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

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

Vassilis S. Verykios
College of Information Science and Technology
Drexel University
Philadelphia, PA 19104, USA.

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

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

Abstract

Very often scientists are faced with the task of locating appropriate solution software for their problems and then selecting from among many alternatives. In [Houstis et al. 1991] we had proposed an approach for dealing with this task by "processing" performance data of the targeted software. The validation of the approach is described in [Weerawarana et al. 1997] and realized by a customized implementation referred to as PYTHIA. 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 this issue, we present in this paper PYTHIA-II — a system which combines a general *knowledge discovery in databases* (KDD) methodology and *recommender*

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.

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 subsets of software from the PELLPACK library [Houstis et al. 1998]. From the end-user perspective, PYTHIA-II allows users to specify the problem to be solved and their computational objectives. In turn, the system (i) selects the software resources available for the user's problem, (ii) suggests parameter values, and (iii) provides phenomenological assessment of the recommendation provided. For the development of "testing" software repositories, PYTHIA-II provides all the necessary facilities to set up database schemas for testing suites and associated performance data. 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 easily applied to any software domain.

1. INTRODUCTION

Complex problems, whether scientific, engineering or societal, are most often solved today by utilizing public domain or commercial libraries or some form of problem solving environments (PSEs). Existing software modules are characterized by a significant number of parameters affecting its 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 that implement the various phases of the computation. Thus, the task of selecting the best software and the associated algorithmic/hardware parameters for a particular problem or 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 validation of the approach is described in [Weerawarana et al. 1997] and realized by an implementation (referred to as PYTHIA) restricted to a specific performance evaluation study. 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 specific domains of scientific software/hardware artifacts. In this paper, we describe the PYTHIA-II architecture and an instance of an RS for PDE software which utilizes the PYTHIA-II infrastructure.

Given a problem description from a known class of problems, along with some performance criteria, PYTHIA-II provides a knowledge based technology for the selection of the most efficient software/machine pair and estimates values for the associated parameters involved. It has the ability to make recommendations by combining attribute-based elicitation of specified problems and matching them against those of predefined "dense" population of similar types of problems. We describe two case studies that cover software for elliptic partial differential equations found in the problem solving environment PELLPACK [Houstis et al. 1998]. The ini-

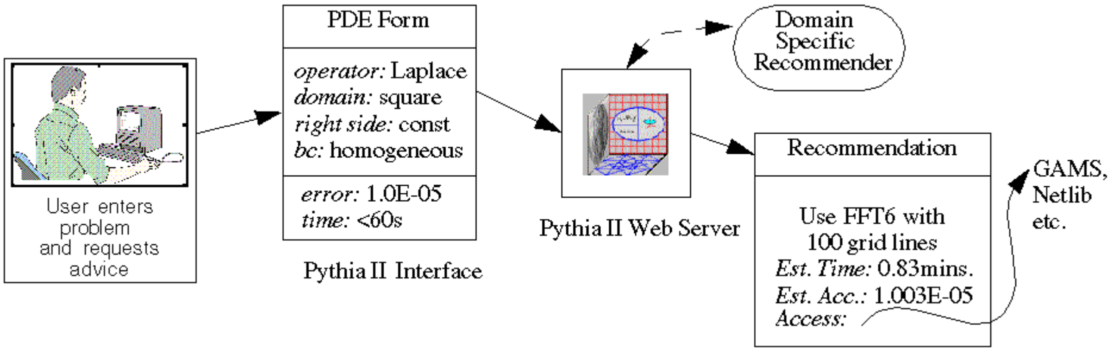

Fig. 1: The recommender component of PYTHIA-II implemented as a web server providing advice to users.

tial version of PYTHIA-II is built as a foundational system that can be enlarged into a national software recommender service for the entire scientific community by making it available as a network server.

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 (and more subdivisions such as linear, first order, if necessary) 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 a time constraint (measured on a Sun SPARCstation 20, for instance) and an error requirement to be satisfied. As shown, the recommender interface contacts the PYTHIA-II (web) server on the user's behalf which, in turn, interfaces with a domain specific RS. The RS uses the knowledge acquired by the learning methodology presented in this paper to perform software selection. Having determined a good algorithm, the RS consults databases of performance data to determine the solver parameters, such as grid lines to use with a PDE discretizer. Estimates of the time and accuracy with the recommended algorithm are also presented. Note that the recommender 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 the PYTHIA-II system. A fundamental software architecture for an RS based on the PYTHIA-II approach is presented in Section 3. For clarity of the presentation only, the functionality of PYTHIA-II is sometimes described in terms of specific targeted PDE software. Towards this end, we include in Section 4 a database schema appropriate for building an RS for elliptic PDE software from the PELLPACK library. The description of the data management subsystem of PYTHIA-II is presented in Section 5. Section 6 outlines the knowledge discovery components of PYTHIA-II. The data flow in PYTHIA-II is illustrated in Section 7. The results of a validation of PYTHIA-II system for two case studies can be found in Sections 8 and 9.

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, and is entirely automated.
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 an intelligent 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 layered methodology is very similar to procedures adopted in the performance evaluation of scientific software.

2. A RECOMMENDER METHODOLOGY FOR SCIENTIFIC SOFTWARE

An RS can be viewed as an intelligent system that 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 that will be of interest [Ramakrishnan 1997; Ramakrishnan et al. 1998; Resnik and Varian 1997]. We define an RS for software/hardware artifacts as a system that uses stored artifact performance data on a population of predefined problems and machines to locate and suggest efficient artifacts for the solution of 'similar' problems. Recommendation becomes necessary when user requests or objectives cannot be properly represented as ordinary database queries. In this paper we present an RS, PYTHIA-II, that assists scientists in selecting suitable software for the problem at hand, in the presence of practical constraints on accuracy, time and cost. Following, we describe the complexity of this problem, the research issues that must be addressed, and a methodology for resolving them.

Awareness of the algorithm selection problem has its origins in an early paper by Rice [Rice 1976]. Even for routine tasks in computational science, this problem is ill-posed and quite complicated. The difficulty in algorithm selection is primarily due to the following factors:

—The space of applicable algorithms for specific problem subclasses is inherently large, complex, ill-understood and often intractable to explore by brute-force means. Approximating the problem space by a representation (feature) space also introduces an intrinsic error in the modeling sense.

—Depending on the way the problem is (re)presented, the space of applicable algorithms changes; some of the better algorithms sacrifice generality for performance

```
-- table no 1
create table FEATURE (
 name       text,    -- record name (primary key)
 nfeatures  integer, -- no. of attributes identifying this feature
 features   text[],  -- numeric/symbolic/textual identification
 forfile    text     -- file-based feature information
);
```

Fig. 2. Schema for the feature record.

```
-- table no 3
create table EQUATION_FEATURE (
 name       text,    -- relation record name (primary key)
 equation   text,    -- name of equation with these features (foreign key)
 feature    text     -- name of record identifying features (foreign key)
);
```

Fig. 3: Schema for an example feature relation record; foreign keys identify the relation between an equation (PDE problem definition object) and its features

.

and have specially customized data structures and routines fine tuned for particular problems or their reformulations.

—Both specific features of the given problem and algorithm performance information need to be taken into account when deciding on 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 should also be able to obtain useful indicators of domain complexity and behavior, such as high level qualitative information about the relative efficacies of algorithms.

—There is an inherent uncertainty in interpreting and assessing the performance measures of a particular algorithm for a particular problem. Minor differences in algorithm implementations can produce large differences in performance measures that render relying on purely analytic estimates impractical.

—Distribution and evolution of the knowledge corpus for problem domains makes it difficult to assimilate relevant information; techniques are required that allow distributed recommender systems to coexist and cooperate together.

A methodology for building an RS for scientific artifacts which uses a *knowledge discovery in databases* (KDD) process is defined in Table I. Its implementation, PYTHIA-II, is discussed in Section 3. Assuming a "densely" distributed set of benchmark problems from the targeted application domain, this methodology uses a three-pronged strategy: feature determination of problem instances, performance evaluation of scientific software, and the automatic generation relevant knowledge for an RS from such data. Following, we address each of these aspects.

2.1 Problem Features

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

Fig. 4: Instances of a feature record (left) and a relation record (right) showing the correspondence between the equation `pde #3` and its feature `opLaplace`.

The applicability and efficiency of algorithms/software depends significantly on the features of the targeted problem domain. Identifying and characterizing problem features of the problem domain is a fundamental problem in software selection. Even if problem features are known, difficulties arise because the overall factors influencing the applicability (or lack) of an algorithm in a certain context are not very well understood. The way problem features affect methods is complex, and algorithm selection might depend in an unstable way on the features. 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 not reflect the underlying mapping well. Furthermore, a good selection methodology might fail because the features are given an attribute-value meaning and assigned measures of cardinality in a space where such interpretations are not appropriate. 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. In the current implementation of PYTHIA-II, this phase is implemented by the knowledge engineer.

Figures 2 and 3 show the database schema for a feature and a feature relation, respectively. The relation record shows how PYTHIA-II represents the connection between problem definition entities (e.g., PDE equations) and their features. Some instances of these records for the PDE case study are shown in Figure 4.

2.2 Performance Evaluation

The performance evaluation phase implemented in PYTHIA-II is based on well established methodologies for scientific software [Rice 1969; Boisvert et al. 1979; Casaletto et al. 1969; Dodson et al. 1968; Dyksen et al. 1984; Houstis et al. 1978; James and Rice 1967; Konig and Ullrich 1990; Moore et al. 1990; Rice 1983; Rice 1990]. While there are many important factors that contribute to the quality of numerical software, we illustrate our ideas using speed and accuracy. Even though more important (and more difficult to characterize) attributes such as reliability, portability, documentation, etc., are ignored in this discussion, our methodology can handle such features as well by utilizing the data storage scheme used in PYTHIA-II.

Accuracy may be measured by several means; we chose either a function of the norm of the difference between the computed solution and the true solution or an estimate of the error guaranteed by an approximation algorithm. Speed is normally measured by the time required to execute the appropriate software/routines in some execution environment. The PYTHIA-II problem evaluation environment ensures that all performance evaluations are made in a consistent manner; their outputs are automatically coded in the form of predicate logic formulas. We deliberately resort to attribute-value encodings when the situation demands it; for instance, the representation of linearized performance profiles for solvers is useful to obtain interpolated values of grid and mesh parameters for PDE problems.

2.3 Reasoning and Learning Techniques for Generating Software Recommendations

There are many approaches to generate recommendations for artifacts. For software selection, we have adopted one that is based on a multi-modal learning approach. Multimodal reasoning methods integrate different AI approaches to leverage their individual strengths. The PYTHIA-II system is a general framework enabling the integration of a range of reasoning and learning techniques. 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]. In the remainder of this section, we describe the CBR and ILP approaches and explain their use. Such learning and reasoning systems can typically be characterized as either 'lazy learning' or 'eager learning' paradigms.

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.) A wealth of evidence from psychology suggests that people compare new problems to ones they have seen before, using some metric of similarity to make judgements. They use the experience gained in solving 'similar' problems to devise a strategy for solving the present one. This strategy might involve a simple retrieval of a strategy (that has worked well in the past), tailoring a stored case to the situation at hand, and/or predictions of the likely outcome if a certain selection is followed. 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. Instance-based approaches such as statistical nearest neighbor selection also form part of the CBR landscape.

ILP systems, on the other hand, are an 'eager mechanism' 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. ILP techniques are also useful in distinguishing between the various features of the problem domain as being suitable for representation vs. discrimination. 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 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. A first restriction to function free horn clauses makes the problem decidable. Most commercial systems (like Golem and PROGOL [Muggleton 1995]) further 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 effect of domain specific restrictions on the induction of hypotheses and analyze several strategies. First, we make use of syntactic and semantic restrictions on the nature of the induced methodology. An example of a syntactic restriction would be 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. This ordering is used to prune the search space for generating plausible hypotheses and to aid in abduction. Finally, since the software architecture of the domain specific RS is augmented with a natural database query interface, we utilize this aspect to provide meta-level patterns for rule generation.

PYTHIA-II also employs more restricted forms of eager-learning paradigms, 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. Algorithms for inducing decision trees follow a greedy divide-and-conquer approach and are outlined as follows:

—Begin with a set of examples called the training set, T. If all examples in T belong to one class, then stop.

—Consider all tests that divide T into two or more subsets. Score each test according to how well it splits up the examples.

—Choose "greedily" the test that scores the highest.

—Divide the examples into subsets and run this procedure recursively on each subset.

A decision tree is a tree-like knowledge representation structure where: (a) every internal node is labeled with the name of one of the predicting attributes; (b) the branches coming out from an internal node are labeled with values of the attribute in that node; (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. A simple tree can be generated by a suitable selection of attributes. In ID3, an information-based heuristic is used to select these attributes. The heuristic selects the attribute providing the highest information gain, i.e., the attribute which minimizes the information needed in the resulting subtrees to classify the elements.

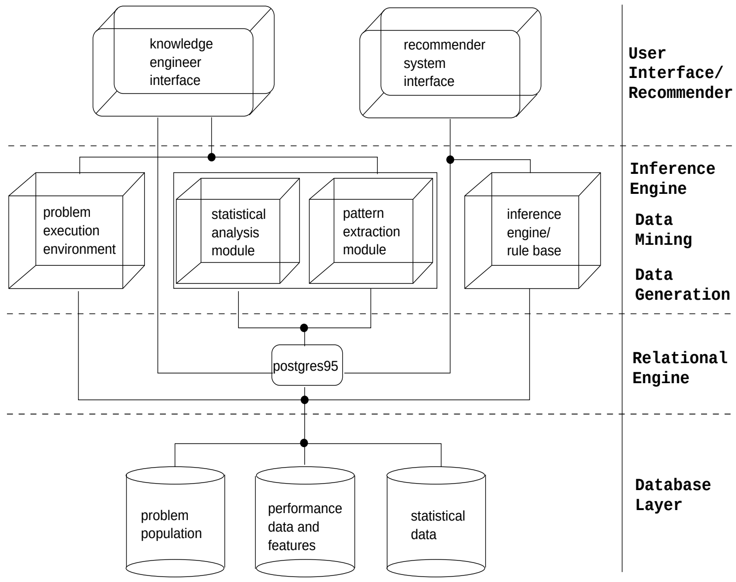

Fig. 5: The system architecture of PYTHIA-II. The recommender component consists primarily of the recommender system interface and the inference engine subsystems. The KDD component consists of the remaining subsystems (knowledge engineer interface down to the database layer).

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 5) based on the recommendation methodology and its components discussed above. Its design objectives include (i) modeling domain specific data into a structured representation as expressed by a database schema, (ii) providing facilities for the generation of system specific performance data by using simulation techniques, (iii) automatically collecting and storing this data, (iv) summarizing, generalizing, and discovering hidden patterns/rules that capture the behavior of the scientific software system by expressing them in a high-level logic-based representation language, and finally (v) incorporating them into the selected inference engine system.

The hierarchical architecture of the system consists of 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 the required functionality for the stored data to be accessible to the upper layers. The third layer consists of three subsystems: the data generation system, the data mining system, and the inference engine.

```
create table EQUATION (
 name       text,    -- record name (primary key)
 system     text,    -- software that solves equations of this type
 nequations integer, -- number of equations
 equations  text[],  -- text describing equations to solve
 forfile    text     -- source code file (used in equation definition)
);
```

Fig. 6: Equation records list the equations; terms are defined using the syntax of the scientific software.

The data generation system accesses the records defining the problem population and procesess them within the problem execution environment, invoking integrated scientific software for solving the problem and generating performance data. The statistical data analysis module and the pattern extraction module comprise the data mining subsystem. The statistical analysis module is a prototype software implementation of a non-parametric statistical method applied to the generated performance data. PYTHIA-II integrates a variety of publicly available pattern extraction tools adhering to the different paradigms implemented by various software packages, such as relational learning, attribute value-based learning, as well as instance based learning techniques. This design allows for pattern finding in diverse domains of features like nominal, ordinal, numerical, etc.

In the highest layer, a graphical user interface allows the knowledge engineer to exploit the capabilities of the system for generating knowledge as well as query the system for facts stored in the database layer. The *recommender* is the RS (end-user) interface, and includes the inference engine. This component of the RS uses the knowledge generated by the lower layers, encoding it appropriately as a knowledge base for an expert system. The facts generated by the knowledge discovery process and stored in the database drive the inference process, allowing the recommender 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. SAMPLE PYTHIA-II INSTANTIATION

For a better understanding of the functionality and implementation of PYTHIA-II components and the data flow among them, we describe a database schema specification used for producing a RS for elliptic PDE software from the PELLPACK library. The components can easily be modified for other domains of scientific software.

—*Problem Population.* The (atomic) entities which describe the PDE problems include equation, domain, boundary_conditions and initial_conditions. Field attributes for these entities must be defined in a manner consistent with the syntax of the targeted scientific software. Solution algorithms are defined by calls to library modules of the software; the modules are represented by entities which include grid, mesh, decompose, discretizer, indexer, linear_system_solver, and triple. In addition, the sequences entity contains an ordered listing of all modules

```
create table SEQUENCES (
 name       text,    -- record name (primary key)
 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, discr, solver)
 names      text[],  -- array of record names (foreign key)
 parms      text[]   -- array of module parameters (foreign key)
);
```

Fig. 7: A solver sequence record lists the order of module processing to solve a PDE problem; the sequence is translated to library calls from software associated with the named system.

```
Field     | Value
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


Field | Value
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"}
```

Fig. 8. Instances of equation and sequence records from the PDE benchmark study.

used in the solution process of a PDE problem. Miscellaneous entities required for the benchmark include output, options and fortran_code. Figures 6 and 7 show the schema for the equation and sequences records, respectively. Instances of an equation and sequence record for the PDE population are shown in Figure 8. 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.* An explanation of the features and their database representation is given in Section 2.1.

—*Experiments.* The experiment is a derived entity which identifies a specific PDE problem and lists a collection of sequences to use in solving it. Generally, the experiment covers a range of solution algorithms with varied parameters; it is translated to a collection of driver programs which are executed to produce performance data corresponding to the solution algorithms and execution platform.

```
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. 9: The experiment record specifies the components of a PDE problem and identifies the collection of sequences to use in solving it.

See Figure 9 for the schema definition.

—*Rundata.* The rundata schema specifies the targeted hardware platforms, their characteristics (operating system, communication libraries, etc) and execution parameters. The rundata and experiment record fully specify an instantiation of performance data.

—*Performance Data.* The performance schema is a very general, extensible representation of data generated by experiments. An instance of performance data generated by the PDE benchmark is shown in Figure 10.

—*Knowledge-related Data.* Processing for the knowledge-related components of PYTHIA-II is driven by the profile and predicate records. These schema represent the set of experiments, problems, methods and features which should be considered for analysis. An instance of the predicate schema is given in Figure 11.

—*Derived Data.* Data resulting from the data mining of the performance database is stored back into the profile and predicate records. This data is processed by visualization and knowledge generation tools.

5. DATA MODELING AND MANAGEMENT COMPONENTS OF PYTHIA-II

The quantity of information generated and manipulated by PYTHIA-II calls for a powerful and adaptable database and database management system with an open

```
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 points", "boundary pieces",
 "problem type"}
featurevals   | {"no", "non-rectangular","3800","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          | {{{"3.1600001","0"},{"2.3499999","0"},{"4.1900001","0"},
                  {"0.11","0"}, {"135.0400043","1.2499995"}},
                {{"3.1300001","0"},{"2.46","0"},{"3.8900001","0"},
                  {"0.09","0"},{"135.4500024","36.74049"}},
                {{"3.1300001","0"},{"2.47","0"},{"3.9100001","0"},
                  {"0.08","0"},{"135.5499933","37.1304893"}},
                {{"3.1700001","0"},{"2.03","0"},{"4.1399999","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. 10. An instance of performance data from the PDE benchmark.

architecture. PYTHIA-II's operational strength relies on the data modeling that supports the data generation, data analysis, automatic knowledge acquisition and inference process. The design requirements of the two lower level layers of the system's architecture can be summarized as follows:

```
Field         | Value
name          | PELLPACK Solution Methods Study
reference     | pellpack
num_rankings  | 1
max_num_blocks| 37
prof_recs     |{{"pde3-1","pde3-2","pde7","pde8-1","pde8-2","pde8-4",
                 "pde9-1", "pde9-2", "pde9-3", "pde10-2", "pde10-3"}}
best          | method
nbest         | 7
bestlist      | {"fft 9pt order 2","fft 9pt order 4","fft 9pt order 6",
                 "5point star & bandge", "herm coll & bandge",
                 "dyakanov-cg", "dyakanov-cg 4"}
featurelist   | {"operator","right-hand-side","domain","bconds","matrix"}
possiblevalues| {{"opLaplace","opPoisson","opHelmholtz","opGeneral"},
                 {"rhsEntire","rhsConstCoeff", "rhsSingular", "rhsAnalytic"}}
recordlist    | {"equation","equation","domain","bcond","perfdata"}
indexlist     | {"featurevals[1]","featurevals[5]"}
```

Fig. 11. Predicate record from the PDE benchmark.

—to provide storage for the problem population (input data to the execution environment) in a structured way, and to keep track of the population parameters, features and constraints,

—to support seamless data access by the user through a graphical interface or by a scripting language,

—to support fully extensible functionality for an environment that keeps changing not only in the size of the data but also in the schema.

The selected system, POSTGRES95 [Stonebraker and Rowe 1986], is an object-oriented, relational DBMS 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 to the user. It also provides facilities for active databases (i.e., alerters can send a message to a user calling for attention to a problem, and triggers can propagate updates in the database to maintain consistency) and inferencing capabilities including forward and backward chaining. It supports the standard SQL language with a number of extensions, and programming interfaces for C, Perl, Python, and Tcl.

PYTHIA-II's database is designed so its relational data model offers an abstraction of the structure of the problem population. This abstraction is (and must be) domain dependent, since the relational model defines benchmark applications from a selected domain which are executed to produce performance data. 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, a possible decomposition of the discrete or continuous domain for parallel execution, various solution modules (e.g., a discretizer or linear system solver), output modules, as well as parameter sets for any of these problem components. Each of the PDE problem specification components constitutes a separate *entity set*. In the relational model, each entity set is mapped into a separate *table* or *relation*.

Apart from these tables, a number of interesting static or dynamic interactions among entities can also be modeled in the relational model by tables representing *relationships*.

In a higher level of abstraction, we use an explicit hierarchy of flat tables to cope with batch execution of experiments and performance data collection, aggregate statistical analysis, and data mining. The *experiment* table is introduced as an intermediate virtual entity that represents a large number of problems in the form of sequences of problem components to be processed at one time by the execution environment for generating performance data. A *profile* table collects sets of performance data records and profile specification information required by the analyzer. A *predicate* table is another virtual entity that identifies a collection of profile and feature records needed for data mining.

In case of the RS for elliptic PDE software considered in the previous section, the current problem population is defined by 13 problem specification tables (equation, domain, bcond, grid, mesh, dec, discr, indx, solver, triple, output, parameter, option) and 21 relationship tables (including 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 to configure the database for PYTHIA-II. Sections 8 and 9 give some examples of these tables definitions within the context of the two case studies considered.

6. KNOWLEDGE DISCOVERY COMPONENTS OF PYTHIA-II

This section describes the functionality of the components of PYTHIA-II contained in the top two layers of Figure 5.

6.1 Data Generation

Information in the performance database drives PYTHIA-II's data analysis and rule generation. The performance database may be a pre-existing store of performance measures or the data may be produced by executing scientific software within the problem execution environment. PYTHIA-II is independent of the characteristics and functionality of the software, and it imposes no requirements or restrictions on the internal operation of the software. In fact, it allows the scientific software to operate entirely as a black box. There are, however, three I/O requirements that must be met by software to be integrated into PYTHIA-II. This section describes these requirements and demonstrates how the PELLPACK software satisfies them. PELLPACK is currently the only scientific software available through the execution environment; it has been used successfully to generate many thousands of performance data records.

First, it must be possible to define the input to the scientific software, (i.e., the problem definition) using only the information contained in an experiment record. The translation of an experiment into an executable program should be handled by a front-end converter written specifically for the software. Its task is to extract the necessary information from the experiment record, and generate the files or drivers required by this software. In the case of 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 converter is written in Tcl and consists of about 250 lines of code. After the .e file is generated, 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 scientific software be able to operate in a "batch" mode when executing PDE programs. In the PELLPACK case, Perl scripts are used to execute PELLPACK programs, both sequential and parallel, on any of the supported platforms. Whatever the number of "programs" defined by a single experiment, that number of programs must be processed and executed without manual intervention.

Finally, the scientific software must produce output files containing values for performance measures that can be used by PYTHIA-II to evaluate the performance of the program. PYTHIA-II does not require any special format since a post-processing program must be written specifically for the software to handle the conversion of the generated output into performance records. Each program execution should result in the insertion of one performance record into the performance database. The PELLPACK data collection program is written in Tcl (350 lines of code) and Perl (300 lines of code), and is responsible for creating performance records that represent the data produced by PELLPACK program executions.

The execution environment is implemented in a modular and flexible way, allowing any or all of the data generation phases (program generation, program execution, data collection) to take place inside or outside of PYTHIA-II. This process is domain dependent since it accesses the domain dependent problem definition records, executes programs by invoking domain specific software and collects data by processing domain specific output files.

6.2 Data Mining

Data mining is the key part of KDD and encompasses the process of extracting and filtering performance data for statistical analysis, generating solver profiles and ranking them, selecting and filtering data for pattern extraction, and generating the knowledge base. The two components involved in this process are the statistical analysis module (analyzer) and the pattern extraction module.

PYTHIA-II runs the analyzer as a separate process, sending it an input file and a set of parameters for output specification. Since the call to the analyzer is configurable, data analyzers can easily be integrated into the system. The statistical analyzer is independent of the problem domain since it operates on the fixed schema of the performance records. The current analyzer was developed in-house.

The task of the statistical analyzer is to assign a ranking to a set of algorithms for a selected problem population based on *a priori* determined performance criteria. It assumes that the algorithms are executed on the selected problems, and that the resulting performance measures for each execution are collected and inserted in the database. The analyzer accesses the database to extract the performance data based on the specification of a selected *predicate* record.

A predicate record defines the complete set of analyzer runs which are to be used as input for a single invocation of the rules generator. The predicate fields of interest to the analyzer are (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. Each profile record specifies how the analyzer should gather and assess the performance measures produced by one

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

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.

problem execution. Table II shows how the analyzer interprets one row of the predicate's profile matrix. The table columns are the specified algorithms, and the table rows are the problems represented by the profiles specified in a single row of the predicate' s profile matrix. The X_{ij} are performance values (see below) computed by the analyzer based on the profile record specification for problem i and algorithm j.

The process for ranking the algorithms uses an analysis for multiple comparisons and contrast estimators based on Friedman rank sums. The two-way layout associated with distribution-free testing is shown in Table II, which assumes nk data values from each of k algorithms for n problems. This assumption is not strictly necessary; the analyzer can "fill in" missing values using various methods, for example, averaging values in the algorithm column. The 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 $R_{\bullet j}$ are the desired algorithm ranks.

It remains to discuss the methods used to compute the X_{ij}. The assignment of a single value to represent the performance of algorithm j for problem i, which can then be compared to other performance values in the framework of the two-way layout, is not a simple matter. Even when comparing elapsed execution time, there are many parameters which should be varied for a serious evaluation of algorithm speed : problem size, execution platform, number of processors (for parallel code), etc. To accommodate these variances in the algorithm execution, the analyzer uses the method of least squares approximation for a collection of observed data over a given variation of problem executions.

A profile is the set of all lines created by a least square approximation to the raw performance data for a given problem over all methods. The analyzer accesses the *profile* records named by the predicate to identify exactly which performance

```
select perfdata.nproc, ' ',
perfdata.time[1:perfdata.nproc][4:4][1:1] from perfdata, sequences
where
   perfdata.solverseq = sequences.name
   and composite_id = 'pde03'
   and rundata = 'IBM SP2'
   and perfdata.memoryvals[2] = '950x950'
   and sequences.names[6] = 'itpack-jacobi cg';
```

Fig. 12. Example analyzer query for retrieving performance data identified by a profile.

measures are to be used for a given problem. This record lists the choices for the x and y axis, and defines which invariants to use in the selection process. In addition, the record identifies where these values are stored in the performance records generated by the execution of the problem. This information produces an analyzer query such as the one in Figure 12 for problem `pde03` executed using algorithm `jacobi cg` on an IBM SP2 machine. The query retrieves observed data for '*time vs num processors*' where the grid size is held invariant.

The goal of the pattern-extraction module is to support the automatic knowledge acquisition process and to extract patterns/models from the data that will be used by a recommender system to provide advice to end users. 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 original PYTHIA methodology presented in [Weerawarana et al. 1997] used a feature vector of numerical features for each problem and some pre-defined classes of problems in order to find a "closest" problem in the knowledge base or the "closest" class of problems to an unseen problem. Having determined a ranking of solution methods for the matching problem or class of problems, the system induced the best method for the new problem. The main limitations of this methodology are that it is mostly a manual process and that it does not scale to larger sets of performance data because of its file-based approach and the low level representation of the induced knowledge.

The relational model of PYTHIA-II, on the other hand, automatically handles the book-keeping of the raw data and offers a unique opportunity for easily generating and storing any amount of raw performance data as well as manipulating them. In order for us to test various learning methodologies, we chose to support a specific format for the data used by the pattern extraction process, and then write filters that transform this format (on the fly) to the format required by the various data mining tools integrated into PYTHIA-II. Since the idea behind knowledge acquisition is to support an RS with as few changes to the automatically generated knowledge as possible, we have integrated mostly systems that generate comprehensible knowledge in the form of logic rules, if-then-else rules or decision trees.

The first learning system we integrated (we present some results using it later on), was GOLEM [Muggleton and Feng 1990], which is classified in [Dzeroski 1996] as an empirical single predicate Inductive Logic Programming (ILP) learning system. It is a batch non-interactive system with noise handling capabilities that implements the *relative least general generalization* principle that can be considered as careful generalization in the search space of possible concept descriptions.

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 by an expert system and constitute its rule base, as we will describe below. In addition to GOLEM, we also integrated the following systems: PROGOL [Muggleton 1995], CN2, PEBLS, OC1 (the latter three are available in the MLC++ library [Kohavi 1996]).

6.3 Inference Engine

The recommender is the end-user component of PYTHIA-II. It answers the user's domain specific questions using an inference engine and facts generated by the knowledge discovery process. The recommender is a form of a decision support system, and is the only component in PYTHIA-II that is both domain dependent and case study dependent. We describe how the recommender has been generated as an interface to the knowledge generated by GOLEM.

GOLEM is a relational learning system that uses positive examples for generalization and negative examples for specialization. Each logical rule generated by GOLEM is associated with an information compression factor f measuring the generalization accuracy of the rule. Its simple formula is $f = p - (c + n + h)$ where p and n are the number of positive and negative examples respectively covered by a specific rule, while c and h are information that is related to the form of the rule. The information compression factor is used for ordering the rules in the rule base in a decreasing order.

Each rule selected by GOLEM covers a number of positive and negative examples. The rules and the set of positive examples covered for each rule are passed to the recommender. The recommender then asks the user to specify the problem features. It uses the CLIPS inference engine to check the rule base for rules that match the specified features. Every rule that is found to match a problem features is selected and is placed into the *agenda*. Rules are sorted in decreasing order based on their generality (number of examples they cover), so the very first rule placed into the agenda will fire at the end of the inference process. This rule determines the best algorithm for the problem the user specifies. Since each rule provided by GOLEM to the recommender is associated with a set of positive examples that are covered by the rule, the recommender goes through the list of positive examples associated with the fired rule and retrieves the example that has the most common features with the user specified problem.

After this example/problem is selected, the fact base of the recommender is processed in order to provide the user with any required parameters for which the user needs advice. The fact base consists of all the raw performance data stored in the database. The recommender accesses this information by submitting queries generated on the fly, based on the user's objectives and selections. If the user objectives cannot be met, then the system decides what "best" answer to give, using weights specified by the user for each performance criterion. Valid performance criteria are, among others, the accuracy, total or communication time, efficiency and speedup. The sum of the weights applied to the criteria equals one. For the case study presented in the next section, the final step is the recommendation of

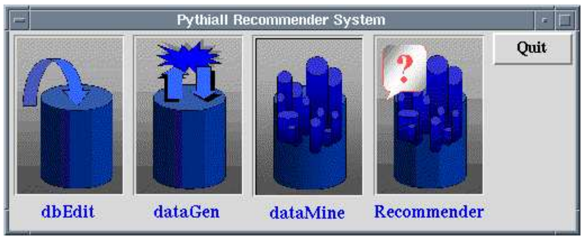

Fig. 13. PYTHIA-II's top level window.

the best numerical method to use, given the problem features specified by the user. It also identifies the grid parameter which satisfies objective imposed by the user: solution accuracy within the given time limitations.

6.4 User Interface

The modular implementation of PYTHIA-II makes it possible to accomplish much of the work involved in knowledge discovery without resorting to a graphical interface, and in some cases this is the preferred way of completing a given task. 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: the program generator is a separate process called from the problem execution environment which is specific to the scientific software. The process is invoked with an argument list describing the I/O for the program generation, and it may be called outside of PYTHIA-II.

(3) Executing programs: the execution process is controlled by scripts invoked by PYTHIA-II. These scripts can also be called outside of PYTHIA-II since they simply operate on the generated program files which reside in a particular directory.

(4) Collecting data: the data collector is called by PYTHIA-II as a separate process, and it is specific to the scientific software. As in (2) above, this process is invoked with an argument list describing its I/O.

With respect to the above items, the graphical interfaces that assist in those tasks are most useful for knowledge engineers who are unfamiliar either with the structure of PYTHIA-II or with the SQL language used by POSTGRES95. In this case, the interfaces provided by PYTHIA-II's *dbEdit* and *dataGEN* are invaluable. The top level window of the PYTHIA-II system is shown in Figure 13 and provides access to these interfaces, besides others.

The graphical interface to the POSTGRES95 database is dbEdit. Each PYTHIA-II record has a corresponding form which is presented to the user when records of that type are selected for editing. The fields are tagged for error checking, and every attempt is made to facilitate data specification. For example, many fields require references to primary records, such as the experiment record which requires the name of an equation, domain, boundary condition and associated parameter records. In dbEdit, the specification of these fields is handled by selection boxes whose contents are determined by *field typing*. If the field type is `equation`, a selection box displaying the current list of available equation records is presented, allowing the user to choose an equation by point and click. This method of editing ensures the correctness of the specification and eliminates costly errors during program generation.

Similarly, dataGEN facilitates the tasks involved in the data generation process, and frees the user from worrying about details such as: where are the generated programs stored, which scripts are available for the selected scientific software, where is the raw output data generated by program execution located, what input is required for invoking the data collection process, and so on. Users familiar with the implementation of the system may prefer to call these processes on their own, but when many users are involved in the (lengthy) data generation process, the graphical interface is most useful.

dataMINE encompasses the statistical analysis of data in selected performance records and the pattern matching process. Even for the most experienced users, it is not possible to attempt either of these tasks outside of PYTHIA-II. A template query is used to extract the performance data of interest in order to generate input for the statistical analyzer. This is accomplished within the graphical interface by choosing the predicate records, and allowing dataMINE to build the query, access perhaps hundreds of performance records to extract the identified fields, and then build the required input file. The input specification for pattern matching is equally difficult to build; it retrieves and matches scores of features across hundreds of performance records, and filters ranking data from the statistical analyzer output. In addition to carrying out essential data preparation tasks that cannot be handled outside of the GUI, dataMINE presents a simple menu system that walks the user through the process of selecting the predicate, calling the statistical analyzer, generating graphical profiles of the ranked methods, and calling the knowledge generator. As a bonus, dataMINE is integrated with DataSplash [Olston et al. 1998] an easy-to-use integrated environment for navigating, creating, and querying visual representations of data. DataSplash is a visualization system that has been built on top of POSTGRES95, therefore interaction with PYTHIA-II's DBMS was built into it.

7. DATA FLOW IN PYTHIA-II

The PYTHIA-II design supports two different user interfaces, one for the knowledge engineer and the other for end users who seek advice about the specific problems they want to solve. This section describes the data flow and I/O interfaces between the main components of the PYTHIA-II system from the perspective of these two interfaces.

7.1 Knowledge Engineer Perspective:

The data flow in PYTHIA-II is shown graphically in Figure 14, where the boxes represent stored entities, the edges represent operations related to the underlying database, and the self-edges represent operations related to various external programs such as statistical analysis, transformations and data filtering.

The knowledge engineer begins with populating the problem specific database

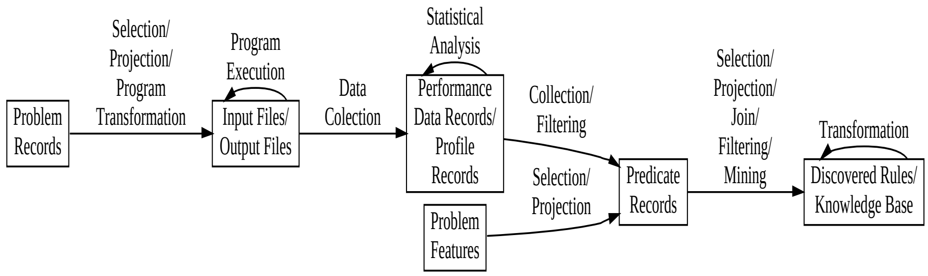

Fig. 14. Data flow and I/O for the knowledge engineer user interface.

tables. In PYTHIA-II, the underlying database schema is fixed, but extensible and dynamic. The knowledge engineer has to specify his understanding of the domain in terms of the relational data model to match PYTHIA-II's database schema. Supporting an extensible and dynamic schema is possible because of some unique features of the POSTGRES95 system, i.e., POSTGRES95 does not have the restriction imposed by the traditional relational model that the attributes of a relation be *atomic*[1], since attributes are allowed to contain sub-values that can be accessed from the query language. In particular, POSTGRES95 allows attributes of an instance to be defined as fixed-length or variable-length multi-dimensional arrays. The knowledge engineer has to specify his understanding of the domain in terms of the relational data model to match PYTHIA-II's database schema. The front-end interface for populating the database includes a full-fledged graphical environment with menus, editors and database specific forms for presentation purposes, very much like those supported by Oracle's SQL*Forms.

An *experiment* database record combines problem records into classes of problems, and a high level problem specification is generated by a program-based transformation of the experiment record into a complete and correct input file specification. These files are passed to the problem execution environment which invokes the appropriate scientific software for problem execution. Currently, PYTHIA-II's execution environment consists of the PELLPACK system which can solve a variety of PDE problems by applying multiple methods for discretization, indexing, domain partitioning and solution, and executing on various sequential and parallel machines. After executing each one of the input files, a corresponding number of output files is generated, each containing information related to the solution of the problem, such as error, memory utilization, execution time per processor (in case of a parallel execution), program traces, etc. Although the variability of the input specification is dealt with by the specific schema of the problem record, the variations in the output format for the files generated during execution are handled by specifying a system specific and user selected file template. The template lists, among other things, the full specification for the program to be called for the collection of the "important" data contained in the output files. This data is automatically collected by the program, and stored in the performance data records for further processing, while all the output files are deleted. These records keep logi-

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

cal references to the problem records in the form of foreign keys. In this manner, performance data can be matched with problem features by executing n-way joins, which is necessary for pattern extraction.

By combining data from a number of performance records, while maintaining all but one of the experimental variables constant (discretizer, indexer, partitioner, solver, problem size, machine size), we can generate a profile that characterizes the behavior of a certain parameter with respect to other parameters. The statistical analyzer uses the instructions for extracting performance data contained in a *profile* database table, which contains the number of experiments deemed necessary by the knowledge engineer for the analyzer to produce rankings of the solver profiles with the required statistical significance. The analyzer submits "canned" SQL queries (Figure 12) to retrieve the data to use for further processing.

After the performance data has been retrieved and combined, it is given to the statistical analyzer for ranking based on the parameter(s) selected by the user for evaluation. 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). This ranking can be used in a number of different ways to drive the pattern extraction process. Before the data is handed over to this process however, yet another abstraction level is used. A *predicate* record defines the collection of profile records to be used in pattern extraction. This means that the knowledge engineer can change the set of input profile records as easily as updating a database record. The predicate also contains all the required information used by the program that creates input for the algorithms used in pattern extraction.

A filter program is called for the selected predicate record to collect and transform the information to the input format required by the pattern extraction programs. For example, our system currently supports, among others, the input formats for GOLEM/PROGOL and the MLC++ (Machine Learning Library in C++) library. After the input data is prepared, the programs generate output in the form of "logic" rules, "if-then" rules or decision trees/graphs for categorization purposes. In this process there is open-ended extensibility reg. the integration of tools like neural networks, genetic algorithms, fuzzy logic tool-boxes, rough set systems, etc. It is only the support for the Recommender that restricts the automatic transformation of the knowledge structures provided by each one of these tools, since building a knowledge base for the Recommender requires that the knowledge induced by the mining process be comprehensible and structured.

7.2 End User Perspective:

The front-end for the Recommender must be configurable and adaptable for satisfying a variety of user needs. It is well understood that end 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. The PYTHIA-II Recommender allows users to specify the characteristics of the problems to solve, as well as the performance objectives or constraints. The system that supports this functionality is CLIPS, an expert system shell tool-box, which uses the induced knowledge, even background knowledge, and facts from the problem, feature, performance, profile and predicate tables to provide the user with the best inferred solution to the problem presented. It is also possible that the user's objective cannot be satisfied. In that case, the user can specify weights for the various objectives, and then the system tries to satisfy the objectives (e.g., accuracy first, then memory constraints) based on the ordering implied by the weights.

8. CASE STUDY 1: EFFECT OF SINGULARITES ON THE PERFORMANCE OF ELLIPTIC PDE SOLVERS

To validate the design and implementation of PYTHIA-II and the underlying KDD process, we consider a performance evaluation database of software modules (PDE solvers) from the PELLPACK [Houstis et al. 1998] library over a population of 2-dimensional, singular, steady state PDE problems defined in [Houstis and Rice 1982]. The algorithm selection problem for this domain can be formally stated as follows:

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

where L is a second order, linear elliptic operator, B is a differential operator involving up to first order partial derivatives of u, Ω is a bounded open region in 2–dimensional space, and θ, T are performance criteria constraints.

8.1 Performance Database Description

In this study, PYTHIA-II collects tables of execution times (in seconds) and errors for each of the selected solvers with respect to various sizes of the grid/mesh over the population of PDE problems. The error is measured as the maximum absolute error on the computational grid/mesh divided by the maximum absolute value of the PDE solution. The PDE software considered from PELLPACK library are abbreviated as follows:

—5PT = 5-point star,

—COLL = Hermite Cubic Collocation,

—DCG2 = Dyakanov CG,

—DCG4 = Dyakanov CG-4,

—FFT2 = FFT9 (order=2),

—FFT4 = FFT9 (order=4) and

—FFT6 = FFT9 (order =6).

In the case of 5PT and COLL, the linear systems are solved with the Band-GE direct solver from PELLPACK with natural ordering. The grids considered are 5x5, 9x9, 17x17, 33x33, and 65x65. More information about this experimental study can be found in [Houstis and Rice 1982].

Defining the PDE population and experiments required 21 equation records with up to 10 parameter sets each, 3 rectangle domain records of differing dimensions, 5 sets of boundary conditions records, 10 grid records defining uniform grids from

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 instance as applied to the PELLPACK singular PDE case study.

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.

coarse to fine, several discretizer, indexing, linear solver and triple records with corresponding parameters, and a set of 40 solver sequence records defining the solution schemes. 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. Figures 6, 7, and 9 depict the structure of these records. The 37 experiments were executed sequentially on a SPARCstation20 with 32MB memory running Solaris 2.5.1 from within PYTHIA-II's execution environment (see Table III. All 37 test cases executed successfully, resulting in the insertion of over 500 performance records into the database.

```
Field          | Value
name           | PELLPACK Solution Methods Study
reference      | pellpack
num_rankings   | 1
max_num_blocks| 37
prof_recs      |{{"pde3-1","pde3-2","pde7","pde8-1","pde8-2","pde8-4",
                  "pde9-1", "pde9-2", "pde9-3", "pde10-2", "pde10-3"}}
best           | method
nbest          | 7
bestlist       | {"fft 9pt order 2","fft 9pt order 4","fft 9pt order 6",
                  "5point star & bandge", "herm coll & bandge",
                  "dyakanov-cg", "dyakanov-cg 4"}
featurelist    | {"operator","right-hand-side","domain","bconds","matrix"}
possiblevalues| {{"opLaplace","opPoisson","opHelmholtz","opGeneral"},
                 {"rhsEntire","rhsConstCoeff", "rhsSingular", "rhsAnalytic"}}
recordlist     | {"equation","equation","domain","bcond","perfdata"}
indexlist      | {"featurevals[1]","featurevals[5]"}
```

Fig. 15. Partial listing of a predicate from the PDE benchmark.

8.2 Data Mining and Knowledge Discovery Process

After the experiment records were defined, dataGEN was used to retrieve them from the database and execute them. Each experiment represented up to 35 PDE programs. When program execution was complete, the raw performance output was located in a specified directory, and the data collection facility was invoked to extract data from the output and trace files and to insert them in the performance database. The dataMINE interface was used to access the performance data according to the specification of the predicate and profile records created for the case study. A portion of the predicate record used to generate profiles and rankings for the seven PELLPACK solvers is shown in Figure 15.

Figure 16 lists the ranking produced by the analyzer for PDE problem pde10-4. The rankings over all PDE problems and their associated features were then used by PROGOL to mine rules. The features considered in this case study are defined in IV, and examples of rules mined by this process are shown in Figure 18. The first rule, for instance, indicates that the method Dyakanov CG4 is best if the problem has a Laplace operator and the right-hand-side is not singular.

8.3 Knowledge Discovery Outcomes

The discovered rules 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 in [Houstis and Rice 1982]. This shows that these simple rules capture much of the complexity of algorithm selection in this domain. Table V summarizes these observations.

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].

```
The rank analysis produces the following comparison
  listed in order from 'best' to 'worst':

  The method ranks

  fft 9 point order 6             : 1
  fft 9 point order 4             : 2
  dyakanov-cg4                    : 3
  fft 9 point order 2             : 4
  hermite collocation and band ge : 5
  dyakanov-cg                     : 6
  5-point star and band ge        : 7
```

Fig. 16. Rankings of the PELLPACK solvers considered for problem pde10-4.

```
Probability for the label 'fft˙9˙point˙order˙4' : 27.03%
Probability for the label 'dyakanov-cg4' : 13.51%
Probability for the label 'fft˙9˙point˙order˙6' : 10.81%
Probability for the label 'hermite˙collocation˙and˙band˙ge' : 21.62%
Probability for the label '5-point˙star˙and˙band˙ge' : 18.92%
Probability for the label 'fft˙9˙point˙order˙2' : 2.70%
Probability for the label 'dyakanov-cg' : 5.41%
```

Fig. 17: Statistical data for the rankings of the seven PELLPACK solvers over the entire PDE population considered for this case study.

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

In this section, we apply PYTHIA-II to the performance database obtained by assigning different boundary condition types to a population of two-dimensional elliptic partial differential equation problems from the study of [Dyksen et al. 1988]. The objective of this performance evaluation can be stated as follows:

> Determine the effect of the presence of derivatives in the boundary conditions on the performance of numerical methods, where the PDE problem is given by
> $$Lu = au_{xx} + cu_{yy} + du_x + eu_y + fu = g \quad \text{on} \quad \Omega$$
> $$Bu = \alpha u + \beta s u_n = t \quad \text{on} \quad \partial\Omega$$
> and α,β control 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 2–dimensional domain with boundary $\partial\Omega$. The selected numerical methods are the following PELLPACK modules:

—5PT = 5-point star,

—COLL = Hermite Cubic Collocation,

—DCG2 = Dyakanov CG,

```
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) :- pdeSmoConst_yes(A), rhsSmoEntire_yes(A)
best_method(A,fft_9_point_order_4) :- solEntire_yes(A), solSmoBoundLayer_yes(A)
best_method(A,fft_9_point_order_4) :- solVarSmooth_yes(A), solSmoSingular_no(A)
best_method(A,fft_9_point_order_4) :- opLaplace_yes(A), rhsAnalytic_no(A),
                        rhsSingDeriv_no(A), rhsConstCoeff_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) :- rhsSmoOscillatory_yes(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)        :- opLaplace_yes(A), rhsEntire_no(A),
                        rhsSingular_no(A), rhsSingDeriv_no(A), rhsOscillatory_no(A)
best_method(A,dyakanov-cg4)        :- pdeJump_yes(A)
best_method(A,dyakanov-cg4)        :- opLaplace_yes(A), rhsAnalytic_no(A),
                        rhsSingDeriv_no(A), rhsPeaked_no(A)
best_method(A,dyakanov-cg4)        :- pdeSmoConst_yes(A), rhsSmoConst_yes(A)
best_method(A,dyakanov-cg4)        :- pdeSmoDiscDeriv_yes(A), rhsSmoConst_no(A)
best_method(A,dyakanov-cg)         :- opLaplace_yes(A), rhsSingDeriv_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) :- opLaplace_no(A), pdeConstCoeff_yes(A),
rhsEntire_no(A)
best_method(A,hermite_collocation) :- pdePeaked_yes(A)
best_method(A,hermite_collocation) :- pdeSmoConst_no(A), rhsSmoSingular_yes(A)
```

Fig. 18. Sample rules generated by PROGOL for the singular PDE study.

—DCG4 = Dyakanov CG-4, and

—MG-00 = Multigrid mg00.

In the case of 5PT and COLL, the linear systems are solved with the Band-GE direct solver from PELLPACK with natural ordering. The PDE problems are restricted to rectangular domains for this case study, and 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 five selected numerical methods using five uniform grids. There are 75 program executions for a given PDE. Data for elapsed solver time and various error measures at the grid points are collected for each problem execution.

9.1 Performance Data Generation, Collection and Analysis

The basic 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 using the basic components and the default PELLPACK program execution environment. All experiments were executed on a SPARCstation-20 SunOS 5.5.1 with 32 MB memory. The standard PELLPACK raw output data was generated and collected, and 600 records were successfully inserted into the performance database.

The statistical analysis and rules generation are handled by dataMINE, which requires as input the predicate and its corresponding profile records. The predicate and profile records identify all important controlling parameters for the tasks involved in data analysis and mining.

The predicate is the highest level controlling agent, and the end result in this case study is knowledge which answers the question stated at the beginning of this section. The predicate names a matrix of profile records that identify the number and type of analyzer invocations. Then it identifies the features of the basic components that are used. In this case, these are boundary_condition features. The analyzer rankings and the predicate feature specifications are handed over to the rules generation process. If the predicate is correctly constructed, the generated rules answer our questions about the effect of derivatives in the boundary conditions on solving PDEs when solved using the selected methods. Table VI lists, in part, the required predicate information.

Although the predicate controls the entire analysis and mining process, the details of the analysis 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 according to the extracted data. In this case, the objective is to study the relative changes in elapsed time as a function of derivative strength in the boundary conditions. Again, the query posed to the database by the profile extracts exactly the information needed by the analyzer to answer this question. Samples of the required retrieval information are listed in Table VI. 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. Other predicates/profiles were built to study relative 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 to generate the knowledge base.

9.2 Knowledge Discovery Outcomes

Here we summarize the results of PYTHIA-II analysis and the interpretation of the rules derived for Case Study 2. They 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.

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.

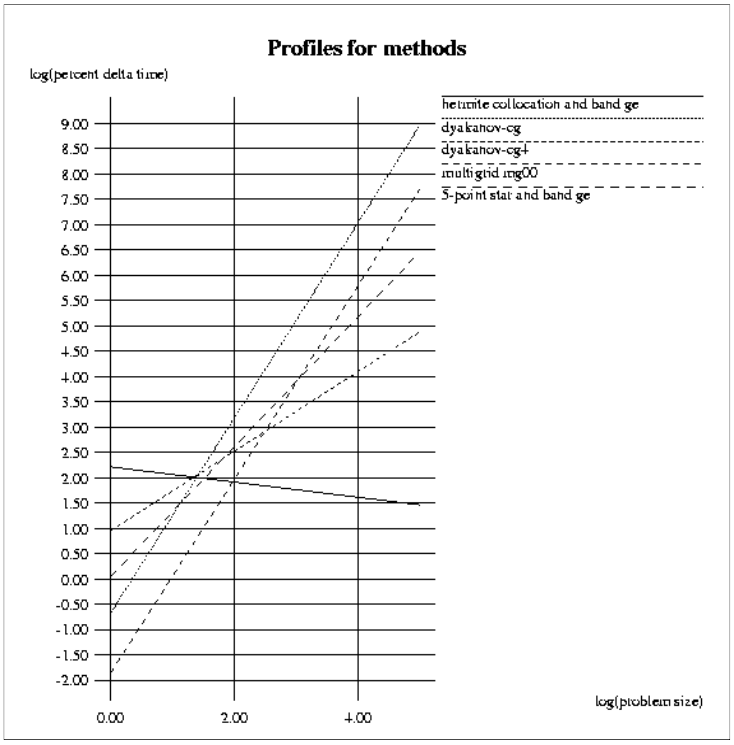

Fig. 19: Profile graph depicting the relative change of execution times between Dirichlet and Mixed problems as a function of the grid size for the five PELLPACK solvers considered.

(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 $\delta T_{dir} = (T_{neu} - T_{dir})/T_{dir}$ 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. One notable exception, however, was the COLL method, for which the derivative term did not introduce a significant increase in elapsed time, resulting in a δT that was very small and which, in some cases, decreased as the problem size increased, as shown in Figure 19.

(2) *The COLL module was least affected.* Specifically, the increase in elapsed time when the derivative term was added was least for COLL. Thus, it was most often ranked first by the analyzer using the relative time profiles. Note that even though the relative elapsed time was least for COLL, the total elapsed time was not. Summary statistics for two of the predicates are given below:

`Rankings for the dir2mix predicate based on relative time`:
Probability for the label 'hermite_collocation' : 57.14%
Probability for the label 'dyakanov-cg4' : 28.57%
Probability for the label 'dyakanov-cg' : 14.29%
Probability for the label '5-point_star' : 0.00%
Probability for the label 'multigrid_mg00' : 0.00%

`Rankings for the dir2neu predicates based on relative time`:
Probability for the label 'hermite_collocation' : 42.86%
Probability for the label 'dyakanov-cg4' : 21.43%
Probability for the label '5-point_star' : 14.29%
Probability for the label 'dyakanov-cg' : 14.29%
Probability for the label 'multigrid_mg00' : 7.14%

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 were chosen 85% and 64% of the time, respectively (see Figure 19 for the method ranking profile for pde04 generated by dir2mix predicate based on relative time). The rankings above also show that fourth order modules were less affected by *mixed conditions* than by *Neumann conditions*, and that MG-00 and 5PT methods performed worst with the addition of a derivative term in the boundary condition.

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 the selected methods differ significantly when ranking with respect to execution times across the collection of PDE problems. With a computed Q value of 25 (see Section 6.2), DCG and COLL show a rank difference of 57; MG-00 and COLL show a rank difference of 84. Methods DCG and 5PT did not behave in a significantly different way. Some analysis results are shown in Figure 20.

10. CONCLUSION

We have presented the architecture, implementation, and validation of the PYTHIA-II software system that facilitates a knowledge discovery in databases (KDD) process for selecting scientific software. It also recommends parameters for a targeted problem class assuming a priori defined "features" and computational objectives. Its architecture is open-ended (i.e., allowing its application to a variety of domain specific software and integration of alternative KDD phase implementations) and scalable (i.e., providing a variety of options to the knowledge engineer for mining data, while storage and retrieval issues are handled by an integrated database system). The modular approach used by PYTHIA-II maximizes the visualization of the entire KDD process, either in parts or as a whole. The high extensibility of the system is facilitated by the large number of alternative paths and tools available at every stage. The accuracy of the underlying KDD process has been demonstrated against two existing case studies. The PYTHIA-II generated data and rules lead to the same conclusions as the cited case studies.

REFERENCES

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.

Casaletto, J., Pickett, M., and Rice, J. 1969. A comparison of some numerical integration programs. *SIGNUM Newsletter 4,* 3.

Dodson, D., Miller, P., Nylin, W., and Rice, J. 1968. An evaluation of five polynomial zero finders. Technical Report CSD-TR-24, Dept. Comp. Sci., Purdue University.

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.

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.

JAMES, R. AND RICE, J. 1967. Experiments on matrix attributes and SOR success. Technical Report CSD-TR-9, Dept. Comp. Sci., Purdue University.

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.

KONIG, S. AND ULLRICH, C. 1990. An expert system for the economical application of self-validating methods for linear equations. In *Intelligent Mathematical Software Systems*, North-Holland, pp. 195–220.

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.

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, J. R. 1969. A set of 74 test functions for nonlinear equation solvers. Technical Report CSD-TR-34, 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.

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.

The rank analysis produces the following comparison
listed in order from 'best' to 'worst':

The Solver Ranks
(avg rank in parenthesis)

multigrid mg00	: 21 (1)
dyakanov-cg	: 48 (2.29)
dyakanov-cg4	: 68 (3.24)
5-point star	: 73 (3.48)
hermite collocation	: 105 (5)

======================

The rank differences and indicated significance
based on the q value,
where Ti-Tj represents the absolute difference
of the Solvers Ti and Tj.

T1: 5-point star T2: hermite collocation T3: dyakanov-cg T4:
dyakanov-cg4 T5: multigrid mg00

Solvers	Rank Diff	Significant?
T2 - T1	32	yes
T3 - T1	25	
T3 - T2	57	yes
T4 - T1	5	
T4 - T2	37	yes
T4 - T3	20	
T5 - T1	52	yes
T5 - T2	84	yes
T5 - T3	27	yes
T5 - T4	47	yes

Distribution of data for each Solver

Solver	Average	Minimum	1st Quart	Median	3rd Quart	Maximum
1	2.394	3.094	2.888	2.676	1.669	1.516
2	3.224	3.506	3.238	3.171	3.101	3.072
3	1.52	2.293	1.797	1.682	0.8285	0.4857
4	1.965	2.43	2.034	1.974	1.806	0.769
5	-0.3241	0.3039	-0.3208	-0.3241	-0.405	-1.164

Fig. 20: Ranking results for the comparison of numerical methods using *grid vs. total elapsed time* profiles.