, , 1–8 ()

Design of the PYTHIA Recommender System

NAREN RAMAKRISHNAN naren@cs.vt.edu
Dept. of Computer Science, Virginia Tech

VASSILIOS S. VERYKIOS verykios@cs.purdue.edu
ELIAS N. HOUSTIS enh@cs.purdue.edu
JOHN R. RICE jrr@cs.purdue.edu
Dept. of Computer Sciences, Purdue University

Editor:

Abstract. In this paper, we outline the design of an algorithm recommender system (PYTHIA). Given a problem in (a specific domain of) scientific computing and performance criteria constraints on its solution, PYTHIA selects the best (or nearly best) algorithm to solve it. It uses a combination of inductive logic programming, statistical analysis, and a database of algorithm performance data to map a feature-based representation of problems (plus performance objectives) to appropriate algorithms. Currently, PYTHIA successfully recommends algorithms for domains such as numerical quadrature, elliptic partial differential equations, domain decomposition and linear algebra.

Keywords: Recommender Systems, Data Mining, Knowledge Discovery, Scientific Computing.

1. Introduction

A recommender system provides automatic advice on selections/choices necessary to optimize its user's performance objectives. The primary motivation for recommender systems (originally called collaborative filtering systems) is to reduce information overload. Examples are advanced web search engines, book and movie recommenders, social network filters, and scientific software selectors. More sophisticated applications abound in the areas of Geographic Information Systems (GIS), traffic regulation, user interfaces and problem solving environments (Kautz, 1998). Building recommenders is now a very fertile business activity due, in part, to the success of electronic commerce and intelligent internet systems. Several companies, including Net Perceptions, Imana Inc., Autonomy, AT&T, Relevance Technologies, Magenta Communications, Firefly specialize in 'personalization' technology and provide custom-made recommender systems for client specifications.

The focus of this paper is, however, a 'scientific recommender system' (PYTHIA) (Ramakrishnan, 1997) whose users are scientists/engineers trying to locate software module(s) appropriate to their needs. Given a scientific problem instance and performance criteria constraints on its solution, PYTHIA recommends an algorithm, estimates its parameters, and identifies a location on the web where a software module implementing the algorithm can be obtained. The current implementations of PYTHIA address the domains of numerical quadrature, elliptic partial differential equations (PDEs) (Ramakrishnan, 1997), domain decomposi-

PROBLEM	$(w\,u_x)_x + (w\,u_y)_y = 1$, where $w = \begin{cases} \alpha, & \text{if } 0 \le x, y \le 1 \\ 1, & \text{otherwise.} \end{cases}$
DOMAIN	$[-1,1] \times [-1,1]$
BC	$u = 0$
OPERATOR	Self–adjoint, discontinuous coefficients
RIGHT SIDE	Constant
BOUNDARY CONDITIONS	Dirichlet, homogeneous
SOLUTION	True solution unknown. Approximate solutions available for $\alpha = 1$, 10, 100. Strong wave fronts for $\alpha \gg 1$.

Figure 1. An example partial differential equation for which an algorithm and associated parameters have to be determined.

tion (Verykios, 1999), and linear algebra. Assume that we would like to solve the problem presented in Fig. 1 under the constraints that the relative error is less than 1.0×10^{-5}, time to solve the PDE is less than 600 seconds (timing measured on a Sun SPARCstation 20) and place equal emphasis on achieving both the time and the error bounds. PYTHIA's recommendation would be: *Use second order Dyakunov conjugate gradient finite differences with* 17 *grid lines. Error Estimate:* 1.00331×10^{-5}. *Time Estimate:* 8.353 *minutes. Confidence:* 0.85. *Software available at:* `http://math.nist.gov/cgi-bin/gams-serve/list-module-components/ELLPACK/12/13060.html`.

PYTHIA's basis for recommendations is a rule-bank mined by spooling performance trace data for various algorithms on benchmark scientific problems and correlating variations in performance to specific characteristics of the problem input. In addition, it interfaces with the GAMS software repository on the web[1] where implementations of the recommended algorithm can be down-loaded. We refer the interested reader to (Drashansky et.al., 1999) for more details about this interface. In this paper, we specifically concentrate on the data mining methodology utilized for the design of PYTHIA.

2. PYTHIA's Methodology

The algorithm/software/resource selection problem, detailed above, is non-trivial, even for specific domains of scientific software. In particular, the space of applicable algorithms for specific problem subclasses is inherently large, complex, ill–understood and often, intractable by brute–force means. Depending on the way the problem is (re)presented, the space of applicable algorithms changes; some of the best algorithms sacrifice generality for performance and have specially customized data structures and routines fine tuned for particular problems or their reformulations. Moreover, there is an inherent uncertainty in interpreting and assessing the performance measures of a particular algorithm for a particular problem. Different implementations of an algorithm produce substantially large variations in performance measures that render relying on purely analytic estimates impractical.

PYTHIA's design borrows heavily from a performance evaluation methodology first formulated in (Boisvert, 1979). Assuming a 'densely' distributed set of benchmark problems from the targeted application domain, PYTHIA uses a four-pronged strategy: feature determination of benchmark problems, performance evaluation of scientific software, statistical analysis of performance data, and the (automatic) knowledge discovery of recommendation rules from software performance[2] (Fayyad et.al., 1996, Matheus et.al., 1993). Data is collected (or is sometimes readily available) from experiments in the domain, mined for recommendations and a recommender system is built according to a conceptual schema of the problem domain. It is then put into production mode in the field and its efficacy is evaluated.

PYTHIA's software architecture is both flexible (allowing extension to newer domains) and scalable (providing a variety of options to the knowledge engineer for mining data, while storage and retrieval issues are handled by an integrated DBMS). PYTHIA works in conjunction with a host problem solving environment (PELLPACK) (Houstis, et. al., 1998) [3] and provides layered subsystems for problem definition, method definition, experiment management, performance data collection, statistical analysis, knowledge discovery (for recommendation rules) and an inference engine. We now describe these issues in the context of algorithm recommendation for PDEs.

3. Design Issues and Decisions

3.1. Database Design and Experiment Management

In order to facilitate the storage and execution of the experiments, the input specification of a PDE problem is decomposed into a set of database tables. There is a one-to-one mapping between the components of the problem specification and the basic entities in PYTHIA's database schema. Features of problem components are also modeled by other tables and tables representing relations are used for designating the constraints in associating features with basic entities. Experiments are also managed by yet another table, in the sense that each experiment record holds all the necessary information for executing a collection of problems in a specific system. PYTHIA currently communicates with the host PELLPACK system via I/O files. In order to avoid redundancy in the implementation, an experiment record uses foreign keys for representing the logical connections with the basic records. Tables that store performance data are used for saving selected parts from the output files produced by running these experiments.

It is important to mention that the implementation of the schema does not require any advanced techniques to be provided by the database management system. The relational model, along with some extensions available in the Postgres95 Object/Relational DBMS, used in our system, can adequately handle all the required functionality as well as extensibility issues. The database tables are accessed by using simple SQL queries, or by embedding SQL queries in the Tcl scripting language. The 'libpgtcl' package is a new front-end library that supports Tcl-based clients to the Postgres95 database. The data collection process has been imple-

mented by using the Perl language, and it includes the extraction of information from specific parts of the output generated by running the experiments, as well as storage of this information in the performance table(s). The graphical user interface is built in Tcl/Tk. In addition, the storage manager of Postgres95 helps in recording archival information into a spooling mechanism that can be efficiently utilized for experiment management.

3.2. Performance Analysis of Algorithms

The first step to solving a PDE is to replace the continuous problem by a discrete set of equations. This discretization step involves, typically, placing a grid on the continuous domain to approximate it. Let us assume that all PDEs are solved on a uniform square N by N grid. The logarithm of the total execution time increases linearly with the logarithm of N, and so the slope of the total execution time versus N (on a log-log chart) is the primary measure of performance. This implies that the algorithm with the smallest slope is the most efficient asymptotically, as N increases. An algorithm's performance profile is thus, the slope obtained by a least-squares approximation to the data points of (total execution time, grid size) on a log-log base for a certain sequence of grid sizes.

The algorithms' performance profiles are ranked by a non-parametric statistical technique — the Friedman, Kendall and Babington-Smith test (Hollander and Wolfe, 1973) — for testing the hypothesis that the PDE algorithms applied to a problem instance are not equally effective and thus there is an ordering/ranking among them. In order to test this hypothesis, the algorithms applied to every problem are ranked, in such a way that the most efficient method (or else the method with the smallest slope) gets a rank of one. After computing the rankings for several problem variations (e.g., using different instantiations of a problem's parameter), PYTHIA computes the average ranking of each method with respect to each problem. By computing a statistic related to the rankings produced, and comparing it with the chi-square statistic corresponding to a certain number of degrees of freedom and significance, it can estimate whether the algorithms differ with respect to performance.

3.3. Mining Recommendation Rules

While this information provides the ranking for different algorithm instances on a given problem instance, generalization across the problem space is achieved by inductive logic programming (ILP) (Dzeroski, 1996) performed with respect to the feature space of the problems (The reasons why a relational encoding is appropriate are detailed in (Ramakrishnan, 1997)). We found ILP to be prohibitively expensive and controlled the complexity of induction by several means — we first use domain specific restrictions for the management of recommendation spaces. For example, PYTHIA uses both syntactic and semantic restrictions on the nature of the induced recommendations. An example of a syntactic restriction is that a PDE algorithm consists of a discretizer, indexer and solver (chained in that order). A different

permutation of algorithm parts does not make syntactic sense[4]. An example of a semantic restriction is consistency checks between algorithms and their inputs. For example, a Dyakunov algorithm assumes that its input is in 'self-adjoint form', so inducing a more general rule will not be fruitful. Moreover, since the software architecture of PYTHIA is augmented with a natural database query interface, we utilize this aspect to provide meta-level patterns for rule generation. An example of a meta-rule is that generalization across problems should be performed with respect to a given accuracy constraint. For example, we would like to induce a rule that infers the best algorithm to use, for a given problem instance and user specified performance criteria. This is because generalizing across the space of constraints is not fruitful in numerical analysis terms.

Our system also supports online and incremental mining. In particular, when a new algorithm is added to the system, PYTHIA can be bootstrapped with the old rule-bank so that new information does not require retraining on the old. Moreover, the stability of our relational encoding ensures that new data does not 'throw' off the ranking. We refer the interested reader to (Ramakrishnan et.al., 1998) for more details.

4. A Case Study

To validate the design and implementation of PYTHIA, a knowledge base was generated for evaluating PELLPACK algorithms based on performance data produced by a population of 2-dimensional, singular, steady state PDE problems. 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 experiment sequences were specified, each defining a collection of PDE programs involving up to 35 solver sequences for a given PDE problem. The 37 sequences were executed sequentially on a Sun SPARCstation 20 with 32MB memory running Solaris 2.5.1 from within PYTHIA's execution environment. All of them executed successfully, resulting in the insertion of hundreds of performance records into the database. The analyzer evaluated the algorithm performance based on generated measures for *time vs problem size* and *time vs error*. The analyzer rankings and problem features were passed to the rules generator which produced relational rules governing algorithm selection for PELLPACK. The recommender was then used to predict the best method and estimate the corresponding parameters for user specified features and performance criteria.

4.1. Usage Scenario

The PYTHIA project web page `http://www.cs.purdue.edu/research/cse/pythia-II` provides an interface to our system. At the outset, there is a facility to provide feature information about a PDE problem. In particular, there are forms that

guide the user in providing details about the operator, function, domain geometry and boundary conditions[5]. Once these details are provided, PYTHIA uses the rule-bank to perform algorithm recommendation. For example, the problem presented in Fig. 1 has a jump discontinuity in its operator, a right side that has constant coefficients, homogeneous and Dirichlet boundary conditions, a square domain, and importantly, a self–adjoint operator. PYTHIA's rules indicate that the 'second order Dyakunov conjugate gradient finite differences' method (DCG) is appropriate. Having determined the algorithm, PYTHIA uses a nearest neighbor technique to identify the closest matching problem to the user's selection from the list of problems covered (these problems have been attached to the rule) by the fired rule. After it finds the closest problem, it consults its database, and uses the matching problem's performance information to predict the parameters of the recommended algorithm, such as grid size, etc[6]. In this example, the closest matching problem is 'p–28–3' from the database. Using the linearized profile of the DCG solver for this instance and the user specified criteria, PYTHIA determines that the number of grid lines to use is 17 which produces a relative error of 1.00331×10^{-5} and a solution time of 8.353 minutes, well within the requested ranges. The user is notified of this selection along with a URL (described in the introduction) from where a software module implementing this solver can be obtained.

4.2. Knowledge Discovery

The recommendation rules mined confirm the statistically observed conclusion in (Houstis and Rice, 1982) that higher order algorithms are better for elliptic PDEs with singularities. They also confirm the general hypothesis that there is a strong correlation between the order of a method and its efficiency. More importantly, the first ten rules discovered impose an ordering of the various algorithms for each of the problems considered in this study. Interestingly, this ranking corresponds almost exactly with the subjective rankings published in (Houstis and Rice, 1982). This shows that very simple rules capture much of the complexity of algorithm selection in this domain. There were several other interesting inferences drawn. Whenever the DCG method is best, so is DCG4. The rule that had the maximum support 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. 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 overall ordering was induced from the entire population of performance records. 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 problems which formed the bulk of our population. In overall, the rules from this study performed best algorithm recommendation for 100% of the cases.

When the nature of problems was changed to 'mixed-boundary-conditions' we found that the same rules were mined but the support (and confidences) of the mined recommendations were found to come down in magnitude. This conclusively supports the hypothesis made in (Dyksen et.al., 1988) that the introduction of the derivative in the boundary conditions results in a degradation of performances of algorithms.

In future work, we plan to extend PYTHIA into newer domains of scientific software. The modular approach subsumed by the system maximizes the ability of the end-user (a scientist/engineer) 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.

Notes

1. GAMS (Guide to Available Mathematical Software) is a virtual cross-index of mathematical software that provides access to over 10,000 implementations of algorithms from four different repositories on the web. It is accessible at `http://www.nist.gov`.
2. Within each sub-domain of mathematical software, there is considerable agreement on what performance measures are deemed important, how test collections should be organized, what kinds of features (of problems) should be included and are relevant and so on. Thus, population definition and feature handcrafting are not discussed in detail in this paper.
3. PELLPACK is an integrated software system that provides all the computational facilities necessary to solve elliptic partial differential equations on multicomputer platforms. It is similar in spirit (but not in scope) to packages like MATLAB and *Mathematica*.
4. This is similar to type restrictions used in induction packages like FOIL and PROGOL. For example, to induce a predicate such as *sibling(X, Y)*, such systems might require that the type of `X` and `Y` be `person`. PYTHIA's syntactic restrictions are more domain-specific and help in provide a 'taxonomical basis' to induction rather than mere type-checking.
5. While some of our research has concentrated on determining this information automatically, we do not address these aspects in this paper.
6. In case no rule's antecedent (from the induced knowledge) matches completely the features selected by the user, an algorithm is selected based on the distance between the problem's description and the rules. For determining the closest matching problem on this case, the process described previously is applied.

References

Drashansky, T.T., Houstis, E.N., Ramakrishnan, N. and Rice, J.R., Networked Agents for Scientific Computing, *Communications of the ACM*, Vol. 42(3), March 1999, pp. 48-54.

Dyksen, W.R., Ribbens, C.J. and Rice, J.R., The Performance of Numerical Methods for Elliptic Problems with Mixed Boundary Conditions, *Numerical Methods for Partial Differential Equations*, Vol. 4, 1988, pp. 347-361.

Dzeroski, S., Inductive Logic Programming and Knowledge Discovery in Databases, *Advances in Knowledge Discovery and Data Mining*, (U.M. Fayyad et.al., editors), AAAI/MIT Press, 1996, pp. 117-152.

Hollander, M. and Wolfe, D.A., *Nonparametric Statistical Methods*, John Wiley and Sons, 1973.

Boisvert, R.F., Rice, J.R. and Houstis, E.N., A System for Performance Evaluation of Partial Differential Equations Software, *IEEE Transactions on Software Engineering*, Vol. SE-5(4), July 1979, pp. 418-425.

Fayyad, U.M., Piatetsky-Shapiro, G. and Smyth, P., From Data Mining to Knowledge Discovery: An Overview, *Advances in Knowledge Discovery and Data Mining*, (U.M. Fayyad et.al., editors), AAAI/MIT Press, 1996, pp. 1-34.

Houstis, E.N. and Rice, J.R., High Order Methods for Elliptic PDEs with Singularities, *International Journal of Numerical Methods in Engineering*, Vol. 18, 1982, pp. 737-754.

Houstis, E.N. et.al., PELLPACK: A Problem Solving Environment for PDE Based Applications on Multicomputer Platforms, *ACM Transactions on Mathematical Software*, Vol. 24(1), 1998, pp. 30-73.

Kautz, H (Ed.)., *Working Notes of the AAAI-98 Workshop on Recommender Systems*, American Association for Artificial Intelligence, Cambridge, Massachusetts, 1998.

Matheus, C.J., Chan, P.K. and Piatetsky-Shapiro, G., Systems for Knowledge Discovery in Databases, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5(6), 1993, pp. 903-913.

Ramakrishnan, N. *Recommender Systems for Problem Solving Environments*, Ph.D. Thesis, Department of Computer Sciences, Purdue University, 1997.

Ramakrishnan, N., Rice, J.R. and Houstis, E.N., GAUSS: An Online Algorithm Recommender System for One-Dimensional Numerical Quadrature, *ACM Transactions on Mathematical Software*, 1999, Communicated, Also available as Technical Report CSD-TR-96-048, Department of Computer Sciences, Purdue University, 1996.

Verykios, V.S., Houstis, E.N. and Rice, J.R., A Knowledge Discovery Methodology for the Performance Evaluation of Scientific Software, *Knowledge and Information Systems*, 1999, to appear. Also available as Technical Report CSD-TR-98-042, Department of Computer Sciences, Purdue University, 1998.