PYTHIA-II: A Knowledge/Database System for Managing Performance Data and Recommending Scientific Software

Elias N. Houstis, Ann C. Catlin, and John R. Rice
Dept. of Computer Sciences, Purdue University, West Lafayette, IN 47906

Vassilios S. Verykios
College of Information Science & Tech., Drexel University, Philadelphia, PA 19104

Naren Ramakrishnan
Dept. of Computer Science, Virginia Tech, Blacksburg, VA 24061

Catherine E. Houstis
Dept. of Computer Science, University of Crete, Heraklion, Greece.

Often scientists need to locate appropriate software for their problems and then select from among many alternatives. We have previously proposed an approach for dealing with this task by processing performance data of the targeted software. This approach has been tested using a customized implementation referred to as PYTHIA. This experience made us realize the complexity of the algorithmic discovery of knowledge from performance data and the management of these data together with the discovered knowledge. To address this issue, we created PYTHIA-II — a modular framework and system which combines a general *knowledge discovery in databases* (KDD) methodology and *recommender* system technologies to provide advice about scientific software/hardware artifacts. The functionality and effectiveness of the system is demonstrated for two existing performance studies using sets of software for solving partial differential equations. From the end-user perspective, PYTHIA-II allows users to specify the problem to be solved and their computational objectives. In turn, PYTHIA-II (i) selects the software available for the user's problem, (ii) suggests parameter values, and (iii) assesses the recommendation provided. PYTHIA-II provides all the necessary facilities to set up database schemas for testing suites and associated performance data in order to test sets of software. Moreover, it allows the easy interfacing of alternative data mining and recommendation facilities. PYTHIA-II is an open-ended system implemented on public domain software and can be applied to any software domain.

Categories and Subject Descriptors: G.1.8 [**Numerical Analysis**]: Partial Differential Equations; H.4.2 [**Information Systems**]: Types of Systems; H.2.8 [**Database Management**]: Database Applications; I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems

Additional Key Words and Phrases: data mining, inductive logic programming, knowledge-based systems, knowledge discovery in data bases, performance evaluation, recommender systems, scientific software

---

This work was supported in part by NSF grant CDA 91-23502, PRF 6902851, DARPA grant N66001-97-C-8533 (Navy), DOE LG-6982, DARPA under ARO grant DAAH04-94-G-0010, and the Purdue Research Foundation.

1. INTRODUCTION

Complex scientific, engineering or societal problems are often solved today by utilizing libraries or some form of problem solving environments (PSEs). Most software modules are characterized by a significant number of parameters affecting efficiency and applicability that must be specified by the user. This complexity is significantly increased by the number of parameters associated with the execution environment. Furthermore, one can create many alternative solutions of the same problem by selecting different software for the various phases of the computation. Thus, the task of selecting the best software and the associated algorithmic/hardware parameters for a particular computation is often difficult and sometimes even impossible. In [Houstis et al. 1991] we proposed an approach for dealing with this task by processing performance data obtained from testing software. The testing of this approach is described in [Weerawarana et al. 1997] using the PYTHIA implementation for a specific performance evaluation study. The approach has also been tested for numerical quadrature sofware [Ramakrishnan et al. 2000] and is being tested for parallel computer performance [Adve et al. 2000; Verykios et al. 1999]. This experience made us realize the high level of complexity involved in the algorithmic discovery of knowledge from performance data and the management of these data together with the discovered knowledge. To address the complexity issue together with scalability and portability of this approach, we present a *knowledge discovery in databases* (KDD) methodology [Fayyad et al. 1996] for testing and recommending scientific software. PYTHIA-II is a system with an open software architecture implementing the KDD methodology, which can be used to build a *Recommender System* (RS) for many domains of scientific software/hardware artifacts [Weerawarana et al. 1997; Ramakrishnan et al. 2000; Verykios 1999; Verykios et al. 2000]. In this paper, we describe the PYTHIA-II architecture and its use as an RS for PDE software.

Given a problem from a known class of problems and some performance criteria, PYTHIA-II selects the best performing software/machine pair and estimates values for the associated parameters involved. It makes recommendations by combining attribute-based elicitation of specified problems and matching them against those of predefined dense population of similar types of problems. Dense here means that there are enough data available so that it is reasonable to expect that a good recommendation can be made. The more dense the population is, the better the recommendation. We describe case studies for two sets of elliptic partial differential equations software found in PELLPACK [Houstis et al. 1998].

We now describe a sample PYTHIA-II session (Figure 1). Suppose that a scientist or engineer uses PYTHIA-II to find software that solves an elliptic partial differential equation (PDE). The system uses this broad categorization to direct the user to a form-based interface that requests more specific information about features of the problem and the user's performance constraints. Figure 1 illustrates a portion of this scenario where the user provides features about the operator, right side, domain, and boundary conditions - integral parts of a PDE - and specifies an execution time constraint (measured on a Sun SPARCstation 20, for instance) and an error requirement to be satisfied. Thus the user wants software that is fast and accurate; it is possible that no such software exists. The RS contacts the

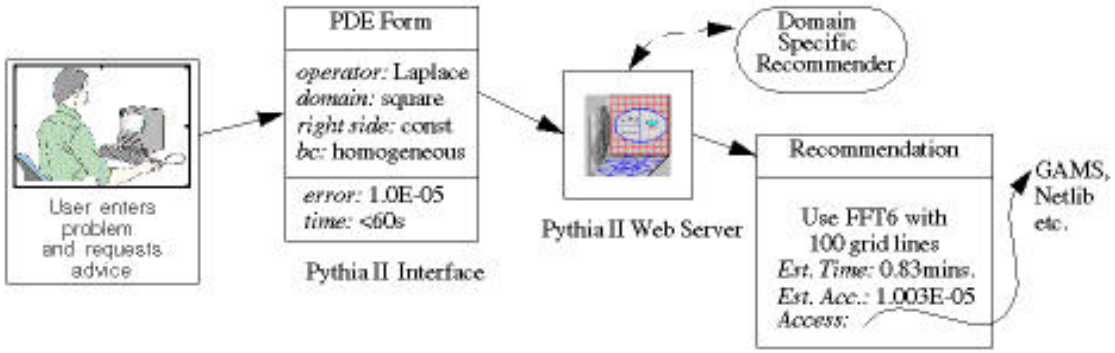

Fig. 1: The recommender component of PYTHIA-II implemented as a web server providing advice to users.

PYTHIA-II (web) server on the user's behalf and uses the knowledge acquired by the learning methodology presented in this paper to perform a selection from a software repository. Then the RS consults databases of performance data to determine the solver parameters, such as grid lines to use with a PDE discretizer, and estimates the time and accuracy using the recommended solver. Note that the RS does not involve the larger databases used in the KDD process, it only accesses special, smaller databases of knowledge distilled from the KDD process.

The paper is organized as follows. Section 2 describes a general methodology for selecting and recommending scientific software implemented in PYTHIA-II. The architecture for an RS based on the PYTHIA-II approach is presented in Section 3. A description of the data management subsystem of PYTHIA-II is presented in Section 4. We include a database schema appropriate for building an RS for elliptic PDE software from the PELLPACK library to illustrate its use. Section 5 outlines the knowledge discovery components of PYTHIA-II. The data flow in PYTHIA-II is illustrated in Section 6. The results of applying PYTHIA-II to two case studies and comparing with earlier results from 1980s can be found in Sections 7 and 8.

2. A RECOMMENDER METHODOLOGY FOR SCIENTIFIC SOFTWARE

An RS uses stored information (user preferences, performance data, artifact characteristics, cost, size, ...) of a given class of artifacts (software, music, can openers, ...) to locate and suggest artifacts of interest [Ramakrishnan 1997; Ramakrishnan et al. 1998; Resnik and Varian 1997]. An RS for software/hardware artifacts uses stored performance data on a population of previously encountered problems and machines to locate and suggest efficient artifacts for solving previously unseen problems. Recommendation becomes necessary when user requests or objectives cannot be properly represented as ordinary database queries. In this section, we describe the complexity of this problem, the research issues to address, and a methodology for resolving them.

The algorithm or software selection problem originated in an early paper by Rice [Rice 1976]. Even for routine tasks in computational science, this problem is ill-posed and quite complicated. Its difficulty is due to the following factors:

—The space of applicable software for specific problem subclasses is inherently

Phases	Description
Determine evaluation objectives	Identify the computational objectives for which the performance evaluation of the selected scientific software is carried out.
Data preparation (1) selection (2) pre-processing	(1) Identify the evaluation benchmark, its problem features, experiments (i.e., population of scientific problems for the generation of performance data). (2) Identify the performance indicators to be measured. (3) Identify the actual software to be tested, along with the numerical values of their parameters. (4) Generate performance data.
Data mining	(1) Transform the data into an analytic or summary form. (2) Model the data to suit the intended analysis and data format required by the data mining algorithms. (3) Mine the transformed data to identify patterns or fit models to the data; this is the heart of the process.
Analysis of results	This is a post-processing phase done by knowledge engineers and domain experts to ensure correctness of the results.
Assimilation of knowledge	Create a user friendly interface to utilize the knowledge and to identify the scientific software (with parameters) for user's problems and computational objectives.

Table I: A methodology for building an RS. This methodology is very similar to previous procedures adopted in the performance evaluation of scientific software.

large, complex, ill-understood and often intractable to explore by brute-force means. Approximating the problem space by a feature space helps, but introduces an intrinsic uncertainty.

—Depending on the way the problem is (re)presented, the space of applicable algorithms changes; some of the better algorithms sacrifice generality for performance and have customized data structures and fine tuned routines.

—Both specific features of the given problem and algorithm performance information affect the algorithm selection strategy.

—A mapping from the problem space to the good software in the algorithm space is not the only useful measure of success; one also needs indicators of domain complexity and behavior, e.g., information about the relative costs.

—There is an inherent uncertainty in assessing the performance measures of a particular algorithm for a problem. Minor mplementation differences can produce large differences in performance that make analytic estimates unreliable.

—Techniques are needed that allow distributed recommender systems to coexist and cooperate together to exploit all relevant information.

The methodology for building PYTHIA-II uses the *knowledge discovery in databases* (KDD) process shown in Table I. Assuming a dense population of benchmark problems from the targeted application domain, this RS methodology uses a three-pronged strategy: feature determination of problem instances, performance evaluation of scientific software, and the automatic generation of relevant knowledge. Note that the dense population assumption can be quite challenging for many application domains. We now address each of these aspects.

2.1 Problem Features

The applicability and efficiency of software depends significantly on the features of the targeted problem domain. Identifying appropriate problem features of the problem domain is a fundamental problem in software selection. The way problem features affect software is complex, and algorithm selection might depend in an unstable way on the features. Thus selections and performance for solving $u_{xx} + u_{yy} = 1$ and $u_{xx} + (1 + xy/10,000)u_{yy} = 1$ can be completely different. Even when a simple structure exists, the actual features specified might not properly reflect the simplicity. For example, if a good structure is based on a simple linear combination of two features $f1$ and $f2$, the use of features such as $f1 * cos(f2)$ and $f2 * cos(f1)$ might be ineffective. Furthermore, a good selection methodology might fail because the features are given inappropriate attribute-value meanings and measures of cardinality. Many attribute-value approaches (such as neural networks) routinely assign value-interpretations to numeric features (such as 1 and 5), when such values can only be interpreted in an ordinal/symbolic sense. PYTHIA-II assumes features are defined by the knowledge engineer.

The database schema defining a feature is of the form name and text as follows:

```
nfeatures  integer  -- no. of attributes identifying this feature
features   text[]   -- numeric/symbolic/textual identification
forfile    text     -- file-based feature information
```

An example relating a feature to a PDE equation is:

```
name       text     -- relation record name
equation   text     -- name of equation with these features
feature    text     -- name of record identifying features
```

where the foreign keys identify the relation between the equation and its features. Two instances from tables for these are:

```
name     | opLaplace                  name     | opLaplace pde #3
nfeatures| 1                          equation | pde #3
features | "Uxx + Uyy (+Uzz) = f"     feature  | opLaplace
```

which shows the correspondence between equation *pde#3* and its feature *opLaplace* (the PDE is the Laplacian).

2.2 Performance Evaluation

There exist well established performance evaluation methodologies for scientific software [Houstis et al. 1978; Boisvert et al. 1979; Houstis et al. 1983; Rice 1983; Dyksen et al. 1984; Moore et al. 1990; Rice 1990]. While there are many important factors that contribute to the quality of numerical software, we illustrate our ideas using speed and accuracy. PYTHIA-II can handle other attributes (reliability, portability, documentation, etc.) in its data storage scheme. Similar performance evaluation methodology and attributes are needed for each application domain.

Accuracy is measured by the norm of the difference between the computed and the true solutions or by a guaranteed error estimate. Speed is measured by the time required to execute the software in a standard execution environment. PYTHIA-II ensures that all performance evaluations are made consistently; their outputs are automatically coded into predicate logic formulas. We resort to attribute-value encodings when the situation demands it; for instance, using straight line approximations to performance profiles (e.g., accuracy vs. grid size) for solvers is useful to obtain interpolated values of grid parameters for PDE problems.

2.3 Reasoning and Learning Techniques for Generating Software Recommendations

PYTHIA-II uses a multi-modal approach by integrating different learning methods to leverage their individual strengths. We have explored and implemented two such strategies: Case-Based Reasoning (CBR) [Joshi et al. 1996] and inductive logic programming (ILP) [Bratko and Muggleton 1995; Dzeroski 1996; Muggleton and Raedt 1994] which we describe in this section.

CBR systems obey a lazy-learning paradigm in that learning consists solely of recording data from past experiments to help in future problem solving sessions. (This gain in simplicity of learning is offset by a more complicated process that occurs in the actual recommendation stage.) Evidence from psychology suggests that people use this approach to make judgements, using the experience gained in solving 'similar' problems to devise a strategy for solving the present one. In addition, CBR systems can exploit *a priori* domain knowledge to perform more sophisticated analyses even if pertinent data is not present. The original PYTHIA system utilized a rudimentary form of case-based reasoning using a characteristic-vector representation for the problem population [Weerawarana et al. 1997].

ILP systems, on the other hand, use an eager learning paradigm in that they attempt to construct a predicate logic formula so that all positive examples of good recommendations provided can be logically derived from the background knowledge, and no negative example can be logically derived. The advantages of this approach lie in the generality of the representation of background knowledge. Formally, the task in algorithm selection is: given a set of positive exemplars and negative exemplars of the selection mapping and a set of background knowledge, induce a definition of the selection mapping so that every positive example can be derived and no negative example can be derived. While the strict use of this definition is impractical, an approximate characterization, called the cover, is utilized which places greater emphasis on not representing the negative exemplars as opposed to representing the positive exemplars. Techniques such as relative least general generalization and inverse resolution [Dzeroski 1996] can then be applied to induce clausal definitions of the algorithm selection methodology. This forms the basis for building RS procedures using banks of selection rules.

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 [Muggleton 1995]) require that background knowledge be ground; meaning that only base facts can be provided as opposed to intensional information. This still renders the overall complexity exponential. In PYTHIA-II, we investigate the use of domain specific restrictions on the induction of hypotheses and analyze several strategies. First, we make syntactic and semantic restrictions on the nature of the induced methodology. For example, we require that a PDE solver should first activate a discretizer before a linear system solver (a different order of PDE solver parts does not make sense). An example of a semantic restriction is consistency checks between algorithms and their inputs. Second, we incorporate a generality ordering to guide the induction of rules and prune the search space for generating plausible hypotheses. Finally, since the software architecture of the domain specific RS has a natural database query interface, we utilize it to provide meta-level patterns for rule generation.

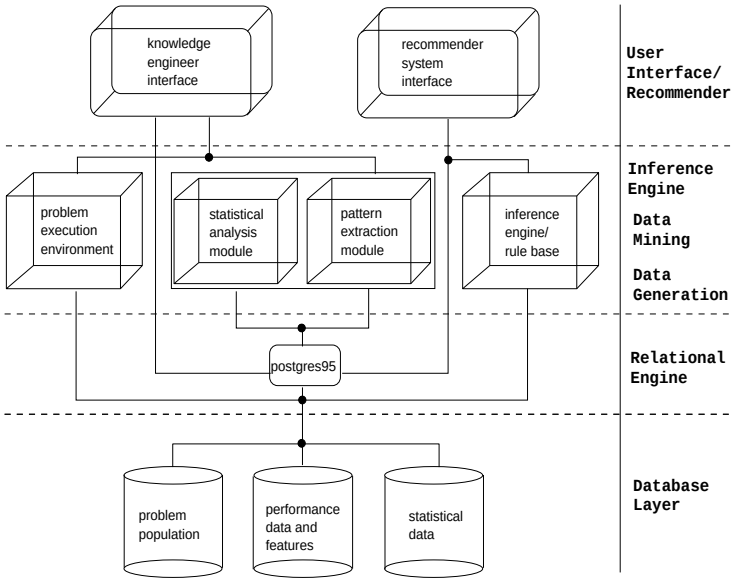

Fig. 2: The system architecture of PYTHIA-II. The recommender component consists of the recommender system interface and the inference engine. The KDD component is the rest.

PYTHIA-II also employs more restricted forms of eager-learning such as the ID3 (Induction of Decision Trees) [Quinlan 1986] system. It is a supervised learning system for top-down induction of decision trees from a set of examples and uses a greedy divide-and-conquer approach. The decision tree is structure where: (a) every internal node is labeled with the name of one of the predicting attributes; (b) the branches from an internal node are labeled with values of the node attribute; (c) every leaf node is labeled with a class (i.e., the value of the goal attribute). The training examples are tuples, where the domain of each attribute is limited to a small number of values, either symbolic or numerical. The ID3 system uses a top-down irrevocable strategy that searches only part of the search space, guaranteeing that a simple – but not necessarily the simplest – tree is found.

3. PYTHIA-II: A RECOMMENDER SYSTEM FOR SCIENTIFIC SOFTWARE

In this section we detail the software architecture of a domain specific RS, PYTHIA-II (see Figure 2), based on the methodology discussed above. Its design objectives include (i) modeling domain specific data into a structured representation using a database schema, (ii) providing facilities to generate specific performance data using simulation techniques, (iii) automatically collecting and storing this data, (iv) summarizing, generalizing, and discovering patterns/rules that capture the behavior of the scientific software system, and (v) incorporating them into the selected inference engine system. The system architecture has four layers:

—user interface layer

—data generation, data mining, and inference engine layer

—relational engine layer, and

—database layer.

The database layer provides permanent storage for the problem population, the performance data and problem features, and the computed statistical data. The next layer is the relational engine which supports an extended version of the SQL database query language and provides access for the upper layers. The third layer consists of three subsystems: the data generation system, the data mining system, and the inference engine. The data generation system accesses the records defining the problem population and procesess them within the problem execution environment to generate performance data. The statistical data analysis and pattern extraction modules comprise the data mining subsystem. The statistical analysis module uses a non-parametric statistical method to rank the generated performance data [Hollander and Wolfe 1973]. PYTHIA-II integrates a variety of publicly available pattern extraction tools such as relational learning, attribute value-based learning, and instance based learning techniques [Bratko and Muggleton 1995; Kohavi 1996]. These tools and our integration methods are discussed in Section 5.2. Our design allows for pattern finding in diverse domains of features like nominal, ordinal, numerical, etc.

The graphical user interface in the top layer allows the knowledge engineer to use the system to generate knowledge as well as query the system for facts stored in the database layer. The *recommender* is the end-user interface, and includes the inference engine. It uses the knowledge generated by the lower layers as an expert system. to answer domain specific questions posed by end users. The architecture of PYTHIA-II is extensible, with well defined interfaces among the components of the various layers.

4. DATA MODELING AND MANAGEMENT COMPONENTS OF PYTHIA-II

PYTHIA-II needs a powerful, adaptable database and management system with an open architecture to support its data generation, data analysis, automatic knowledge acquisition and inference process. The design requirements are summarized as follows:

—provide storage for the problem population (input to the execution environment) in a structured way, along with its parameters, features and constraints,

—support seamless data access by the user,

—support full extensibility to accomodate changes in the data size and schema.

PYTHIA-II uses POSTGRES95 [Stonebraker and Rowe 1986], an object-oriented, relational DBMS (Database Management System) which supports complex objects and which can easily be extended to new application domains by providing new data types, new operators, and new access methods. It also provides facilities for active databases and inferencing capabilities including forward and backward chaining. It supports the standard SQL language and has interfaces for C, Perl, Python, and Tcl. PYTHIA-II's relational data model offers an abstraction of the structure of the problem population which must be domain dependent. For example, the abstraction of a standard PDE problem includes the PDE system, the boundary conditions, the physical domain and its approximation in a grid or mesh format, etc. Each of the PDE problem specification components constitutes a separate

entity set which is mapped into a separate *table* or *relation*. Interactions among entities can also be modeled by tables representing *relationships*. In a higher level of abstraction, we use tables for batch execution of experiments and performance data collection, aggregate statistical analysis, and data mining. The *experiment* table represents a large number of problems as sequences of problem components to be executed at one time. A *profile* table collects sets of performance data records and profile specification information required by the analyzer. A *predicate* table identifies a collection of profile and feature records needed for data mining.

To illustrate the data modeling and management of PYTHIA-II, we now describe an example database schema specification for an RS for elliptic PDE software from the PELLPACK library. Throughout the remainder of this paper, we use this example to describe some aspects of the components of PYTHIA-II. The overall design of the system, however, is independent of the particular case study, and the elements of the system that are case study dependent will always be clearly indicated. In the data modeling component of PYTHIA-II, the schema specification must be modified for a each domain of scientific software. The PYTHIA-II database mechanisms are independent of the application domain, but the problem population, performance measures and features do depend on the domain.

—*Problem Population.* The atomic parts of a PDE problems are the equation, domain, boundary_conditions and initial_conditions. These entities must be defined consistent with syntax of the targeted scientific software. Solution algorithms are defined by a sequence of calls to library modules whose parts are grid, mesh, decompose, discretizer, indexer, linear_system_solver, and triple. The sequences entity contains an ordered listing of all these. Miscellaneous entities required for the benchmark include output, options and fortran_code. The schema for the database records for equation and sequence are as follows:

```
EQUATION
   name       text     -- record name
   system     text     -- software to solve equation
   nequations integer  -- number of equations
   equations  text[]   -- text describing equations to solve
   forfile    text     -- source code file (used in definition)
SEQUENCES
   name       text     -- record name
   system     text     -- software that provides the solver modules
   nmod       integer  -- number of modules in the solution scheme
   types      text[]   -- array of record types (e.g., grid, solver)
   names      text[]   -- array of module record names
   parms      text[]   -- array of module parameters
```

Instances of these from the case studies are as follows:

```
   name      | pde #39
   system    | pellpack
   nequations| 1
   equations | {"uxx + uyy + ((1.-h(x)**2*w(x,y)**2)/(&b))u = 0"}
   forfile   | /p/pses/projects/kbas/data-files/fortran/pde39.eq

   name  | uniform 950x950 proc 2 jacobi cg
   system| pellpack
   nmod  | 6
   types | {"grid","machine","dec","discr","indx","solver"}
   names | {"950x950 rect","machine_2","runtime grid 1x2",
            "5-point star","red black","itpack-jacobi cg"}
   parms | {"","","","","","itmax 20000"}
```

```
create table EXPERIMENT (
 name       text,    -- record name (primary key)
 system     text,    -- software identification used for program generation
 nopt       integer, -- number of options
 options    text[],  -- array of option record names (foreign key)
 noptparm   integer, -- number of parameter specific options
 optparm    text[],  -- array of option record names
 equation   text,    -- equation record which defines the equation
 neqnparm   integer, -- number of equation parameters
 eqnparm    text[],  -- array of equation parameter names
 domain     text,    -- domain record on which the equation is defined
 ndomparm   integer, -- number of domain parameters
 domparm    text[],  -- array of domain parameter names
 bcond      text,    -- boundary condition record
 nbcparm    integer, -- number of bcond parameters
 bcparm     text[],  -- array of bcond parameter names
 nparm      integer, -- number of parameters applied across all definitions
 parm       text[],  -- array of problem-wide parameters (no. of programs)
 sequences  text[],  -- names of the sequence records containing soln. schemes
 nout       integer, -- number of output records
 output     text[],  -- array of output record names
 nfor       integer, -- number of source code files to include
 fortran    text[]   -- names of the files to include
);
```

Fig. 3: The Experiment table specifies an experiment by listing the components of a PDE problem and sets of solvers (collection of Sequence records) to use in solving it.

The equation field attribute in the equation record uses the syntax of the PELLPACK PSE. The `&b` in the specification is for parameter replacement and the `forfile` attribute provides for additional source code to be attached to the equation definition. The sequences record shows an ordered listing of the module calls used to solve a particular PDE problem. For each module call in the list, the sequence identifies the module type, name and parameters.

—*Features.* Features and their representations are given in Section 2.1.

—*Experiments.* The experiment is a derived entity which identifies a specific PDE problem and a collection of PDE solver sequences. Generally, the experiment varies the solution algorithms parameters. This information is used to produce a set of driver programs to execute and produce performance data. See Figure 3 for the schema definition of an example experiment.

—*Rundata.* The rundata schema specifies the targeted hardware platforms, their characteristics (operating system, communication libraries, etc) and execution parameters.

—*Performance Data.* The performance schema is a very general, extensible representation of data generated by experiments. An instance of performance data generated by a PDE experiment is shown in Figure 4.

—*Knowledge-related Data.* Processing for the knowledge-related components of PYTHIA-II is driven by the profile and predicate records (not illustrated) which represent the experiments, problems, methods and features to be analyzed.

—*Derived Data.* Results from the data mining of the performance database are also put into the profile and predicate records. This data is processed by visualization and knowledge generation tools.

```
Field         | Value
-------------------------------------------
name          | pde54 dom02 fd-itpack-rscg SP2-17
system        | pellpack
comp_db       | linearalgebra
composite_id  | pde54 domain 02 fd-itpack-rscg
perfind_set   | pellpack-std-par-grd
pid           | 1432
sequence_no   | 17
eqparms       | pde #54 parameter set 5
solverseq     | 950x950 proc 4 reduced system cg
rundata       | IBM SP2 with 18 compute nodes
nfeature      | 5
featurenames  | {"matrix symmetric", "domain type", "boundary pieces", "problem type"}
featurevals   | {"no", "non-rectangular","8", "FD"}
nperf         | 1
perfnames     | {"number of iterations"}
perfvals      | {"830"}
nproc         | 4
nperfproc     | 0
nperfproc2    | 0
nmod          | 5
modnames      | {"domain processor","decomposer","discretizer","indexer","solver"}
ntimeslice    | 2
timeslice     | {"elapsed","communication"}
time          | {{{"3.16","0"},{"2.3","0"},{"4.2","0"},{"0.11","0"},{"135.4","1.24"}},
 {{"3.1","0"},{"2.46","0"},{"3.89","0"}, {"0.09","0"},{"135.4500024","36.74049"}},
 {{"3.1","0"},{"2.47","0"},{"3.91","0"}, {"0.08","0"},{"135.5499933","37.1304893"}},
 {{"3.1","0"},{"2.03","0"},{"4.139","0"}, {"0.04","0"},{"136.1499939","88.7300339"}}}
ntotal        | 4
total         | {"150.1600037","149.9700012","150.0200043","149.6300049"}
nmemory       | 4
memorynames   | {"number of equations", "x grid size","y grid size", "problem size"}
memoryvals    | {"224676","950","950","902500"}
nerror        | 3
errornames    | {"max abs error","L1 error","L2 error"}
errorvals     | {"0.0022063255","0.00011032778","0.00022281437"}
```

Fig. 4. An instance of performance data from a PDE experiment.

In this sample PYTHIA-II instantiation, the problem population has 13 problem specification tables (equation, domain, bcond, grid, mesh, dec, discr, indx, solver, triple, output, parameter, option) and 21 relationship tables (equation-discr, mesh-domain, parameter-solver, etc). Additional tables define problem features and execution related information (machine and rundata tables). In all, 44 table definitions are used for the PYTHIA-II database. Sections 7 and 8 give some examples of these tables.

5. KNOWLEDGE DISCOVERY COMPONENTS OF PYTHIA-II

We now describe the PYTHIA-II components in the top two layers of Figure 2.

5.1 Data Generation

The PYTHIA-II performance database may be pre-existing performance measures or the data may be produced by executing scientific software using PYTHIA-II. The scientific software operates entirely as a black box except for three I/O requirements that must be met for integration into PYTHIA-II. This section describes these requirements and illustrates how the PELLPACK software satisfies them.

First, it must be possible to define the input (i.e., the problem definition) us-

	Algorithm 1	Algorithm 2	···	Algorithm k
Problem 1	X_{11}	X_{12}	···	X_{1k}
Problem 2	X_{21}	X_{22}	···	X_{2k}
⋮	⋮	⋮	⋮	⋮
Problem n	X_{n1}	X_{n2}	···	X_{nk}
Rank	R_1	R_2	···	R_k
Average Rank	$R_{\bullet 1}$	$R_{\bullet 2}$	···	$R_{\bullet k}$

Table II: Algorithm ranking table based on Friedman rank sums using the two-way layout. X_{ij} is the performance of algorithm j on problem i and R_i and $R_{\bullet i}$ are the rank measures.

ing only information in an experiment record. The translation of an experiment into an executable program should be handled by a script written for the software which extracts the necessary information from the experiment record and generates the files or drivers for the software. For PELLPACK, the experiment record is translated to a *.e file*, which is the PELLPACK language definition of the PDE problem, the solution scheme, and the output requirements. The script is written in Tcl and consists of about 250 lines of code. The standard PELLPACK *preprocessing* programs convert the .e file to a Fortran driver and link the appropriate libraries to produce an executable program. The second requirement is that the software is able to operate in a batch mode. In the PELLPACK case, Perl scripts are used to execute PELLPACK programs, both sequential and parallel, on any number of platforms. The programs must be created and executed without manual intervention. Finally, the software must produce performance measures as output. A post-processing program must be written specifically to convert the generated output into PYTHIA-II performance records. Each program execution should insert one record into the performance database. The PELLPACK post-processing program is written in Tcl (350 lines of code) and Perl (300 lines of code).

Data generation (program generation, program execution, data collection) may take place inside or outside of PYTHIA-II. This process is domain dependent since problem definition records, software, and output files depdend on the domain.

5.2 Data Mining

Data mining in PYTHIA-II is the process of extracting and filtering performance data for analysis, generating solver profiles and ranks, selecting and filtering data for pattern extraction, and generating the knowledge base. Its principal components are the statistical analysis module (analyzer) and the pattern extraction module.

PYTHIA-II runs the analyzer as a separate process with a configuable input call, so various data analyzers can easily be integrated into it. The statistical analyzer is problem domain independent as it operates on the fixed schema of the performance records. All the problem domain information is distilled to one number measuring the performance of a program for a problem. The analyzer assigns a performance ranking to a set of algorithms applied to a problem population. It accesses the the performance data using a selected *predicate* record which defines the complete set of analyzer results used as input for a single invocation of the rules generator. The predicate contains (1) the list of algorithms to rank, and (2) a profile matrix, where each row represents a single analyzer run and the columns identify the *profile*

records to be accessed for that run. Table II illustrates the predicate's profile matrix; its columns represent algorithms and its rows represent problems as specified by a profile record. The X_{ij} are performance values (see below) computed by the analyzer. PYTHIA-II currently ranks the preformance of algorithms with Friedman rank sums [Hollander and Wolfe 1973]. This distribution-free ranking assumes nk data values from each of k algorithms for n problems. The analyzer can "fill in" missing values using various methods. The Friedman ranking proceeds as follows:

—For each problem i rank the algorithms' performances. Let r_{ij} denote the rank of X_{ij} in the joint rankings of $X_{i1}, ...X_{ik}$ and compute $R_j = \sum_{i=1}^{n} r_{ij}$.

—Let $R_{\bullet j} = \frac{R_j}{n}$ where R_j is the sum over all problems of the ranks for algorithms j, and then $R_{\bullet j}$ is the average rank for algorithm j. *Use the $R_{\bullet j}$ to rank the algorithms over all problems.*

—Compute $Q = q(\alpha, k, \infty)\sqrt{\frac{n \cdot k \cdot (k+1)}{12}}$ where $q(\alpha, k, \infty)$ is the critical value for k independent algorithms for experimental error α. $\mid R_u - R_v \mid > Q$ implies that algorithms u and v differ significantly for the given α.

The assignment of a single value, X_{ij}, to represent the performance of algorithm is not a simple matter. Even when comparing execution times, there are many parameters which should be varied for a serious evaluation: problem size, execution platform, number of processors (for parallel code), etc). The analyzer uses the method of least squares approximation of observed data to accomodate variations of problem executions. Thus, the relations between pairs of variables (e.g., time and grid size, time and number of processors) are represented linearly as seen in Figure 7 for Case Study 2. These profiles allow a query to obtain data of one variable for any value of another.

The pattern-extraction module provides automatic knowledge acquisition (patterns/models) from the data to be used by an RS. This process is independent of the problem domain. PYTHIA-II extends the PYTHIA methodology to address the algorithm selection problem by applying various neuro-fuzzy, instance-based learning and clustering techniques. The relational model of PYTHIA-II automatically handles any amount of raw data related manipulation. It has a specific format for the data used by the pattern extraction process, and filters transform this format (on the fly) to the format required by the various data mining tools integrated into PYTHIA-II. The goal is to accumulate tools that generate knowledge in the form of logic rules, if-then-else rules or decision trees.

PYTHIA-II first used GOLEM [Muggleton and Feng 1990], an empirical single predicate Inductive Logic Programming (ILP) learning system. It is a batch system that implements the *relative least general generalization* principle. We have experimented with other learning methods, e.g., fuzzy logic or neural networks, and have not found large differences in their learning abilities. We chose ILP because it seemed to be the easiest to use in PYTHIA-II; our selection of it is not the result of a systematic study of the effectiveness of learning methods. PYTHIA-II is designed so the learning component can be replaced if necessary. GOLEM generates knowledge in the form of logical rules which one can model in a language like first order predicate logic. These rules can then be easily utilized as the rule base of an expert system. We have also integrated PROGOL [Muggleton 1995], CN2, PEBLS,

and OC1 (the latter three are available in the MLC++ library [Kohavi 1996]).

5.3 Inference Engine

The recommender component of PYTHIA-II answers the user's questions using an inference engine and facts generated by the knowledge discovery process. It is is both domain dependent and case study dependent. We describe the recommender that uses knowledge generated by GOLEM. Each GOLEM logical rule has an information compression factor f measuring its generalization accuracy. Its simple formula is $f = p-(c+n+h)$ where p and n are the number of positive and negative examples, respectively, covered, while c and h are related to the form of the rule. The information compression factor is used for sorting the rules in decreasing order. The rules and the set of positive examples covered for each rule are passed to the recommender which then asks the user to specify the problem features. It uses the CLIPS inference engine to check for rules that match the specified features. Every rule found in this way is placed into the *agenda*. Rules are sorted in decreasing order based on the number of examples they cover, so the very first rule covers the most examples and will fire at the end of the inference process and determine the best algorithm. The recommender then goes through the list of positive examples associated with the fired rule and retrieves the example that has the most features in common with the user's problem.

The fact base of the recommender is then processed for this example to provide parameters for which the user needs advice. The fact base consists of all the raw performance data stored in the database. This information is accessed by queries generated on the fly, based on the user's objectives and selections. If the user objectives cannot be met, then the recommender decides what "best" answer to give, using weights specified by the user for each performance criterion. For the case studies in Sections 7 and 8, the final step is the recommendation of the best PDE solver to use. It also provides solver parameters such as the grid needed to achieve the solution accuracy within the given time limitations.

5.4 User Interface

PYTHIA-II can accomplish much of the work of knowledge discovery without using a graphical interface, for example

(1) Creating database records for the problem population and experiments: the SQL commands can be given directly inside the POSTGRES95 environment.
(2) Generating executable programs from the experiments: this is a separate process called from the domain specific execution environment, and can be called outside of PYTHIA-II.
(3) Executing programs: this process is controlled by scripts invoked by PYTHIA-II and can be called outside of PYTHIA-II since they operate on the generated files in some directory.
(4) Collecting data: the data collector is a separate domain specific process called by PYTHIA-II.

Graphical interfaces that assist in these tasks are useful for knowledge engineers unfamiliar with the structure of PYTHIA-II or the POSTGRES95 SQL language. These interfaces are provided by PYTHIA-II and shown in Figure 5.

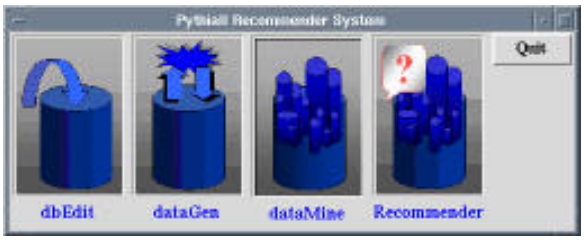

Fig. 5. PYTHIA-II's top level window.

The graphical interface to the POSTGRES95 database is dbEdit. Each PYTHIA-II record has a form presented when records of that type are selected for editing. Similarly, dataGEN facilitates the tasks involved in the data generation process, and frees the user from worrying about details such as: where the generated programs are stored, which scripts are available, where raw output data is located, and so on. DataMINE encompasses the data analysis and knowledge discovery. Even experienced users must perform these tasks inside PYTHIA-II. A template query is used to extract the performance data for the statistical analyzer. The query uses a profile record and may access hundreds of performance records to build the anlyzer input file. The pattern matching input specification for pattern matching is equally difficult to build. DataMINE presents a simple menu system that walks the user through all these steps. It is integrated with DataSplash [Olston et al. 1998] an easy-to-use integrated visual environment which is built on top of POSTGRES95 and therefore interacts with PYTHIA-II's database naturally.

6. DATA FLOW IN PYTHIA-II

PYTHIA-II has one interface for the knowledge engineer and another for end users. We describe the data flow and I/O interfaces between the main components of PYTHIA-II from the perspective of these two interfaces.

6.1 Knowledge Engineer Perspective:

The data flow in PYTHIA-II is shown in Figure 6, where boxes represent stored data, edges represent operations on the database, and self-edges represent external programs. The knowledge engineer begins with populating the problem database. The knowledge engineer specifies the domain in terms of the relational data model to match PYTHIA-II's database schema. Extensible and dynamic schema are possible. POSTGRES95 does not have a restriction imposed by the traditional relational model that the attributes of a relation be *atomic*[1].

An *experiment* combines problem records into groups, and a high level problem specification is generated by a program-based transformation of the experiment record into an input file for execution. The problem execution environment invokes the appropriate scientific software to generate data. For the example instantiation referred to in Sections 4 and 5, the execution environment consists of PELLPACK. The execution generates a number of output files, each containing performance and other information related to solving the problem. The input uses the specific

[1]This is sometimes referred to as the First Normal Form (1NF) of database systems.

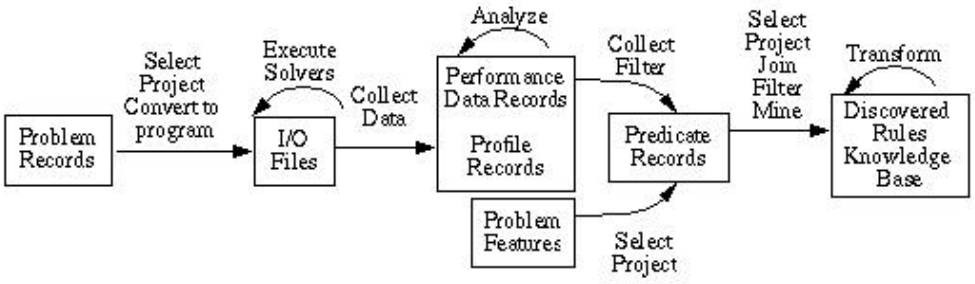

Fig. 6. Data flow and I/O for the knowledge engineer user interface.

schema of the problem record and the output format is specified by a system specific and user selected file template. The template lists the program used to collect the required output data. These data records keep logical references (called foreign keys) to the problem definition records so that performance can be matched with problem features by executing n-way joins during pattern extraction.

The statistical analyzer uses the performance data for ranking based on the parameter(s) selected by the user. The ranking produces an ordering of these parameters which is statistically significant (i.e., if the performance data shows no significant difference between parameters, they are shown as tied in rank). A *predicate* record defines the collection of profile records to be used in pattern extraction and allows a knowledge engineer to change the set of input profile records as easily as updating a database record. A filter program converts data to the input format required by the pattern extraction programs. PYTHIA-II currently supports GOLEM/PROGOL, the MLC++ (Machine Learning Library in C++) library, and others. These programs generate output in the form of logic rules, if-then rules or decision trees/graphs for categorization purposes. This process is open-ended, and tools like neural networks, genetic algorithms, fuzzy logic tool-boxes, rough set systems can be used.

6.2 End User Perspective:

The Recommender interface must adapt to a variety of user needs. Users of an RS for scientific computing are most interested in questions regarding accuracy of a solution method, performance of a hardware system, optimal number of processors to be used in a parallel machine, how to achieve certain accuracy by keeping the execution time under some limit, etc. PYTHIA-II allows users to specify problem characteristics plus performance objectives or constraints. The system uses facts to provide the user with the best inferred solution to the problem presented. If the user's objective cannot be satisfied. the system tries to satisfy the objectives (e.g., accuracy first, then memory constraints) based on the ordering implied by the user's performance weights.

7. CASE STUDY 1: PERFORMANCE EFFECTS OF SINGULARITES FOR ELLIPTIC PDE SOLVERS

To validate PYTHIA-II and its underlying KDD process, we reconsider a performance evaluation for a population of 2-dimensional, singular, elliptic PDE problems

[Houstis and Rice 1982]. The algorithm selection problem for this domain is

Select an algorithm to solve
$Lu = f \quad \text{on} \quad \Omega$
$Bu = g \quad \text{on} \quad \partial\Omega$
so that relative error $\epsilon_r \leq \theta$ and time $t_s \leq T$

where L is a second order, linear elliptic operator, B is a differential operator with up to first order derivatives, Ω is a rectangle, and θ, T are performance criteria constraints.

7.1 Performance Database Description

In this study, PYTHIA-II collects tables of execution times and errors for each of the given solvers using various grid sizes. The error is the maximum absolute error on the grid divided by the maximum absolute value of the PDE solution. The grids considered are 5x5, 9x9, 17x17, 33x33, and 65x65. The PDE solvers are from PELLPACK :

—5PT = 5-point star plus band Gauss elimination

—COLL = Hermite cubic collocation plus band Gauss elimination

—DCG2 = Dyakanov conjugate gradient for order 2

—DCG4 = Dyakanov conjugate gradient for order 4

—FFT2 = FFT9 (order=2) Fast Fourier transform for 5-point star

—FFT4 = FFT9 (order=4) Fast Fourier transform for 9-point star

—FFT6 = FFT9 (order=6) Fast Fourier transform for 6th order 9-point star

Defining the population of 35 PDEs and the experiments required 21 equation records with up to 10 parameter sets each, 3 rectangle domain records, 5 sets of boundary conditions records, 10 grid records, several discretizer, indexing, linear solver and triple records with corresponding parameters, and a set of 40 solver sequence records. Using these components, 37 experiments were specified, each defining a collection of PDE programs involving up to 35 solver sequences for a given PDE problem. Examples of these records are given in Section 4. The 37 experiments were executed on a SPARCstation20 with 32MB memory running Solaris 2.5.1 from within PYTHIA-II's execution environment (see Table III.) Over 500 performance records were created.

7.2 Data Mining and Knowledge Discovery Process

When the execution finished, the performance database was created. The dataMINE interface was used to access it using the predicate and profile records created for the case study. The rankings produced by the analyser for PDE problem 10-4 are, for example:

```
1. FFT6, 2. FFT4, 3. DCG4, 4. FFT2, 5. COLL, 6. DCG2, 7. 5PT.
```

The frequency for each solver being best for these 35 PDEs is

```
FFT4:  27.0 %          FFT6 : 10.8 %
COLL:  21.6 %          DCG2 :  5.4 %
5PT :  18.9 %          FFT2 :  2.7 %
DCG4:  13.5 %
```

Phases	Description	Implementation
Determine evaluation objectives	Evaluate the efficiency and accuracy of a set of solution methods and their associated parameters with respect to elapsed time, error and problem size.	Manual
Data preparation (1) selection (2) pre-processing	(1) problem population (2) measures: elapsed solver time, discretization error. (3) methods (4) Generate performance data.	POSTGRES95 SQL Tcl/Tk PERL
Data Mining	(1) Collect the data for error and time across all solvers, grid sizes (2) Use the method of least squares to develop linear approximations of time vs error across all grid sizes. Develop profiles of the methods for all problems, and rank the methods. (3) Use the rankings and the problem features to identify patterns and generate rules.	TCL/Tk PERL In-house statistical software PROGOL
Analysis of results	Domain experts ensure correctness of the results.	Manual
Assimilation of knowledge	Create an intelligent interface to utilize the knowledge to identify the "best method" with associated parameters for user's problems and computational objectives.	CLIPS

Table III. The PYTHIA-II process applied to Case Study 1.

Problem Component	Features
Equation	*first tier operator*: Laplace, Poisson, Helmholtz, self-adjoint, general *second tier operator*: analytic, entire, constant coefficients, *operator smoothness tier*: constant, entire, analytic *right-hand-side tier* : entire, analytic, singular(infinite), singular derivatives, constant coefficients, nearly singular, peaked, oscillatory, homogeneous, computationally complex *right-hand-side smoothness tier*: constant, entire, analytic, computationally complex, singular, oscillatory, peaked
Domain	unit square, $[a, b] \times [a + x, b + x]$, where x can vary $[a, b] \times [a + c, b + c]$, where c is a constant
Boundary Conditions	$U = 0$ on all boundaries $AU = f$ on all boundaries $BU_n = f$ on some boundaries $AU + BU_n = f$ on some boundaries constant coefficients, non-constant coefficients

Table IV. Features for the problem population of the benchmark case study.

Note that some solvers are not applicale to many of the PDEs. These rankings over all PDE problems and their associated features (see Table IV) were then used to mine rules. Examples of these rules are shown below. The first rule indicates that the method Dyakanov CG4 is best if the problem has a Laplace operator and the right-hand-side is singular.

```
best_method(A,dyakanov-cg4)        :- opLaplace_yes(A), rhsSingular_yes(A)
best_method(A,fft_9_point_order_4) :- opHelmholtz_yes(A), pdePeaked_no(A)
best_method(A,fft_9_point_order_4) :- solVarSmooth_yes(A), solSmoSingular_no(A)
best_method(A,fft_9_point_order_2) :- solSingular_no(A), solSmoSingDeriv_yes(A)
best_method(A,fft_9_point_order_6) :- opLaplace_yes(A), rhsSingular_no(A),
                  rhsConstCoeff_no(A), rhsNearlySingular_no(A), rhsPeaked_no(A)
best_method(A,fft_9_point_order_6) :- pdeSmoConst_yes(A), rhsSmoDiscDeriv_yes(A)
best_method(A,dyakanov-cg4)        :- opSelfAdjoint_yes(A), rhsConstCoeff_no(A)
best_method(A,dyakanov-cg4)        :- pdeJump_yes(A)
best_method(A,dyakanov-cg)         :- pdeSmoConst_yes(A), rhsSmoDiscDeriv_yes(A)
best_method(A,hermite_collocation) :- opGeneral_yes(A)
best_method(A,hermite_collocation) :- pdePeaked_yes(A)
```

PDE	5PT	COLL	DCG2	DCG4	FFT2	FFT4	FFT6
3–1	7 (7)	6 (4)	4 (6)	5 (5)	2 (2)	1 (3)	3 (1)
3–2	6 (6)	7 (7)	1 (5)	3 (3)	4 (4)	2 (2)	5 (1)
7–1	7 (7)	6 (3)	2 (5)	3 (5)	1 (4)	4 (1)	5 (1)
8–2	7 (7)	6 (5)	1 (4)	5 (6)	2 (2)	4 (3)	3 (1)
9–1	6 (6)	5 (5)	3 (4)	2 (3)	4 (2)	1 (1)	-
9–2	6 (6)	5 (5)	4 (4)	3 (3)	2 (2)	1 (1)	-
9–3	6 (6)	4 (5)	5 (3)	3 (3)	2 (2)	1 (1)	-
10–2	6 (6)	7 (7)	5 (5)	2 (4)	3 (3)	1 (2)	4 (1)
10–3	6 (6)	7 (7)	5 (5)	3 (4)	4 (3)	2 (2)	1 (1)
10–4	7 (5)	5 (7)	6 (4)	3 (6)	4 (3)	2 (2)	1 (1)
10–7	6 (6)	5 (7)	4 (5)	3 (3)	7 (3)	1 (2)	2 (1)
11–2	7 (7)	6 (6)	5 (5)	1 (3)	2 (3)	3 (2)	4 (1)
11–3	7 (6)	6 (6)	4 (5)	3 (4)	5 (3)	1 (2)	2 (1)
11–4	6 (6)	7 (7)	5 (5)	3 (4)	4 (3)	2 (2)	1 (1)
11–5	6 (6)	7 (7)	5 (4)	3 (4)	4 (3)	2 (2)	1 (1)
13–1	3 (3)	4 (4)	2 (1)	1 (1)	-	-	-
15–1	2 (2)	1 (1)	-	-	-	-	-
15–2	2 (2)	1 (1)	-	-	-	-	-
17–1	7 (7)	6 (6)	3 (5)	1 (3)	2 (4)	4 (2)	5 (1)
17–2	6 (6)	7 (7)	4 (4)	2 (5)	5 (3)	1 (2)	3 (1)
17–3	6 (6)	7 (7)	4 (4)	5 (5)	3 (3)	2 (2)	1 (1)
20–1	1 (1)	2 (2)	-	-	-	-	-
20–2	1 (1)	2 (2)	-	-	-	-	-
28–2	3 (2)	-	1 (1)	2 (2)	-	-	-
30–4	1 (1)	2 (2)	-	-	-	-	-
30–8	2 (2)	1 (1)	-	-	-	-	-
34–1	4 (4)	3 (2)	2 (2)	1 (1)	-	-	-
35–1	4 (4)	3 (2)	2 (2)	1 (1)	-	-	-
36–2	2 (1)	1 (1)	-	-	-	-	-
39–2	1 (1)	2 (2)	-	-	-	-	-
39–4	2 (2)	1 (1)	-	-	-	-	-
44–2	2 (2)	1 (1)	-	-	-	-	-
44–3	2 (2)	1 (1)	-	-	-	-	-
47–2	6 (6)	6 (6)	3 (5)	2 (4)	4 (3)	1 (2)	5 (1)
49–3	2 (2)	1 (1)	-	-	-	-	-
51–1	1 (1)	2 (2)	-	-	-	-	-
54–1	1 (1)	2 (2)	-	-	-	-	-

Table V: A listing of the rankings generated by PYTHIA-II and, in parentheses, the subjective rankings reported in [Houstis and Rice, 1982].

7.3 Knowledge Discovery Outcomes

The rules discovered confirm the assertion (established by statistical methods) in [Houstis and Rice 1982] that higher order methods are better for elliptic PDEs with singularities. They also confirm the general hypothesis that there is a strong correlation between the order of a method and its efficiency. More importantly, the rules impose an ordering of the various solvers for each of the problems considered in this study. Interestingly, this ranking corresponds closely with the subjective rankings published earlier (see Table V.) This shows that these simple rules capture much of the complexity of algorithm selection in this domain.

8. CASE STUDY 2: THE EFFECT OF MIXED BOUNDARY CONDITIONS ON THE PERFORMANCE OF NUMERICAL METHODS

We apply PYTHIA-II to analyze the effect of different boundary condition types on the performance of elliptic PDE solvers considered in the study of [Dyksen et al.

Record	Controlling information	Field data
Predicate	How many invocations of the analyzer?	24
	Profiles to be used for each invocation.	pde01_Dir-vs-Mix, pde01_Dir-vs-Neu, pde01_Mix-vs-Neu, pde02_Dir-vs-Mix, ...
	Items to rank.	numerical methods : DGC, DCG4, MG-00, 5PT, COLL
	Features to base rules on.	ElapsedTimeEffect_Dir2Mix, ElapsedTimeEffect_Dir2Neu, ...
Profile	Experiments used in a single analyzer run?	pde01-dirichlet, pde01-mixed, ...
	Profile graph x-axis values?	grid sizes
	Profile graph y-axis values?	relative increase in mixed execution elapsed time vs Dirichlet execution elapsed time : $(T_{mix} - T_{dir})/T_{dir}$
	Matching record identifier for profile graph building.	use perfdata record and match fields: classparms = dir vs. mix select on numerical methods
	Name of SQL query template.	dir.vs.mix

Table VI: Sample predicate and profile information for the relative elapsed times analysis for mixed vs. Dirichlet problem executions.

1988]. The PDEs for this performance evaluation are of the form

$$Lu = au_{xx} + cu_{yy} + du_x + eu_y + fu = g \quad \text{on} \quad \Omega$$
$$Bu = \alpha u + \beta s u_n = t \quad \text{on} \quad \partial\Omega$$

The parameters α and β determine the strength of the derivative term. The coefficients and right hand sides, a, c, d, e, f, g, s and t, are functions of x and y, and Ω is a restangle. The numerical methods considered are the modules (5PT, COLL, DCG2, DCG4) listed in Section 7.1 plus MG-00 (Multigrid mg00). The boundary condition types are defined as follows

—Dirichlet: $u = t$ on all sides.

—Mixed : $\alpha u + s u_n = t$ where $\alpha = 0$ or $\alpha = 2$ on one or more sides

—Nearly Neumann : $\alpha u + \beta s u_n = t$ where either $\alpha = 1$, $\beta = 1000$ or $\alpha = 0$, $\beta = -1$ on one or more sides.

Every PDE equation is paired with all three boundary condition types and is associated with three experiments. Each experiment consists of a problem defined by the PDE equation and boundary condition, which is solved by the five methods using five uniform grids. There are 75 program executions for each PDE. Performance data on elapsed solver time and various error measures are collected.

8.1 Performance Data Generation, Collection and Analysis

The PYTHIA-II database records (equations, domains, boundary_conditions, parameters, modules, solver_sequences and experiments) are defined using dbEdit, and the PDE programs are built and executed with PYTHIA-II's dataGen and the PELLPACK problem execution environment. All experiments were executed on a SPARCstation-20 SunOS 5.5.1 with 32 MB memory. About 600 records were

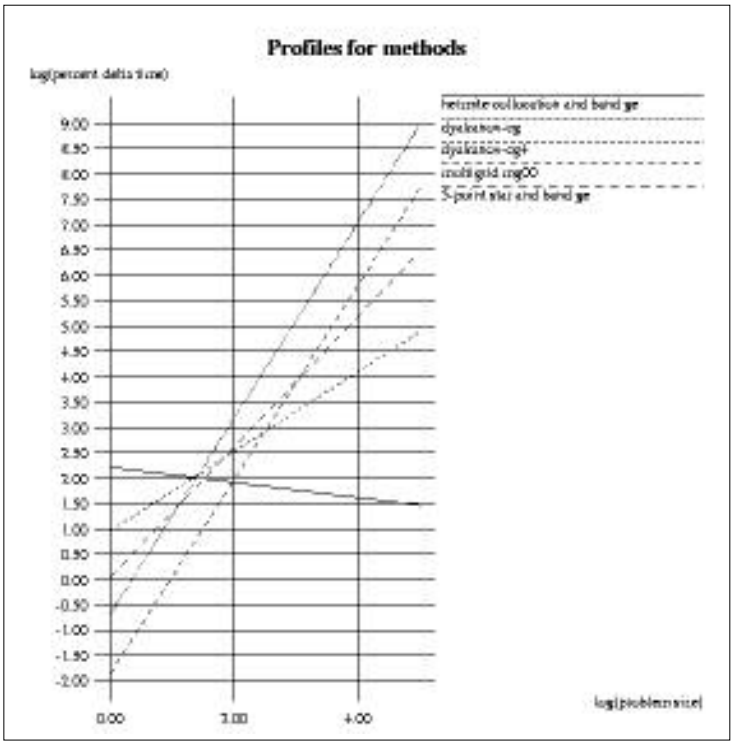

Fig. 7: Profile graph depicting the relative change of execution times between Dirichlet and Mixed problems as a function of the grid size for the five PDE solvers considered.

inserted into the performance database. The statistical analysis and rules generation are handled by dataMINE using the appropriate predicate and profile records which identify all parameters controlling the tasks.

The predicate names a matrix of profile records that identify the number and type of analyzer invocations. Then it identifies the boundary condition features used. The analyzer rankings and the predicate feature specifications are handed over to the rules generation process. Table VI lists, in part, the required predicate information. The predicate controls the overall analysis and the details are handled by the profile records. Each profile record identifies which fields of performance data are extracted, how they are manipulated, and how the experiment profiles for the analyzer are built. The result of the analysis is a ranking of method performance for the selected experiments. The query posed to the database by the profile extracts exactly the information (see Table VI) needed by the analyzer to answer this question. The complex query used for building the analyzer's input data is determined by profile field entries for x-axis, y-axis and field matching. In this case, the profile record builds sets of (x, y) points for each numerical method, where the x values are grid points and the y values are relative elapsed time changes for mixed boundary conditions with respect to Dirichlet conditions, changes in elapsed time for Neumann conditions with respect to Dirichlet conditions, and relative changes in error

for derivative conditions with respect to Dirichlet conditions. In all, 6 predicates and more than a hundred profiles were used.

8.2 Knowledge Discovery Outcomes

The rules derived in Case Study 2 are consistent with the hypothesis and conclusions stated in [Dyksen et al. 1988]. For the analysis, we use rankings based on the relative elapsed time profiles described above.

(1) *The performance of the numerical methods is degraded by the introduction of derivatives in the boundary conditions.* Profile graphs of the values for relative elapsed time changes δT for the mixed and Neumann problems with respect to the Dirichlet problems, $\delta T_{mix} = (T_{mix} - T_{dir})/T_{dir}$ and $\delta T_{neu} = (T_{neu} - T_{dir})/T_{dir}$ were generated by the analyzer for all methods over all grid values. It is observed that the values of $\delta T >> 0$ for most methods over all problem sizes. Thus, the presence of derivative terms slows the execution substantially except for the COLL solver for which the derivative term does not introduce a significant increase in elapsed time (see Figure 7.)

(2) *The COLL module was least affected.* Specifically, the increase in elapsed time when the derivative term was added was least for COLL. Note that even though the relative elapsed time was least for COLL, the total elapsed time was not. The frequencies for each solver to be best considering least relative time increase for changing from Dirichlet to mixed conditions, are:

```
COLL: 57.1 %          5PT:   0 %
DCG4: 28.6 %          MG-00: 0 %
DCG2: 14.3 %
```

The frequencies for each solver to be best for changing from Dirichlet to Neumann conditions, are

```
COLL: 42.9 %          DCG2:  14.3 %
DCG4: 21.4 %          MG-00:  7.1 %
5PT:  14.3 %
```

The final rules generated by PYTHIA-II for the elapsed time predicates are:

```
best_method(A,hermite_collocation) :  dir2mix(A).
best_method(A,hermite_collocation) :  dir2neu(A).
```

(3) *The fourth order modules COLL and DCG4 are less affected than second order modules.* The above statistics show that the fourth order modules are best 85% and 64% of the time. (see Figure 7 for the method ranking profile for pde04 generated by dir2mix predicate based on relative time). The rankings also show that fourth order modules are less affected by *mixed conditions* than by *Neumann conditions*, and that MG-00 and 5PT methods perform worst with the addition of derivatives in the boundary conditions.

Next, we consider ranking the methods for all PDE-boundary condition pairs using profile graphs involving problem size vs. elapsed time. The analysis does not consider relative increase in execution time for different boundary condition types, it ranks all methods over all PDE problems as in Case Study 1. The analysis ranks MG-00 as best method. It was selected 72% of the time as the faster method over all PDE problems. The analysis also showed that all methods had the same best-to-worst ranking for a fixed PDE equation and all possible boundary conditions.

In addition, these results show that some of these methods differ significantly when ranking with respect to execution times across the collection of PDE problems.

REFERENCES

Adve, V. S., Bagrodia, R., Brown, J. C., Deelman, E., Dube, A., Houstis, E. N., Rice, J. R., Sakellariou, R., Surdaram-Stukel, D., Teller, P. J., and Vernon, M. K. 2000. POEMS: End-to-end performance of large parallel adaptive computational systems. *IEEE Trans. Soft. Eng., to appear*.

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.

Bratko, I. and Muggleton, S. 1995. Applications of inductive logic programming. *Comm. ACM 38,* 11, 65–70.

Dyksen, W., Houstis, E., Lynch, R., and Rice, J. 1984. The performance of the collocation and galerkin methods with hermite bicubics. *SIAM Journal of Numerical Analysis 21*, 695–715.

Dyksen, W., Ribbens, C., and Rice, J. 1988. The performance of numerical software methods for elliptic problems with mixed boundary conditions. *Numer. Meth. Partial Differential Eqs. 4*, 347–361.

Dzeroski, S. 1996. Inductive logic programming and knowledge discovery in databases. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining*, pp. 117–152. AAAI Press/MIT Press.

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. 1996. From data mining to knowledge discovery: an overview. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining*, pp. 1–34. AAAI Press/MIT Press.

Hollander, M. and Wolfe, D. 1973. *Non-parametric Statistical Methods.* John Wiley and Sons.

Houstis, C., Houstis, E., Rice, J., Varadaglou, P., and Papatheodorou, T. 1991. Athena: a knowledge based system for //ELLPACK. *Symbolic–Numeric Data Analysis and Learning*, 459–467.

Houstis, E., Lynch, R., and Rice, J. 1978. Evaluation of numerical methods for elliptic partial differential equations. *Journal of Comp. Physics 27*, 323–350.

Houstis, E., Rice, J., Weerawarana, S., Catlin, A., Gaitatzes, M., Papachiou, P., and Wang, K. 1998. Parallel ELLPACK: a problem solving environment for PDE based applications on multicomputer platforms. *ACM Trans. Math. Soft. 24,* 1, 30–73.

Houstis, E. and Rice, J. R. 1982. High order methods for elliptic partial differential equations with singularities. *Inter. J. Numer. Meth. Engin. 18*, 737–754.

Houstis, E. N., Mitchell, W., and Papatheodorou, T. 1983. Performance evaluation of algorithms for mildly nonlinear elliptic partial differential equations. *Inter. J. Numer. Meth. Engin. 19*, 665–709.

Joshi, A., Weerawarana, S., Ramakrishnan, N., Houstis, E., and Rice, J. 1996. Neuro–fuzzy support for PSEs: a step toward the automated solution of PDEs. *Special Joint Issue of IEEE Computer & IEEE Computational Science and Engineering Vol. 3,* 1, pages 44–56.

Kohavi, R. 1996. MLC++ developments: data mining using MLC++. In S. e. a. Kasif (Ed.), *Working Notes of the AAAI-96 Fall Symposia on 'Learning Complex Behaviors in Adaptive Intelligent Systems'*, pp. 112–123. AAAI Press.

Moore, P., Ozturan, C., and Flaherty, J. 1990. Towards the automatic numerical solution of partial differential equations. In *Intelligent Mathematical Software Systems*, North-Holland, pp. 15–22.

Muggleton, S. 1995. Inverse entailment and PROGOL. *New Generation Computing Vol. 13*, pages 245–286.

Muggleton, S. and Feng, C. 1990. Efficient induction of logic programs. In S. Arikawa, S. Goto, S. Ohsuga, and T. Yokomori (Eds.), *Proceedings of the First International Conference on Algorithmic Learning Theory*, pp. 368–381. Japanese Society for Artificial Intelligence, Tokyo.

Muggleton, S. and Raedt, L. D. 1994. Inductive logic programming: theory and methods. *Journal of Logic Programming 19,* 20, 629–679.

Olston, C., Woodruff, A., Aiken, A., Chu, M., Ercegovac, V., Lin, M., Spalding, M., and Stonebraker, M. 1998. Datasplash. In *Proceedings of the ACM-SIGMOD conference on management of data*, Seattle, Washington, pp. 550–552.

Quinlan, J. R. 1986. Induction of decision trees. *Machine Learning 1,* 1, 81–106.

Ramakrishnan, N. 1997. Recommender systems for problem solving environments. Ph. D. thesis, Dept. of Computer Sciences, Purdue University.

Ramakrishnan, N., Houstis, E., and Rice, J. 1998. Recommender Systems for Problem Solving Environments. In H. Kautz (Ed.), *Working notes of the AAAI-98 workshop on recommender systems*. AAAI/MIT Press.

Ramakrishnan, N., Rice, J., and Houstis, E. N. 2000. GAUSS: An on-line algorithm recommender system for one-dimensional numerical quadrature. *ACM Trans. Math. Soft., to appear*.

Resnik, P. and Varian, H. 1997. Recommender systems. *Communications of the ACM Vol. 40,* 3, pages 56–58.

Rice, J. 1983. Performance analysis of 13 methods to solve the Galerkin method equations. *Lin. Alg. Appl. 53*, 533–546.

Rice, J. 1990. Software performance evaluation papers in TOMS. Technical Report CSD-TR-1026, Dept. Comp. Sci., Purdue University.

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

Stonebraker, M. and Rowe, L. A. 1986. The design of POSTGRES. In *Proceedings of the ACM-SIGMOD Conference on Management of Data*, pp. 340–355.

Verykios, V. S. 1999. Knowledge discovery in scientific databases. Ph. D. thesis, Computer Science Department, Purdue University.

Verykios, V. S., Houstis, E. N., and Rice, J. R. 1999. Mining the performance of complex systems. In *ICIIS' 99, IEEE International Conference on Information, Intelligence and Systems*, pp. 606–612. IEEE Computer Society Press.

Verykios, V. S., Houstis, E. N., and Rice, J. R. 2000. A knowledge discovery methodology for the performance evaluation of scientific software. *Neural, Parallel & Scientific Computations, to appear*.

Weerawarana, S., Houstis, E. N., Rice, J. R., Joshi, A., and Houstis, C. 1997. PYTHIA: a knowledge based system to select scientific algorithms. *ACM Trans. Math. Soft. 23*, 447–468.