PYTHIA II: A Knowledge/Data Base System for Recommending and Testing Scientific Software

Elias N. Houstis, Vassilis S. Verykios, Ann C. Catlin,
Naren Ramakrishnan and John R. Rice

Abstract

Very often scientists are faced with the task of locating appropriate solution software for their problems and then selecting from among many alternatives. Issues related to how someone specifies problems, extracts content information, builds knowledge bases, infers answers, and identifies software resources are crucial to any scientific computing development today. In [Houstis et al. 1991] we had proposed an approach for dealing with these issues by "processing" performance data obtained from "testing" software. Reliable testing requires identification of "dense" benchmarks that cover many of the application domain "features", systematic testing procedures and automatic ways to collect and analyze the results of this process. Testing constitutes a significant investment of effort and expertise that cannot be duplicated easily by an average scientific or engineering group. In this paper, we present the architecture and implementation of a knowledge/data base system that makes software recommendations based on problem specifications and computational objectives such as accuracy, cost or time, and memory requirements. The system is referred to as PYTHIA II, and is designed to (i) identify and select the software/hardware resources available for a user's problem, (ii) locate these resources and provide information about their usage, availability, cost and related information, (iii) suggest parameter values, and (iv) provide an assessment of the recommendation. In addition, PYTHIA II can be used to generate "testing" software repositories, since it provides all the necessary facilities to set up database schemas for testing benchmarks and associated performance data, with a number of tools for visualization, statistical ranking, data mining, knowledge representation, and recommendation generation.

Contents

1. INTRODUCTION

Complex problems, whether scientific or engineering, are most often solved today by utilizing public domain or commercial libraries or some form of problem solving environments (PSEs) [Gallopoulos et al. 1994]. Most extant software systems are characterized by a significant number of parameters affecting efficiency and applicability which 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 implements the various phases of the computation. Thus, the task of selecting the best software for a particular problem or computation is often difficult and sometimes even intractable. In [Houstis et al. 1991] we had proposed an approach for dealing with these issues by "processing" performance data obtained from "testing" software. Reliable testing requires systematic testing procedures and automatic ways to collect and analyze the results of this process. Testing constitutes a significant investment of effort and expertise that cannot be duplicated easily by an average scientific or engineering group.

In this paper, we present the architecture and implementation of a knowledge/data base (K/DB) system, referred to throughout as PYTHIA II[1], whose design objectives attempt to address most of the above issues. Specifically, from the end-user perspective, PYTHIA II will allow users to specify the problem to be solved and their computational objectives such as accuracy, cost or time, memory requirements. The system will (i) identify and select the software/hardware resources available for the user's problem, (ii) locate these resources and provide information about their usage, availability, cost and related information, (iii) suggest parameter values, and (iv) provide an assessment of the recommendation. To support the development of the "testing" software repositories, PYTHIA II provides a highly extensible database schema for testing suites and associated performance data, with a number of tools for visualization, statistical ranking, data mining, knowledge representation, and recommendation generation.

The realization of PYTHIA II requires us to

(1) develop and analyze methodologies and tools for generating knowledge of specific domains (e.g. linear solvers, linear elliptic PDEs, mesh decomposition) of scientific software (algorithms),
(2) address the issue of intelligent integration and presentation of information,
(3) devise a software architecture for PYTHIA II, and
(4) integrate methodologies to provide advice for solving classes of scientific problems and indicate the available software/hardware resources, including an estimation of the parameters involved.

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 estimation of software/hardware parameters involved. Due to its ability to make recommendations by combining attribute-based elicitation of a specified problem features and matching them

[1]PYTHIA II is a successor to the PYTHIA system [Weerawarana et al. 1997] for selecting scientific algorithms using exemplar based reasoning.

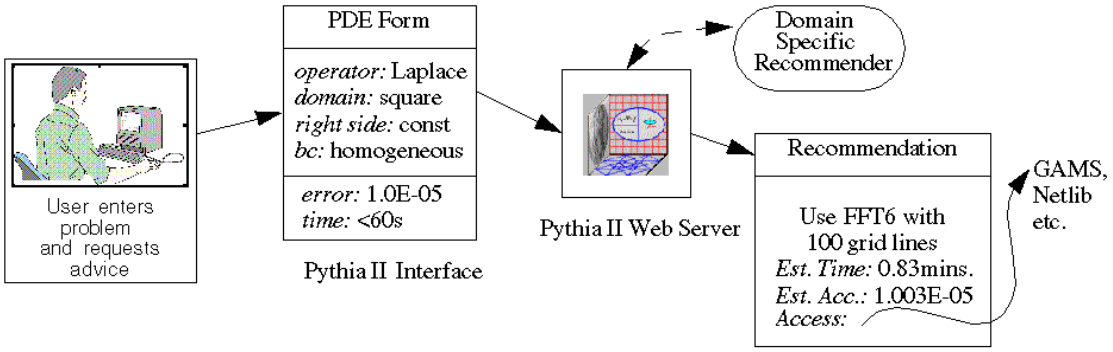

Fig. 1. Schematic of a sample interaction with the PYTHIA II web server.

against those of a predefined "dense" population of similar type of problems, we classify PYTHIA II as a recommender system [Ramakrishnan et al. 1998]. We describe an operational recommender system with a case study that covers software for elliptic partial differential equations found in the problem solving environment PELLPACK [Houstis et al. 1998]. The initial version of PYTHIA II is built as a foundational system that will be enlarged into a national software recommender service for the entire scientific community by making it available as a network server.

One of the core research issues in building PYTHIA II is understanding the fundamental processes by which knowledge about scientific problems and their solutions is created, validated, accumulated, and communicated. Some of this knowledge will come directly from experts-scientists and engineers-in-the-field. Other knowledge will be mined from experimental data. Yet further knowledge will be learned from the experience gained by the system itself as it extracts performance knowledge about software components running on various platform and applied to various problems. The methodology employed for extracting knowledge from performance data is implemented as a *knowledge/database (K/DB) process* which utilizes database, statistical, data mining, and rule generation technologies.

We now describe a sample PYTHIA II session. 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 constrains. 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 interface contacts the PYTHIA II (web) server on the user's behalf which, in turn, interfaces with a domain specific recommender. The recommender uses the knowledge acquired by the learning methodology presented in [Houstis et al. 1991; Ramakrishnan 1997; Ramakrishnan et al. 1998] to perform the software selection. Having determined a good algorithm, the recommender 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.

The rest of the paper is organized as follows: The motivation for and a general methodology of building recommender systems is introduced in Section 2. The system architecture of a recommender system is also presented in this section. Section 3 concretizes these ideas further by addressing these issues with specific reference to the PYTHIA II system. A case study with the prototype system for a benchmark suite of test problems and algorithms is outlined in Section 4. Section 5 concludes by providing pointers for future research and development.

2. RECOMMENDER SYSTEMS FOR SCIENTIFIC SOFTWARE: METHODOLOGY

In the context of human artifacts, a recommender system (RS) can be viewed as an intelligent system that uses stored user preferences for a given class of artifacts to locate and suggest artifacts that will be of interest to associated users. Throughout this paper we define a recommender system 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 that will be compatible with the solution of "similar" problems. Recommendation becomes necessary when user's requests or objectives cannot be properly represented as database queries. In this paper we attempt to design and implement a RS that assists scientists in selecting suitable software for the problem at hand, in the presence of practical constraints on accuracy, time and cost. In other words, it is necessary to adaptively select, recommend and locate software to conform to the performance requirements set by the user [Rice 1969]. We refer to this as the algorithm/software recommendation problem. 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]. Given a task in scientific computation, with performance criteria constraints on its solution (such as accuracy, time, cost, etc.), it is necessary to decide on an algorithm to achieve the desired objectives. 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 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 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

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 Buidling Recommender Systems. This layered methodology is very similar to procedures adopted in the performance evaluation of scientific software.

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. Different implementations of an algorithm produce substantially large variations 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 and network relevant information; techniques are required that will allow distributed recommender systems to coexist and cooperate together.

A methodology for generating an RS for scientific artifacts is defined in Table I. The layered approach suggested by this methodology is akin to similar strategies put forth for the performance evaluation of scientific software. Its implementation, illustrated by PYTHIA II, is discussed in Section 3. Assuming a 'dense' benchmark of problems from the targeted application domain, this methodology is based on a three-pronged strategy: the feature determination of problem domain, performance evaluation of scientific software, and the automatic generation of recommender systems from such data. Following, we described each of these in more detail.

2.1 Problem Features

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

```
-- 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

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 f_1 and f_2, the use of features such as $f_1 * cos(f_2)$ and $f_2 * cos(f_1)$ might not reflect the underlying mapping succinctly. 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 base comparisons on features values (such as 1 and 5), erroneously concluding that the magnitude of the latter is five times that of the former. Comparing features, on the other hand, might not be possible, or it may be that their 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 data base schema for a feature and a feature relation, respectively. The relation record shows how PYTHIA II represents the correspondence 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,

```
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*.

portability, documentation, etc., are ignored in this discussion, our methodology represents these aspects as well. Other claseses of performance objectives for software are handled more simply, e.g., code language, public or proprietary, licensing availability, or member of library X.

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 a particular 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. Diagnostic information like error reports, fail codes, etc., is also provided in the form of logic formulas so that they may influence the algorithm selection methodology. Some of the most important performance measures appear to be - and are - quite hardware and systems infrastructure dependent. Our philosophy is that a recommendation should be made that is close to best. If one wants to be sure about the best, one has to generate data for the particular computing environment to be used, and this almost always involves more computation than using a close to best algorithm.

How can performance data from many different machines be used to make a recommendation for a new, unknown, machine? We use machine specific performance factors and feature matching to compare execution times on different machines [Houstis and Rice 1980]. Although these are approximate, we believe our comparison mechanism is valid.

2.3 Reasoning and Learning Techniques in PYTHIA II

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. Specifically, it provides the following three broad learning strategies:

—*Case Based Reasoning (CBR):* A case based reasoning system [Kolodner 1993; Riesbeck 1996; Riesbeck and Schank 1989; Watson 1977] records 'cases' of past experience and uses them to guide problem solving in future analogous situations.

These cases might reflect a useful solution approach, a bad strategy or estimations of the likely outcomes in a state-based environment. The original PYTHIA system [Weerawarana et al. 1997] utilized a rudimentary form of case based reasoning where the cases correspond to characteristic-vector descriptions of PDE problems and algorithms. Such systems are advantageous for their 'stored library' paradigm, where it is assumed that a case library can be constructed that covers the actual problems and situations encountered. In addition, case based reasoning can be used to 'evolve' new cases (in environments where data is scarce), suggest directions for continued exploration (in an unknown and large environment) and form the basis for recommender systems via the case bank. CBR has been successfully applied in previous advisory systems such as the SQUAD system at NEC, a system using approximately 30,000 cases to provide advice to software quality control engineers [Kitano and Shimazu 1996].

—*Inductive Logic Programming (ILP):* ILP systems [Bratko and Muggleton 1995; Dzeroski 1996; Muggleton and Raedt 1994], on the other hand, 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 reproduced. 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 recommender procedures using banks of selection rules. This methodology has been adopted in [Ramakrishnan 1997].

—*Decision-Tree Induction:* Decision trees are a precursor to ILP systems and while limited in their representation capabilities, are advantegous for their ability to handle noise, outliers and use attribute-value based comparisons to influence decision making. The ID3 [Quinlan 1986] is one such system that we have investigated for inclusion in the PYTHIA II system. ID3 is a supervised learning system for top-down induction of decision trees using a greedy algorithm. This algorithm is based on a simple information-theoretic consideration of the classifiability of a given training set with respect to several of its attributes. The result of this process is a tree-like knowledge representation structure where: (a) every internal node (including the root) bases its decision on the value of some attribute; (b) every leaf node identifies a specific class. It is very advantegeous in domains where attributes have a mixed symbolic-numeric flavor and the underlying structure is simple enough to be accomodated in a tree-based representation.

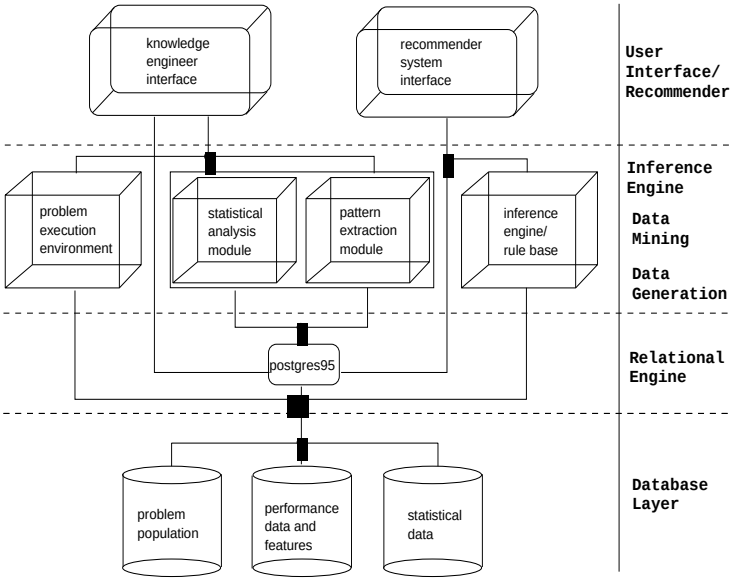

Fig. 5. System architecture of PYTHIA II.

2.4 Domain Specific Recommender Systems: Architecture

In this section we detail the software architecture of a domain specific recommender system based on the recommendation methodology and its components discussed above. The design objectives of a RS for scientific software includes (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 that generates the performance data by expressing them in a high level logic based representation language, and finally (v) incorporating them into the intensional/deductive part of the underlying relational DBMS in the form of relation views. An instance of these operational components of the PYTHIA II system is depicted in Figure 5.

Two of the basic components of a recommender system are the stored rule base and an inference engine to support its deduction capabilities. The rule base contains rules generated using one of automated learning process described above. In the recommender system production framework we envision having a highly integrated software system for knowledge acquisition and maintenance that spans the domains of databases, statistical analysis, inductive learning and a deductive-like approach, coupled with a high level user interface that facilitates easy access and reasonable learning curves for the knowledge engineer that plans to update and maintain a domain specific recommender system. We propose a fully automated system for generation and maintenance of domain specific recommender systems, but do not neglect human intervention throughout the process, especially when the generalization accuracy attained by the machine learning system is of low quality.

Domain experts apply their feedback/evaluation (i.e., sanity check) to the induced rules, since it is easy for them to judge the general applicability and reasonableness of rules, even as it is beyond human capability to identify rules by searching through very large databases. We propose a modularized approach for building recommender system cores (e.g., Figure 5) with the interfaces between the various modules as points where human support or iteration can easily take place.

By modeling and collecting all the information related to a specific scientific computing domain in a database system, our integrated approach easily synthesizes input programs on demand. All the required information exists in a structured way in the database which transforms the programs to the input format required by the execution environment. The database system then executes them in an automatic, possibly batch manner. Simulation techniques applied to the appropriately transformed programs will generate performance data to be collected, cleaned and converted to a format suitable to the performance schema defined beforehand for storage in the database. A statistical analysis phase can be applied to some suites of performance data to summarize the data and to extract information about the various trends or patterns that are known to exist. The objective of such a statistical analysis might be some ranking, or a discretization of continuous variables (since we know where percentiles are located we can reasonably split a continuous variable if required by the system using the data) and so on. At this point, the core of the inductive rule generation and case based reasoning processes begins. Appropriately selected data are retrieved from the database and are fed into the knowledge discovery system that attempts to mine novel patterns hidden in the data, expressing the results in a high level representation language. We expect that different methods will be applicable to different problem domains. At the termination of the rule generation process, the domain expert decides whether the knowledge generated in the form of rules is satisfactory (the sanity check), or else the process is repeated.

The intensional part of the underlying DBMS includes capabilities to define rules (those automatically generated by the learning process), which can deduce or infer additional information from the facts that are stored in the database. Rules in our case are relational views. They specify virtual relations that are not actually stored but can be formed from the facts by applying inference mechanisms based on the rule specification. An SQL (the standard for database query and modification) interface at this stage is enough to provide the user with domain specific recommendations. A user of the recommender system can use the SQL engine of the DBMS to retrieve data and recommendations either from facts stored as simple relations or from the relational views that consist of a simple encoding of the discovered knowledge. A simple text based or graphical SQL based form interface can be used by an end user to access the services of a recommender system.

Our recommender system requires the support of an object-oriented, relational database to provide storage, retrieval and processing for atomic entities, experiments, performance data, knowledge-related data and derived data. Atomic entities are domain specific since they represent the problem definition objects of a targeted domain, but the performance and knowledge-related data schema extend easily to other problem domains. Following we describe the data base schema specification used for producing a recommender system for elliptic PDE software. Their

```
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.

```
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.

modification for other domains of scientific software can be easily derived. They are presented in itemized form.

—*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, a sequences entity was defined to contain an ordered listing of all modules 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 [Houstis et al. 1998]. The &b in the specification allows for parameter replacement and the forfile attribute allows 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 was given in section 2.1.

—*Experiments.* The experiment is a derived entity which identifies a specific PDE

```
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.

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. 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.

3. PYTHIA II: A REALIZATION OF THE RECOMMENDER METHODOLOGY

3.1 System Design

Specifically, we describe the realization of the proposed architecture of a kernel RS system for scientific software in terms of the database and programming infrastructure used to implement it.

3.1.1 *Architecture.* The modular design of PYTHIA II is shown in Figure 5. The hierarchical architecture of the system consists of four layers:

```
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.

—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 occupied by 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. 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.

```
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

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 also resides in the top layer. It 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

```
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. Partial listing of a predicate from the PDE benchmark.

for answering 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. The interfaces of these components are discussed in Section 3.1.2, and their functionality and implementation are described in Section 3.2.

For storage and database management, we selected the POSTGRES95 relational database and used *PgTcl* as the front-end interface between PYTHIA II and the POSTGRES95 back-end. Using Tcl/Tk as the basic programming environment for the implementation of PYTHIA II allows the database to be accessed in a transparent and intuitive way. PgTcl is extremely efficient for database access, since it communicates with the back-end directly via the front-end-back-end protocol, without the need for intermediate C libraries (similar to Oracle Pro*C). It also handles multiple back-end connections from a single front-end application. The implementation code can either use library calls for connecting/selecting/reading from the database, or can execute embedded SQL statements, making the data access simple and flexible.

3.1.2 *Data Flow.* The PYTHIA II design presented in Section 3.1.1 supports two different user interfaces, one for the knowledge engineer and the other for end users who query the recommender for domain specific advice about the 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.

Knowledge engineer perspective: The data flow is depicted graphically in Figure 12, 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 automated knowledge discovery process begins with populating the problem

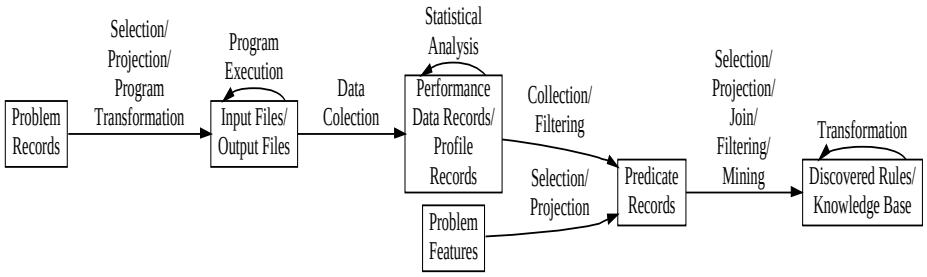

Fig. 12. Data flow and I/O Interfaces.

specific database tables. In PYTHIA II, the underlying database schema is fixed, but extensible and dynamic. Supporting an extensible and dynamic schema is possible based on some unique features of the POSTGRES95 system, i.e., POSTGRES95 does not have the restriction imposed by the relational model that the attributes of a relation are *atomic*, 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 simulation PDE problems, applying multiple methods for discretization, indexing, domain partitioning and solution, in 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 logical 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

```
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. 13. Example analyzer query for retrieving performance data identified by a profile.

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 13) to retrieve the data to use for further processing.

After the performance data has been retrieved and combined, it is provided to the statistical analyzer for ranking based on the domain parameter 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 then they are shown as tied in rank). The 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, 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 regarding 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 system 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.

End user perspective: The front-end for a recommender must be configurable and adaptable for satisfying a variety of user needs. It is well understood that end users of a recommender 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 user specified limit, etc. The PYTHIA II recommender interface 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 that was 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 will try to satisfy the objectives (e.g., accuracy first, then memory constraints) based on the ordering implied by the weights.

3.1.3 *Data Modeling and Management.* The quantity of information generated and manipulated by PYTHIA II calls for a powerful and adaptable database and database management system (DBMS) with an open 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 functionality to be provided by the two lower level layers of the system's architecture is summarized as follows:

—to provide storage for the problem population input data to the execution environment in a structured way, and to keep track of constraints implied by the specification language of the execution environment, and even the physical characteristics of the application,

—to support seamless data access by the user through a graphical interface or by a programming system like 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 and Relational DBMS (ORDBMS) which supports complex objects and is easily extensible by providing new data types, new operators, and new access methods to the user so that it can be used in new application domains. It also provides facilities for active databases (i.e., alerters can send a message to a user calling for his 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 will be 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 mod-

ules, as well as parameter sets for any of the problem components. Each of the PDE problem specification components constitutes a separate *entity*. In the relational model, each entity 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 introduce 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.

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 were used to configure the database for PYTHIA II. Section 4 gives some examples of these tables definitions within the context of the case study.

3.2 System Components

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

3.2.1 *Data Generation.* Information in the performance database drives PYTHIA II' s data analysis and rule generation. The performance database may be a preexisting store of performance measures or the data may be produced by executing scientific software within the problem execution environment. PYTHIA II does not need to understand 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 any software that is a candidate for integration 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 was translated to a *.e file*, which is the PELLPACK language definition of the PDE problem, the solution scheme, and the output requirements. The converter was written in Tcl and consists of about 250 lines of code. After the .e file was generated,

the standard PELLPACK *preprocessing* programs took over, converting the .e file to a FORTRAN driver and linking the appropriate libraries to produce an executable program.

The second requirement is that the scientific software should be able to operate in a "batch" mode when executing PDE programs. In the PELLPACK case, Perl scripts were 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 written specifically for the software and must 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 was written in Tcl (350 lines of code) and Perl (300 lines of code), and was responsible for creating performance records that represented the data produced by PELLPACK program executions.

The execution environment has been 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 processesing domain specific output files.

3.2.2 *Data Mining.* Data mining 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 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. The analyzer assumes that the algorithms have been executed on the selected problems, and that the resulting performance measures for each execution have been 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

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

Table II. Algorithm ranking table based on Friedman Rank Sums using the two-way layout.

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 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 values computed by the analyzer based on the profile record specification for Problem i. The computation of the X_{ij} will be covered later in this section, but for the sake of the following discussion, assume these values exist.

The process for ranking the algorithms was developed from an analysis for multiple comparisons and contrast estimators using procedures 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. 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 $R_{\bullet j}$ is the average rank for algorithms j. *Use $R_{\bullet j}$ 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 threshold α.

The $R_{\bullet j}$'s 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 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 13 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 for efficient use of the scientific software. This process is independent of the problem domain. PYTHIA II is an extension of the PYTHIA methodology used to address the algorithm selection problem by applying various neuro-fuzzy, semantic networks, instance-based learning and clustering techniques. The idea of this methodology is to use a feature vector of numerical features for each problem and some pre-defined classes of problems in order to find a "closest" problem 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 could induce the best method for the unseen problem. The main limitations of this methodology was that it was mostly a manual process and it could 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 we use completely solves 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 have decided to support a specific format for the data that will be used by the pattern extraction process, and write filters that transform this format (on the fly) to the format required by the various integrated data mining tools. Since the idea behind knowledge acquisition was to support a recommender system 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 even decision trees.

The first system we integrated and for which we will present some results later on, was GOLEM [Muggleton and Feng 1990], which has been 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. The task of empirical single predicate learning in ILP can be formulated as follows:

Given a set of training examples $\mathcal{E}$, consisting of true $\mathcal{E}^+$ ground facts (examples of correct recommendations), false $\mathcal{E}^-$ ground facts (corresponding to 'bad' recommendations), a concept description language $\mathcal{L}$, specifying syntactic restrictions on the definitions of the predicate (say p) to be mined (in this case, the predicate that relates problem features to the best method recommendation), background knowledge $\mathcal{B}$ (identifying problem features and performance criteria), the task for

ILP is to find a definition of p that is both *complete* and *consistent*. Completeness refers to the property of the mined rule to conform to the positive examples and consistency refers to its ability to exclude negative examples from its cover.

Such rules generated by GOLEM can be processed in a language like first order predicate logic. These rules can be easily utilized by an expert system and constitute its rule base, as we will describe below. In addition to GOLEM, we have already written filter programs to integrate the following systems: PROGOL, MLC++ library, CN2, PEBLS, OC1.

3.2.3 *Inference Engine.* The recommender is a form of a decision support system, and is the only module in PYTHIA II that is case study dependent as well as domain dependent. We will describe how a recommender system has been generated as an interface for the knowledge generated by GOLEM.

GOLEM, as described in the previous section, 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 which is related to the generalization accuracy of the rule. The formula for this metric is simple: $f = p - (c + n + h)$ 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 will be 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 set of positive examples covered for each rule along with the rules, is one part of the input given to the recommender. The recommender asks the user to specify the characteristics of the problem he wants to solve in the form of problem features. The recommender, using the CLIPS inference engine, checks its rule base to find a rule that matches its left-hand side which specifies the problem features. Every rule that is found to match the problem features specified by the user is selected and is placed into the *agenda*. Because the rules have been sorted in decreasing order based on their significance (number of examples they cover), it is only the very first rule placed into the agenda that will be fired to determine 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. This step aids in subsequent parameter estimation.

After the example/problem has been selected, the fact base of the recommender is processed in order to provide the user with any required set of parameters for which the user asks advice. The fact base consists of all the raw performance data that are 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 specified objectives cannot be met, then the system has to decide what "best" answer to give the user. In order for the recommender to be able to decide upon this issue, the user has to specify the weight to be placed on each performance criteria when selecting the best method. Valid performance criteria for the recommender are, among others, the accuracy, total or communication time, efficiency and speedup. The user can a specify a set of weights for each of the above

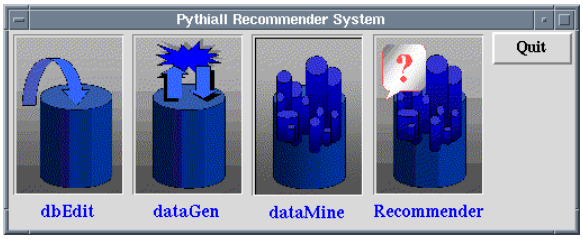

Fig. 14. PYTHIA II's top level window.

parameters in such a way that their sum equals one. For the recommender used in the case study presented in the next section, the final step is the recommendation of a certain method, machine, or number of processors, as the best method to use to satisfy the given conditions. It also indicates what problem size should be used to achieve the specified the accuracy within the time limitations imposed by the user.

3.2.4 *User Interface.* The modular implementation of PYTHIA II has made it possible to accomplish much of the work involved in knowledge discovery without resorting to the 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 used to solve the problems. 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 scientic 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 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. The top level window of the PYTHIA II system is shown in Figure 14.

4. CASE STUDY : A RECOMMENDER FOR ELLIPTIC PDE SOFTWARE

To validate the design and implementation of PYTHIA II, a knowledge base was generated for evaluating PELLPACK [Houstis et al. 1998] solvers based on performance data produced by a population of 2-dimensional, singular, steady state PDE problems. The algorithm recommendation problem for this domain can be formally stated as follows:

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

where L is a second order, linear elliptic operator, B is a differential operator 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. In this study, we

Problem Component	Generalized Forms	Parameterization
Equation	coef1$(x,y) * U_{xx}$ + coef2$(x,y) * U_{yy}$ coef3$(x,y) * U_x$ + coef4$(x,y) * U_y$ coef5$(x,y) * U$ = f(x,y)	operator coefficients are specified in the database as parameter records and right-hand-sides are specified as Fortran routines in data files referenced by the database equation records.
Domain	unit square, square $[-1,1] \times [-1,1]$ rectangle $[0,.5] \times [0,.75]$ rectangle $[a,b] \times [c,d]$ rectangle $[a,b] \times [a+c, b+c]$	endpoints are specified in the database as parameter records
Boundary Conditions	$u = 0$ on outer boundary u = true(x,y) on outer boundary	true(x,y) is specified as Fortran routines in data files referenced by database equation records

Table III. Required problem population for the case study.

Module Type	Module Names	Performance Criteria
Grid	5 x 5, 9 x 9, 17 x 17, 33 x 33, 65 x 65	
Discretizer	5-point star, hermite collocation	
Indexer	as is, red-black	
Linear System Solver	band ge, itpack-jacobi cg	
Triple	fft 9 point, dyakanov-cg, dyakanov-cg4	
Solver sequence	grid, 5-point star, as is, band ge grid, fft 9 point (orders 2,4,6) grid, hermite collocation, as is, band ge grid, dyakanov-cg grid, dyakanov-cg 4	error, elapsed time

Table IV. Available methods and solver sequences for the case study.

restrict ourselves to rectangular domains. Accuracy is measured as the maximum absolute error on the rectangular mesh divided by the maximum absolute value of the PDE solution. Performance studies are conducted and the amount of time required to obtain three levels of accuracy — 10^{-3}, 10^{-4} and 10^{-5} — are collected by the PYTHIA II system.

Table III shows the general form of the problems which were included in the study. In Table IV, the solver modules and solver sequences which were applied to the problems are listed. Table V identifies the features of the problem components used to drive the rules generation and form the basis for user inquiries to the recommender. Table VI uses the "raw data" descriptions in Tables III and IV to demonstrate how the recommender methodology was applied to the PELLPACK case study.

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 coarse to fine, several discretizer, indexing, linear solver and triple records with corresponding parameters, and a set of 40 solver sequence records defining the

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 V. Features for the problem population of the case study.

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: Table III (2) measures: elapsed solver time, discretization error. (3) methods: Table IV (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 GOLEM
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 VI. Building a Recommender for the PELLPACK case study.

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.

The 37 experiments were executed sequentially on a SPARCstation5 with 32MB memory running Solaris 2.5.1 from within PYTHIA II' s execution environment. All 37 test cases executed successfully, resulting in the insertion of over 500 performance records into the database. The analyzer evaluated the solver performance based on generated measures for *time vs problem size* and *time vs error*. The analyzer rankings and problem features were passed to the rules generator which produced logic-based rules governing method selection for PELLPACK solvers. The recom-

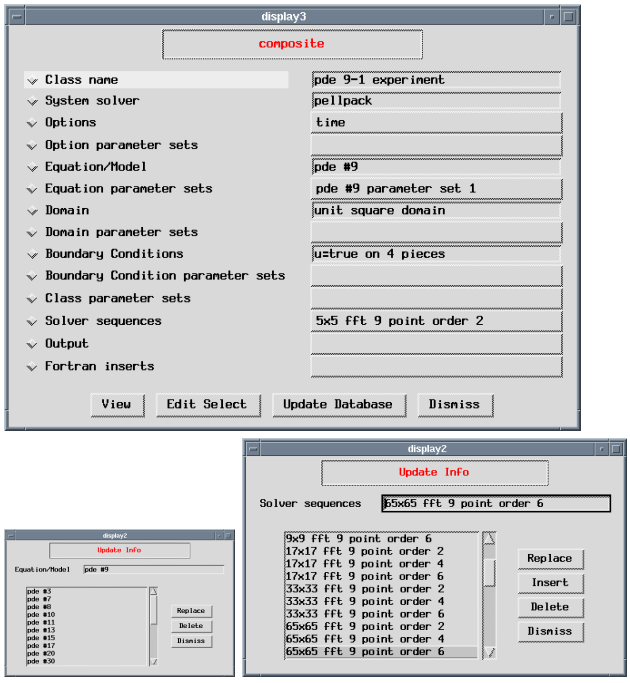

Fig. 15: The pde 9-1 experiment record defines a class of 35 PDE programs. The selection mechanism for experiment definition fields is shown for the equation specification.

mender was then used to predict the best method and estimate the corresponding parameters for user specified features and performance criteria. Specifically, if an end-user identified a problem with features such as "Poisson equation" with "computationally complex" right-hand-side on a unit square having "mixed boundary conditions", and specified that the error should not exceed "10^{-4}" with execution time less than .5 CPU seconds, the recommender predicted the best grid size and solver which satisfied the performance criteria for a problem with those features. It also listed the expected error and execution time, and identified the "closest" matching problem from the rules base.

The advantage of this demonstrator was that it corresponded to existing studies [Rice et al. 1981; Weerawarana et al. 1997; Houstis and Rice 1982], allowing validation of the rules and predictions.

The POSTGRES95 database was populated with 44 records defining problems, features, methods, and experiments. Each record had a corresponding form in the PYTHIA II graphical interface which was used to create and edit the records. Three record definitions are shown in Figures 6, 7, and 9. The dbEdit gui is the interface to the database for editing problem, method and experiment records. A dbEdit form corresponding to the experiment record in Figure 9 is shown in Figure 15.

The experiment record in Figure 15 defines 35 solver sequences for `pde09` with parameter set 1 on the unit square. The sequences cover five different grid sizes for each of three triples (`fft, dyakanov-cg` and `dyakanov-cg 4`) and two discretizer-solvers (`5 point star` and `hermite collocation`, paired with the `direct band-ge`

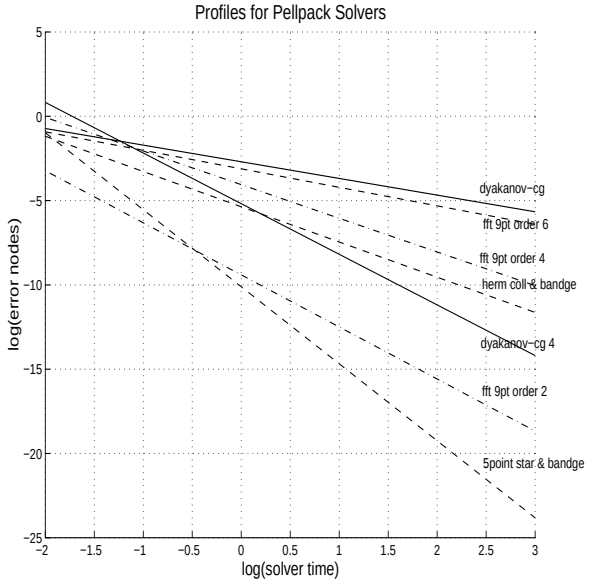

Fig. 16. Solver profile graph for experiment pde10-3 from the case study.

solver). Specification of equations, domains, parameters, solver sequences, etc for experiment record editing is facilitated by the field typing mechanism described earlier. Figure 15 demonstrates this with the selection of the equation and solver sequences from the selection boxes as they appeared during the definition of experiment 9-1.

After the experiment records were defined, dataGEN was used to select 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 target directory, and the data collection facility was invoked to extract data from the output and trace files and insert it in the performance database. The dataMINE gui accessed the performance data according to the specification of the predicate and profile records created for the case study. A portion of the predicate record is shown in Figure 11. The predicate specified *all* problems and methods so that the data available to the recommender for making inferences based on user inquiries was as broad as possible. The analyzer used this predicate to generate profiles and rankings for the seven PELLPACK solvers. Figure 16 shows a profile graph of the seven solvers for `pde10-7`, and Figure 17 lists the ranking produced by the analyzer for all solvers over all methods. The rankings and features were used by GOLEM to define rules.

Example of rules mined by this process include:

```
R1:     best(A,FFT6) :- dom_us(A), op_laplace(A).
R2:     best(A,P3C1C) :- rs_s(A), op_general(A).
R3:     best(A,PS5) :- rs_s(A), smo_cc(A).
...
```

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

  The Linear Solver Ranks
  (avg rank in parenthesis)

  5pt star & bdge   :  60 (1.67)
  herm coll & bdge  :  60 (1.67)
  fft 9pt order 2   : 132 (3.67)
  dyakanov-cg       : 132 (3.67)
  fft 9pt order 6   : 186 (5.17)
  dyakanov-cg 4     : 192 (5.33)
  fft 9pt order 4   : 246 (6.83)

Distribution of slopes for each Linear Solver
-----------------------------------------------
Linear Solver    Average  Minimum  1st Quart   Median   3rd Quart  Maximum
-------------------------------------------------------------------------
fft 9pt order 2   -1.896   -2.541     -1.896    -1.896     -1.529   -1.423
fft 9pt order 4   -3.958   -5.216     -3.958    -3.958     -3.095   -2.952
fft 9pt order 6   -2.944   -5.549     -2.944    -2.944     -1.701   -1.436
5pt star & bdge   -1.007   -1.613     -0.989   -0.8025     -0.779  -0.5201
herm coll & bdge -0.9616   -1.098    -0.9885   -0.9015    -0.8858  -0.8327
dyakanov-cg       -1.878   -2.021     -1.878    -1.878     -1.771   -1.721
dyakanov-cg 4     -2.534   -3.004     -2.534    -2.534     -2.405   -2.075
```

Fig. 17. Rankings for the PELLPACK solver case study.

The first rule R1, for instance, indicates that the method FFT6 is good for a certain problem if the problem has a Laplacian operator and the domain under consideration is a unit square[2].

When the rules generation process was complete, we placed high-level inquiries to the recommender regarding problem features and performance criteria. Figure 18 shows the recommender interface where users specify problem characteristics and performance objectives. Fig. 19 details the recommendations made for a specific problem instance and performance objectives. It can be seen that the recommendation satisfies the desired criteria set by the user.

4.1 Knowledge Discovery

The rules discovered confirm the statistically discovered conclusion in [Houstis and Rice 1982] that higher order methods are better for elliptic PDEs with singularities (which was a subset of the population used in our study). 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

[2]While these rules appear to use a hard-wired absolute ranking encoded by the `best` predicate, they can be easily updated to reflect new data, via the cover heuristic detailed in Section 2.3. The exact algorithm for effecting this 'online' capability is beyond the scope of this paper.

No.	PDE	First Method (from [Houstis and Rice 1982])	First Method (from PYTHIA II)	Second Method (from [Houstis and Rice 1982])	Second Method (from PYTHIA II)
1	3–1	FFT6	FFT6	FFT2	FFT4
2	3–2	FFT6	FFT6	FFT4	FFT4
3	7–1	FFT6, FFT4	FFT6	—	FFT4
4	8–2	FFT6	FFT6	FFT2	FFT4
5	9–1	FFT4	FFT4	FFT2	FFT2
6	9–2	FFT4	FFT4	FFT2	FFT2
7	9–3	FFT4	FFT4	FFT2	FFT2
8	10–2	FFT6	FFT6	FFT4	FFT4
9	10–3	FFT6	FFT6	FFT4	FFT4
10	10–4	FFT6	FFT6	FFT4	FFT4
11	10–7	FFT6	FFT6	FFT4	FFT4
12	11–2	FFT6	FFT6	FFT4	FFT4
13	11–3	FFT6	FFT6	FFT4	FFT4
14	11–4	FFT6	FFT6	FFT4	FFT4
15	11–5	FFT6	FFT6	FFT4	FFT4
16	13–1	DCG4, DCG	DCG	—	DCG4
17	15–1	P3C1C	P3C1C	PS5	PS5
18	15–2	P3C1C	P3C1C	PS5	PS5
19	17–1	FFT6	FFT6	FFT4	FFT4
20	17–2	FFT6	FFT6	FFT4	FFT4
21	17–3	FFT6	FFT6	FFT4	FFT4
22	20–1	PS5	PS5	P3C1C	P3C1C
23	20–2	PS5	PS5	P3C1C	P3C1C
24	28–2	DCG	DCG	PS5, DCG4	PS5
25	30–4	PS5	P3C1C	P3C1C	P3C1C
26	30–8	P3C1C	P3C1C	PS5	P3C1C
27	34–1	DCG4	DCG4	DCG2, P3C1C	DCG4
28	35–1	DCG4	DCG4	DCG2, P3C1C	DCG4
29	36–2	PS5,P3C1C	P3C1C	—	P3C1C
30	39–2	PS5	PS5	DCG4, DCG2	P3C1C
31	39–4	P3C1C	PS5	PS5, DCG4, DCG2	P3C1C
32	44–2	P3C1C	P3C1C	PS5	P3C1C
33	44–3	P3C1C	P3C1C	PS5, DCG4, DCG2	P3C1C
34	47–2	FFT6	FFT6	FFT4	FFT4
35	49–3	P3C1C	P3C1C	PS5	PS5
36	51–1	PS5	PS5	P3C1C	P3C1C
37	54–1	PS5	PS5	P3C1C	P3C1C

Table VII: A comparison betwen two different rankings of problem solving modules for elliptic PDEs. The third and fifth columns depict the subjective rankings made in an earlier study. The fourth and sixth columns depict those inferred by our knowledge methodology. The correspondence between these rankings is readily seen.

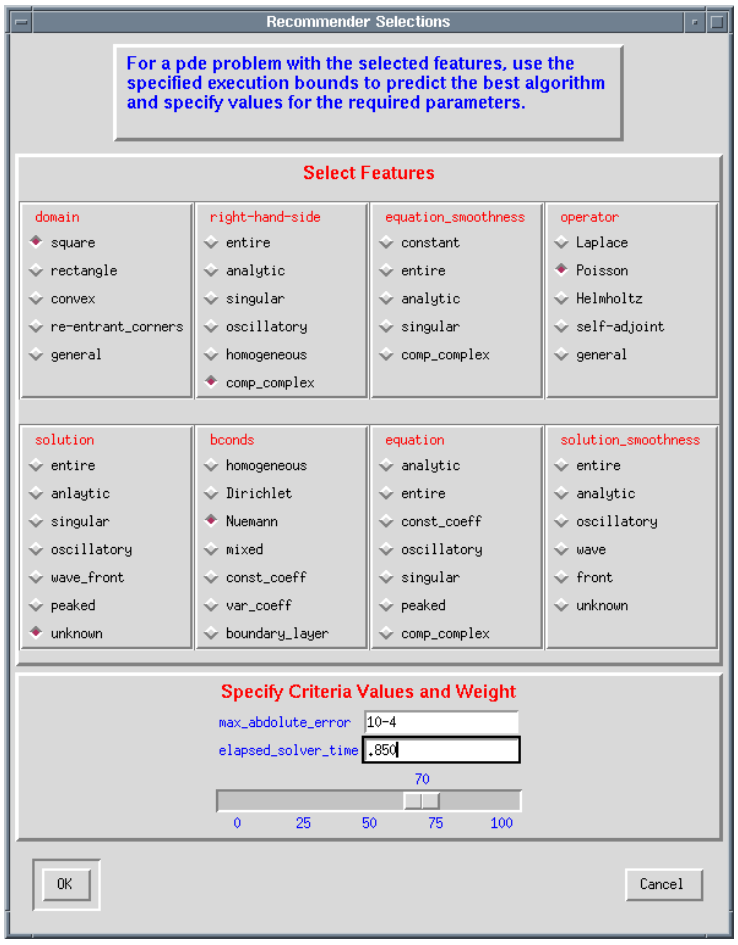

Fig. 18. End-user interface for the Recommender in the PELLPACK solver study.

corresponds almost exactly 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 VII compares these results. There were several other interesting inferences drawn. Whenever the DCG method is best, so is DCG4. The rule that had the maximum cover from the data was the one which stated that FFT6 is best for a PDE if the PDE has a Laplacian operator, homogeneous and Dirichlet boundary conditions and discontinuous derivatives on the right side. This can also be seen from rule R1, which recognizes the significant presence of a Laplace operator in a majority of the PDE population. Other rules also indicated when a certain method is inappropriate for a problem. The FFT6 module, for example is a 'bad' method whenever the problem has boundary conditions with variable coefficients. There are many more such interesting observations and we mention only the most interesting here. Finally, an approximate ordering was requested for the overall population. This gave rise to the ordering — FFT6, FFT4,

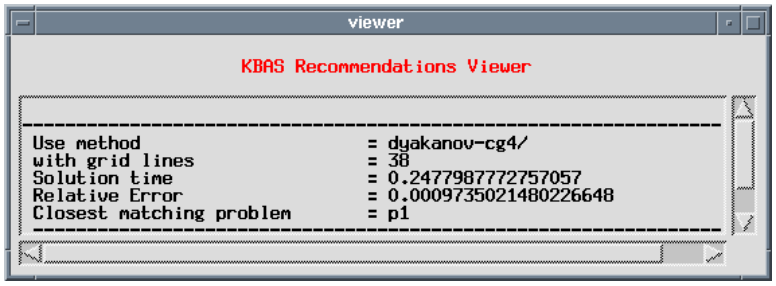

Fig. 19: A sample recommendation obtained from PYTHIA II. Notice that in addition to the recommendation, PYTHIA II provides estimates of relevant criteria and also the closest matching problem from the database.

FFT2, DCG4, DCG2, PS5. This is pertinent because this ranking corresponds most closely to that for Poisson problems which formed the bulk of our population. Furthermore, all the selections made by PYTHIA II are 'valid' (a selection is considered 'invalid' if the method is inappropriate for the given problem or if any of the parameters do not apply correctly to the method). In prior research, accuracy of algorithm selection was measured as the fraction of the valid selections that are also correct (a correct selection is one where the selected method and parameters does result in solutions satisfying the requested criteria). In overall, the rules from this study performed best algorithm recommendation for 100% of the cases.

5. CONCLUSION

The PYTHIA II recommender system, modeled after a systematic performance evaluation and testing methodology, facilitates the KDD process for manipulating performance data related to scientific computing applications. Its architecture is both flexible (allowing extension to newer domains) and scalable (providing a variety of options to the knowledge engineer for mining data, while storage and retrieval issues are handled by an integrated DBMS). The modular approach subsumed by the system maximizes the ability of an end-user to visualize 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 available at every stage of the KDD process.

In future, we plan to augment the functionality of the system to support incremental learning, distributed data mining techniques and mediating between multiple recommenders for the same problem. Some first steps in this direction are presented in [Joshi et al. 1998]. In addition, we plan to explore recommendations made using incomplete, uncertain and continually varying information.

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.

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.

Gallopoulos, E., Houstis, E., and Rice, J. 1994. Computer as Thinker/Doer: Problem-Solving Environments for Computational Science. *IEEE Computational Science and Enginerring Vol. 1,* 2, pages 11–23.

Houstis, C., Houstis, E., Rice, J., Varadaglou, P., and Papatheodorou, T. 1991. Athena: A Knowledge Based System for //ELLPACK. In E. Diday and Y. Lechavallier (Eds.), *Symbolic–Numeric Data Analysis and Learning,* pp. 459–467. Nova Science.

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. and Rice, J. 1980. An experimental design for the computational evaluation of elliptic partial differential equation solvers. In M. Delves and M. Hennell (Eds.), *The Production and Assessment of Numerical Software,* pp. 57–66. Academic Press.

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., Ramakrishnan, N., and Houstis, E. 1998. Multi–Agent Support for Networked Scientific Computing. *IEEE Internet Computing Vol. 2,* 3, pages 69–83.

Kitano, H. and Shimazu, H. 1996. H. kitano and h. shimazu. In D. Leake (Ed.), *Case-Based Reasoning: Experiences, Lessons, and Future Directions.* AAAI Press/MIT Press, Menlo Park, CA.

Kolodner, J. 1993. *Case-Based Reasoning.* Morgan Kaufmann, San Francisco.

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. 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.

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, J. R., Houstis, E., and Dyksen, W. 1981. A Population of Linear, Second Order, Elliptic Partial Differential Equations on Rectangular Domains. *Mathematics of Computation 36*, 475–484.

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

Riesbeck, C. 1996. What next? the future of cbr in postmodern ai. In D. Leake (Ed.), *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. AAAI Press/MIT Press, Menlo Park, CA.

Riesbeck, C. and Schank, R. 1989. *Inside Case-Based Reasoning*. Lawrence Erlbaum, Hillsdale, NJ.

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.

Watson, I. 1977. *Applying Case-Based Reasoning: Techniques for Enterprise Systems*. Morgan Kaufmann.

Weerawarana, S., Houstis, E. N., Rice, J. R., Joshi, A., and Houstis, C. 1997. Pythia: A Knowledge Based System to Select Scientific Algoritms. *ACM Trans. Math. Soft. 23*, 447–468.