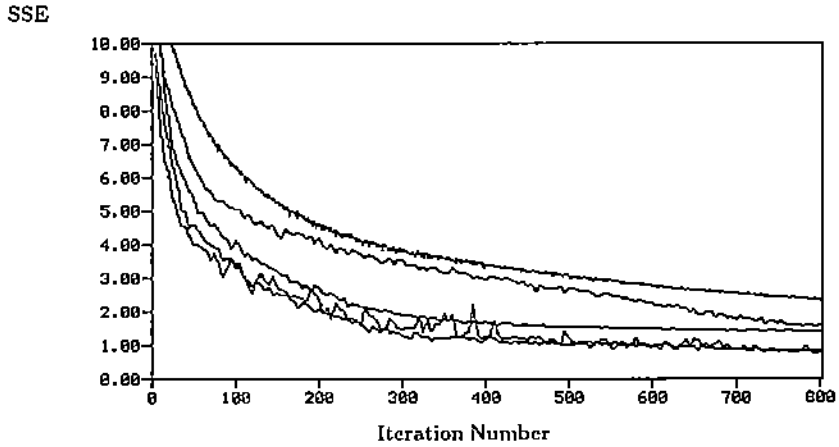

Fig. 13. Magnification of the first part of Figure 12.

A popular variation on the standard backpropagation is to use backpropagation with momentum and flat spot elimination. This involves using another term to modify the weights, which is the second derivative of the error function. Also, a small constant term is added to the derivative to avoid local minima. This is the second training algorithm used. The value of the learning rate is fixed at 0.2 from the previous experiments, and the flat spot elimination constant is chosen as 0.05. The parameter multiplying the momentum term is varied and once again plots of SSE vs. iteration number are made. The best performance seems to be for the momentum term at about 0.8 (these plots are not shown here). It is also clear that the addition of the momentum and flat spot elimination lead, as expected, to lower SSE values than the standard backpropagation, and to much faster convergence.

Clearly the network learns to correctly classify all the problems it is trained on. However, one of the reasons we wished to use neural networks is that they could generalize. The next step therefore is to verify that our network can generalize correctly. To do this, we split the total of 167 vectors that we have into two sets, one set consisting of 111 vectors (the "larger set"), and the other consisting of 56 vectors (the "smaller set"). First, the network is trained for 2000 iterations with the larger set. The learning rate used is 0.2, the momentum 0.8, and the flat spot elimination constant 0.05. After training, the network is presented with all the 167 vectors, and its output recorded. To interpret and analyze the output, we compute the least square norm of the expected and actual outputs for each of the 167 cases. Figure 14 shows a scatter plot of the results, with the X axis showing the vector number (from 1 through 167), and the Y axis showing the L_2 error norm. As the figure indicates, the network correctly classifies most of the vectors.

To further study the generalization power of such networks, we use the smaller problem set as a training set instead of the larger set as in the previous experiment. The parameters and the number of iterations remain the same as in the previous case and the testing is once again done with the complete set of vectors. As expected, training with fewer samples leads to a degradation of performance with respect to the previous case.

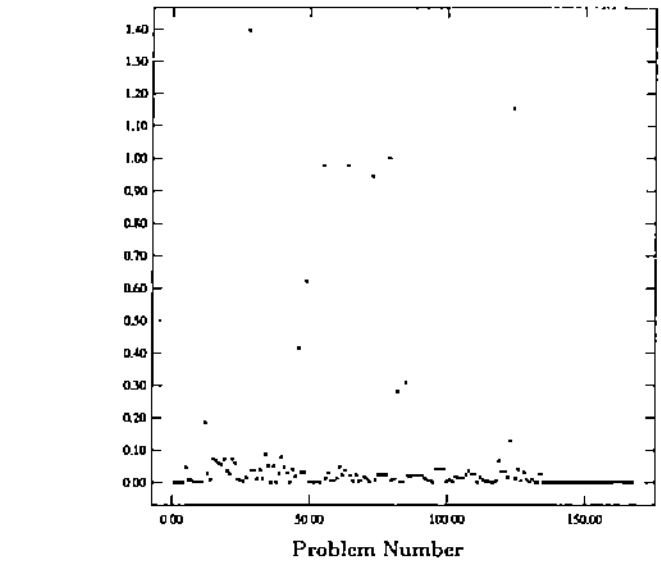

Fig. 14: Scatter plot of error vs. vector number for the larger training set with the final training algorithm.

Training Set	Threshold Value			
	0.2	0.1	0.05	0.005
Large	157	155	143	67
Small	142	132	113	37

Table II: Number of correctly classified vectors with various error thresholds for the two training sets. For example, there are 157 vectors correctly classified when the error threshold is 0.2 and when the larger set is used to train the network.

Tables II and III present some further analysis of this data. We first choose an arbitrary threshold for the L2 error norm. Error norms above the threshold are assumed to imply that the corresponding input had been misclassified. In Table II, we show the number of correctly classified vectors (out of 167) for different values of the threshold. We can clearly see that the network is fairly successful in classifying correctly, especially with the larger training set. The same trend is reflected in Table III, where we present the mean and median values for the error norms in the two cases. Notice that while the mean value of error for the smaller training set is slightly higher, the median value remains small. This clearly illustrates that while there are outliers, most of the problem do get classified correctly.

10.2 Performance of Method Selection and Parameter Prediction

These experiments study the effectiveness of the method selection and performance prediction methodology once a problem is identified as being in a certain class.

Training Set	Mean	Median
Large	0.0652	0.0095
Small	0.1367	0.0235

Table III. Statistics for the error norms with various training sets.

In the first experiment, we used the class of problems studied in [Houstis and Rice 1982] as the test class which we denote by c. Then, we use different parts of c as a training set to make predictions for the rest of c.

In the first test, we use one quarter of c as a training set. We request selections to achieve relative errors less than 10^{-3}, 10^{-4}, and 10^{-5} in at most 10 minutes with equal weight on achieving the error bound and on achieving the time bound. For rule weights, we selected the default values chosen to emphasize the overall performance of methods (vs. peak performance) in making decisions. The results in this test are not very satisfactory: Only 15% of the selections were actually valid. (A selection is considered "invalid" if the suggested method is not applicable to the problem or if the any of the parameters do not apply correctly to the method.) However, it is encouraging that all the valid selection are in fact correct– solving the problems with the selected method and parameters does result in solutions with the requested error within the requested time. It should be noted that since equal weight is placed on the correctness of the error bound and on the correctness of the time bound, it is not necessary for both predictions (or either prediction) to fall within the originally requested bound for the selection to be considered correct.

In the second test, we use $\frac{c}{2}$ as a training set and specified the same error and time bounds as well as the same rule weights. The results here are much more satisfying: All the selections made are in fact valid and 90% of them are also correct (i.e., the requested error is achieved within the requested time).

In the second experiment, we used the class of problems with mixed boundary conditions studied in [Dyksen et al. 1988]. We used a total of 21 test problems (c) which we broke to two groups of 9 and 12 problems.

In the first test, we train with the smaller set of 9 problems and repeat the same tests as above. All the selections made by PYTHIA were valid and 86% of the predictions were correct. In the second test, we train with the larger set of 12 problems and found again that all the predictions were valid with 92% of them being correct.

10.3 The Overhead of PYTHIA

The response time for a prediction from PYTHIA is generally in the order of seconds. We estimate the sizes of the various knowledge bases that PYTHIA uses to judge how much processing is done in this time.

For each problem, we have one instance of a "problem" class during the training phase and one fact during the consultation phase. Since only one phase is active at a time, there is one object per problem. Let the number of problems in the database be np. For each problem p, there are $nmp_p \times nc$ performance data facts, where nmp_p is the number of methods with which p can be solved and nc is the number of criteria with which the performance of a method is measured. Thus,

there are a total of

$$np + \sum_{p \in P} nmp_p \times nc$$

problem–related facts in the database.

Each class is represented by an instance of a "problem-class" class during the training phase. During the consultation phase, a class is represented by an ordered fact indicating the members, an ordered fact indicating the class characteristic vector and the class performance facts described earlier. Let nc be the number of classes. For each class c, there are $nr \times nmc_c \times ne \times nc$ class performance facts, where nr is the number of comparative ranks for which performance comparisons are available, nmc_c is the number of methods with which all the problems in class c have been solved, ne is the number of error levels at which performance data was gathered and nc is as before. Thus, there are a total of

$$2 \times nc + \sum_{c \in C} nmc_c \times nr \times nc \times ne$$

class–related facts in the database (where nmc_c is the number of methods with which all problems in class c have been solved).

For the ELLPACK PDE population described earlier, $np \approx 275$, $nc \approx 5$ (for the data we have generated), $nmp_p \approx 15$ (although 15 or more possible solution methods per problem are easily feasible with the //ELLPACK library, due to resource limitations we have generated data for only about $nmp_p \approx 5$), $nmc_c \approx 5$ (again, for the data we have been able to generate), $nr \approx 5$, $nc = 2$, $ne = 3$. That is, a grand total of about 8,000 rules were generated for the 275 problems.[3]

11. CONCLUSION

In this paper we have described a knowledge based system to assist in the selection of a PDE solver that satisfies a user's objectives for error and computational cost. The selection is one of a multitude of (grid, method) pairs and is based on performance information from a population of PDE problems. The methodology has been implemented as an integrated part of the //ELLPACK system. We have validated the accuracy of PYTHIA's predictions for some classes of PDE problems for which we know *a priori* the performance of applicable solvers. We have also validated PYTHIA's class selection methodology for the neural network based scheme. The results indicate that PYTHIA's accuracy increases when the knowledge base (number of rules) increases. PYTHIA's selections almost coincide with the results of various known performance evaluation studies. PYTHIA's paradigm is seen to be a good alternative to support "intelligence" in PSEs for PDE based applications. PYTHIA is currently being extended to parallel elliptic solvers. We are also working on using novel neuro–fuzzy techniques for the prediction mechanisms.

ACKNOWLEDGMENTS

We would like to acknowledge the contributions made by Amy L. S. Ng and Narendran Ramakrishnan to the realization of the PYTHIA system. Amy contributed

[3]While, this is what we would like to generate, we have yet only produced those rules necessary for the tests described earlier.

to the design and implementation of the various user interfaces and Narendran contributed to the testing and verification of PYTHIA.

REFERENCES

BAREISS, R. 1986. *Exemplar-Based Knowledge Analysis.* Academic Press.

BOISVERT, R. F., RICE, J. R., AND HOUSTIS, E. N. 1979. A system for performance evaluation of partial differential equations software. *IEEE Transactions on Software Engineering SE–5*, 4, 418–425.

CARPENTER, G. A. AND GROSSBERG, S. 1988. The art of adaptive pattern recognition by a self–organizing neural network. *Computer 21*, 77–88.

COOPER, T. AND WOGRIN, N. 1988. *Rule–based Programming with OPS5.* Morgan Kaufmann.

DURBIN, R. AND WILLSHAW, D. 1987. An analogue approach to the travelling salesman problem using an elastic net method. *Nature 326*, 689–691.

DYKSEN, W. R. AND GRITTER, C. R. 1989. Elliptic expert: An expert system for elliptic partial differential equations. *Mathematics and Computers in Simulation 31*, 333–343.

DYKSEN, W. R. AND GRITTER, C. R. 1992. Scientific computing and the algorithm selection problem. In E. N. HOUSTIS, J. R. RICE, AND R. VICHNEVETSKY (Eds.), *Expert Systems for Scientific Computing*, pp. 19–31. North–Holland.

DYKSEN, W. R., RIDDENS, C. J., AND RICE, J. R. 1988. The performance of numerical methods for elliptic problems with mixed boundary conditions. *Numerical Methods for Partial Differential Equations 4*, 347–361.

GALLOPOULOS, E., HOUSTIS, E. N., AND RICE, J. R. 1994. Problem–solving environments for computational science. *IEEE Computational Science and Engineering 1*, 11–23.

GIARRATANO, J. C. 1991. *CLIPS User's Guide, Version 5.1.* NASA Lyndon B. Johnson Space Center.

HOPFIELD, J. J. AND TANK, D. 1986. Computing with neural circuits. *Science 233*, 625–633.

HOUSTIS, E. N. AND RICE, J. R. 1982. High order methods for elliptic partial differential equations with singularities. *International Journal for Numerical Methods in Engineering 18*, 737–754.

HOUSTIS, E. N., RICE, J. R., CHRISOCHOIDES, N. P., KARATHANASIS, H. C., PAPACHIOU, P. N., SAMARTZIS, M. K., VAVALIS, E. A., WANG, K. Y., AND WEERAWARANA, S. 1990. //ELLPACK: A numerical simulation programming environment for parallel MIMD machines. In J. SOPKA (Ed.), *Proceedings of Supercomputing '90*, pp. 96–107. ACM Press.

HOUSTIS, E. N., RICE, J. R., CHRISTARA, C. C., AND VAVALIS, E. A. 1988. Performance of scientific software. In J. R. RICE (Ed.), *Mathematical Aspects of Scientific Software*, pp. 123–155. Springer–Verlag.

JOSHI, A., WEERAWARANA, S., AND HOUSTIS, E. N. 1994. The use of neural networks to support "intelligent" scientific computing. In *Proceedings Int. Conf. Neural Networks, World Congress on Computational Intelligence*, Volume IV, pp. 411–416. (Orlando, Florida).

KAMEL, M. S., MA, K. S., AND ENRIGHT, W. H. 1993. ODEXPERT: An expert system to select numerical solvers for initial value ode systems. *ACM Transactions on Mathematical Software 19*, 44–62.

KÖNIG, S. AND ULLRICH, C. 1990. An expert system for the economical application of self–validating methods for linear equations. In E. N. HOUSTIS, J. R. RICE, AND R. VICHNEVETSKY (Eds.), *Intelligent Mathematical Software Systems*, pp. 195–220. North–Holland.

MOORE, P. K., OZTURAN, C., AND FLAHERTY, J. E. 1990. Towards the automatic numerical solution of partial differential equations. In E. N. HOUSTIS, J. R. RICE, AND R. VICHNEVETSKY (Eds.), *Intelligent Mathematical Software Systems*, pp. 15–22. North–Holland.

PEARL, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann.

RAMAKRISHNAN, N., JOSHI, A., WEERAWARANA, S., HOUSTIS, E. N., AND RICE, J. R. 1995. Neuro-Fuzzy Systems for Intelligent Scientific Computing. Technical Report TR-95-026, Dept. Comp. Sci., Purdue University.

RICE, J. R. 1976. The algorithm selection problem. *Advances in Computers 15*, 65–118.

RICE, J. R. 1979. Methodology for the algorithm selection problem. In L. D. FOSDICK (Ed.), *Performance Evaluation of Numerical Software*, pp. 301–307. North–Holland.

RICE, J. R. AND BOISVERT, R. F. 1985. *Solving Elliptic Problems Using ELLPACK*. Springer–Verlag.

RICE, J. R., HOUSTIS, E. N., AND DYKSEN, W. R. 1981. A population of linear, second order, elliptic partial differential equations on rectangular domains, part I. *Mathematics of Computation 36*, 475–484.

RUMELHART, D. E. AND MCCLELLAND, J. L. 1986. *Parallel Distributed Processing, Explorations into the Microstructure of Cognition*. MIT Press.

SIU, K. Y., ROYCHOWDHURY, V., AND KALATH, T. 1995. *Discrete Neural Computation*. Prentice Hall.

WEERAWARANA, S. 1994. Problem solving environments for partial differential equation based applications. Ph. D. thesis, Department of Computer Sciences, Purdue University.

WEERAWARANA, S., HOUSTIS, E. N., AND RICE, J. R. 1992. An interactive symbolic-numeric interface to parallel ELLPACK for building general PDE solvers. In B. R. DONALD, D. KAPUR, AND J. L. MUNDY (Eds.), *Symbolic and Numerical Computation for Artificial Intelligence*, Chapter 13, pp. 303–322. Academic Press.

WEERAWARANA, S., HOUSTIS, E. N., RICE, J. R., CATLIN, A. C., CRABILL, C. L., CHUI, C. C., AND MARKUS, S. 1994. PDELab: An object-oriented framework for building problem solving environments for PDE based applications. In *Proceedings of the Second Annual Object–Oriented Numerics Conference*, Rogue–Wave Software, pp. 79–92.

WIDROW, B. AND WINTER, R. 1988. Neural nets for adaptive filtering and adaptive pattern recognition. *Computer 21*, 25–39.

ZELL, A., MACHE, N., HÜBNER, R., MAMIER, G., VOGT, M., HERRMANN, K.-U., SCHMALZL, M., SOMMER, T., HATZIGEORGIOU, A., DÖRING, S., AND POSSELT, D. 1993. Stuttgart Neural Network Simulator User Manual, Version 3.1. Technical Report 3, University of Stuttgart.