Mining and Visualizing Recommendation Spaces for Elliptic PDEs with Continuous Attributes

NAREN RAMAKRISHNAN and CALVIN J. RIBBENS
Virginia Polytechnic Institute and State University

In this paper we extend previous work in mining recommendation spaces based on symbolic problem features to PDE problems with continuous-valued attributes. We identify the research issues in mining such spaces, present a dynamic programming algorithm from the data-mining literature, and describe how *a priori* domain metaknowledge can be used to control the complexity of induction. A visualization aid for continuous-valued recommendation spaces is also outlined. Two case studies are presented to illustrate our approach and tools: (i) a comparison of an iterative and a direct linear system solver on nearly singular problems, and (ii) a comparison of two iterative solvers on problems posed on nonrectangular domains. Both case studies involve continuously varying problem and method parameters which strongly influence the choice of best algorithm in particular cases. By mining the results from thousands of PDE solves, we can gain valuable insight into the relative performance of these methods on similar problems.

Categories and Subject Descriptors: G.1.8 [**Numerical Analysis**]: Partial Differential Equations; H.4.2 [**Information Systems**]: Types of Systems; I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems

General Terms: Theory, Performance

Additional Key Words and Phrases: Performance evaluation, data mining, associations, recommender systems

1. INTRODUCTION

The algorithm selection problem was first seriously formulated and analyzed by John R. Rice nearly 25 years ago [Rice 1976]. Having posed the problem of selecting a good method for a particular problem instance as a scientific problem itself, Rice and coworkers, along with many others, have

The work of the second author was performed in part under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract no. W-7405-Eng-48.
Authors' address: Department of Computer Science, Virginia Polytechnic Institute and State University, VA 24061; email: naren@cs.vt.edu.

long worked to address this problem using the tools of the scientific method. Hence, they have used traditional theoretical and mathematical tools (e.g., theorems, convergence rates) as well as empirical and experimental approaches. Rice has been a leader in building frameworks in which large performance-evaluation studies could be conducted and from which new insights into the algorithm selection problem could be gained, e.g., an early PDE solving performance evaluation system [Boisvert et al. 1979], the ELLPACK system [Rice and Boisvert 1985], a population of parameterized test problems [Rice et al. 1981], and a population of parameterized PDE domains and solutions [Rice 1984; Ribbens and Rice 1986]. More recently, approaches to solving the algorithm selection problem whose roots are in the artificial intelligence community have been developed [Addison et al. 1991; Lucks and Gladwell 1992; Kamel et al. 1993; Weerawarana et al. 1996; Houstis et al. 2000]. A particular recent emphasis has been on the implementation of algorithm recommender systems in specialized domains.

One of the main results of this work is PYTHIA [Houstis et al. 2000]—a design framework that supports the rapid prototyping of algorithm recommender systems [Ramakrishnan 1999]. PYTHIA works in conjunction with problem-solving environments (PSEs) such as ELLPACK [Rice and Boisvert 1985] and PELLPACK [Houstis et al. 1998], 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. PYTHIA also supports the incorporation of new learning algorithms that facilitate alternative methods of data analysis and mining.

1.1 Contributions of this Paper

We show how the basic PYTHIA framework presented in Houstis et al. [2000] can be extended to mining recommendation spaces for PDE problems with continuous-valued attributes. Traditionally, continuous-valued attributes have been handled by one of two approaches: (i) using function approximations to model mappings (e.g., neural networks, regression, polynomial networks), or (ii) using techniques such as decision trees that perform sampling or discretization (of the feature space) to design new (symbolic) features that can be subsequently utilized in the induced generalizations. In Ramakrishnan and Valdés-Pérez [2000] we showed why both these approaches are inadequate for profiling mathematical algorithms and mining recommendation spaces. In this paper, we identify the research issues in mining such spaces, present a dynamic programming algorithm from the data-mining literature, and demonstrate how *a priori* domain metaknowledge is factored to control the complexity of induction. A visualization aid for recommendation spaces is also outlined.

Two case studies are presented to illustrate the use of our approach and tools:

(1) a comparison of the performance of an iterative and a direct linear system solver on nearly singular problems and

(2) a comparison of two iterative solvers on problems posed on nonrectangular domains.

Both case studies focus on algorithm selection for the problem of solving linear systems, the dominant step in a typical numerical PDE computation. The problem of choosing good algorithms and software for linear system solving can be approached from a variety of directions. Proposers of new methods typically demonstrate their advantages by using mathematical analysis and illustrative examples on model problems. For example, Greenbaum [1997] and Saad [1996] survey the state of the art in iterative solvers and preconditioners primarily from this perspective. Others have executed performance evaluation studies or modeling of solvers on problems arising from particular application domains [Schmid et al. 1995; Zhang 1996], or in the presence of certain architectural features [Pommerell and Fichtner 1994; de Sturler 1996]. Here, we propose and illustrate another, complementary approach, which applies data-mining techniques to the problem. Both our case studies involve continuously varying problem and method parameters which strongly influence the choice of best algorithm in particular cases. By mining data sets containing the results from thousands of PDE solves, we gain valuable insight into the likely relative performance of these methods on similar problems.

1.2 Organization of the Paper

Section 2 presents a quick overview of a class of techniques appropriate for mining recommendation spaces with continuous attributes. Section 3 identifies various practical considerations and implementation details for the effective application of these techniques. It also emphasizes the role of domain metaknowledge in data mining. The case studies are presented in Section 4. Population definition, experiment processing, and postprocessing are covered in detail here. Section 5 identifies various directions for future research.

2. MINING AS SEARCH

The use of data mining to induce generalizations is an active area of current research. In this section, we present the basics of an important class of single-table data-mining algorithms called *association rule mining*. The associations mined by such algorithms refer to connections between (sets of) entities that occur together (frequently) in a given database. For example, associations in a business context refer to items frequently purchased together in a supermarket, while associations in an algorithm recommender system refer to connections between features of PDE problem instances and the algorithm(s) that performed best (or well) on such instances. Barring any specific constraints on the nature of such associations, one way to generate them is to systematically explore and analyze the space of possible patterns. This idea of "generalization as search" was first proposed in Mitchell [1982], and the specific emphasis on mining database tuples is due to Agrawal et al. [1993]. We present here the salient features of the technique. For more details, we refer the interested reader to Agrawal et al. [1993].

Problem	Feature Information	Algorithm
$p1$	$\{f, g, h\}$	$a1$
$p2$	$\{f, g\}$	$a1$
$p3$	$\{j, k, h\}$	$a2$
$p4$	$\{j, k, i\}$	$a2$
$p5$	$\{j, m, k\}$	$a2$
$p6$	$\{f, h\}$	$a1$

Fig. 1. An RDBMS table that describes an instance of an association rule-mining problem. Each tuple in the table records an experiment and identifies a problem instance, its features, and the algorithm that performed best on that instance.

Definition 2.1. An instance of an *association rule problem* is

—a finite set of features, $F = \{f_1, f_2, \cdots, f_m\}$,

—a finite set of algorithms, $A = \{a_1, a_2, \cdots, a_n\}$,

—a finite set of experiments, $E = \{e_1, e_2, \cdots, e_t\}$ where $\forall i \in [1 \cdots t]$, $\exists j \in [1 \cdots n]$ such that $e_i = (S_i, a_j)$, $S_i \subset F$. In the above formulation, a feature corresponds to a property such as "Operator is Laplace," "Boundary Condition is Dirichlet," etc.; an experiment corresponds to the solution of a PDE problem and the algorithm that was declared the "winner" in the experiment with respect to some performance measure(s) (details of performance evaluation are provided in future sections). Ties between, say two, algorithms are processed either (i) by declaring no winners, or (ii) indicating that both algorithms are winners. The latter case is typically encoded as two different experiments.

For example, Figure 1 describes an RDBMS (relational database management system) table where each experiment is assigned a tuple, and the individual a_j entries denote the choice of an algorithm that best satisfies a performance constraint. We assume, for the moment, that F contains only discrete features, and that algorithms do not have features. The goal of the mining algorithm is two-fold:

—Find all sets $I \subset F$ such that I occurs in at least η% of the tuples, where η is a user-defined *support* level. These are referred to as *frequent itemsets* in Agrawal et al. [1993], a name derived from the original application of data mining to commercial market basket data, since each customer's transaction is modeled as a set of items. Notice that frequent itemsets only contain problem features, not algorithms.[1]

—For all such frequent itemsets I, output a rule $I \rightarrow J$; $J \in A$, if the number of tuples that contain $I \cup J$ (as a fraction of the number of tuples that contain I) is at least θ% where θ is a user-defined *confidence* level.

[1]The reader will note that our definition of support is more restrictive than the definition used in the data mining community. As mentioned before, we have adapted it to mining recommendation spaces for algorithms.

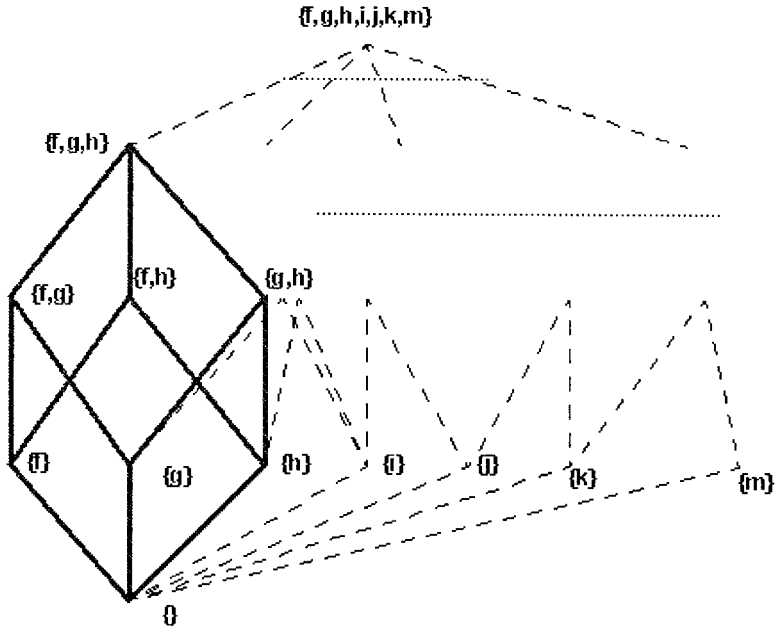

Fig. 2. Lattice induced by the subset relation on the set {f,g,h,i,j,k,m}. The darkened lines indicate the sublattice for the subset {f,g,h} (and, by definition, for {f}, {g}, {h}, {f,g}, {f,h}, and {g,h}).

Notice that support and confidence denote different measures on the nature of the induced patterns. Support determines how representative the frequent itemset is in the experiment database whereas confidence reflects the strength of the implication induced in the second step. A rule can have extremely high confidence but might have been derived from an itemset of low support. For example, algorithm Z might perform well whenever feature X is present, but X might be a very infrequent entry in the database, leading to low support. Conversely, a rule can have low confidence with high/moderate support. For example, features X and Y might occur together in 90% of the tuples, but it might not be possible to obtain a rule with more than 10% confidence that has $X \cup Y$ in its antecedent. Thus, data-mining applications typically exhibit a support-confidence trade-off. For example, one study [Steinacher 1998] cites that mining transactional data on the Internet never results in itemsets with more than 5% support. There are applications where support is more important (business data, since it is required to justify actionability), and there are others where confidence is more important (as in building recommender systems for scientific software). As a result, researchers have explored various linear combinations of these two measures as evaluation criteria for data-mining systems [Fukuda et al. 1996].

To illustrate how an association rule algorithm functions, consider the lattice induced by the subset relation on F. The lattice diagram for the data in Figure 1 is shown in Figure 2 and can be used as the basis to find the

frequent itemsets. One could systematically try either a top-down or a bottom-up approach and explore this space to find the frequent itemsets. However, as identified in Agrawal et al. [1993], the definition of the support function above provides a useful pruning criterion that can be effectively utilized in a bottom-up process. For example, if a given subset of size s does not have support, then no superset of this subset can have support in future generations. Conversely, for a set of size s to have support, all subsets of size $(s - 1)$ must also be frequent itemsets. This is also referred to as *antimonotonicity* in the database literature [Han et al. 1999]. Thus, the lattice framework provides both (i) a means to systematically generate candidate itemsets, and (ii) a pruning criterion for the exploration of this space. The worst-case complexity of this approach is $O(X \cdot |E|)$ where X denotes the sum of the sizes of all itemsets considered by the technique (which is exponential in $|F|$). (Many database algorithms such as closure checking and dependency verification require time proportional to the sum of the sizes of sets considered.) In experimental studies, this is not a bottleneck due to (i) the number of frequent itemsets in higher generations decreases substantially, even for moderate values of support; (ii) database systems provide various primitives for the efficient implementation of this technique: sampling [Agrawal et al. 1996], "pushing algorithms into the database address space" [Sarawagi et al. 1998], constraint-based checking [Han et al. 1999], efficient hash-based indexing data structures for implementing the subset function [Park et al. 1995], and specialized query languages that can selectively reorder operations for higher efficiency [Imielinski and Mannila 1996]; and (iii) efficient online reformulations of the basic technique also exist [Hidber 1999] that can terminate early, once results of the desired quality are achieved, and can be viewed as *anytime algorithms* [Ramakrishnan and Grama 1999] due to their interruptibility and the monotonic improvement of the quality of the answer with time.

Example 1. We now illustrate the operation of the mining technique by application to the data in Figure 1. The first stage involves the computation of the frequent itemsets, and the second stage augments these itemsets to obtain rules that achieve a desired level of confidence. At the end of the first iteration of the first stage, the individual itemsets and their support are given in Figure 3. Assume that we prune subsets at the 50% support level. Thus, only the sets $\{f\}$, $\{j\}$, and $\{k\}$ are considered for computing the itemsets in the next generation. Continuing in this manner, the final frequent itemsets are

$$\{f\}, \{j\}, \{k\}, \{j, k\}$$

The next stage augments these itemsets (with algorithms) to form rules that can be used in deductive inference. We thus mine the rules shown in Figure 4. The confidence measure is provided alongside the rules.

Itemset	Support
$\{f\}$	50.0%
$\{g\}$	33.3%
$\{h\}$	33.3%
$\{i\}$	16.6%
$\{j\}$	50.0%
$\{k\}$	50.0%
$\{m\}$	16.6%

Fig. 3. First generation of frequent itemsets determined by the first step of the association-mining technique.

Rule	Confidence
$f \rightarrow a1$	100%
$j \rightarrow a2$	100%
$k \rightarrow a2$	100%
$j, k \rightarrow a2$	100%

Fig. 4. Rules induced by the association rule-mining algorithm for the sample database in Figure 1.

2.1 Numeric Attributes

The above example utilized only symbolic attributes defined on problem properties. However, many real-world applications involve continuous-valued problem and algorithm parameters which have significant implications for performance. For example, geometric features such as holes, interfaces, and corners can have a significant influence on the performance of discretization methods. Similarly, difficult PDE operator or solution features such as point singularities, boundary layers, and shocks are often present in real-world problems. The relative importance of these features is better represented by continuous parameters than by symbolic or discrete ones. The performance of algorithms is also heavily dependent on continuous-valued method parameters, e.g., acceleration parameters, Krylov subspace dimension, convergence criteria, fill-in levels for incomplete factorization-based preconditioners, etc. (*Note*: We use the term "continuous-valued" to encompass numeric attributes that can be grouped into ranges.)

The analysis presented above fails when we allow database tuples and attributes to take numeric values in a continuous range. This is due to the lack of an effective pruning criterion for continuous valued attributes. Assume that we preprocess numeric-valued attributes by (i) first sorting and bucketing the measured values of the feature, and (ii) subsequently designing tests (with a boolean/symbolic value) based on the class distribution and the subsets induced by the test. For example, we could design new features such as "in the range $[a, b]$," for varying values of a and b. When operating with such ranges, however, constructing a lattice according to the partial order *is-contained-in* does not allow us to use the same partial order as a pruning criterion. As an example, consider the case when one of the features can take values in the range $[1 \cdots 10]$. If a given part of this range $[3 \cdots 5]$ does not have support, we will still need to consider ranges containing this range, such as $[2 \cdots 6]$, $[1 \cdots 5]$, $[1 \cdots 9]$, and so on. In the

1	2	3	4	5	6	7	8
10	8	3	**14**	**10**	8	10	10

Fig. 5. A uniform bucketing of a continuous attribute. Each entry denotes the number of times a certain algorithm was best (out of a maximum of 20). The emphasized bucket range [4, 5] indicates the solution obtained by Bentley's algorithm with a confidence of 0.6.

absence of an effective greedy strategy, Fukuda et al. [1996] showed that the only efficient technique for mining association rules with numeric feature ranges is via dynamic programming by exhaustively checking the space of possible ranges.

Example 2. We now present an example where the database involves a single continuous-valued problem feature f and multiple algorithms. To simplify the analysis, we assume a discretization of the original data into equidepth buckets such that each bucket satisfies the support constraint (which implies that all sequences of buckets will also satisfy the constraint). We then proceed to find the range that maximizes the confidence measure. For example, Figure 5 indicates a continuous attribute discretized into eight buckets of size 20 with each entry denoting the number of times a certain algorithm was best (the support fraction is thus 20/160, or 12.5%). As discussed in Fukuda et al. [1996], Bentley's linear-time algorithm [Bentley 1986] can be used to find a contiguous range that maximizes the confidence. It scans the data from left to right, maintains both the range ending at the scan point as well as the best range seen so far, and uses these ranges to incrementally compute the ranges for the next scan point; see Manber [1992] for a description of this algorithm.

2.2 Two-Dimensional Mining

Extending this scheme to two dimensions (with more than one continuous-valued feature) is simple if the goal is to find a contiguous rectangular region. The straightforward extension of the 1D case results in a $O(N^3)$ time algorithm (since there are N^2 ways to choose the limits of the rectangle and within each rectangle, the columns/rows can be collapsed to yield a 1D problem (which requires O(N) time)). A variation of this approach, popular in image processing and data mining [Fukuda et al. 1996], is to allow nonrectangular regions whose intersection with a family of isothetic axes (vertical or horizontal) is continuous. Such regions can be found in $O(N^2)$ time—an improvement over the regular rectangular algorithm. The technique makes use of a monotonicity property of connected regions[2] and is particularly appropriate for mining two-dimensional recommendation spaces, where confidence-optimizing regions are frequently nonrectangular.

Example 3. Consider a case with two continuous-valued attributes, f_1 and f_2, where f_1 varies in the range $[0 \cdots 25]$ and f_2 varies in the range

[2]This is different from the antimonotonicity nature of the constraints used for data mining.

Table I. Number of Experiments Conducted for Various Values of Two Continuous-Valued Attributes, f_1 and f_2

	$f_2 \in [0,1)$	$[1,10)$	$[10,100)$	$[100,1000]$
$f_1 \in [20,25]$	500	500	500	500
$[15,20)$	500	500	500	500
$[10,15)$	500	500	500	500
$[5,10)$	500	500	500	500
$[0,5)$	500	500	500	500

Table II. Number of *Hits* for Algorithm X for the Experiments Presented in Table I

	$f_2 \in [0,1)$	$[1,10)$	$[10,100)$	$[100,1000]$
$f_1 \in [20,25]$	500	410	0	25
$[15,20)$	500	500	316	75
$[10,15)$	500	500	428	0
$[5,10)$	500	500	500	500
$[0,5)$	500	500	500	500

$[0 \cdots 1000]$. The first step is to discretize these ranges into buckets (this can either be obtained from the process that generated the data or be achieved by equidepth sampling techniques). The goal of this step is to ensure that all buckets satisfy a minimum support constraint. Let us assume that equidepth buckets are obtained for

$$f_1 \in \{[0, 5), [5, 10), [10, 15), [15, 20), [20, 25]\}$$

$$f_2 \in \{[0, 1), [1, 10), [10, 100), [100, 1000]\}$$

Table I describes the equidepth bucketing, and Table II describes the number of *hits* in each bucket for a certain algorithm X, i.e., these are regions for which it performed best. The 2D algorithm described in Fukuda et al. [1996] identifies a connected region, as shown on the right in Figure 6. The mined recommendation space shows that as f_1 increases, algorithm X works well *only* for correspondingly lower values of f_2. This phenomenological observation can then be used as the basis of a knowledge-based decision support system (that selects X for such values of (f_1, f_2)). We refer the interested reader to Fukuda et al. [1996] for more details about the mining algorithm, and to Houstis et al. [2000] for a description of a recommender system facility.

3. IMPLEMENTATION CONSIDERATIONS

In this section, we outline various implementation decisions and considerations for the successful application of this technique to mining recommendation spaces for PDE problems and solvers.

3.1 Domain-Specific Restrictions

The previous section outlined a bottom-up approach to computing itemsets and rules. Concurrent with this approach, a top-down scheme that makes

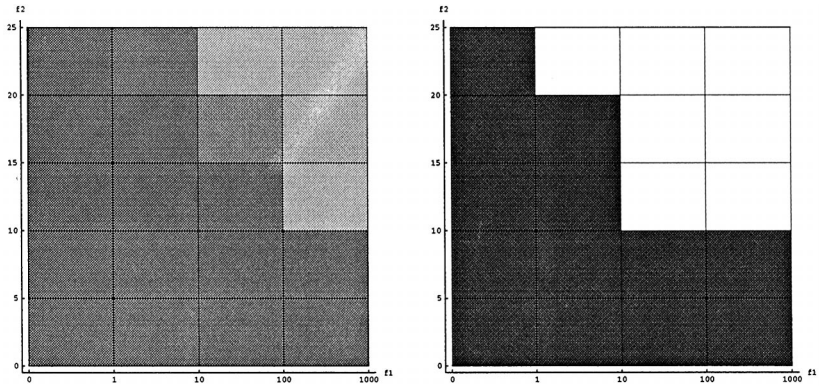

Fig. 6. (left) The data in Table II modeled by a colormap that associates the confidence measure in each bucket with the intensity of pixels. Thus, the darker regions reflect a greater number of hits. (right) The confidence-optimizing region mined by the algorithm of Fukuda et al. [1996].

use of *a priori* domain metaknowledge can suggest directions to explore and collect further data. This is the experiment generation problem studied in Subramanian and Feigenbaum [1986]. Examples of metaknowledge for this domain include the following:

—**Factorization:** If the recommendation space can be factored into multiple, independent recommendation subspaces. For example, if we knew that the effects of feature f_1 on algorithm applicability are completely independent of the effects of feature f_2, we can attempt to learn two different patterns (and combine them later) by a divide-and-conquer approach. Subramanian et al. refer to such patterns as *factorizable concepts* [Subramanian and Feigenbaum 1986]. Factorization can also be used to a limited extent if the features impose a hierarchical structure. It can also be applied to the case when both symbolic and numeric feature attributes are present.

—**Subspace Elimination:** If the recommendation space contains *impossible subspaces*, due to semantic considerations arising from problem definition. For example, consider domain #20 from the population of nonrectangular PDE domains defined by Rice [1984]. This domain involves two intersecting circles controlled by two continuous-valued parameters. The radius of one of the circles is fixed at 1, while the other radius is allowed to vary (which is the second parameter f_2). In addition, the first parameter (f_1) controls the degree of overlap between the two circles. This domain poses the constraint that the condition

$$f_2 > \sqrt{1 - f_1^2}$$

be satisfied for every valid PDE problem. Thus, data points belonging to subspaces that do not satisfy this constraint need not even be evaluated.

```
hit(f1,f2), ~hit(f1+1,f2), f3 > f2 -> ~hit(f1+1,f3).
~hit(f1,f2) -> ~hit(f1+1,f2).
~hit(f1,f2) -> ~hit(f1,f2+1).
```

Fig. 7. Declarative modeling of a staircase constraint for recommendation spaces, where the computations are performed w.r.t. bucket numbers. `hit(f1, f2)` is true if the bucket corresponding to $(f1, f2)$ is a "hit," false otherwise.

—**Constraints:** If the application domain poses constraints on the nature of the induced spaces. Consider again the recommendation space mined in Figure 6 where f_2 has been shown to have a "staircase" effect on the applicability of algorithm X with respect to feature f_1. Modeling this constraint beforehand can assist in exploring the recommendation space. There are two main approaches to encoding such knowledge: (i) in the control flow of the mining algorithm, or (ii) as declarative constraints in the database environment. We prefer the latter which allows for the use of active and rule-based elements to aid in interactive exploration. For instance, the staircase constraint can be modeled via the rule base in Figure 7. The first line in Figure 7 indicates that if the bucket corresponding to (f_1, f_2) is part of the recommendation space but not the one to its right $((f_1 + 1, f_2))$, then no bucket to the top of $(f_1 + 1, f_2)$ can be part of the recommendation space, and so on.

—**Selective Focusing:** This facility is most useful when coupled with incremental computation of recommendation spaces by techniques such as those described in Hidber [1999]. Assume that we obtain a "coarse" image of the recommendation space by specifying a moderate level of support. One could then zoom in to regions that look promising without evaluating other portions of the space. Selective focusing is also facilitated by database-sampling techniques that can dynamically prefetch data based on user preferences [Hellerstein et al. 1999].

—**Bootstrapping:** If a recommendation space between features f_1 and f_2 has been computed for a coarse level of discretization (of the feature space), the patterns mined from this space can be used to bootstrap the study for a finer level of discretization. We have actively utilized this bootstrapping technique for many of the datasets presented in this paper. For example, if a 5×5 bucketing of the feature space reveals a spike along the y-axis, bootstrapping the mining algorithm with this information can narrow down the search space for a higher level of discretization. Bootstrapping can also be employed when we move from a lower-dimensional feature space to higher dimensions.

3.2 Visualizing Recommendation Spaces

Since the mined recommendation spaces could be of arbitrary shape, visualization of the induced regions constitutes an important stage in this

experimental mode of investigation (in contrast, purely symbolic attributes restrict the mined regions to the corners of an m-dimensional feature hypercube). We have implemented a visualization facility using the `DensityGraphics` and `RGBColor` primitives provided in *Mathematica*. These functions enable the creation of colormaps that map regions of the recommendation space to proportions of color components for pixels displayed in each box. One such design is presented in Fukuda et al. [1996] and uses the mapping $f(x) = \{x, 1 - x, 0\}$, where each component of $f(x)$ reflects the components according to the RGB model (x denotes the "hit ratio" for a certain algorithm). The advantages of this mapping are threefold: (i) it uses intensity to provide visual clues that aid in human perception of recommendation spaces; (ii) by setting the blue component to zero, it minimizes the number of colors required to provide acceptable discrimination; and (iii) the colors are not too saturated that perception of detail is impossible. A potential disadvantage of this approach is that it limits the number of different algorithms (choices) that can be represented in a single image. However, from our experiments, the efficiency of the visual search process is significantly enhanced by restricting this number to 2. This factor appears to be heavily application-dependent, and our observation mirrors many psychophysical studies of the use of color in identification tasks [Nowell 1997; Smallman and Boynton 1990]. An example of the $f(x)$ mapping is depicted in the left part of Figure 6. The right part of Figure 6 depicts the region automatically mined by the algorithm of Fukuda et al. [1996]. We use the mapping $g(x) = \{0, 0, 1\}$ ("blue") for the subregions that optimize confidence and $g(x) = \{1, 1, 1\}$ ("white") otherwise.

4. CASE STUDIES

In this section, we present two case studies which illustrate our approach to mining and visualizing recommendation spaces for PDEs with continuously varying parameters. The simple examples here do not necessarily yield significant new insight into the behavior of particular numerical methods, but rather suggest how this approach can be used in the PDE problem-solving context. In both examples, we explore data sets involving linear, second-order, two-dimensional elliptic PDEs, discretized using standard $O(h^2)$ accurate finite differences, with linear systems solved using Krylov subspace solvers preconditioned with incomplete factorizations. We consider nonsymmetric problems with features that can often cause difficulties for iterative solvers. The first example explores the influence of one problem parameter and one method parameter on whether to use an iterative or a direct method. In the second example, we look for patterns in the relative performance of two widely used iterative methods on problems posed on a nonrectangular domain, with one problem parameter controlling the shape of the domain and one influencing the PDE coefficients.

The framework used to carry out these case studies includes the ELLPACK system [Rice and Boisvert 1985], SPARSKIT [Saad 1990], and several scripts which generate and process the data. A script generates an

ELLPACK program which defines a particular PDE problem and solution method. The ELLPACK discretization module "`5 point star`" is used throughout. SPARSKIT solvers `gmres` and `bcgstab`, and preconditioners `ilut` and `ilutp`, were incorporated into ELLPACK and used as solvers and preconditioners, respectively. All experiments were carried out on a 500MHz Digital Alpha workstation.

4.1 Iterative or Direct Linear Solver?

One of the attractions of direct solution methods (e.g., Gaussian elimination or one of its variants) for solving linear systems of equations is that they are essentially guaranteed to work. This "guarantee" ignores the possibility of various numerical disasters, of course, as well as significant performance problems that may stem from the nonzero structure of the matrix, the data structures used, memory hierarchy, etc. But nonetheless, it is true, especially for nonsymmetric problems, that iterative methods introduce a question direct methods do not, namely, "Will it converge?", and if so, "In how much time?" Much is known theoretically about the convergence behavior of iterative methods on model problems. But for more general problems, the choice of a good Krylov solver and preconditioner is often made in an *ad hoc* manner. In this case study, we illustrate how mining a large database of already solved problems can yield helpful insight into the choice between iterative and direct methods.

Consider the scalar linear elliptic equation

$$-\nabla^2 u + \frac{\alpha}{(\beta + x + y)^2} u_x + \frac{\alpha}{(\beta + x + y)^2} u_y = f(x, y),$$

with Dirichlet boundary conditions on the unit square, where $\beta > 0$ and $f(x, y)$ is chosen so that the true solution is the smooth function

$$u(x, y) = (\cos(y) + \sin(x - y)) * (1 + \sin(x/2)).$$

Discretized with centered ("five-point") finite differences, this problem results in a linear system that is increasingly ill-conditioned for large α and small β, respectively. The singularity in the operator is just outside the domain, along the line $\beta + x + y = 0$. We observed in preliminary experiments that the Krylov solvers have difficulty as this problem approaches the singular case, i.e., as α/β^2 grows. What is not so clear is how quickly the performance will degrade, and how much preconditioning can help.

We solved this problem using a uniform grid with spacing $h = 1/100$, resulting in 9801 discrete equations and unknowns. We compared the time taken by the ELLPACK band Gaussian elimination direct solver to the time taken by a typical Krylov solver with fixed memory requirements, namely `gmres(10)` with `ilutp` preconditioning and a relative residual reduction of 10^{-6}. (The notation `gmres(10)` indicates gmres with a restart parameter of 10.) The `ilutp` preconditioner is a variant of `ilut` which uses

partial pivoting [Saad 1996, Chap. 10]. Both `ilut` and `ilutp` rely on a dual thresholding strategy controlled by two parameters, `lfill` and `droptol`. When constructing the preconditioner, these methods drop any element whose absolute value is less than `droptol` relative to the size of the diagonal element. At the same time, only the `lfill` largest off-diagonal elements in each row of the factors L and U are kept. In this way, `lfill` is used to limit the storage requirements of the preconditioner, while `droptol` is used to select only the largest elements for inclusion in the preconditioner. Half of the data in this case study was generated with `droptol` = 0.0 and the other half with `droptol` = 0.001. When `droptol` = 0, the thresholding procedure essentially disappears, with no matrix elements being thrown away because they are too small. The method then reduces to the simple strategy of keeping the largest `lfill` elements on each row of the factors L and U. For each of the two choices for `droptol` we set

$$\beta = 0.01, 0.02, \ldots, 0.05$$

$$\texttt{lfill} = 2, 4, 6, \ldots, 30$$

and let $\log_{10}(\alpha)$ vary randomly in [0, 3]. Each combination of the problem and solution parameters corresponds to a separate PDE solve—45,000 (5 values of $\beta \times$ 15 values of `lfill` $\times$ 2 values of `droptol` $\times$ 300 values of α) in all for this study. In this and the next study, we set the minimum support fraction such that every bucket satisfies the requirement.

In Figures 8 and 9 we show the results for $\alpha \in [1, 200]$, for `droptol` = 0 and `droptol` = 0.001, respectively. The top row in Figure 8 shows the "hits" for `gmres` and a direct solve (left and right, respectively), where a darker shading means a higher percentage of wins for that method. In the bottom row of Figure 8, we show the results of the confidence-optimizing region-mining algorithm for each method for a confidence level $\theta = 0.9$. The importance of the parameter `droptol` is clearly seen. When `droptol` = 0, `lfill` must fall within a relatively narrow interval in order for the iterative method to be preferred, and this is increasingly true as α increases. Eventually, for large enough α, the direct solve is always the best method. In contrast, we see in Figure 9 that setting `droptol` = 0.001 makes `ilutp` much more robust with respect to changes in both α and `lfill`. In this case, as long as `lfill` is sufficiently large the iterative method is preferred. There is no penalty for choosing `lfill` too large because the `droptol` parameter ensures that less-important elements are not kept, no matter how large `lfill` is. Furthermore, the results give some guidance regarding what a "sufficiently large" value for `lfill` might be, for a given α.

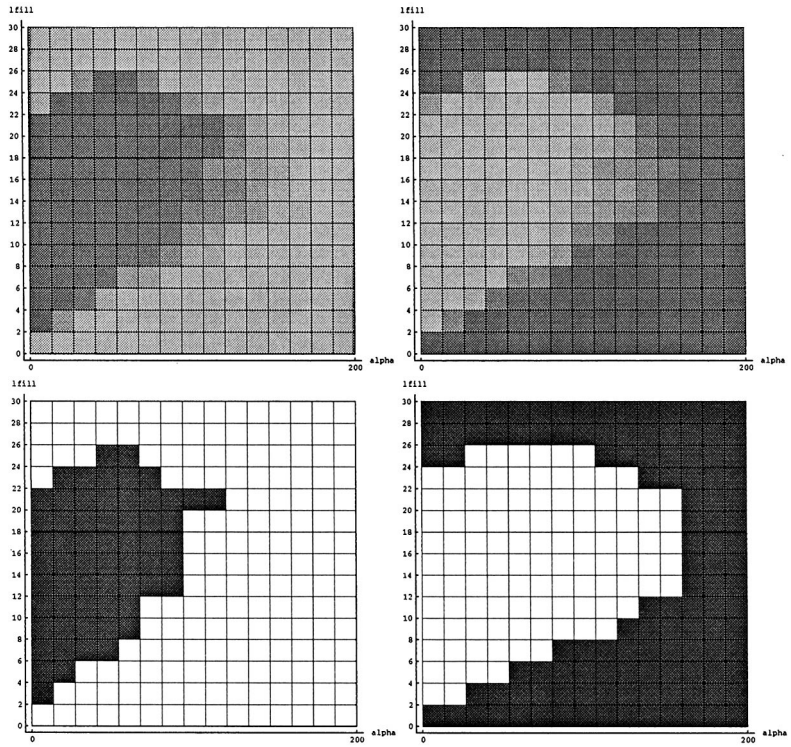

Fig. 8. Case Study 1 results: relative frequency of hits for `gmres` (top left) and direct solve (top right); $\theta = 0.9$ confidence regions for `gmres` (bottom left) and direct solve (bottom right), for `droptol` = 0. Notice that the intersection of the region on the right with a horizontal or vertical segment can lead to a discontinuity; it was obtained by two consecutive runs of the mining algorithm, *i.e.*, once a region is obtained by the algorithm, the data corresponding to this region is removed and the algorithm run again. The end result is a superposition of the two regions.

4.2 Which Iterative Method?

A second case study considers the relative performance of two well-known Krylov solvers on problems posed on a nonrectangular region with a step function in the PDE coefficients. The differential equation solved here is

$$-(w(x, y)u_x)_x - (w(x, y)u_y)_y = 2.0,$$

with homogeneous Dirichlet boundary conditions on the boundary of the region Ω shown in Figure 10. This is Domain 20 from the population defined in Rice [1984]. Note that Ω is parameterized by p. The domain consists of two overlapping circles of radius 1.0 which intersect at $x = p$. The coefficient $w(x, y)$ is defined as a step function, taking on the value α if (x, y) is in the region where the two circles overlap, and returning the value 1.0 otherwise.

Again we discretize using second-order accurate finite differences, this time using a 151×151 uniform grid. The resulting systems of linear

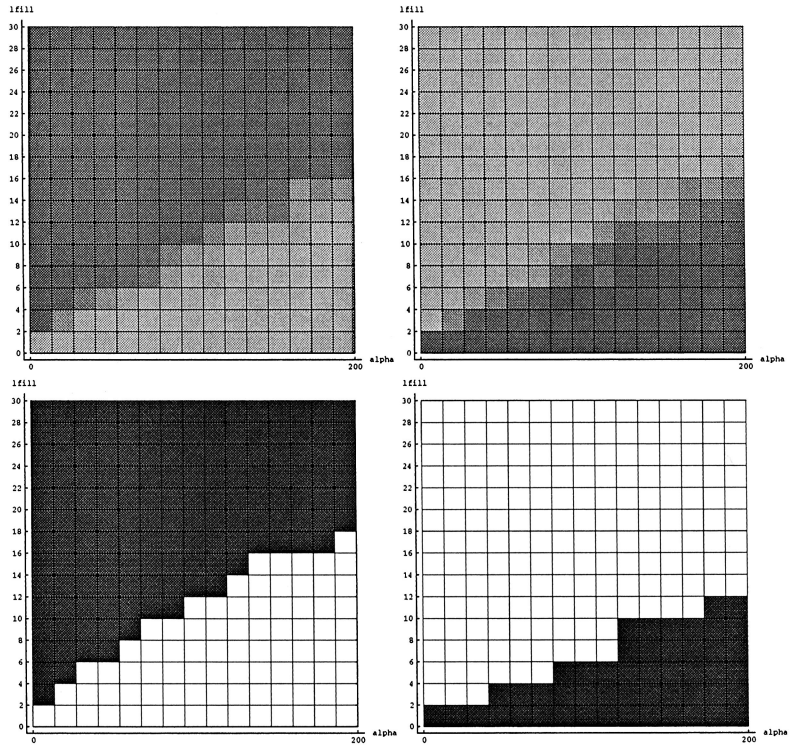

Fig. 9. Case Study 1 results: relative frequency of hits for `gmres` (top left) and direct solve (top right); $\theta = 0.9$ confidence regions for `gmres` (bottom left) and direct solve (bottom right), for `droptol` = 0.001.

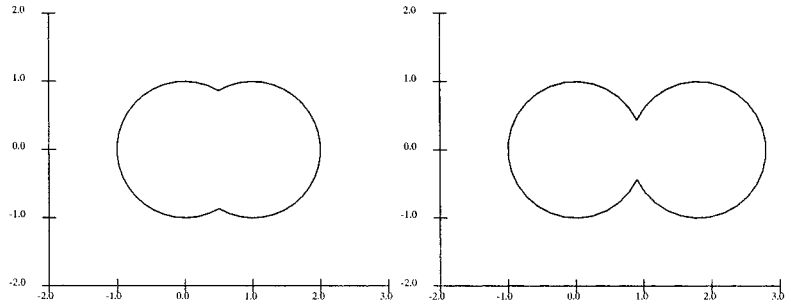

Fig. 10. Domain for Case Study 2: $p = 0.5$ (left) and $p = 0.9$ (right).

equations are of dimension approximately 18,500, the size varying slightly with p. We compare the time taken by SPARSKIT's `gmres(10)` and `bcgstab`, both preconditioned with `ilut`. (We select restart parameter 10 for `gmres` so that the storage requirements of the two iterative methods are approximately the same.) For this case study, `droptol` is fixed at 0.001 while `lfill` is allowed to vary. The results shown in Figures 11 and 12 are from a data set generated by letting p vary randomly in the interval [0.75, 0.95].

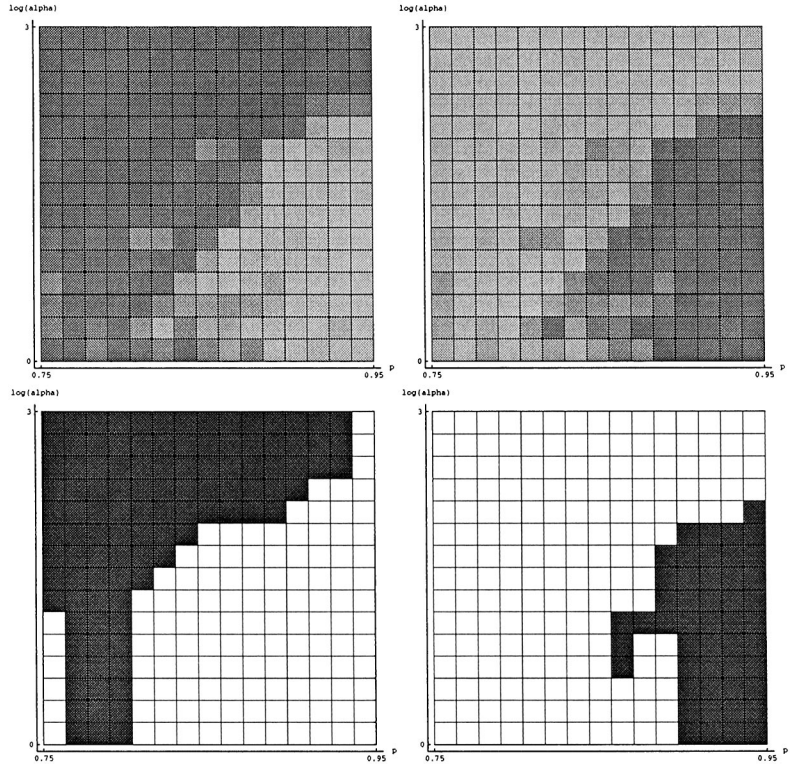

Fig. 11. Case Study 2 results with `lfill` = 20: relative frequency of hits for `bcgstab` (top left) and `gmres(10)` (top right); $\theta = 0.9$ confidence regions for `bcgstab` (bottom left) and `gmres(10)` (bottom right).

Each figure represents 10,000 PDE solves. Figure 11 reflects cases where `lfill` is fixed at 20, while $\log_{10}(\alpha)$ varies randomly in $[0, 3]$. The figure suggests that `gmres(10)` is only competitive for large values of the domain parameter p and when α is not large. In other words, `bcgstab` is more robust with respect to large jumps in the PDE coefficient w. In Figure 12 we view the results with `lfill` and p as the varying parameters—`lfill` varying uniformly from 2 to 30—and α fixed at 10. Again we see that `gmres` is competitive only for large values of p, as long as `lfill` is sufficiently large. It is apparent that `gmres` benefits more from increasing `lfill`, or equivalently, that `bcgstab` does not require as large a value of `lfill`, evidence that `gmres` performs better for the largest values of p, corresponding to the smallest overlap region between the two circles in Ω.

5. CONCLUDING REMARKS

We have demonstrated the applicability of a region-finding algorithm to mining and visualizing recommendation spaces for elliptic PDEs with continuous attributes. While our discussion has been restricted to two-dimensional spaces (for ease of presentation), the ideas presented in

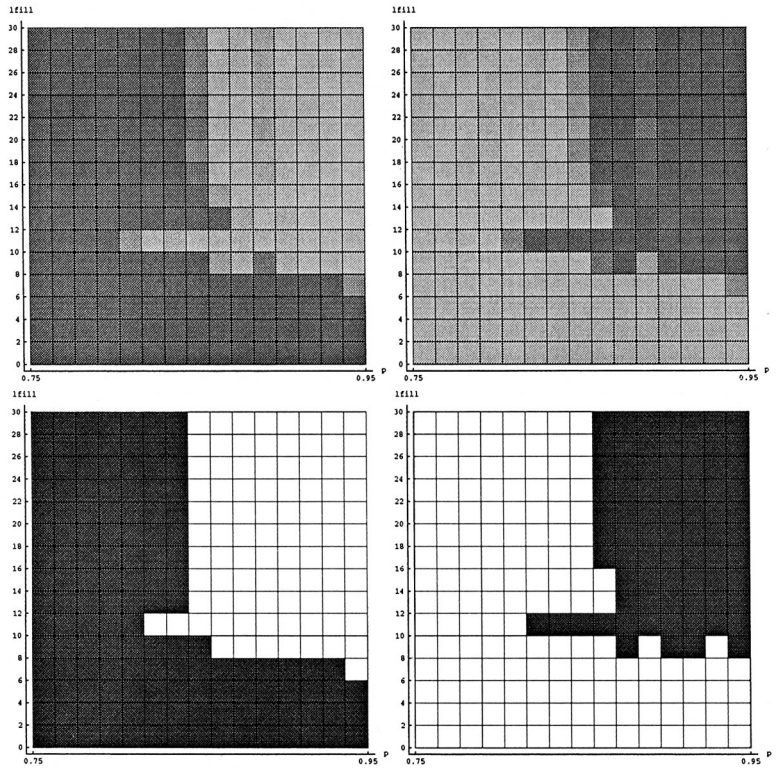

Fig. 12. Case Study 2 results with $\alpha = 10$: relative frequency of hits for `bcgstab` (top left) and `gmres(10)` (top right); $\theta = 0.9$ confidence regions for `bcgstab` (bottom left) and `gmres(10)` (bottom right). Notice again that the region on the right is obtained by two consecutive runs of the mining algorithm, as described earlier.

Section 3.1 can be used to scale up to higher dimensions. In addition, most evaluation criteria for data-mining systems involve limitations that bound the recommendation space from both above and below. Bayardo and Agrawal [1999] showed that the computation of the 2D range described here contains both a positive and a negative border. Thus, one could maintain two sets of patterns (at the expense of optimality), and update each set dynamically in opposite directions till the borders are reached. While we have not utilized this technique in this paper, we believe that this will be necessary in scaling up to more than two dimensions.

Our future work focuses on automating many aspects of the tools presented here—dynamic selection of sampling criteria, using augmented data structures for in-core computations of large datasets, and more detailed characterization of the scientific datasets that are particularly amenable to such techniques. The encouraging results presented in this paper arise from limiting the nature of the induced recommendation spaces. Various studies are planned to address the important issue of *knowledge compilation*—"how to tractably represent/encode application-specific knowledge without compromising efficiency?"—within this frame-

work. Finally, the results of the second case study give a hint that `gmres` has an advantage over `bcgstab` as the outer iteration in an overlapping domain decomposition (Schwarz) method, for example. Although the study was designed to study the relationship between nonrectangular geometries and iterative solvers in a rather general way, it is interesting that we are led to this possibility for future investigations into the relationship between subdomain overlap and iterative methods.

REFERENCES

ADDISON, C., ENRIGHT, W., GAFFNEY, P., GLADWELL, I., AND HANSON, P. 1991. Algorithm 687: A Decision Tree for the Numerical Solution of Initial Value Ordinary Differential Equations. *ACM Transactions on Mathematical Software Vol. 17*, 1, pp. 1–10.

AGRAWAL, R., IMIELINSKI, T., AND SWAMI, A. 1993. Mining Association Rules Between Sets of Items in Large Databases. *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 207–216. Washington, D.C.

AGRAWAL, R., MANNILA, H., SRIKANT, R., TOIVONEN, H., AND VERKAMO, A. I. 1996. Fast Discovery of Association Rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy Eds., *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, Cambridge, MA.

BAYARDO, R. J., JR. AND AGRAWAL, R. 1999. Mining the Most Interesting Rules. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

BENTLEY, J. L. 1986. *Programming Pearls*. Addison-Wesley.

BOISVERT, R., RICE, J., AND HOUSTIS, E. 1979. A System for Performance Evaluation of Partial Differential Equations Software. *IEEE Transactions on Software Engineering Vol. SE-5*, 4 (July), pp. 418–425.

DE STURLER, E. 1996. A Performance Model for Krylov Subspace Methods on Mesh-Based Parallel Computers. *Parallel Computing 22*, pp. 57–74.

FUKUDA, T., MORIMOTO, Y., MORISHITA, S., AND TOKUYAMA, T. 1996. Data Mining Using Two-Dimensional Optimized Association Rules: Schema, Algorithms and Visualization. *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 13–23.

GREENBAUM, A. 1997. *Iterative Methods for Solving Linear Systems*. SIAM, Philadelphia.

HAN, J., LAKSHMANAN, L. V. S., AND NG, R. T. 1999. Constraint-Based, Multidimensional Data Mining. *IEEE Computer 32*, 8, pp. 46–50.

HELLERSTEIN, J. M., AVNUR, R., CHOU, A., HIDBER, C., OLSTON, C., RAMAN, V., ROTH, T., AND HAAS, P. J. 1999. Interactive Data Analysis: The Control Project. *IEEE Computer 32*, 8, pp. 51–59.

HIDBER, C. 1999. Online Association Rule Mining. *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 145–156.

HOUSTIS, E., RICE, J., WEERAWARANA, S., CATLIN, A., PAPCHIOU, P., WANG, K., AND GAITATZES, M. 1998. Parallel ELLPACK: A Problem Solving Environment for PDE Based Applications on Multicomputer Platforms. *ACM Transactions on Mathematical Software 24*, 1, pp. 30–73.

HOUSTIS, E. N., CATLIN, A. C., RICE, J. R., VERYKIOS, V. S., RAMAKRISHNAN, N., AND HOUSTIS, C. E. 2000. PYTHIA II: A Knowledge/Database System for Managing Performance Data and Recommending Scientific Software. *ACM Transactions on Mathematical Software*. appears in this issue.

IMIELINSKI, T. AND MANNILA, H. 1996. A Database Perspective on Knowledge Discovery. *Communications of the ACM*, pp. 58–64.

KAMEL, M., MA, K., AND ENRIGHT, W. 1993. ODEXPERT: An Expert System to Select Numerical Solvers for Initial Value ODE Systems. *ACM Transactions on Mathematical Software 19*, pp. 44–62.

LUCKS, M. AND GLADWELL, I. 1992. Automated Selection of Mathematical Software. *ACM Transactions on Mathematical Software Vol. 18*, 1, pp. 11–34.

MANBER, U. 1992. *Introduction to Algorithms: A Creative Approach*. Addison-Wesley.

MITCHELL, T. M. 1982. Generalization as Search. *Artificial Intelligence 18*, 2, pp. 203–226.

NOWELL, L. T. 1997. *Graphical Encoding for Information Visualization: Using Icon Color, Shape, and Size to Convey Nominal and Quantitative Data.* Ph.D. thesis, Virginia Polytechnic Institute and State University.

PARK, J. S., CHEN, M.-S., AND PHILIP, S. Y. 1995. An Effective Hash Based Algorithm for Mining Association Rules. *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 175–186.

POMMERELL, C. AND FICHTNER, W. 1994. Memory Aspects and Performance of Iterative Solvers. *SIAM J. Sci. Comput. 15*, pp. 460–473.

RAMAKRISHNAN, N. 1999. Experiences with an Algorithm Recommender System. In P. Baudisch Ed., *Proceedings of the CHI'99 Workshop on Recommender Systems.* ACM SIGCHI Press.

RAMAKRISHNAN, N. AND GRAMA, A. 1999. Data Mining: From Serendipity to Science. *IEEE Computer 32*, 8, pp. 34–37. Guest Editors' Introduction to the Special Issue on Data Mining.

RAMAKRISHNAN, N. AND VALDÉS-PÉREZ, R. E. 2000. Note on Generalization in Experimental Algorithmics. *ACM Transactions on Mathematical Software.* to appear.

RIBBENS, C. J. AND RICE, J. R. 1986. Realistic PDE Solutions for Non-Rectangular Domains. Technical Report CSD-TR-639, Department of Computer Sciences, Purdue University, West Lafayette, IN.

RICE, J. 1976. The Algorithm Selection Problem. *Advances in Computers 15*, pp. 65–118. Academic Press, New York.

RICE, J. 1984. Algorithm 625: A Two-Dimensional Domain Processor. *ACM Transactions on Mathematical Software 10*, pp. 453–462.

RICE, J., HOUSTIS, E., AND DYKSEN, W. 1981. A Population of Linear, Second Order, Elliptic Partial Differential Equations on Rectangular Domains, Part I. *Mathematics of Computation 36*, pp. 475–484.

RICE, J. R. AND BOISVERT, R. F. 1985. *Solving Elliptic Problems Using ELLPACK.* Springer-Verlag, New York.

SAAD, Y. 1990. SPARSKIT: A Basic Tool Kit for Sparse Matrix Computations. Technical Report 90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffet Field, CA.

SAAD, Y. 1996. *Iterative Methods for Sparse Linear Systems.* PWS Publishing, Boston.

SARAWAGI, S., THOMAS, S., AND AGRAWAL, R. 1998. Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications. *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 343–354.

SCHMID, W., PAFFRATH, M., AND HOPPE, R. H. W. 1995. Application of Iterative Methods for Solving Nonsymmetric Linear Systems in the Simulation of Semiconductor Processing. *Surv. Math. Indust. 5*, pp. 1–26.

SMALLMAN, H. S. AND BOYNTON, R. M. 1990. Segregation of Basic Colors in an Information Display. *Journal of the Optical Society of America 7*, 102, pp. 1985–1994.

STEINACHER, S. 1998. Data Mining: What Your Data Would Tell You if It Could Talk. http://news400.com.

SUBRAMANIAN, D. AND FEIGENBAUM, J. 1986. Factorization in Experiment Generation. *Proceedings of the National Conference on Artificial Intelligence (AAAI'86)*, pp. 518–522.

WEERAWARANA, S., HOUSTIS, E., RICE, J., JOSHI, A., AND HOUSTIS, C. 1996. PYTHIA: A Knowledge Based System to Select Scientific Algorithms. *ACM Transactions on Mathematical Software 22*, pp. 447–468.

ZHANG, J. 1996. Preconditioned Krylov Subspace Methods for Solving Nonsymmetric Matrices from CFD Applications. Technical Report 280-98, Department of Computer Science, University of Kentucky, Lexington, KY. To appear in Computer Methods in Applied Mechanics and Engineering.

Received October 1999; revised March 2000 and May 2000; accepted May 2000