A Data Mining Environment for Modeling the Performance of Scientific Software *

Elias N. Houstis, Vassilios S. Verykios, Ann C. Catlin
Department of Computer Sciences, Purdue University

Naren Ramakrishnan
Computer Science Department, Virginia Tech

John R. Rice
Department of Computer Sciences, Purdue University

March 2, 1999

Abstract

Complex problems, whether scientific or engineering, are most often solved today by utilizing public domain or commercial libraries or some form of problem solving environment. The task of "selecting" the best software for a targeted application or computation is often difficult and sometimes even intractable. We have proposed an approach for dealing with this issue by "mining" performance data of scientific software to generate knowledge that can be used to select software for a particular scientific problem, assuming some computational objectives. In this paper we describe a framework together with its software implementation for mining performance data of scientific software and use the results to generate knowledge necessary to solve the software selection problem.

1 Introduction

Very often scientists are faced with the task of locating appropriate solution software for their problems and then selecting from among many alternatives. 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

*This work was supported in part by NSF grant CDA 91-23502, PRF 6902851, DARPA grant N 66001-97-C-8533 (Navy), DOE LG-6982, and the Purdue Research Foundation.

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 1: A KDD methodology for modeling the performance evaluation of scientific software.

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 [HHR+91] we had proposed an approach for dealing with these issues by "mining" performance data obtained from "testing" software.

In [VHR98] we proposed a knowledge discovery in databases (KDD) [FPSS96, FHS96]) methodology for modeling the performance of scientific software. For completeness the process is summarized in Table 1. Assuming a "densely" distributed set of benchmark problems from the targeted application domain, this KDD methodology is based in a four-pronged strategy: feature determination of benchmark problems, performance evaluation of scientific software, data mining of performance data, and the automatic knowledge discovery of software performance. In this paper we describe the implementation of the KDD methodology as an open-ended software environment so that various users can customize the system for different case studies (i.e., modeling the performance of various software domains) or plug and evaluate different data mining techniques or knowledge discovery strategies. Figure 1 depicts the software architecture of PYTHIA-II.

Specifically, from the end-user perspective, PYTHIA-II will allow users (i) to import performance data according to a pre-defined schema (for various a priori defined software domains), (ii) to specify new database schemas for new software domains, (ii) to utilize existing facilities (PELLPACK [HRW+98]) to automatically generate performance data, (iii) to mine the performance data using statistical techniques to identify performance profiles and rankings of software together with confident intervals, (iv) to use the data mining results to automatically generate knowledge rules that model the performance of the software under user specified conditions [1].

The paper is organized as follows. Sections 2 and 3 present the design and components

[1] PYTHIA-II is an extension and a new implementation of the PYTHIA framework [WHR+97] for selecting scientific algorithms utilizing database, customized data mining techniques and well established knowledge discovery strategies.

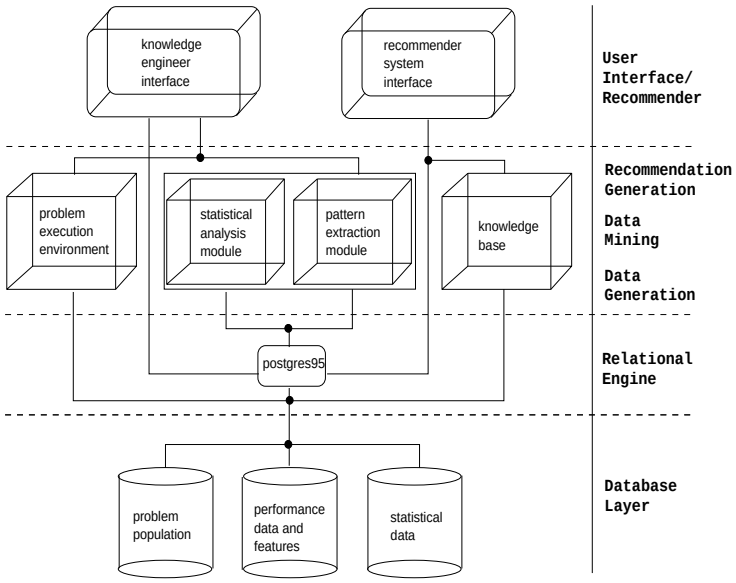

Figure 1: A block diagram of PYTHIA-II displaying the software components and the software architecture of the system.

of PYTHIA-II system. Section 4 describes the use of PYTHIA-II to model the performance of elliptic partial differential equation software. Some concluding remarks are incorporated in Section 5.

2 System Design

In this section we describe the overall design of the system in terms of its components and structure and the data flow.

2.1 Architecture

The modular design of PYTHIA-II is shown in Figure 1. The hierarchical architecture of the system consists of four layers:

- user interface layer
- data generation, data mining, and recommendation generation layer
- relational engine layer, and
- database layer.

The database layer provides permanent storage for the problem population, the performance data and problem features, and the computed statistical data. The next layer is the relational engine which supports an extended version of the SQL database query language and provides the required functionality for the stored data to be accessible to the upper layers. The third layer consists of three subsystems: the data generation system, the data

mining system, and the recommendation generation system. The data generation system accesses the records defining the problem population and processes 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 different learning paradigms.

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 end-user interface 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 or recommender system. The facts stored in the database drive the process of 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 2.2, and their functionality and implementation are described in Section 3.

For storage and database management, we selected the POSTGRES95 relational database and used *PgTcl* as the application programming interface of 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 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.

2.2 Data Flow

The PYTHIA-II design presented above supports two different user interfaces, one for the knowledge engineer and the other for end users who request 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 2, 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 specific database tables. In PYTHIA-II, the underlying database schema is fixed, but extensible and dynamic. The knowledge engineer has to specify his understanding of the domain in terms of the relational data model to match PYTHIA-II's database schema. 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.

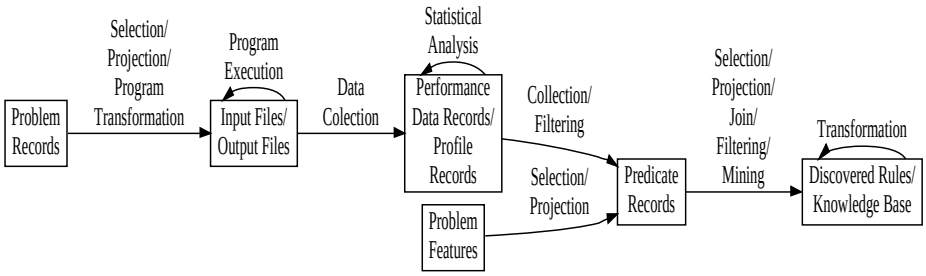

Figure 2: Data flow and I/O interfaces.

```
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';
```

Figure 3: Example analyzer query for retrieving performance data identified by a profile.

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 program execution. 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 utilizing 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 generated by the program execution. This data is automatically collected by the data collection program, and stored in the performance data records for further processing, while all the output files are deleted. The performance data 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 all but one of the experimental variables constant, 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 profiles with the required statistical significance. The analyzer submits "canned" SQL queries to retrieve the data to use for further processing. Figure 3 presents an instance of this process for the case study considered in Section 4.

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. 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.

End user perspective: The Recommender is the module within PYTHIA-II which is accessed by the end-user for requesting domain specific advice. The front-end for the recommender system must be configurable and adaptable for satisfying a variety of user needs. It is well understood that end users of a recommender system for scientific computing are most interested in questions regarding accuracy of a solution method, performance of a hardware system, optimal number of processors to be used in a parallel machine, how to achieve certain accuracy by keeping the execution time under some limit, etc. The PYTHIA-II recommender interface allows users to specify the characteristics of the problems to solve, as well as the performance objectives or constraints. The software system that supports this functionality in PYTHIA-II is CLIPS [**?**]. This is an expert system shell tool-box, which uses the induced knowledge, even background knowledge provided by domain experts, and facts from the problem, feature, performance, profile and predicate tables to provide the user with the best recommended solution to the problem presented. It is also possible that the user's objective cannot be satisfied. In that case, the user can specify weights for the various objectives, and then the system tries to satisfy the objectives (e.g., accuracy first, then memory constraints) based on the ordering implied by the weights.

3 System Components

This section describes the functionality of the components of PYTHIA-II contained in the top two layers of Figure 1.

3.1 Data Generation

Information in the performance database drives PYTHIA-II's data analysis and rule generation. The performance database may be a pre-existing store of performance measures or the data may be produced by executing scientific software within the problem execution environment. PYTHIA-II is independent of the characteristics and functionality of the software used for execution purposes, and it imposes no requirements or restrictions on the internal

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

Table 2: Algorithm ranking table based on Friedman Rank Sums using the two-way layout. X_{ij} is the performance of algorithm j on problem i, and R_j, $R_{\bullet j}$ are the rank assignments.

operation of this software. In fact, it allows the scientific software to operate entirely as a black box as long as it has well defined interfaces for communicating with the other parts of the PYTHIA-II system.

3.2 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 (analyzer) and the pattern extraction module.

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

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

A predicate record defines the complete set of analyzer runs which are to be used as input for a single invocation of the rules generator. The predicate fields of interest to the analyzer are (1) the list of algorithms to rank, and (2) a profile matrix, where each row represents a single analyzer run and the columns identify the *profile* records to be accessed for that run. Each profile record specifies how the analyzer should gather and assess the performance measures produced by one problem execution. Table 2 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 and algorithm j (see below for the discussion of the methods used to compute the X_{ij}).

The process for ranking the algorithms uses multiple comparisons and contrast estimators based on Friedman rank sums [HW73]. The two-way layout associated with distribution-free

testing is shown in Table 2, 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' performance. Let r_{ij} denote the rank of X_{ij} in the joint rankings of $X_{i1}, ...X_{ik}$ and compute $R_j = \sum_{i=1}^{n} r_{ij}$.
- Let $R_{\bullet j} = \frac{R_j}{n}$ where R_j is the sum over all problems of the ranks for algorithms j, and then $R_{\bullet j}$ is the average rank for algorithm j. *Use $R_{\bullet j}$ 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 record 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.

The goal of the pattern-extraction module is to support the automatic knowledge acquisition process and to extract patterns/models from the data to be used by the recommender to provide advice to end users. This process is independent of the problem domain.

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

The first learning system we integrated was GOLEM [MF90]. It can be classified as an empirical single predicate Inductive Logic Programming (ILP) learning system [Dze96]. 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.

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 as its rule base, as described below. In addition to GOLEM, we have also integrated the following learning systems into PYTHIA-II: PROGOL, MLC++ library, CN2, PEBLS, and OC1.

3.3 Recommendation Generator

The recommender system is the end-user module of the PYTHIA-II system. It is a form of 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 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 measuring the generalization accuracy of the rule. Its simple formula is $f = p - (c + n + h)$ where p and n are the number of positive and negative examples respectively covered by a specific rule, while c and h are information that is related to the form of the rule. The information compression factor is used for ordering the rules in the rule base in a decreasing order.

Each rule selected by GOLEM covers a number of positive and negative examples. The set of positive examples covered for each rule along with the rules, is one part of the input given to the recommender system. The recommender system asks the user to specify the features of the problem he wants to solve. It then uses the CLIPS inference engine and checks its rule base to find a rule whose left-hand side satisfies the user selected problem features. Every rule that is found to match the problem features specified by the user is selected and is placed into the *agenda*. Rules are sorted in decreasing order based on their generality (number of examples they cover), and the very first rule in the agenda is fired to determine the best algorithm for the problem the user specifies. Since each rule provided by GOLEM to the recommender system is associated with a set of positive examples that are covered by this rule, the recommender system 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 this example problem is selected, the fact base of the recommender system 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 stored in the database. The recommender system accesses this information by submitting queries generated on the fly, based on the user's objectives and selections. If the user objectives cannot be met, a recommendation is provided as described at the end of Section 2.2. For the recommender system used in the case study presented in the next section, the final step is the recommendation of a certain method, as the best method to use to satisfy the given conditions. It also indicates what grid size should be used to achieve the specified the accuracy within the time limitations imposed by the user.

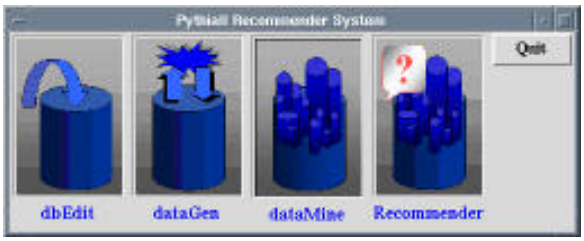

Figure 4: PYTHIA-II's top level window. A web-based version of the front-end recommender system can be accessed from: `http://www.cs.purdue.edu/research/cse/pythia-II`.

3.4 User Interface

The modular implementation of PYTHIA-II makes it possible to accomplish much of the work involved in knowledge discovery without resorting to 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. The process is invoked with an argument list describing the I/O for the program generation, and it may be called outside of PYTHIA-II.

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

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

With respect to the above items, the graphical interfaces that assist in those tasks are most useful for knowledge engineers who are unfamiliar either with the structure of PYTHIA-II or with the SQL language used by POSTGRES95. In this case, the interfaces provided by PYTHIA-II's *dbEdit* and *dataGEN* are invaluable. The top level window of the PYTHIA-II system is shown in Figure 4.

The graphical interface to the POSTGRES95 database is called *dbEdit*. *dbEdit* supports browsing, editing, deletion, and insertion of all kinds of records contained in the database. Records are displayed as *forms* very much like those supported by ORACLE's SQL*Forms, implemented in TCL/Tk in such a way that half of the form displays static information related to the schema of the record retrieved, and the rest displays data kept by the record which has been accessed based on a user specified selection. In addition to the above functionality, we have incorporated in our environment the immediate access of the POSTGRES95 backend through the subset of SQL supported by the underlying DBMS and a text-based window.

dataGEN facilitates the tasks involved in the data generation process. 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 extraction process. Even for the most experienced users, it is impractical 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. The input specification for pattern extraction 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 graphical user interface, *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 [OWA+98], 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 is built into it.

4 Case Study : Modeling the Performance of Elliptic PDE Software

To validate the design and implementation of PYTHIA-II, a knowledge base was generated for evaluating PELLPACK [HRW+98] solvers based on performance data produced by a population of 2-dimensional, singular, steady state PDE problems. This case study which corresponds to existing studies [RHD81, WHR+97, HR82], allows for validation of the adopted KDD process. The algorithm selection 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.

4.1 Performance Database Description

In this study, we 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

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 3: General form of the PDE problems included in the 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 4: Methods and solver sequences used for the case study.

to obtain three levels of accuracy — 10^{-3}, 10^{-4} and 10^{-5} — are collected by the PYTHIA-II system.

Table 3 shows the general form of the PDE problems included in the study. In Table 4, the solver modules and solver sequences which were applied to the problems are listed. Table 5 identifies the features of the problem components used to drive the rules generation and form the basis for user inquiries to the PYTHIA-II recommender system. Table 6 uses the "raw data" descriptions in Tables 3 and 4 to demonstrate how the recommender methodology of PYTHIA-II was applied to the 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 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

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 5: 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 3 (2) measures: elapsed solver time, discretization error. (3) methods: Table 4 (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 6: Applying PYTHIA-II to the PELLPACK case study.

passed to the rules generator which produced logic-based rules governing method selection for PELLPACK solvers. The recommender system 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 system 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 fact base.

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 5, 6, and 7. The dbEdit interface is used for editing problem, method and experiment records.

```
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)
);
```

Figure 5: 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)
);
```

Figure 6: 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.

4.2 Data Mining and Knowledge Discovery Process

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 them in the performance database. The dataMINE interface was used to access the performance data according to the specification of the predicate and profile records created for the case study. A portion of the predicate record is shown in Figure 9. The predicate specified *all* problems and methods so that the data available to the recommender system 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 8 lists the ranking produced by the analyzer for all solvers over a parameterized problem. The rankings and features were used by GOLEM to generate 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 first rule R1, for instance, indicates that the method FFT6 is best if the problem has a Laplacian operator and the domain under consideration is a unit square[2].

[2]While these rules appear to use a hard-wired absolute ranking encoded by the `best` predicate, they can

```
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
);
```

Figure 7: The experiment record specifies the components of a PDE problem and identifies the collection of sequences to use in solving it.

4.3 Knowledge Discovery Outcome

The rules discovered confirm the statistically discovered conclusion in [HR82] 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 corresponds almost exactly with the subjective rankings published in [HR82]. This shows that these simple rules capture much of the complexity of algorithm selection in this domain. Table 7 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, FFT2, DCG4, DCG2, PS5. This is pertinent because this ranking corresponds most closely to that for Poisson

be easily updated to reflect new data, via the cover heuristic detailed in Section 3.3. The exact algorithm for effecting this "incremental learning" capability is beyond the scope of this paper.

```
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       Avg   Min  Quart_1  Med  Quart_3   Max
-------------------------------------------------------------
fft 9pt order 2   -1.89  -2.54  -1.89  -1.89  -1.52  -1.42
fft 9pt order 4   -3.95  -5.21  -3.95  -3.95  -3.09  -2.95
fft 9pt order 6   -2.94  -5.54  -2.94  -2.94  -1.70  -1.43
5pt star & bdge   -1.00  -1.61  -0.98  -0.80  -0.77  -0.52
herm coll & bdge -0.961  -1.09  -0.98  -0.90  -0.88  -0.83
dyakanov-cg       -1.87  -2.02  -1.87  -1.87  -1.77  -1.72
dyakanov-cg 4     -2.53  -3.00  -2.53  -2.53  -2.40  -2.07
```

Figure 8: Ranking for the PELLPACK solvers for a parameterized problem from the PDE population.

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 software environment, facilitates the knowledge discovery in databases (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 database system). 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.

```
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]"}
```

Figure 9: Partial listing of a predicate record from the PDE benchmark.

References

[Dze96] Saso Dzeroski. Inductive logic programming and knowledge discovery in databases. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 117–152. AAAI Press/MIT Press, Menlo Park, CA, 1996.

[FHS96] U. Fayyad, D. Haussler, and P. Stolorz. Mining scientific data. *Comm. ACM*, 39(11):51–57, 1996.

[FPSS96] U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 1–34. AAAI Press/MIT Press, Meno Park, CA, 1996.

[HHR+91] C.E. Houstis, E.N. Houstis, J.R. Rice, P. Varadaglou, and T.S. Papatheodorou. Athena: A Knowledge Based System for //ELLPACK. In E. Diday and Y. Lechavallier, editors, *Symbolic–numeric data analysis and learning*, pages 459–467. Nova Science, 1991.

[HR82] E.N. Houstis and J. R. Rice. High order Methods for elliptic partial differential equations with singularities. *Inter. J. Numer. Meth. Engin.*, 18:737–754, 1982.

[HRW+98] E.N. Houstis, J.R. Rice, S. Weerawarana, A.C. Catlin, M. Gaitatzes, P. Papachiou, and K. Wang. PELLPACK: A problem solving environment for PDE based applications on multicomputer platforms. *ACM Trans. Math. Soft.*, 24(1):30–73, 1998.

[HW73] M. Hollander and D.A. Wolfe. *Nonparametric statistical methods.* John Wiley and Sons, 1973.

No.	PDE	First Method (from [HR82])	First Method (from PYTHIA-II)	Second Method (from [HR82])	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 7: A comparison between two different rankings of problem solving modules for elliptic PDEs. The third and fifth columns give the subjective rankings made in an earlier study. The fourth and sixth columns give those inferred by our knowledge methodology. The very high correlation between these rankings is readily seen.

[MF90] S. Muggleton and C. Feng. Efficient induction of logic programs. In S. Arikawa, S. Goto, S. Ohsuga, and T. Yokomori, editors, *Proceedings of the First International Conference on Algorithmic Learning Theory*, pages 368–381. Japanese Society for Artificial Intelligence, Tokyo, 1990.

[OWA+98] C. Olston, A. Woodruff, A. Aiken, M. Chu, V. Ercegovac, M. Lin, M. Spalding, and M. Stonebraker. Datasplash. In *Proceedings of the ACM-SIGMOD Conference on Management of Data*, pages 550–552, ACM, New York, NY, 1998.

[RHD81] J. R. Rice, E.N. Houstis, and W.R. Dyksen. A population of linear, second order, elliptic partial differential equations on rectangular domains. *Mathematics of Computation*, 36:475–484, 1981.

[VHR98] V. S. Verykios, E. N. Houstis, and J. R. Rice. A knowledge discovery methodoloy for the performance evaluation of scientific software. Technical Report TR-98-031, Dept. Comp. Sci., Purdue University, 1998.

[WHR+97] S. Weerawarana, E. N. Houstis, J. R. Rice, A. Joshi, and C. Houstis. Pythia: A knowledge based system to select scientific algoritms. *ACM Trans. Math. Soft.*, 23:447–468, 1997.