CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2002; **14**:1481–1505 (DOI: 10.1002/cpe.684)

MyPYTHIA: a recommendation portal for scientific software and services

E. N. Houstis[1,*,†], A. C. Catlin[1], N. Dhanjani[1], J. R. Rice[1],
N. Ramakrishnan[2] and V. Verykios[3]

[1] *Department of Computer Sciences, Purdue University, West Lafayette, IN 47907, U.S.A.*
[2] *Department of Computer Science, Virginia Tech, Blacksburg, VA 24061, U.S.A.*
[3] *College of Information Science and Technology, Drexel University, Philadelphia, PA 19104, U.S.A.*

SUMMARY

We outline the design of a recommendation system (MyPYTHIA) implemented as a Web portal. MyPYTHIA's design objectives include evaluating the quality and performance of scientific software on Grid platforms, creating knowledge about which software and computational services should be selected for solving particular problems, selecting parameters of software (or of computational services) based on user-specified computational objectives, providing access to performance data and knowledge bases over the Web and enabling recommendations for targeted application domains. MyPYTHIA uses a combination of statistical analysis, pattern extraction techniques and a database of software performance to map feature-based representations of problem instances to appropriate software. MyPYTHIA's open architecture allows the user to customize it for conducting individual case studies. We describe the architecture as well as several scientific domains of knowledge enabled by such case studies. Copyright © 2002 John Wiley & Sons, Ltd.

KEY WORDS: Grid computing environments; problem solving environments; recommender systems; Web portals; data mining; knowledge discovery

1. INTRODUCTION

The new economic realities require the rapid prototyping of manufactured artifacts and rapid solutions to problems with numerous interrelated elements. Thus, computational modeling and simulation has

[*] Correspondence to: Elias N. Houstis, Department of Computer Sciences, Purdue University, West Lafayette, IN 47907, U.S.A.
[†] E-mail: enh@cs.purdue.edu

Contract/grant sponsor: NSF; contract/grant number: CDA 9123502, EIA-9974956, EIA-9984317, PRF 6902851
Contract/grant sponsor: DARPA (Navy); contract/grant number: N66001-97-C-8533
Contract/grant sponsor: DOE; contract/grant number: LG-6982
Contract/grant sponsor: DARPA under ARO; contract/grant number: DAAH04-94-G-0010
Contract/grant sponsor: Purdue Research Foundation

Received 26 July 2001
Revised 6 January 2002

already become the main tool of ‘big’ and ‘small’ science, and it is shifting from the single physical component design to the design of a whole physical system. Increasingly, such simulations are being dynamically composed by scientists working collaboratively on the Grid, even though they are geographically distributed [1–6].

The realization of this simulation scenario requires that Grid users be able to identify relevant software and its parameters before implementation on a remote computing resource. However, the obstacles to selecting the best solution approach for a particular problem, and subsequently finding an appropriate software implementation, are often difficult, and sometimes even impossible, to surmount. What is required is a knowledge-based technology for the location and assembly of scientific software components. We refer to such services as ‘recommendation services’ and the underlying software selection engines as recommendation systems or ‘recommender systems’ [7–14].

In this paper, we describe the MyPYTHIA Web recommendation portal that evolved from our earlier efforts in scientific recommendation systems [7,14]. Our design philosophy for scientific recommendation systems can be characterized as data-driven, i.e. a database of performance data on a benchmark set of problem instances is accumulated, and this database is mined to form the basis of a recommendation for future problem instances. A necessary assumption, thus, is that the performance database accumulated for mining is representative of the set of problem instances (and performance indicators) that will be encountered in the future. This assumption is valid in several problem domains where the Grid has enabled a recurring methodology of computation (e.g. solving linear algebra problems using component-based software from an OO library [15], solving partial differential equations using PSEs [16] and solving optimization problems using a service such as NEOS [17]). Hence, our focus is on application domains where there is substantial agreement over the methodology of computation and for which a large class of specialized algorithms and software have been developed. Later in the discussion, we address how MyPYTHIA’s design and architecture can be extended to support other domains. Just as recommender systems in electronic commerce reduce information overload, recommender systems in the Grid setting provide targeted access to scientific software.

The precursor to MyPYTHIA is the original PYTHIA system [14]. This facility used case-based reasoning and made recommendations of PDE methods/software by comparing performances on similar problem instances. In [7], we extended this methodology into the more general PYTHIA-II database architecture, with well-defined interfaces for learning and reasoning. The recommendation problem was ‘configurable’, in that the knowledge engineer had to define appropriate database records and specify interactions with the underlying execution environment. PYTHIA-II then provided all the necessary enabling technologies to prototype recommender systems. MyPYTHIA is the final step in this evolution and is meant to be a truly Web-accessible system that can be customized by any Grid user desiring recommendation services. MyPYTHIA addresses the following design objectives:

- evaluate the quality and performance of software/machines pairs; MyPYTHIA allows users to incorporate native performance data and provides all necessary facilities for their evaluation and data mining;
- create knowledge about which software and computational services should be selected for solving particular problems or realizing application components; MyPYTHIA provides data mining procedures to reduce the original data and generate rules from the original performance data;

- select the parameters of software (or computational service) based on user-specified computational objectives; MyPYTHIA estimates parameters of software/hardware that satisfy given user-defined objectives such as time and/or memory requirements;
- provide performance data and knowledge for specific software domains over the Web; MyPYTHIA can serve as a knowledge base for *a priori* defined case studies;
- provide recommendation services for targeted application domains.

Outline of paper

Section 2 motivates and describes the algorithmic basis of the MyPYTHIA recommendation methodology. Section 3 identifies the major components of MyPYTHIA, along with system implementation details. Section 4 describes three case studies that have been conducted with MyPYTHIA. Section 5 elaborates on one of these studies, providing details of the analysis and descriptions of the results. Section 6 identifies opportunities for future research.

2. MyPYTHIA METHODOLOGY

One of the basic research issues in designing recommendation services is understanding the fundamental processes by which knowledge about scientific problems and their solutions is created, validated and communicated. Some of this knowledge will come directly from experts—scientists and engineers—in the field. Other knowledge can be mined from experimental data arising from simulations. Yet further knowledge will be learned from the experience gained by the recommendation service itself as it extracts performance knowledge about software components running on the Grid and applied to various problems. Thus, it is not surprising that the word ‘recommendation’ is interpreted differently in the scope of different projects.

The typical approach to recommendation uses a polyalgorithm [18,19], which is a decision procedure hardwired to choose among different solution methods. This can be viewed as a first level of recommendation service. It is, however, a very restrictive one since the decision procedure is statically hardwired to the application and, except in artificial problem domains, does not generalize to new situations. In many cases, it does not take into account all available problem features and cannot incorporate the latest solution methods. The recommendations have to be ‘over-cautious’ to prevent various numerical disasters [20]. Furthermore, recommendations provided in this way cannot be used independently of the accompanying software/modules (and hence do not constitute a Grid service by themselves).

Our approach explicitly recognizes the Grid context in which scientific computations are performed. This means that performance evaluation of scientific software is effected in a Grid setting and features utilized for making recommendations are also cognizant of the Grid context for problem solving. To be as widely applicable as possible, MyPYTHIA allows the definition of performance metrics and parameterized scientific software as appropriate to the underlying application domain.

The proper way to distinguish MyPYTHIA from other Grid services is by its support for modeling the recommendation problem. Interaction with MyPYTHIA begins by the knowledge engineer defining records that are appropriate to the underlying application. Consider an application domain that involves running a nuclear physics code on a cluster of workstations using Globus. Assume that there are several possibilities of conducting this simulation, with choices pertaining to problem domain partitioning. The knowledge engineer first defines a benchmark set of problems and a suitable set of performance metrics in MyPYTHIA. This allows the engineer to define his study as a collection of (parameterized) database records. Simulation data collected by running the code (for the parameterized study) is tabulated and stored in a MyPYTHIA database session. Execution interfaces to the underlying application are assumed to be provided by the engineer, independent of MyPYTHIA. A performance database is thus accumulated for the benchmark set of problems. The engineer then uses the machine learning algorithms supplied in MyPYTHIA to mine this database. One preferred embodiment of this stage is the determination of high-level rules that correlate problem (and machine) characteristics to application performance. Such rules can then be used to determine appropriate recommendations for a new set of problems. This will help the engineer, for instance, to recommend a suitable partitioning strategy for unseen problem instances.

The reader will notice that some of the goals of MyPYTHIA are similar to performance databases such as the PDS server [21], in that they address the problem of mathematical software evaluation. However, the main focus of such database services is on profiling solution methods and they do not aid in problem solving beyond the tuning of architectural parameters. A richer variety of features are modeled by rating services and online taxonomies of mathematical software (e.g. GAMS), but such services are more information-based and cannot be easily invoked in a computational pipeline due to differences in vocabulary and mismatches in problem domain representation.

Before we describe the MyPYTHIA methodology, it is useful to define some notation. We denote by:

- $\mathcal{P}$, a collection of problems or problem space;
- $\mathcal{F}$, a feature space for problems in $\mathcal{P}$;
- $\mathcal{A}$, a collection of parameterized scientific software for simulating a specific class of problems in $\mathcal{P}$;
- $\mathcal{M}$, a collection of parameterized machines (hardware);
- $\mathcal{V}$, a set of n performance metrics;
- s, a mapping from $\mathcal{P} \times \mathcal{A} \times \mathcal{M}$ to $\mathcal{U}^n \subset \mathcal{R}^n$ where the points in the range of this mapping denote the measurements for the metrics in $\mathcal{V}$ obtained by running (a, m) pairs ($(a, m) \in \mathcal{A} \times \mathcal{M}$) on the benchmark space $\mathcal{P}$, $\mathcal{R}$ is the set of reals; and
- $\mathcal{W}$, a collection of profiles for each software/machine pair defined from pairs of metrics from $\mathcal{V}$.

Profiles are high-level summarizations of performance data that are useful in analysis and mining operations. An example of a profile is a low-degree polynomial approximation which can be used to obtain interpolated data over predefined regions. MyPYTHIA's methodology addresses the partition (clustering) of the performance data space $\mathcal{U}^n$ into subclasses $\{\mathcal{C}_i\}_{i=1}^k$ with respect to a set of profiles (or pairs of metrics) and the description (classification) of those classes in terms of problem features from $\mathcal{F}$. The classification part can be used to identify the best (a, m) pair for all $p \in \mathcal{P}$ or a user defined subclass of $\mathcal{P}$. We refer to this problem as the method/machine selection problem. In addition, MyPYTHIA can address the problem of determining the associated parameters of the best (a, m) pair (parameter estimation problem) along with predicting the performance of the selected pair to the user's problem by a performance mapping.

Assuming a dense benchmark of problems from a targeted application domain, MyPYTHIA uses a four-pronged processing phase to realize the solution of the above problems: (i) feature classification of benchmark problems and applicable software, (ii) performance evaluation of scientific software, (iii) statistical analysis of performance data, and (iv) the (automatic) knowledge discovery of recommendation rules from software performance. These phases have already been defined in previous papers [7,14,22] and address many of the difficulties in algorithm recommendation outlined previously. In the Web portal implementation of MyPYTHIA, only the interface of the portal and the underlying database facilities have been redesigned. The new design allows users to import their own data necessary for the MyPYTHIA methodology to conduct performance evaluation and for the creation of knowledge for a class of software/machine pairs.

3. MyPYTHIA SYSTEM COMPONENTS

We now provide detailed descriptions of MyPYTHIA's system components. The MyPYTHIA portal is served by an Apache Webserver [23]. PHP [24] is the scripting language of choice and enables a modular design with its support for object oriented class definitions. PHP's built-in support for sessions helps provide the user with an easy to navigate interface. Throughout, we use the term 'method' to refer to software that simulates a targeted class of problems.

3.1. Data management and retrieval

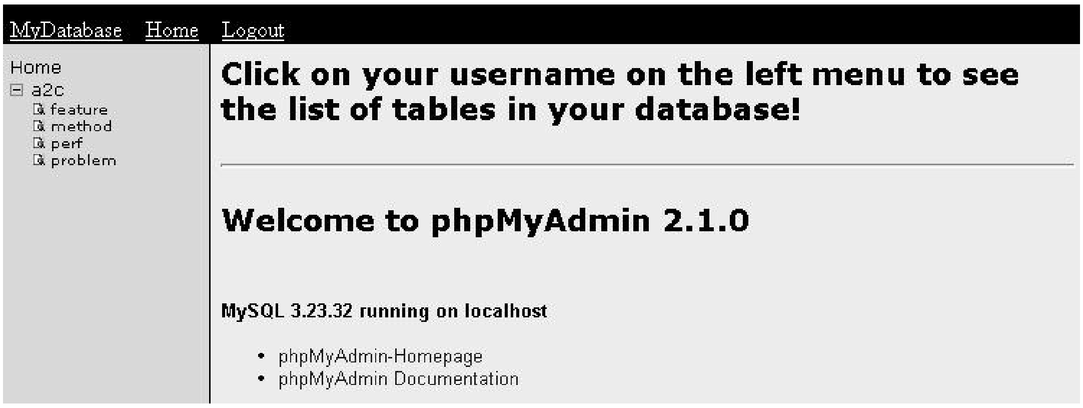
Each MyPYTHIA user is provided with an isolated view of their own database. Connection to the underlying MySQL [25] (open source RDBMS) database is supported by available PHP database connectivity drivers. When new users attempt to view their database contents for the first time, a simple SQL script is run which populates the isolated database with relevant default data. The end-user Web-enabled database interface is made possible by tailoring the PhpMyAdmin package [26] to suit the security needs of the MyPYTHIA implementation. The MyPYTHIA database interface indicating the MySQL and PHP supporting components is shown in Figure 1.

The database schema which is used to model and manage the data for the MyPYTHIA recommender has been designed in a very general way, so that the analysis and data mining process can easily be extended to any knowledge domain. Performance data must be generated, collected and classified external to the MyPYTHIA system. For instance, interfaces to Grid-based problem solving environments and remote computing servers can be utilized here. The performance data is then imported into the MyPYTHIA storage manager using an XML-like interface. This eliminates the domain-specific representations required to internally generate performance data [7]. Our earlier schema (described elsewhere [7]) required users to define all domain-specific elements of the problems and methods so that the system itself could create and execute the programs for performance data acquisition. This assumption is relaxed in MyPYTHIA.

The underlying representations required by MyPYTHIA to support the generation of inference data are listed as follows.

- *Benchmark problem population:*

```
name         varchar() -- problem name (primary key)
description  text      -- high level description of the problem
code         text      -- user description of the problem code
features     text      -- keys for accessing problem characteristics
featurevals  text      -- values for problem characteristics
```



Click on your username on the left menu to see the list of tables in your database!

Welcome to phpMyAdmin 2.1.0

MySQL 3.23.32 running on localhost

- phpMyAdmin-Homepage
- phpMyAdmin Documentation

Figure 1. MyPYTHIA database homepage for creating the database, importing performance data, and creating records which identify the benchmark problems, the collection of methods, and the features of interest.

Note that the *code* field is used for descriptive purposes only. Users may enter a description of the code, a path to the code source, or even the code itself in this field, but MyPYTHIA does not use it in the analysis process. The list of features and their values are the essential structures required in the analysis of the benchmark population.

- *Methods:*

```
name         varchar() -- method name (primary key)
description  text      -- high level description of the method
features     text      -- keys for accessing method characteristics
featurevals  text      -- values for method characteristics
measures     text      -- performance measures to use in method evaluation
```

Features and measures are the critical elements required by MyPYTHIA to analyze the methods and generate rules concerning their application.

- *Features:*

```
name            varchar() -- feature name (primary key)
description     text      -- high level description of the feature
possiblevalues  text      -- values assumed by this feature
```

This is a generic schema for features associated with problems or methods. Features may even be defined to correspond directly to a set of performance data. That is, the feature may occur as a result of problem-method execution, and may not be identified specifically with either a problem or method. An example set of *possiblevalues* is {‘yes’, ‘no’} for the feature ‘problem uses a uniform grid in all dimensions’.

- *Performance data:*

```
name          varchar() -- performance record  name (primary key)
description   text      -- high level description of the method
problem       text      -- benchmark problem used to generate this data
method        text      -- method/software used to generate this data
features      text      -- problem-method execution characteristics
featurevals   text      -- values for characteristics
machine       text      -- execution environment, machine characteristics
perfarray     text      -- list of performance measure names
perfvalues    text      -- list of performance values
```

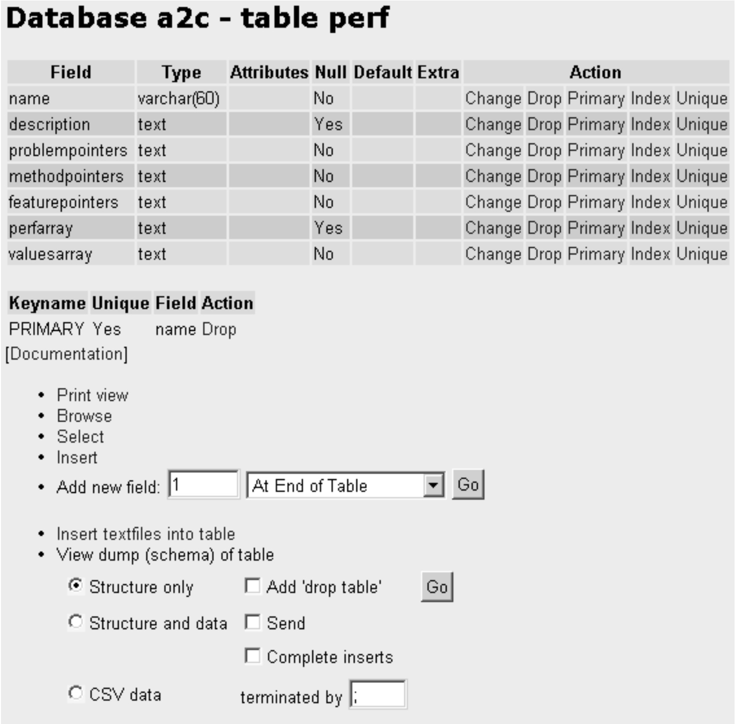
  The problem and method fields are used to access the features associated with the (problem and method) input used in producing the performance data. They may also be accessed by the end-user to get more comprehensive information about the kinds of problems and methods which are the source for the generated data. The performance measures which are available for analyzing the methods are named in *perfarray*, and their values are listed in *perfvalues*.

- *Feature-measure context:*

```
name           varchar() -- context record  name (primary key)
description    text      -- high level description of the context
runs           int       -- number of analyzer runs
problemlist    text      -- list of problems classified by run
methodlist     text      -- list of methods to evaluate
measurelist    text      -- list of performance measures to extract
profileparm    text      -- problem-to-method profile definition
featurelist    text      -- list of features to consider
```

  The context record determines the strategy for analyzing the data. The data must be collected and sent to the analyzer in a way that guarantees the formulation of rules that can answer the target recommendation queries. The selected problem-method performance data must be classified according to problem parameter sets (one or more parameterized classes are combined to determine one analyzer run) so that the correct grouping of features will be achieved in the ranking of the methods. Finally, the *profileparm* must identify one particular problem parameter to be used in creating the performance profiles. The measure values are collected for all variations of the 'profile parameter' which are associated with a given problem-parameter-method.

The typical usage of MyPYTHIA assumes that a substantial body of performance data already exists for the user's knowledge domain, and that the user can import this data into MyPYTHIA. The performance data should represent values that quantify the distinguishing characteristics of the collection of methods (speed of execution, accuracy, scalability, etc.) and include all important measures for evaluating which methods are better than others for a given benchmark problem. The benchmark features represent the interesting and important characteristics of problems that scientists are interested in solving. As far as possible, the collection of methods should be applied to all the benchmark problems and the measures should be collected across all applications of the methods (statistical pre-processing techniques can help overcome any lack of coverage for the underlying problem population). It remains the task of the domain expert to assemble the benchmark problems and the collection of methods, and to identify the salient, relevant features of both problems and methods. Performance measures should capture the significance of changes in problem features on method execution.



Database a2c - table perf

Field	Type	Attributes	Null	Default	Extra	Action
name	varchar(60)		No			Change Drop Primary Index Unique
description	text		Yes			Change Drop Primary Index Unique
problempointers	text		No			Change Drop Primary Index Unique
methodpointers	text		No			Change Drop Primary Index Unique
featurepointers	text		No			Change Drop Primary Index Unique
perfarray	text		Yes			Change Drop Primary Index Unique
valuesarray	text		No			Change Drop Primary Index Unique

Keyname	Unique	Field	Action
PRIMARY	Yes	name	Drop

[Documentation]

- Print view
- Browse
- Select
- Insert
- Add new field: 1 At End of Table ▾ Go

- Insert textfiles into table
- View dump (schema) of table

● Structure only ☐ Add 'drop table' Go
○ Structure and data ☐ Send
☐ Complete inserts
○ CSV data terminated by ;

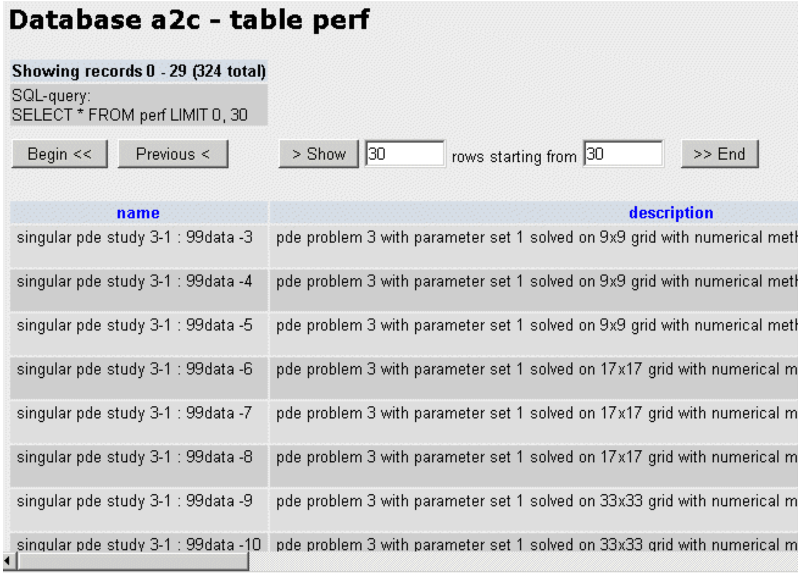
Figure 2. End-user interface for importing the performance data.

The features that characterize the problems, the methods and the execution create the context of the inference data that subsequent recommendations are based upon. More specifically, a particular instance of inference data (which the end-user chooses for eliciting a recommendation) is based entirely on the feature-measure context. The performance measures determine the ranking of methods as applied to the selected benchmark population and the features must explain (through the MyPYTHIA rules) why the methods behaved differently.

As an example, scientists may want MyPYTHIA to recommend methods that are scalable. A problem feature, 'size', could be used to identify methods for which performance deteriorates as problem size increases, but only if the domain expert included performance measures for length of execution time over a range of problem sizes.

3.2. Performance data generation and collection

As described earlier, MyPYTHIA factors out the underlying execution environment for characterizing application/software performance.



Database a2c - table perf

Showing records 0 - 29 (324 total)

SQL-query:
SELECT * FROM perf LIMIT 0, 30

Show 30 rows starting from 30

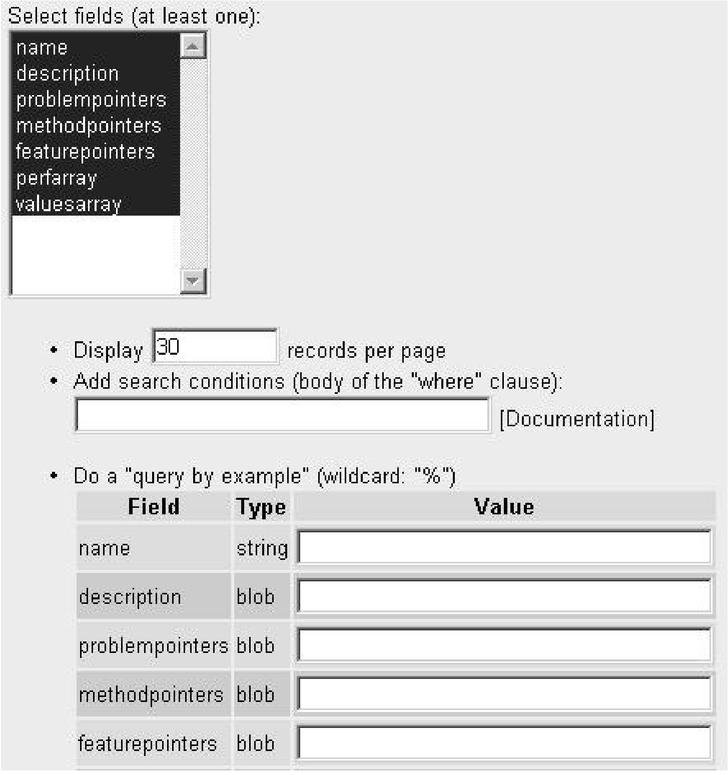
name	description
singular pde study 3-1 : 99data -3	pde problem 3 with parameter set 1 solved on 9x9 grid with numerical meth
singular pde study 3-1 : 99data -4	pde problem 3 with parameter set 1 solved on 9x9 grid with numerical meth
singular pde study 3-1 : 99data -5	pde problem 3 with parameter set 1 solved on 9x9 grid with numerical meth
singular pde study 3-1 : 99data -6	pde problem 3 with parameter set 1 solved on 17x17 grid with numerical m
singular pde study 3-1 : 99data -7	pde problem 3 with parameter set 1 solved on 17x17 grid with numerical m
singular pde study 3-1 : 99data -8	pde problem 3 with parameter set 1 solved on 17x17 grid with numerical m
singular pde study 3-1 : 99data -9	pde problem 3 with parameter set 1 solved on 33x33 grid with numerical m
singular pde study 3-1 : 99data -10	pde problem 3 with parameter set 1 solved on 33x33 grid with numerical m

Figure 3. End-user interface for browsing the performance database.

The simple schema of MyPYTHIA records and the XML-like import format provide the mechanism for populating the user's MyPYTHIA database with large collections of existing performance data. Users can build retrieval scripts with XML filters to apply to their current data storage environment, extracting the targeted data and inserting the XML definition tags. The result is then imported directly into the MyPYTHIA database manager. A sample XML definition for importing a specific instance of performance data into MyPYTHIA is now given.

```
<complexType name="MyPYTHIARecord">
  <element name="Performance" type="varchar()" />
  <element ref="Description" item="singular pde study 10-2 : 99data -3"/>
  <element ref="perfarray" item="number of iterations, elapsed time,
               max-grid-delta, number of equations, problem size,
               abs max error at nodes, discrete L2 error at nodes,
               discrete L2 residual at midpoints, relative error"/>
  <element ref="perfvalues" item=" 120, 1.73, 1.56E-02, 3969, 38909,
               2.06E-04,2.52E-05,2.29E-01, 0.00328"/>
</complexType>
```

The supporting records for problems, methods and features are defined by users through the MyPYTHIA database manager interface. These supporting records, along with the imported performance data drive the data analysis, rule generation and recommendation processes. Figure 2



Select fields (at least one):

name
description
problempointers
methodpointers
featurepointers
perfarray
valuesarray

- Display 30 records per page
- Add search conditions (body of the "where" clause):
  [Documentation]
- Do a "query by example" (wildcard: "%")

Field	Type	Value
name	string	
description	blob	
problempointers	blob	
methodpointers	blob	
featurepointers	blob	

Figure 4. End-user interface for accessing the performance data.

shows the user interface for importing performance records into a sample MyPYTHIA database. Figures 3 and 4 show MyPYTHIA browsing and edit interfaces.

3.3. Data mining subsystems

Data mining encompasses the process of extracting and filtering performance data for statistical analysis, generating method profiles and ranking them, selecting and filtering data for pattern extraction and finally, populating the rule base of the recommender system. There are a variety of ways to configure data mining and our experience has been that different data mining algorithms are appropriate for particular types and forms of performance data. It is beyond the scope of this article to identify these selective superiorities; we refer the reader to [7–10] for details. We instead present the algorithmic underpinnings of the statistical analysis and pattern extraction stages of data mining.

3.3.1. Statistical analysis

The task of the analyzer is to rank a set of methods for a selected sample from the problem population, based on *a priori* specified performance criteria. This ranking (ordering) is a function of the user's computational objectives (error, memory, total execution time). The generation of this ordering is based on the performance curves or profiles such as error versus total elapsed time (on a log–log basis) for each method, machine, and problem triple. These profiles are generated by linear least squares approximations to raw performance data. MyPYTHIA runs the analyzer as a separate process, sending it an input file containing performance data and a set of parameters which specify the mode of operation for the analyzer. In MyPYTHIA, we have borrowed the approach presented in [14], where a non-parametric statistical technique is considered for the ranking of methods with respect to a given set of problem instances. The process for ranking the methods uses multiple comparisons and contrast estimators based on Friedman rank sums [27].

3.3.2. Pattern extraction algorithms

A variety of induction algorithms have been incorporated into MyPYTHIA using off-the-shelf freely available software such as MLC++ [28] and PROGOL [29]. In the current implementation, the user is expected to choose an appropriate mining algorithm from among several categories. These routines are I/O intensive and often polynomial in the space of parameters (of the mining algorithm). Currently, they are executed on a special server, though one possibility is to harness Grid resources for this purpose.

Clustering. An area where tremendous progress has been made in inductive learning is clustering—a fundamental procedure in pattern recognition that looks for regularities in training exemplars. By revealing associations between individuals of a population, it can be used to provide a compact representation of the input problem domain. The basic goal of clustering is to obtain a c-partition of the input data that exhibits categorically homogeneous subsets, where n is the number of training exemplars and $2 \leq c \leq n$. In MyPYTHIA, by representing problem instances as characteristic vectors, we can use clustering to reveal regularities in the underlying problem population. Recommendations of algorithms can be made by testing for membership of problems in given cluster(s) and choosing the algorithm that best satisfies performance criteria (for other problems in the same cluster).

Neural networks. More general forms of functional relationships can be modeled by neural networks [30], which can approximate any function to any required level of accuracy (perhaps with exponential increase in complexity). Neural networks use one or more layers of intermediate functional elements to model the dependence of an output signal(s) on given input parameters. The task of learning in this scenario is thus to determine the interconnection strengths and threshold biases (weights) that will result in acceptable approximation. While the general problem of determining weights has been shown to be NP-complete, neural networks have emerged as a valuable tool for pattern recognition and function approximation. In MyPYTHIA, neural networks have been used to associate problem characteristic features with method and machine parameters [31]. Given a training set, MyPYTHIA can functionally model the dependence of problem features on solution properties, which can lead to the recommendation of a suitable algorithm.

Inductive logic programming. While neural networks are attribute-value based techniques, more expressive schemes can be obtained by harnessing the representational power of logic (specifically, first-order logic). In this formalism, facts and measurements are represented as logical facts and data mining corresponds to forming an intensional rule that uses relational representations to model the dependence of a 'goal' predicate on certain input predicates (relations). This 'inductive logic programming' (ILP) approach [29] is one of the most popular in MyPYTHIA and with the aid of software such as PROGOL, it can be used to find patterns such as:

```
method(X,'FFT9') :- laplace(X), constcoeff(X).
```

This rule states that the `FFT9` method is best for problem `X` if `X` is Laplacian and has constant coefficients. Notice the easy comprehensibility of the rule which could be later used in diagnostics and what-if analyses. In addition, such rules can also be recursive, a feature which makes ILP amenable to automated program synthesis and discovery of complex relationships [29]. In addition, ILP allows the incorporation of *a priori* background knowledge, a necessary pre-requisite for mining in complex structured domains.

ILP systems typically take a database of positive examples, negative examples and background knowledge and attempt to construct a predicate logic formula (such as `method(X,Y)`) so that all (most) positive examples can be logically derived from the background knowledge and no (few) negative examples can be logically derived.

In the most general setting of first-order logic, relational induction as described above is undecidable. A first restriction to function-free horn clauses results in decidability (this is the form of programs used in the programming language PROLOG). Decidability here is with respect to induction; for deduction, first-order logic is semi-decidable. However, ILP is often prohibitively expensive and the practice in MyPYTHIA is to restrict the hypothesis space to a proper subset of first-order logic.

Other propositional representations. Various attribute-value learning systems like ID3 [32], C4.5 [33], IBL [34], which express the discovered knowledge in attribute-value languages, have also been incorporated into MyPYTHIA. These approaches have the representational ability of propositional logic. Propositional languages are limited and do not allow for representing complex structured objects and relations among objects or their components. The domain (background) knowledge that can be used in the learning process is also of a very restricted form, compared to ILP.

For all of these approaches, an intermediate filter converts required information from MyPYTHIA databases to the format required by each of the data mining packages. Recommendations are then made based on the methodology applicable in the particular learning context. For instance, with an ILP system, we attach to each rule all the positive (true) examples that are covered by the rule. After a rule has fired, we apply a nearest neighbor technique to identify the stored example that most closely matches (closest example) the user's example. After the closest example has been identified, we retrieve the performance data for this example and the recommended best method by the fired rule. The performance profiles for the method and the problem are consulted to provide estimates for the method's parameters and the performance indicators. If it is impossible for the system to satisfy the user's imposed constraints, then it uses the weights selected by the user to determine the best solution that satisfies approximately the constraints with respect to the indicated weights.

3.4. Recommendation and inference engine

The recommender system is the end user module of the MyPYTHIA system. It is a form of decision support system and is the only module in MyPYTHIA 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 the ILP approach in the pattern extraction module.

Each rule selected by the data mining program 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 first rule in the agenda is fired to determine the best algorithm for the problem the user specifies. Since each rule 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 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 default recommendation is provided. For some of the case studies presented here, the final step is the recommendation of a certain method as the best method to use in satisfying the given conditions. It also indicates what parameter(s) would be used to achieve the computational objectives imposed by the user.

3.5. User interface components of MyPYTHIA

Users access MyPYTHIA through a Web-based interface which provides: (i) user management and access authorization; (ii) a Web-enabled database interface to the underlying MySQL database; (iii) a guided selection process for identifying how performance data will be classified and retrieved for knowledge creation processing; and (iv) the recommender graphical interface.

When users access MyPYTHIA for the first time (http://shrine.cs.purdue.edu/pythia3), they must register for authorization and password assignment. When an authorized, registered user enters the MyPYTHIA environment with ID and password, a new database can be created or an already existing database can be selected. The MyPYTHIA database interface supports a full range of functionality for creating, editing, browsing, retrieving and importing MyPYTHIA records. Both graphical and SQL mechanisms are available for managing the stored data.

The specification of the feature-measure context is the starting point for the knowledge building process. MyPYTHIA supports both graphical and SQL methods for identifying which performance data will form the basis for analysis and rules generation. Various graphical selection processes are available, such as the following.

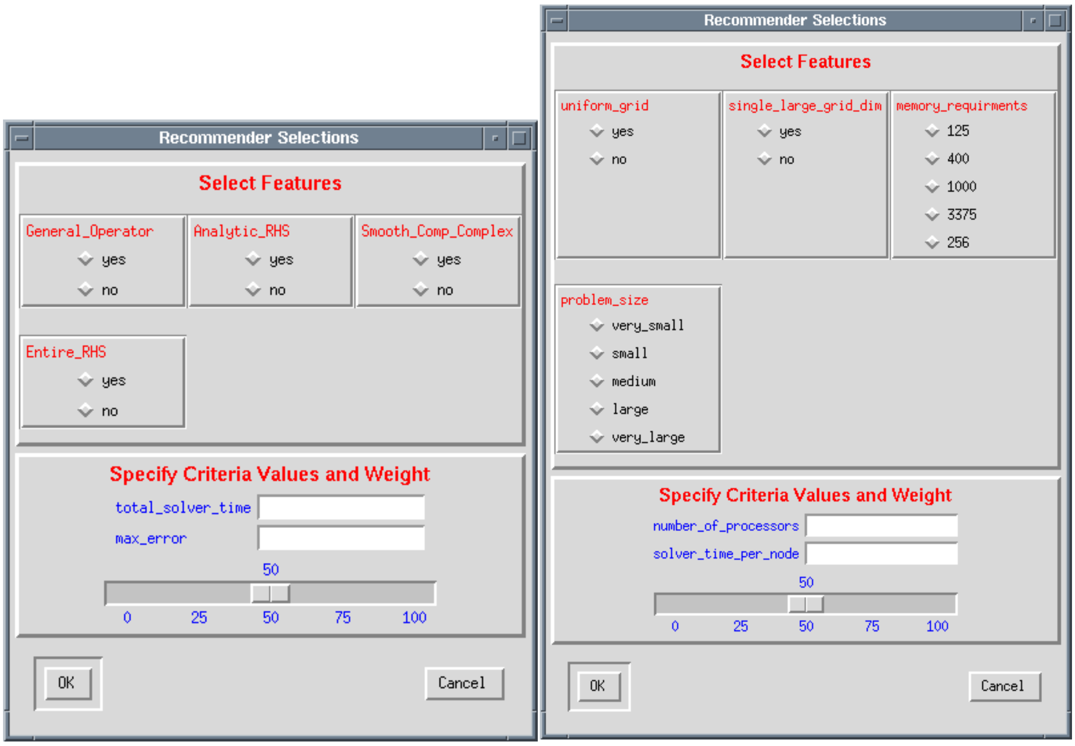


Figure 5. Recommender interfaces for two MyPYTHIA case studies: (left) PDE case study, (right) SWEEP3D application performance characterization.

- **Selection by features:** users can identify the problem and method features to be studied, and the system will select all performance records associated with problems and methods possessing those features.
- **Selection by problems and methods:** users can select the collection of all problems and methods to be considered in the study. The system will select all performance records naming the chosen problems and methods. Features associated with the selected problems and methods are considered as the basis for the rules generation process.
- **Selection by performance data:** users can select any subset of performance records to consider in the study. Features associated with the problems and methods named in the performance records are considered as the basis for rules generation.

Once the collection of performance records and corresponding features are established, the performance measures are selected. Users select from the available measures listed in the *perfarray* field of the performance record collection. The selected measures must be available for all performance records in the collection, since these measures apply to all analyzer runs. The measures are used to evaluate which methods are better than others, and the measures become the basis for the computational objectives which users specify when requesting software recommendations. The final phase of the feature-measure context specification is the grouping of performance records for analyzer runs. The information specified for the feature-measure context is saved into the context record, and is then used to guide the data retrieval for analysis and method ranking. Users can also browse the context record to realize the basic input to MyPYTHIA's inference data generation process.

The recommendation interface is created dynamically, using the inference context to identify the relevant features and measures. Examples of generated MyPYTHIA recommender interfaces for two application case studies are shown in Figure 5.

4. CURRENT APPLICATIONS COVERAGE OF MyPYTHIA

We now describe the chief application case studies that have used MyPYTHIA. One of these studies—involving PDE benchmarks—is described in greater detail in the next section.

4.1. Numerical quadrature

The recommendation problem addressed by this study is to select an method to evaluate a given integral $I = \int_a^b f(x)\,\mathrm{d}x$ so that relative error $\epsilon_r < \theta$ and N_i is minimized; where θ is an user-specified error requirement and N_i is the number of times $f(x)$ is evaluated in $[a, b]$ to yield the desired accuracy. For this testcase of MyPYTHIA, we have utilized 124 routines from the GAMS cross-index [35], where the category H2a is for the evaluation of one-dimensional integrals. A collection of 286 test problems and 124 methods were identified. The libraries from which the routines are obtained are QUADPACK, NAG, IMSL, PORT, SLATEC, JCAM and the collected methods of the ACM (TOMS). The test problems were selected so that they exhibit interesting or common features such as smoothness (or its absence), singularities, discontinuities, peaks and oscillation. Some of the functions were selected so that they satisfy the special considerations on which some methods are designed. For example, routine QDAWO requires that its argument contain a sine or a cosine. Most of the functions are parameterized, which generates families of integrands with similar features and characteristics—this aids in the generalization of the system. For each routine and each applicable integrand, experiments were conducted with varying requirements on the relative error accuracy ϵ_r. The number of accuracy levels was 10 and the strictest error requirement used was 10^{-8}.

Detailed results are presented elsewhere [36]. We briefly summarize that the information mined by our methodology corresponds to accepted general knowledge about numerical integration routines. It was observed, for example, that the adaptive methods use fewer function evaluations to achieve high-accuracy results than their non-adaptive counterparts; conversely, they use more evaluations to meet low-accuracy constraints. A high-accuracy adaptive method has been found to be more suitable for an oscillating integrand. This could possibly be due to the fact that in an oscillating function, subdivisions are spread over the entire domain of integration and hence a smaller number of subdivisions are required to achieve a fairly high degree of accuracy. Conversely, integrands with singularities or peaks are more amenable to low- and medium-accuracy adaptive routines. There are many more such observations, and we have reproduced only the most interesting here. Finally, this implementation has helped identify 'redundant' methods, i.e. methods which perform almost exactly the same for the test functions considered in this work. For example, the rules selecting the methods DPCHIA, DCSITG and DCSPQU contained the same antecedents. DPCHIA evaluates the given integral using piecewise cubic Hermite functions, DCSITG evaluates the integral of a cubic spline and DCSPQU also uses spline interpolation. One interpretation for this result is that quadrature methods using fitted functions often utilize similar mechanisms for obtaining data-dependent break points (for use in Hermite cubics or as knots in spline-based methods).

4.2. PDE software

The recommendation problem addressed by this case study is to select a method to solve $Lu = f$ on Ω, $Bu = g$ on $\partial\Omega$, such 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 two-dimensional space, and θ, T are performance criteria constraints. 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 to obtain three levels of accuracy—10^{-3}, 10^{-4} and 10^{-5}—is tabulated. We use linear least squares approximation to the profile data as this allows us to interpolate between the discrete values of the meshes to specify the size necessary to obtain a specified accuracy. We use the population in [37] of 56 linear second-order elliptic partial differential equations. The primary motivation for developing this population was to aid in the performance evaluation of numerical software. Forty-two of the PDEs from this population are parameterized which gives rise to a huge number of PDEs, numbering nearly 200. We use several features of these PDEs to aid in method selection. The principal characteristics used are those of the operator and right side (analytic, constant coefficients, nearly singular behavior, oscillatory, jump discontinuity, singular, peaked, singular derivatives, entire), boundary conditions (as being mixed, homogeneous, Dirichlet, Neumann, constant coefficients, etc.) and those of the domain (unit square, variable square, rectangle, etc.).

This case study is described in greater detail in Section 5. We briefly summarize the results here. The rules discovered confirm the statistically discovered conclusion in [38] that higher order methods are better for elliptic PDEs with singularities. They also confirm the general hypothesis that there is a strong correlation between the order of a method and its efficiency. There were several other interesting inferences drawn. The rule that had the maximum cover from the data was the one which stated that the nine-point fast Fourier transform (FFT) method is best for a PDE if the PDE has a Laplacian operator, homogeneous and Dirichlet boundary conditions and discontinuous derivatives on the right side. Finally, an approximate ordering was induced for the overall population. This gave rise to the ordering whose posited ranks correspond most closely to that for Poisson problems which formed the bulk of our population. In overall, the rules from this study performed best method recommendation for 100% of the cases.

4.3. Performance modeling of large scientific codes

Our final case study involved assessing performance models of Sweep3D, a complex ASCI benchmark for the solution of 3D time-independent, neutron particle transport equations [39]. Available in FORTRAN code in the public domain, it forms the kernel of an ASCI application and involves solving a group of time-independent discrete-ordinate (Sn), neutron particle transport equations on a XYZ Cartesian cube [40]. The XYZ geometry is represented by an IJK logically rectangular grid of cells. The angular dependence is handled by discrete angles with a spherical harmonics treatment for the scattering source. The solution involves two steps: the streaming operator is solved by sweeps for each angle and the scattering operator is solved iteratively.

The first recommendation goal is to facilitate the selection of computational parameters of the application/machine pair in order to achieve pre-specified performance goals. For example, in the context of SWEEP3D application, it can be used to obtain the parameters of the underlying algorithm (e.g. grid size, spacing, scattering order, angles, k-blocking factor, convergence test), system (e.g. I/O switches) and the target machine (e.g. processor configuration). Thus, in the case of a fixed application/architecture pair where the parameters are *a priori* known and there are performance data for various values of the parameters, then MyPYTHIA can be used: (i) to identify the best pair from the given performance data for different levels of performance; (ii) to identify the best pair for a specified performance objective (i.e. execution time, cost); and (iii) to identify the best pair including an estimate of its parameters by specifying certain *a priori* selected features of the pair.

The second recommendation goal is to automatically classify the various designs/implementations based on their performance data by using well-designed benchmarks with specific features and with respect to range values of some performance indicators or features. Here, the user might want to see a classification of the data with respect to performance levels or certain features (i.e. architectural model, shared memory, distributed memory, etc.)

The third goal is to predict the performance of a conceptual design by comparing it with the performance of existing 'similar' designs/implementations and assuming some user defined computational goals and design features.

5. A COMPLETE MyPYTHIA APPLICATION

We present a case study that demonstrates how a knowledge engineer can use MyPYTHIA to discover and evaluate the pertinent knowledge needed to solve the software selection problem for a certain population of PDEs.

For this case study we have selected 30 benchmark PDE problems from [38] for which the solutions exhibit singularities or near singularities. Each of the 30 problems is parameterized by three sets of equation coefficients and five different grid sizes, resulting in a total population of $30 \times 3 \times 5 = 450$ unique PDE problems. The seven methods to be evaluated by this case study of MyPYTHIA for solving the problems are listed in Table I. Each of the 450 problems represents a subset of the feature space which contains 64 features. These features have been categorized into six classes: features for the PDE operator, the PDE, the right-hand side, the solution, the boundary conditions and the domain. Each class of features is shown in a separate table: Table II lists the operator features, Table III lists the PDE features, Table IV lists the right-hand side features, Table V lists the boundary condition features, Table VI lists the solution features and Table VII lists the domain features.

As described in a previous section, the performance data must be generated by the PDE expert external to the MyPYTHIA system, and then imported into MyPYTHIA in an XML-like format according to the underlying database schema for the MyPYTHIA performance records. In the rest

Table I. Numerical methods to be evaluated in MyPYTHIA.

Program name	Method implemented
5-POINT STAR	Ordinary second-order finite differences; Gauss elimination
P3C1-COLLOCATION	Fourth-order collocation with Hermite bicubics; Gauss elimination
DYAKANOV CG	Ordinary second-order finite differences; iteration with generalized marching method and conjugate gradient method
DYAKANOV CG-4	Dyakanov CG with Richardson extrapolation to achieve fourth-order accuracy
FFT9(IORDER=2)	Ordinary second-order finite differences and fast Fourier transform
FFT9(IORDER=4)	Fourth-order finite difference method and fast Fourier transform
FFT9(IORDER=6)	Sixth-order finite difference method and fast Fourier transform

Table II. Symbolic names and descriptions of features for the operator.

Feature name	Description
opConstCoeff	Constant coefficient
opGeneral	General operator
opHelmholtz	$AU_{xx} + CU_{yy} + fU = 0$
opLaplace	$U_{xx} + U_{yy}(+U_{zz}) = f$
opSelfAdjoint	$(PU_x)_x + (QU_y)_y + (RU_z)_z + Sf = g$
opPoisson	$U_{xx} + U_{yy}(+U_{zz}) = 0$
opSmoConst	Constant operator smoothness

of this section, we present some of the important principles related to the modeling and analysis of this domain using MyPYTHIA.

The ‘feature-measure context’ for this case study contains logical references to 30 different analyzer ranking runs. Each analyzer run considers one of the benchmark PDEs across the three coefficient parameter sets and five grid size parameters. Each run addresses the two measures selected by the PDE expert for analyzing the effectiveness and efficiency of the seven numerical methods: accuracy (error in the computed solution) and elapsed CPU time in solving the problem. As each analyzer run is initiated, the values for these two measures are retrieved from the performance database records for the parameterized variations of a specific benchmark problem. Thus, $(3 \times 5) \times 7 = 105$ problem-method instances of elapsed time and error are extracted for each run, and $450 \times 7 = 3150$ problem-method instances are accessed for the complete ranking analysis.

The full list of features to be used in the knowledge discovery process is also identified in the ‘feature-measure context’. We have selected 64 features covering the entire suite of classes for the operator, PDE, right-hand side, solution, boundary conditions and domain. All of the features are

Table III. Symbolic names and descriptions of features for the PDE.

Feature name	Description
pdeAnalytic	PDE problem is analytic
pdeEntire	PDE problem is entire
pdeConstCoeff	PDE problem has constant coefficients
pdeNearlySingular	PDE problem is nearly singular
pdeOscillatory	PDE problem is oscillatory
pdeJump	PDE problem has a jump discontinuity
pdeSingular	PDE problem is singular (infinite)
pdePeaked	PDE problem is parameterized or peaked
pdeCompComplex	PDE problem is computationally complex
pdeSingDeriv	PDE problem has singular derivatives
pdeSmoConst	Smoothness of PDE problem is constant
pdeSmoEntire	Smoothness of PDE problem is entire
pdeSmoAnalytic	Smoothness of PDE problem is analytic
pdeSmoDiscDeriv	Smoothness of PDE problem has discontinuous derivatives
pdeSmoSingular	Smoothness of PDE problem is singular
pdeSmoOscillatory	Smoothness of PDE problem is oscillatory
pdeSmoCompComp	Smoothness of PDE problem is computationally complicated

Table IV. Symbolic names and descriptions of features for the right-hand side.

Feature name	Description
rhsEntire	Right-hand side is entire
rhsAnalytic	Right-hand side is analytic
rhsSingular	Right-hand side is singular (infinite)
rhsSingDeriv	Right-hand side has singular derivatives
rhsConstCoeff	Right-hand side has constant coefficients
rhsNearlySingular	Right-hand side is nearly singular
rhsPeaked	Right-hand side is parameterized or peaked
rhsOscillatory	Right-hand side is oscillatory
rhsHomogeneous	Right-hand side is homogeneous
rhsCompComplex	Right-hand side is computationally complex
rhsSmoConst	Smoothness of right-hand side is constant
rhsSmoEntire	Smoothness of right-hand side is entire
rhsSmoAnalytic	Smoothness of right-hand side is analytic
rhsSmoDiscDeriv	Smoothness of right-hand side has discontinuous derivatives
rhsSmoSingular	Smoothness of right-hand side is singular
rhsSmoOscillatory	Smoothness of right-hand side is oscillatory
rhsSmoPeaked	Smoothness of right-hand side is peaked
rhsSmoCompComp	Smoothness of right-hand side is computationally complicated

Table V. Symbolic names and descriptions of features for the boundary condition.

Feature name	Description
bcHomogeneous	$U = 0$ on boundary
bcDirichlet	$AU = f$ on boundary
bcNeumann	$BU_n = f$ on some boundary
bcMixed	$AU + BU_n = f$ on some boundary
bcConstCoeff	Boundary condition has constant coefficients
bcVarCoeff	Boundary condition has non-constant coefficients

Table VI. Symbolic names and descriptions of features for the solution.

Feature name	Description
solEntire	Solution is entire
solAnalytic	Solution is analytic
solSingular	Solution is singular (infinite)
solSingDeriv	Solution has singular derivatives
solOscillatory	Solution is oscillatory
solWaveFront	Solution has a wave front
solBoundLayer	Solution has a boundary layer
solPeaked	Solution is parameterized or peaked
solUnknown	Solution is unknown
solNearSingular	Solution is nearly singular
solVarSmooth	Solution has variable smoothness
solSmoEntire	Smoothness of the solution is entire
solSmoAnalytic	Smoothness of the solution is analytic
solSmoSingDeriv	Smoothness of the solution has singular derivatives
solSmoOscillatory	Smoothness of the solution is oscillatory
solSmoWaveFront	Smoothness of the solution has wave front
solSmoDiscDeriv	Smoothness of the solution has discontinuous derivatives
solSmoSingular	Smoothness of the solution is singular
solSmoBoundLayer	Smoothness of the solution has a boundary layer
solSmoPeak	Smoothness of the solution has a peak
solSmoTabled	Smoothness of the solution is tabled

Table VII. Symbolic names and description of features for the domain.

Feature name	Description
domVariableSquare	$(a, b) \times (a + x, b + x)$, where x can vary
domUnitSquare	$(0, 1) \times (0, 1)$
domSquare	$(a, b) \times (a + c, b + c)$ where c is a constant
domRectangle	$(a, b) \times (c, d)$
domVariableRectangle	$(x, y) \times (z, u)$

```
New best node (1 evals) #0[]: error: 70.00% +- 10.48% (0.00% - 100.00%).
  Test Set: 70.00% +- 8.51% [52.12% - 83.34%].  Bias: 0.00% cost: 10
   complexity: 0
...................................................................
New best node (68 evals) #66[1, 14, 21]: error: 36.67% +- 7.78%
    (0.00% - 66.67%).
  Test Set: 33.33% +- 8.75% [19.23% - 51.22%].  Bias: 3.33% cost: 10
   complexity: 3
................................................................
New best node (132 evals) #86[1, 14, 19, 21]: error: 30.00% +- 5.98%
    (0.00% - 66.67%).
  Test Set: 26.67% +- 8.21% [14.18% - 44.45%].  Bias: 3.33% cost: 10
   complexity: 4
```

Figure 6. Output generated by the feature subset selection process for the feature space of the PDEs with (near) singularities case study.

identified by symbolic names and are associated with Boolean values, i.e. the 'possible values' assumed by these features listed in the FEATURE record is true/false. The Boolean value indicates whether the named feature belongs to the PDE problem or not.

The 'feature-measure context' controls the entire process of analyzer ranking and rules generation. The analyzer builds performance profiles for each numerical method applied to each unique PDE problem, maintaining the resulting profile data in the database. The profiles are used by the statistical analyzer to rank the methods for each of the 30 benchmarks, in each case using a rank sum analysis applied to the rankings across the parameter variations of a given benchmark problem. After the method ranking for each benchmark record has been determined, the benchmark's features to be considered (according to the 'feature-measure context') are retrieved and a predicate is generated attaching the method ranked first as the `best` method for the corresponding benchmark problem.

The predicates generated in this way, along with the predicates stating the features of each benchmark, are saved. Based on the knowledge discovery process selected by the domain expert, different filters are used by the system to transform the predicate information to an appropriate format for the knowledge discovery tool. MyPYTHIA system utilizes many tools for the knowledge discovery phase, but in this case study we discuss only the PROGOL system, because it is the only one that is currently interfaced with MyPYTHIA's Web Recommender system.

PROGOL [29] induces logical descriptions of relations in a subset of first order predicate logic. This means that the induction process is extremely expensive. For this reason, the concept space to be explored by PROGOL must be well suited to the domain under consideration. This becomes obvious if we consider that PROGOL immediately fails to induce any rule for the 64-dimensional space of our case study due to resource limits. A solution to this problem, then, is to reduce the high-dimensional space of this study. In order to accomplish this, we make use of the feature subset selection utility that is offered by the `MLC++` library. The feature subset selection is a wrapper inducer (in the sense that it wraps around a regular inducer) and selects a good subset of features for improved accuracy performance. This topic has been fully covered in [28].

Table VIII. Confusion matrix for the wrapped ID3 inducer.

(a)	(b)	(c)	(d)	(e)	(f)	(g)	Classes
4.00	0.00	0.00	0.00	0.00	0.00	0.00	(a) 5-point_star_and_band_ge
1.00	3.00	0.00	0.00	0.00	0.00	0.00	(b) hermite_collocation_and_band_ge
0.00	0.00	0.00	0.00	0.00	0.00	0.00	(c) dyakanov-cg
0.00	0.00	0.00	6.00	0.00	1.00	2.00	(d) dyakanov-cg4
0.00	0.00	0.00	1.00	0.00	0.00	0.00	(e) fft_9_point_order_2
0.00	0.00	0.00	0.00	0.00	5.00	1.00	(f) fft_9_point_order_4
0.00	0.00	0.00	1.00	0.00	1.00	4.00	(g) fft_9_point_order_6

```
best_method(p2,fft_9_point_order_2).
best_method(p9,fft_9_point_order_6).
best_method(p19,fft_9_point_order_4).
best_method(A,5-point_star_and_band_ge) :- opGeneral_yes(A).
best_method(A,hermite_collocation_and_band_ge)
    :- pdeSmoCompComp_yes(A). f=278,p=40,n=1,h=0
best_method(A,fft_9_point_order_4) :- rhsEntire_yes(A).
best_method(A,dyakanov-cg4) :- opGeneral_no(A),
       pdeSmoCompComp_no(A).
best_method(A,fft_9_point_order_6) :- opGeneral_no(A),
       pdeSmoCompComp_no(A), rhsEntire_no(A).
[Total number of clauses = 8]
```

Figure 7. Rule set generated by PROGOL rule generation program along with coverage statistics.

By applying this methodology to the feature space in this case study we end up with only four features that are chosen by the feature subset selection process as the most important for the identification of the classes. The results from the feature subset selection process are illustrated in Figure 6.

The most important information to be acquired from the feature subset selection process is the list of numbers inside the square brackets which specify the index of the features considered as the most important by the program, along with an estimation of the error attained for each one of the subsets of features selected as best each time. The indices of the selected features are: 1, 14, 19 and 21. The indexing scheme assumed by the method considers the first feature as having index 0. For this reason the selected features are: opGeneral, pdeSmoCompComp, rhsAnalytic, and rhsEntire. We also observe that the error falls from 70% to 30%. The wrapped inducer in this case is the ID3 inducer. The final error for the ID3 inducer using a 10-fold cross validation is only 26.667%. Table VIII shows the confusion matrix for the problem instances used in this case study.

The rule set produced by PROGOL along with coverage statistics of each non-trivial (compressed) rule is shown in Figure 7. Each logical rule generated by PROGOL is associated with an information compression factor measuring the generalization accuracy of the rule. Its simple formula

```
[Partial accuracy= 149/210] Contingency table=
                      _________A_________~A
                     P|        24|        55|         79
                      |(   11.3)|(   67.7)|
                    ~P|         6|       125|        131
                      |(   18.7)|(  112.3)|
                       ~~~~~~~~~~~~~~~~~~~~
                              30       180         210
[Overall accuracy= 70.95% +/- 3.13%]
[Chi-square = 24.72] [Without Yates correction = 26.79]
[Chi-square probability = 0.0000]
```

Figure 8. Contingency matrix representing accuracy information and statistics for the rule set generated for the PDE case study.

is $f = p - (c + n + h)$ where p and n are the number of positive and negative examples respectively covered by a specific rule, c is the number of predicates in the body of the clause/rule, and h is the number of predicates which must be added to the clause to complete the relationship between variables in the head of the clause. The information compression factor is used for ordering the rules in the rule base in a decreasing order and values for each one of these rule parameters are shown in Figure 7.

The confusion matrix by using the leave-one-out accuracy estimation technique for the rule set shown in Figure 7 is illustrated in Figure 8. The column labeled A in the contingency table represents the examples that are known to belong to the set of positive examples while the column labeled Ã consists of negative examples. The positive and negative examples, in sequel, are separated into two different categories: the category P, which includes the number of examples that they were predicted as positive and P̃, the examples that were predicted as negative examples. Additional information related to confidence and overall accuracy is also shown in Figure 8.

6. DISCUSSION

As a recommendation portal, MyPYTHIA enables the complete automation, generation and maintenance of domain specific recommender systems, but does not neglect human intervention throughout the process. Our modularized system enables a data-driven approach to Grid computing, by clearly defining the interfaces at which human support or iteration can take place. Our open database architecture can easily scale with the quantity and variety of information that will be generated by recommendation studies. By facilitating efficient storage and retrieval for data-intensive computing in scientific simulations, MyPYTHIA serves as a complementary approach to other projects in Grid computing that provide decision-support in the form of data management facilities [42–45]. MyPYTHIA's system design and architecture are novel in that they formalize the scientific software recommendation problem and point in the direction of connecting such services with Grid information services and other knowledge-based portals.

Our long-term goal is to gain acceptance for the data-driven methodology presented here. The implementations described have assumed that the methodology of scientific computation is well established; however, MyPYTHIA can be viewed more basically as an approach to managing and harnessing large simulation runs. For domains that lack a perfect theory for selecting algorithms or problem solving strategies, MyPYTHIA's approach will be especially pertinent.

One of our directions of future research is to use MyPYTHIA as an information service for large-scale application composition. Preliminary results are described in [39]. This will require renewed emphasis on performance modeling of scientific codes on the Grid. Another direction of future work is to exploit Grid resources for scheduling the data mining runs required by MyPYTHIA. This will also enable us to study the scalability of our current MyPYTHIA implementation and its extension to a distributed data mining setting.

ACKNOWLEDGEMENTS

This work was supported in part by NSF grants CDA 9123502, EIA-9974956, EIA-9984317, PRF 6902851, DARPA grant N66001-97-C-8533 (Navy), DOE LG-6982, DARPA under ARO grant DAAH04-94-G-0010 and the Purdue Research Foundation.

REFERENCES

1. Casanova H, Dongarra J. NetSolve: A network server for solving computational science problems. *International Journal of Supercomputer Applications and High Performance Computing* 1997; **11**(3):212–223.
2. Chandy KM. The scientist's infosphere. *IEEE Computational Science and Engineering* 1996; **3**(2):43–44.
3. Foster I, Kesselman C. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications and High Performance Computing* 1997; **11**(2):115–128.
4. Foster I, Kesselman C. The Grid: Blueprint for a new computing infrastructure. Morgan Kaufmann: San Mateo, CA, 1998.
5. Grimshaw AS, Ferrari A, Lindahl G, Holcomb K. MetaSystems. *Communications of the ACM* 1998; **41**(11):46–55.
6. Grimshaw AS, Weismann JB, West EA, Loyot Jr EC. MetaSystems: An approach combining parallel processing and heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing* 1994; **21**(3):257–270.
7. Houstis EN, Catlin AC, Rice JR, Verykios VS, Ramakrishnan N, Houstis CE. PYTHIA-II: A knowledge/database system for managing performance data and recommending scientific software. *ACM Transactions on Mathematical Software* 2000; **26**(2):227–253.
8. Ramakrishnan N. Recommender systems for problem solving environments. *PhD Thesis*, Department of Computer Sciences, Purdue University, 1997.
9. Ramakrishnan N, Ribbens CJ. Mining and visualizing recommendation spaces for elliptic PDEs with continuous attributes. *ACM Transactions on Mathematical Software* 2000; **26**(2):254–273.
10. Verykios VS. Knowledge discovery in scientific databases. *PhD Thesis*, Department of Computer Sciences, Purdue University, 1999.
11. Verykios VS, Houstis EN, Rice JR. Mining the performance of complex systems. *Proceedings IEEE International Conference on Information Intelligence and Systems*. IEEE Computer Society Press, 1999; 606–612.
12. Verykios VS, Houstis EN, Rice JR. A knowledge discovery methodology for the performance evaluation of scientific software. *Journal of Neural, Parallel and Scientific Computations* 2000; **8**(2):115–132.
13. Verykios VS, Houstis EN, Tsoukalas LH, Pantazopoulos KN. Automating the analysis of option pricing algorithms through intelligent knowledge acquisition approaches. *IEEE Transactions on Systems, Man and Cybernetics: Part A* 2001; **31**(6):573–586.
14. Weerawarana S, Houstis EN, Rice JR, Joshi A, Houstis CE. PYTHIA: A knowledge based system to select scientific algorithms. *ACM Transactions on Mathematical Software* 1996; **22**:447–468.
15. Gannon D, Bramley R, Stuckey T, Villacis J, Balasubramanian J, Akman E, Breg F, Diwan S, Govindaraju M. The linear system analyzer. *Enabling Technologies for Computational Science*, Houstis EN, Rice JR, Gallopoulos E, Bramley R (eds.). Kluwer: Dordrecht, 2000; 123–134.
16. Markus S, Weerawarana S, Houstis EN, Rice JR. Scientific computing via the World Wide Web: The Net PELLPACK PSE server. *IEEE Computational Science and Engineering* 1997; **4**(3):43–51.

17. Ferris MC, Mesnier MP, Moré JJ. NEOS and Condor: Solving optimization problems over the Internet. *ACM Transactions on Mathematical Software* 2000; **26**(1):1–18.
18. Rice JR. A metalgorithm for adaptive quadrature. *Journal of the ACM* 1973; **22**(1):61–82.
19. Rice JR. The algorithm selection problem. *Advances in Computers* 1976; **15**:65–118.
20. Addison CA, Enright WH, Gaffney PW, Gladwell I, Hanson PM. Algorithm 687: A decision tree for the numerical solution of initial value ordinary differential equations. *ACM Transactions on Mathematical Software* 1991; **17**(1):1–10.
21. LaRose B. The development and implementation of a performance database server. *Technical Report CS-93-195*, University of Tennessee, August 1993.
22. Houstis EN, Verykios VS, Catlin AC, Ramakrishnan N, Rice JR. A data mining environment for modeling the performance of scientific software. *Enabling Technologies for Computational Science: Frameworks, Middleware and Environments*, ch. 21. Kluwer: Dordrecht, 2000; 261–271.
23. Apache Webserver. http://www.apache.org/ [2001].
24. PHP. http://www.php.net/ [2001].
25. MySQL. http://www.mysql.com/ [2001].
26. PhpMyAdmin. http://www.phpwizard.net/projects/phpMyAdmin/ [2001].
27. Hollander M, Wolfe DA. *Nonparametric Statistical Methods*. John Wiley & Sons, 1973.
28. Kohavi R. Wrappers for performance enhancement and oblivious decision graphs. *PhD Thesis*, Department of Computer Science, Stanford University, 1995.
29. Bratko I, Muggleton S. Applications of inductive logic programming. *Communications of the ACM* 1995; **38**(11):65–70.
30. Jordan MI, Bishop CM. Neural networks. *The Computer Science and Engineering Handbook*, Tucker A (ed.). CRC Press: Boca Raton, FL, 1997; 536–556.
31. Joshi A, Weerawarana S, Ramakrishnan N, Houstis EN, Rice JR. Neuro-fuzzy support for PSEs: A step toward the automated solution of PDEs. *IEEE Computational Science and Engineering* 1996; **3**(1):44–56.
32. Quinlan JR. Induction of decision trees. *Machine Learning* 1986; **1**(1):81–106.
33. Quinlan JR. *C4.5: Programs for Machine Learning*. Morgan Kaufmann: San Mateo, CA, 1993.
34. Aha DW, Kibler D, Albert MK. Instance-based learning algorithms. *Machine Learning* 1991; **6**(1):37–66.
35. Boisvert RF. The NIST guide to available mathematical software. http://gams.nist.gov [1996].
36. Ramakrishnan N, Rice JR, Houstis EN. GAUSS: An online algorithm selection system for numerical quadrature. *Advances in Engineering Software* 2001; **33**(1):27–36.
37. Rice JR, Houstis EN, Dyksen WR. A population of linear, second order, elliptic partial differential equations on rectangular domains, part I. *Mathematics of Computation* 1981; **36**:475–484.
38. Houstis EN, Rice JR. High order methods for elliptic partial differential equations with singularities. *International Journal for Numerical Methods in Engineering* 1982; **18**:737–754.
39. Adve VS, Bagrodia R, Browne JS, Deelman E, Dube A, Houstis EN, Rice JR, Sakellariou R, Sundaram-Stukel DJ, Teller PJ, Vernon MK. POEMS: End-to-end performance design of large parallel adaptive computational systems. *IEEE Transactions on Software Engineering* 2000; **26**(11):1027–1048.
40. Koch KR, Baker RS, Alcouffe RE. Solution of the first-order form of the 3-D discrete ordinates equation on a massively parallel processor. *Transactions of the American Nuclear Society* 1992; **65**:198–199.
41. Foster I, Insley J, von Laszewski G, Kesselman C, Thiebaux M. Distance visualization: Data exploration on the Grid. *IEEE Computer* 1999; **32**(12):36–43.
42. Karin S, Graham S. The high performance computing continuum. *Communications of the ACM* 1998; **41**(11):32–35.
43. Moore RW, Baru C, Marciano R, Rajasekar A, Wan M. Data-intensive computing. *The Grid: Blueprint for a New Computing Infrastructure*, Kesselman C, Foster I (eds.). Morgan Kaufmann: San Mateo, CA, 1998; 105–129.
44. Moore RW, Prince TA, Ellisman M. Data-intensive computing and digital libraries. *Communications of the ACM* 1998; **41**(11):56–62.
45. Ramakrishnan N, Grama AY. Mining scientific data. *Advances in Computers* 2001; **55**:119–169.