Personalizing the GAMS Cross-Index

Saverio Perugini Priya Lakshminarayanan Naren Ramakrishnan

Department of Computer Science
Virginia Tech, VA 24061
Email: {sperugin, plakshmi, naren}@cs.vt.edu

March 18, 2000

Abstract

The NIST Guide to Available Mathematical Software (GAMS) system at `http://gams.nist.gov` serves as the gateway to thousands of scientific codes and modules for numerical computation. We describe the PIPE personalization facility for GAMS, whereby content from the cross-index is specialized for a user desiring software recommendations for a specific problem instance. The key idea is to (i) mine structure, and (ii) exploit it in a programmatic manner to generate personalized web pages. Our approach supports both content-based and collaborative personalization and enables information integration from multiple (and complementary) web resources. We present case studies for the domain of linear, second-order, elliptic partial differential equations that indicate strong empirical evidence for the usefulness of our semi-automatic approach.

1 Introduction

The explosive growth of the World Wide Web (WWW) and the concomitant increase in online content has greatly exacerbated information overload. Search engines, while attempting to alleviate this problem, harness only a small fraction of the indexable web (one study estimates this to be $< 30\%$ [6]), and still require users to sift through a multitude of results to determine relevant selections. The low coverage of search engines is attributed to at least two reasons: (i) a majority of web pages are dynamically generated [3] (and hence not directly accessible via hyperlinks), and (ii) lack of sophisticated conceptual models for web information retrieval (but see [2] for some interesting work in this direction). To counter information overload, recommender systems and personalization technology have emerged as a major segment of the Internet economy. They also serve as an effective mechanism to retain customers in E-commerce [10].

The online mathematical software community is not insured from information overload, either. The wide acceptance of repositories such as Netlib (`http://www.netlib.org`), cross-indices such as the Guide to Available Mathematical Software (GAMS at `http://gams.nist.gov`), and problem solving environments (PSEs) has increased the number of online algorithms available to the computational scientist. However, much of this information is accessible only if we know 'what we want' and 'where it is.' The obstacles in selecting the best algorithm for a particular problem and subsequently finding an appropriate software implementation are often difficult and sometimes even impossible to surmount. As an example, it is estimated that there are nearly 10 million software

modules for numerical quadrature that are potentially interesting and significantly different from one another!

Various approaches have been proposed to conduct personalization. Two broad flavors of research can be identified:

1. The *information filtering* research provides content-based schemes that use keywords, string matching, link patterns, and manually compiled identifiers to provide simple 'web query languages' for personalization (e.g. "Find all books on the NY times best-seller list for more than 7 consecutive weeks"). An example from the CS&E domain would be to use GAMS to "find all algorithms that are applicable to helmholtz-type elliptic partial differential equations (PDEs)."

2. The *collaborative filtering* approaches bring in preferences/ratings/experiences into the personalization process. The average Internet user will be familiar with book recommender systems that mine user buying patterns to provide customizations ("Since you liked 'Sense and Sensibility', you might want to buy 'Pride and Prejudice' too"). In the CS&E arena, the GAUSS recommender system [9] selects algorithms for numerical quadrature by organizing a battery of benchmark problems & algorithm executions and mining it to obtain high-level rules that can form the basis of a recommendation. A similar facility for elliptic PDEs is provided by the PYTHIA system [4]. For example, with a given PDE (and performance criteria constraints on its solution), PYTHIA might suggest *'Use the 5–point star algorithm with a* 200×200 *grid on an NCube/2 using* 16 *processors. Confidence: 0.85 (based on performance on similar problem p-36).'* We refer the interested reader to [4] for more details about this system.

However, both these approaches are limited in that they (i) must be individually handcrafted for particular domains and, more importantly, (ii) do not provide facilities to integrate information from multiple web sources. It is well understood that to deliver compelling experiences, personalization systems need to operate on multiple web sites, cross-indices, topic-specific recommenders, individual pages, resource lists, meta search-tools, and internet repositories. Two main technical bottlenecks are (i) the lack of a uniform programming model for building personalization systems, and (ii) integrating the design of such information systems with the task model(s) underlying the (assumed) interaction scenario. This issue is further complicated by the inherent difficulties in information integration — synonymy ('numerical quadrature' in one web site is referred to as 'numerical integration' in another), polysemy (the interpretation of 'linear' depends on the context), and lack of apriori schema in web site organization.

In this article, we present a system for creating personalized recommendations about mathematical and scientific software on the web. One of the main research issues here is understanding and modeling the fundamental processes by which knowledge about scientific problems and algorithms is created, validated, and communicated. Our prototype system (PIPE at `http://pipe.cs.vt.edu`) personalizes content from the GAMS cross-index and the PYTHIA recommender system (at `http://www.cs.purdue.edu/research/cse/pythia`), and integrates them into a unified framework. We demonstrate how PIPE produces results that cannot be achieved by either of the two facilities alone. Moreover, PIPE's integrated methodology is applicable to a wide variety of domains and not restricted to the implementation presented here.

The rest of this article is organized as follows: We first briefly review certain research issues in information personalization and motivate the PIPE methodology. Implementation considerations and the design of a web portal are described next. Finally, experimental results and evaluation of our prototype are addressed.

Web Sources	**Examples**	**Data Models**	**Mining Algorithms**
Unstructured	User Pages Documents	Vector-Space Probabilistic Case-Based Reasoning	LSI [1] n-grams
Semi-structured	XML Labeled Hyperlinks	Graph-Based	Typing Rules Hypergraph Clustering

Table 1: Comparison of Data Models, Web Sources, and Mining Algorithms.

2 Design of the PIPE System

The key idea in PIPE is to (i) extract structure from web sources (such as GAMS), and (ii) exploit it in a programmatic manner to yield customized content. At a high level, web sites can be characterized by a phenomenon of 'implicit but loose structure', denoting the irregular and constantly evolving schema of information (contrast this to relational database systems where the schema is set *apriori*). Various *semi-structured* data models have been proposed, particularly graph-based schemes that use directed labeled arcs to model the connection between web pages and between web sites. At the other extreme, textual content in individual pages is highly unstructured and traditional approaches such as the vector-space model are appropriate. It is not surprising that these diverse models are amenable to different mechanisms for extracting and mining for structure (see Table 1). Mining unstructured data has been well-addressed in a previous issue of *CiSE* [1], and will not be explored further here. We mention briefly that the basic idea is to use linear algebraic matrix transformations and approximations to identify hidden structures in word usage, thus enabling searches that go beyond simple keyword matching. In this article, we concentrate on the graph-based schemes for abstracting and modeling semi-structured data and, in particular, the algorithm of Nestorov et al. [7], as they are more relevant for our purposes.

Consider the hypothetical web site depicted in the top left part of Fig. 1. The individual web pages are denoted by `M1`, `M2` etc., the pre-leaf nodes by `P1`, `P2` etc., while the other internal nodes are represented as `S1`, `S2` etc. Notice that the links are assumed to be tagged via some labeling mechanism, say XML tags (more on this later). The web pages are treated as atomic objects, and the links between web pages are modeled as relations between the atomic objects. The first step in extracting structure from the web site is to proceed to *type* the data. For example, the `S2` node can be typed as: `S2(Y) :- S1(X), link(X,Y,'a'), P1(Z), link(Y,Z,'e')`, which indicates that it is reachable from `S1` (using the `a` tag), and has a link to `P1` (via the label `e`). Such a typing (expressed in the form of a logic program) might not yield any compression to the original data, so various approximations and simplifications are employed to reduce its size for further processing. We first identify commonalities due to encountering the same page multiple times (top right of Fig. 1). Next, the algorithm of Nestorov et al. [7] uses program-theoretic techniques to find the minimal set of types necessary to accurately represent the original data. And finally, allowing one type to be expressed as the superposition of multiple other types helps further reduce the size of the logic program. The end-result of this process (see bottom right of Fig. 1) is a logic program whose interpretation refers to the organization of information in the web sources. For example, the data in Fig. 1 (bottom right) results in:

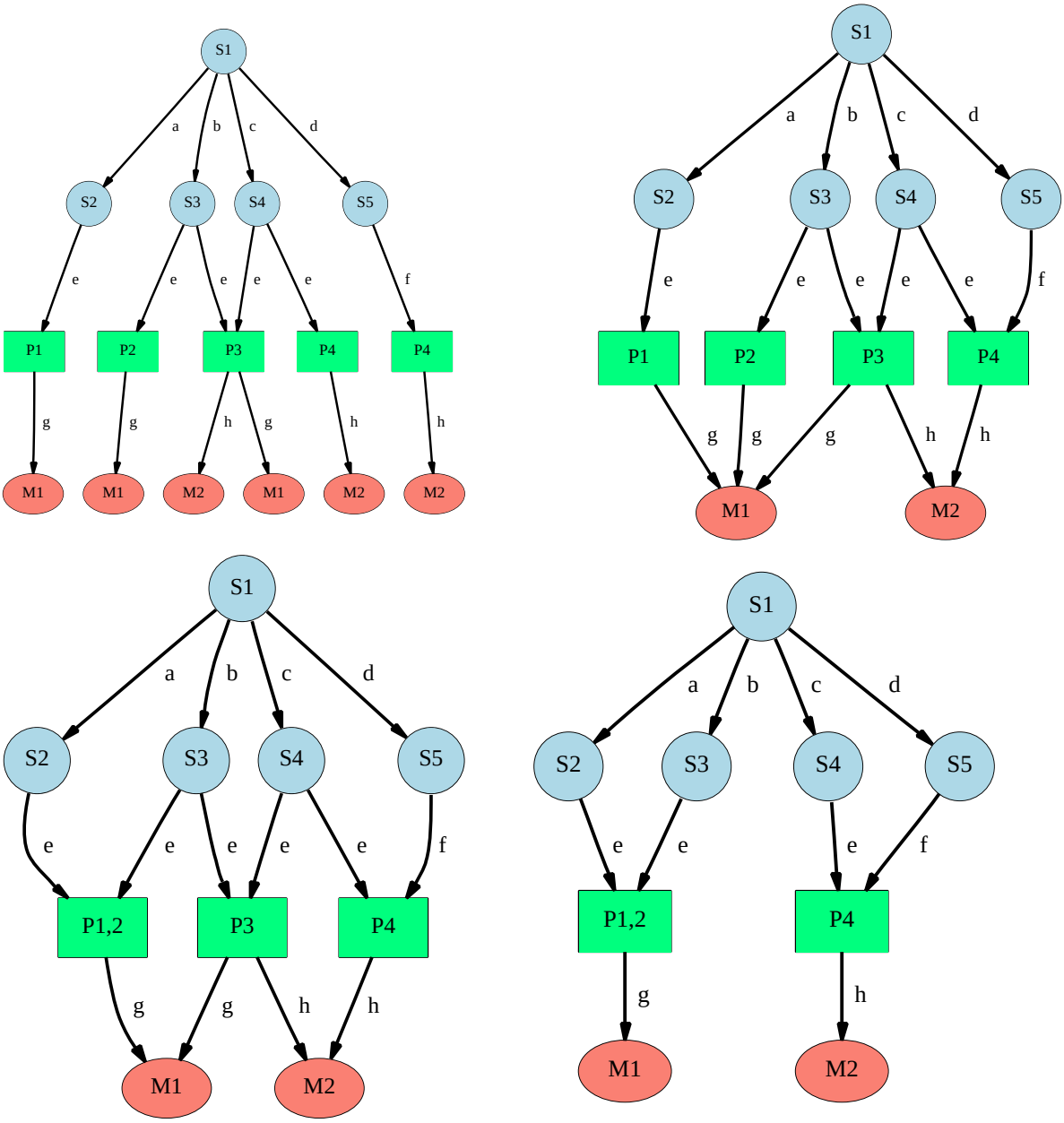

Figure 1: Four stages in extracting structure from a semi-structured data source. The input is assumed to be a graph-structure with labeled and directed edges (top left). Commonalities encountered in tree-building are factored first (top right). At this stage, multiple internal nodes may possess the same input and output labels (for example, `P1` and `P2`). The algorithm then proceeds to type the data, thus collapsing `P1` and `P2` (bottom left). Finally, nodes are allowed to belong to multiple types, rendering `P3` to be redundant (bottom right).

```
S2(Y) :- S1(X), link(X,Y,'a').
S3(Y) :- S1(X), link(X,Y,'a').
S4(Y) :- S1(X), link(X,Y,'c').
S5(Y) :- S1(X), link(X,Y,'d').

P1,2(Y) :- S2(X), S3(Z), link(X,Y,'e'),link(Z,Y,'e'),...
...
```

It should be noted that the final predicates `S2`, `P1,2` etc. are obtained by data mining using the `link` information, and are merely the most natural/obvious interpretations for the types discovered by the program (*i.e. they are not obtained by the algorithm*). We refer the reader to [7] for a more detailed explanation. Notice that while we have only modeled and abstracted the link information between web pages, nonstructural information such as text can be stored alongside using augmented data structures. Such logic programs (from diverse sources), can then be merged, taking care to ensure that entities referred to in different ways by individual web sources are correctly merged together. This involves the effective design of mediators that resolve naming conflicts and issues of synonymy (typically addressed by domain specific conventions in applications) [3]. These steps constitute the off-line aspect of the methodology.

Once web pages and sources are abstracted as logic programs, the next step is to personalize content for the user by creating specialized programs (and reconstructing web pages from them, in turn). Our main contribution is the use of *partial evaluation* to automatically personalize the logic program, by using personalization criteria (from a consultation session with the user). The basic goal of partial evaluation is to automatically specialize programs, given incomplete information about their input. The input to a partial evaluator is a program and (some) static information about its arguments. Its output is a specialized version of this program (typically in the same language), that uses the static information to 'pre-compile' as many operations as possible. A simple example is how the C function `pow` can be specialized to create a new function, say `pow2`, that computes the square of an integer. Consider for example, the definition of a `pow`er function shown in the left part of Fig. 2 (grossly simplified for presentation purposes). If we knew that a particular user will utilize it only for computing squares of integers, we could specialize it (for that user) to produce the `pow2` function. Thus, `pow2` is obtained automatically (not by a human programmer) from `pow` by precomputing all expressions that involve `exponent`, unfolding the for-loop and by various other compiler transformations such as copy propagation and forward substitution. Its benefit is obvious when we consider a higher-level loop that would invoke `pow` repeatedly for computing, say, the squares of all integers from $1 \cdots 100$. Partial evaluation is now used in a wide variety of applications (scientific computing, database systems etc.) to achieve speedup in highly parameterized environments. Automatic program specializers are available for C, FORTRAN, PROLOG, LISP, and several other important languages. We refer the interested reader to [5] for a good introduction. While the traditional motivation for using partial evaluation is to achieve speedup and/or remove interpretation overhead, partial evaluation can also be viewed as a technique to remove inapplicable, unnecessary and 'uninteresting' information from a program.

3 Experimental Results

Our implementation of PIPE personalizes content from the GAMS cross-index for the domain of elliptic PDEs. GAMS is a tree-structured taxonomy that indexes nearly $10,000$ algorithms (from over 100 software packages) for most areas of scientific software. It functions in an interactive

```
int pow(int base, int exponent) {
  int prod = 1;
  for (int i=0;i<exponent;i++)
    prod = prod * base;
  return (prod);
}
```

```
int pow2(int base) {
  return (base * base)
}
```

Figure 2: Illustration of the partial evaluation technique. A general purpose `power` function written in C (left) and its specialized version (with `exponent` = 2) to handle squares (right). Such specializations are performed automatically by partial evaluators such as C-Mix [5].

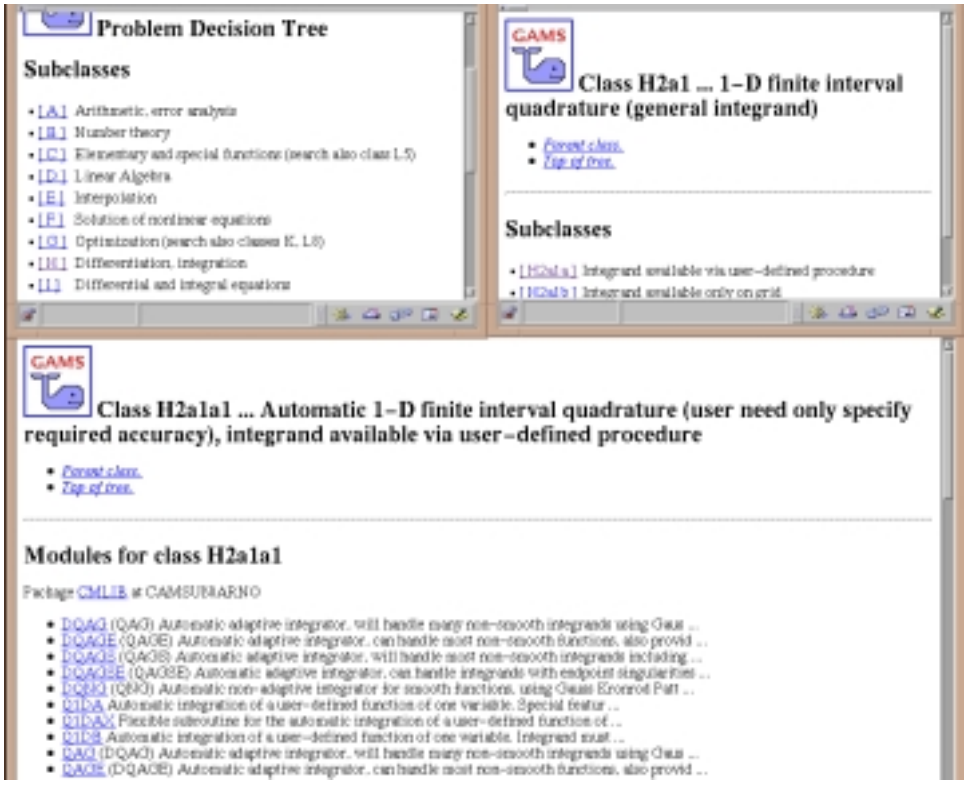

Figure 3: Snapshot of the GAMS Interface (`http://gams.nist.gov`) at three levels of the hierarchy.

fashion, guiding the user from the top of a classification tree to specific modules as the user describes the problem in increasing detail. During this process, many features of the problem are determined, indirectly from the user. However, at the 'leaves', there still exist several choices of algorithms for a specific problem. Fig. 3 describes three screen shots during a typical GAMS session. At first glance, the regularity of the GAMS taxonomy is unequivocally judged by the average web surfer to be a highly structured web site. However, closer inspection reveals that it is best abstracted via semi-structured models:

- **Absence of Apriori Schema:** GAMS organizes its taxonomy into a hierarchy consisting of Classes, Subclasses, Packages, and Modules. However, not all levels of the hierarchy contain the same type of pages. For example, some Subclass pages link to not only other Subclasses, but also directly to Packages.

- **Cross-References:** For certain problem types, more than one category in GAMS is pertinent. For example, optimization software reside not only in the subtree rooted at G ('optimization'), but also K ('approximation'), and L8 ('regression'). Some of this information is provided via "Search also" labels on the corresponding links. Modeling this aspect with graph-based schema introduces additional links that aid in the mining process.

- **Duplication of Common Module Sets:** We also create a virtual page for every instance of a distinct set of modules associated with a package and pointed to by a non-leaf node. For example, different subsets of the CMLIB package are reproduced multiple times at various leaves. Mapping these into a virtual entry provides an additional source for commonalities at the lower levels of the taxonomy.

We designed a web crawler that employs a recursive depth first search complemented with a Perl hash[1] to drive the mining algorithm. Since the same web page may be encountered multiple times, a hash indexed by page URL helps identify commonalities during tree building. The text processing capabilities of the `lynx` browser are used to populate individual hash table entries. Notice that while links in GAMS (and most web sites) are not *typed*, we interpret the text anchoring the '`a href`'s in the web pages as the label when following the associated link. Furthermore, the labels for certain links (typically to modules) are very long that they cannot be listed intelligibly on the originating page. In such cases, the label is suffixed with "..." and continued on the page pointed to. For the purposes of personalization, we cannot ignore the continuation of the label as it may contain important keywords that describe the module.

The compressions arising from mining GAMS schema are of two main flavors: (i) reductions due to factoring common nodes at the pre-leaf level (typically module sets), and (ii) reductions arising from links that violate the tree taxonomy. The first kind are captured by the tree-building process while the second are addressed by the typing algorithm(s). For example, results from the `I2b1a` (11%; Fig. 4) and `L` subtrees (51%; which contains a number of "Search also" links) fall primarily under the first category. Results from the `D` subtree, for instance, arise due to compressions from typing (14%). In addition, the drastic reduction obtained for the `L` subtree is due to the broad, encompassing nature of its category ('Statistics and Probability') and the software modules indexed by it. A more comprehensive tabulation of results for various GAMS subtrees is provided in Table. 2.

In this implementation of PIPE, we focus on the GAMS subtree `I2b1a` (linear, second-order elliptic PDEs). The next step is to merge the schema extracted from this subtree with the PYTHIA rule base for recommending individual solvers for PDEs (information integration). Fig. 5 describes

[1]Perl hashes use arrays indexed by strings, in this case the URL of the web page.

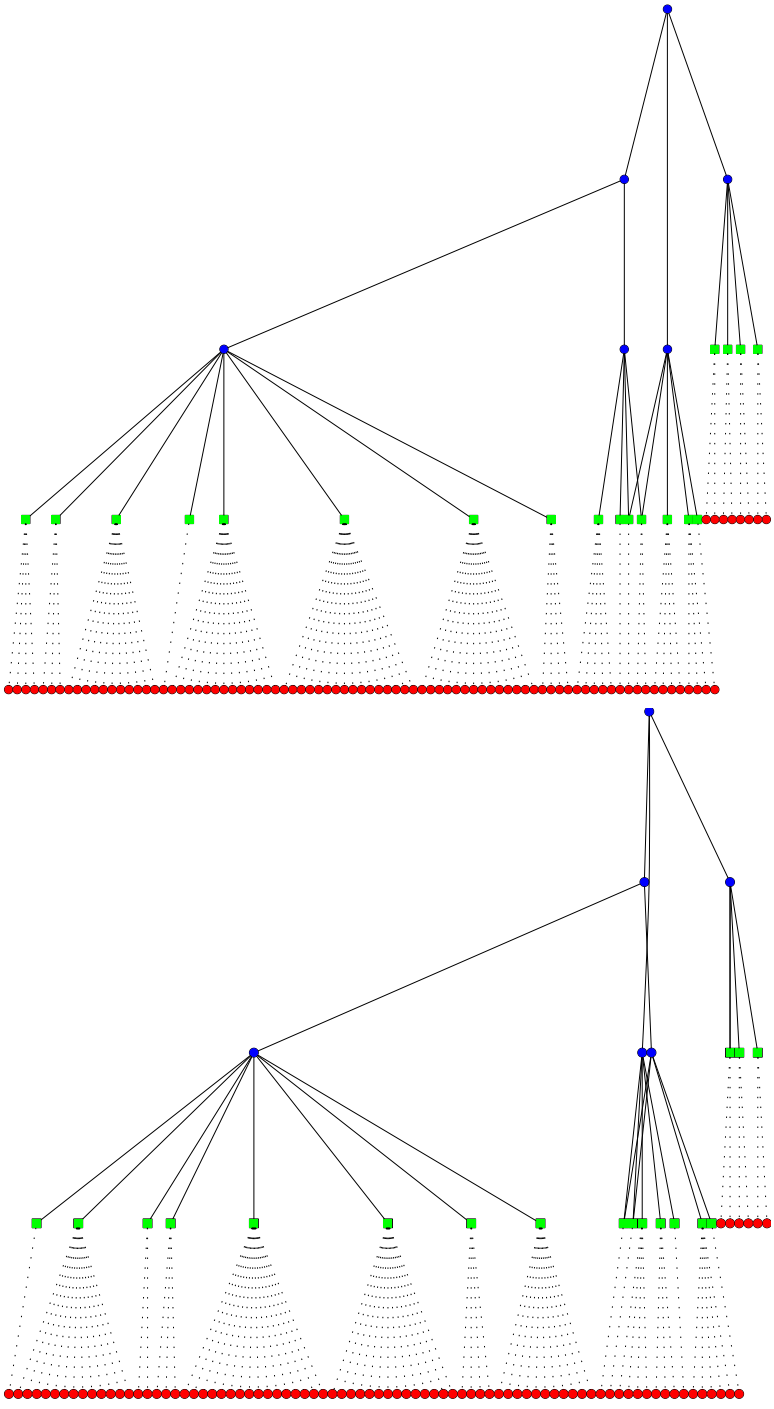

Figure 4: The I2b1a subtree before (top) and after (bottom) the processing of the mining algorithm. Notice the reductions arising from factoring module sets at the pre-leaf level. In other subtrees of the GAMS hierarchy, higher levels of compression are observed.

Subclass	T1	T2	T3	T4	Savings
Example Graph in Fig. 1	10	9	8	7	30 %
A (Arithmetic, Error Analysis)	67	59	58	58	13 %
B (Number Theory)	3	3	3	3	0 %
C (Elementary and Special Functions)	306	283	273	268	12 %
D (Linear Algebra)	883	835	784	763	14 %
E (Interpolation)	85	82	82	82	4 %
F (Solution of Nonlinear Equations)	41	41	39	39	5 %
G (Optimization)	191	186	185	185	3 %
H2a (1-D Quadrature)	80	74	69	69	14 %
I2b1a (Linear, 2nd Order, Elliptic PDEs)	27	25	24	24	11 %
J (Integral Transforms)	66	61	58	58	12 %
K (Approximation)	239	225	220	220	8 %
L (Statistics, Probability)	2706	1340	1330	1330	51 %
M (Simulation, Stochastic Modeling)	11	11	11	11	0 %
N (Data Handling)	101	99	96	96	5 %
O (Symbolic Computation)	6	6	6	6	0 %
P (Computational Geometry)	9	9	9	9	0 %
Q (Graphics)	14	14	14	14	0 %
R (Service Routines)	48	47	45	45	6 %
S (Software Development Tools)	16	16	15	15	6 %
Z (Other)	7	7	7	7	0 %
GAMS	7014	2901	2819	2793	60 %

Table 2: Summary results for various GAMS subtrees. The T's indicate the number of internal nodes, as input (T1), after removing commonalities (T2), after minimal perfect typing (T3), and after factoring nodes that can be expressed as a combination of multiple types (T4). The savings denote the fraction (T1-T4)/T1. Notice that reductions in the L subtree occur during tree building, while subtree D presents opportunities for the mining algorithm.

```
...
(defrule I2b1a-rule1
        ?s <- (type start)
        ?web <- (url http://pipe.cs.vt.edu/start.html)
        (feature opSeparable_yes)
        (feature opPoisson_yes)
=>
        (assert (type type1))
        (retract ?web ?s)
        (assert (url http://pipe.cs.vt.edu/type1.html))
)
...
(defrule pythiarule-r9
        ?web <- (url http://pipe.cs.vt.edu/type4.html)
        ?p <- (problem unsolved)
        (feature opLaplace_yes)
        (feature rhsSingDeriv_yes)
        (type type4)
=>
        (retract ?p)
        (assert (best method is dyakanov-cg))
        (retract ?web)
        (assert (url http://pipe.cs.vt.edu/ell12.html))
        (assert (closest exemplar is p2))
        (assert (problem is solved))
)
```

Figure 5: Example program input to the partial evaluator. The `defrule`s in the above CLIPS program are either automatically mined by the procedure described earlier, or integrated from the recommender rules of PYTHIA. Notice also the annotations in the rules that will be used to decompose the final specialized program to the originating web sources.

a partial listing of the composite program obtained by mining the respective web sources and integrating them. This is represented in the CLIPS programming language, which provides procedural, rule-based, and object-oriented paradigms for representation. The overall architecture of the personalization system is depicted in Fig. 6. The end-user interface for the personalization system is available at `http://pipe.cs.vt.edu` and an example is provided in Fig. 7. As shown, the user provides the input problem in self-describing mathematical terms (the creation of the logic program is effected off-line and needs to be done only once for all personalization sessions with this system). PIPE first parses the input to obtain as many features as possible. The automatic determination of these features is a complex issue, and various techniques are discussed in [8]. Partially evaluating the original problem with these features removes a majority of the original information. Some of the features are used to evaluate the the rules from PYTHIA, providing the final recommendation. At this stage, the 'links' in the specialized program are decomposed into the original web sources. In the general case, this involves having the ability to decompose queries across individual web sources. The key research issue is how to restrict decomposition to a small number of web sources and to determine an exact algorithmic strategy. This is still an open research problem in various domains. Finally, the specialized program is used by an HTML generator to produce the output

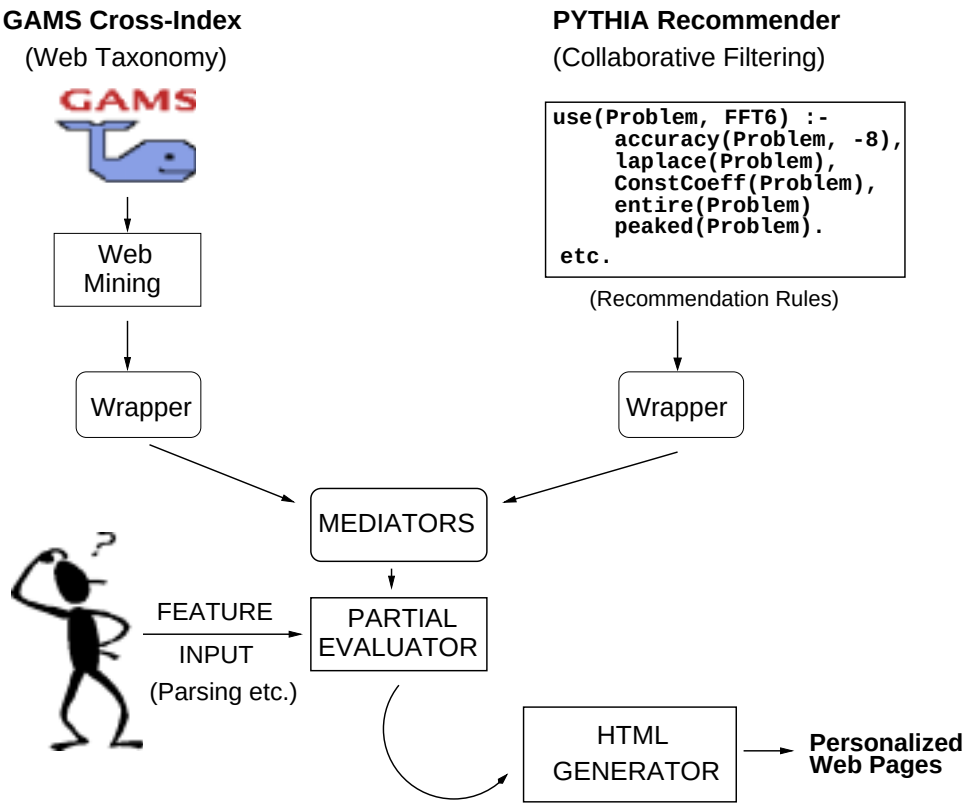

Figure 6: Implementation of the PIPE methodology to create personalized software recommendations. Notice that both content-based and collaborative techniques are harnessed in this example.

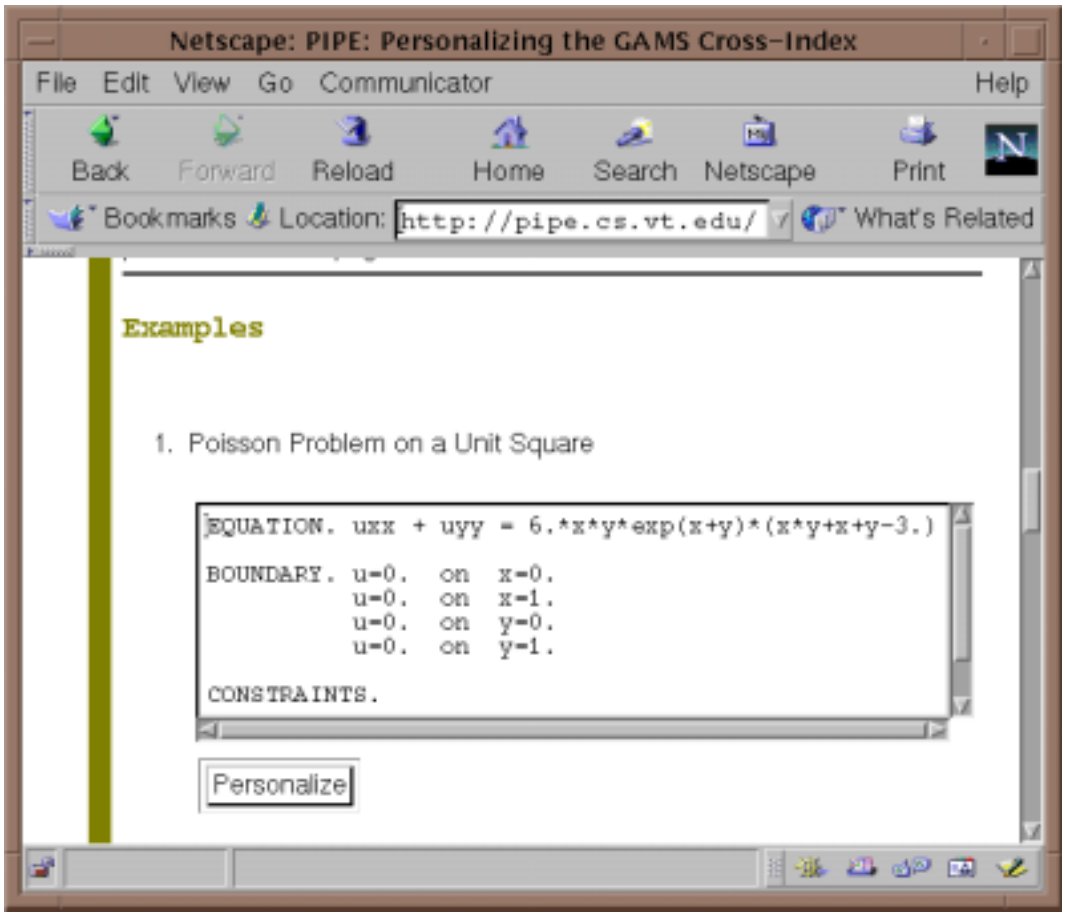

Figure 7: Sample session with the PIPE system, personalizing content for a partial differential equation problem.

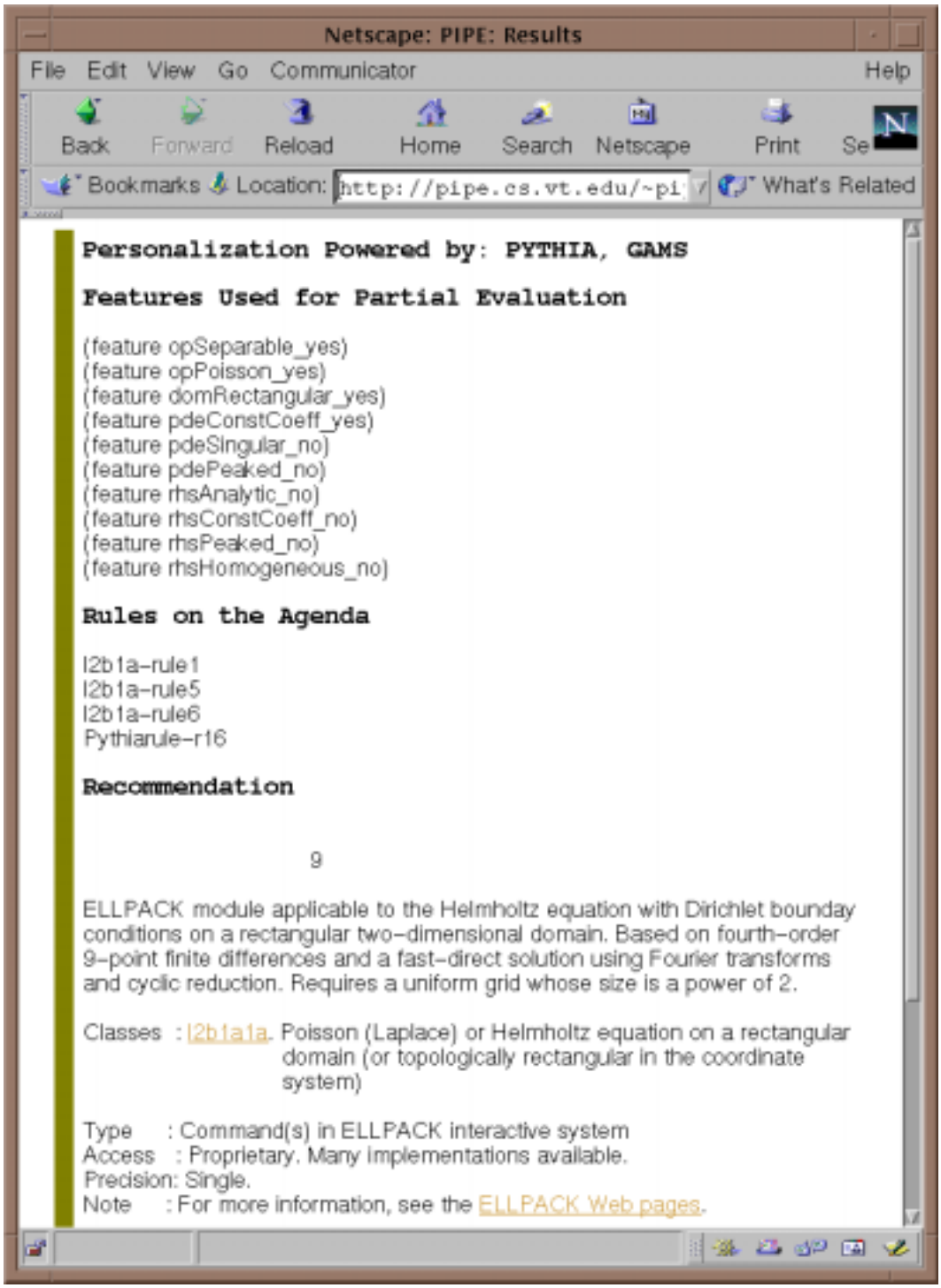

Figure 8: Personalized output for the problem in Fig. 7. Information is available about the features used for partial evaluation and the rules that were evaluated.

shown in Fig. 8. Such generators typically take a specification of a web site as input (graphs, rules etc.) and define the structure of the site in terms of the underlying data model. Our HTML generator is written using the text manipulation capabilities provided in Perl.

PIPE's integrated approach is thus applicable to various other domains where a lot of meta-data (and recommenders) are available, and where the logic program creation can be performed statically; furthermore it still allows the user to choose the rating mechanism (value-neutrality). In our case, this is the PYTHIA recommender system. A different system, trained on qualitatively different examples, might be appropriate in another situation. Also notice that if partial evaluation does not lead to substantial (or any) reduction in program size, we can still reconstruct the appropriate web pages (all of them, resp.) from the original program.

3.1 Evaluation of PIPE

The domain presented here was chosen for its importance and immediate relevance to the computational scientist. While there exist individual recommenders designed for specialized domains of scientific software, there does not exist any comprehensive personalization facility as described in this article. This implementation of PIPE is hence a novel application of personalization technology. In our evaluation of PIPE, we used a benchmark set of problems described in [4] to characterize the final recommendations obtained. Since PYTHIA's selection rules recommend only algorithms from the ELLPACK suite, personalization is most effective with these algorithms. We briefly describe these results below: First, all selections made by PIPE are 'valid' (a selection is considered 'invalid' if the algorithm is inappropriate for the given problem). In addition, we recorded the accuracy of the final selections (a selection is accurate if the selected algorithm does result in solutions satisfying the requested criteria). PIPE selects the best algorithm for 100% of the test cases in the PDE benchmark. Another interesting aspect of PIPE is its ability to recommend algorithms, even in the absence of full coverage from the GAMS site. For example, the fourth-order ELLPACK Dyakunov algorithm DCG4 is identified by GAMS under the category 'Nonseparable problems', whereas PIPE's recommendations show that it is equally applicable to helmholtz type separable problems (see `http://pipe.cs.vt.edu`). This indicates that information integration is a powerful technique to provide comprehensive recommendations.

4 Concluding Remarks

The use of partial evaluation to provide personalization is no *silver bullet*. We emphasize that it is most appropriate for situations where heterogeneity, diversity, and integration of web sources is important. We also note that the existence of an ontology such as GAMS serves two useful purposes: (i) to guide the personalization process, and (ii) to overcome conflicts during information mediation. Systems like PIPE will become increasingly relevant to provide compelling personalization experiences. They will also eliminate the need for restructuring when more web sources or additional rating mechanisms are introduced. The implementation presented here can be extended to include more web sources, such as Netlib, as appropriate.

References

[1] A. Booker, M. Condliff, M. Greaves, F.B. Holt, A. Kao, D.J. Pierce, S. Poteet, and Y.-J.J. Wu. Visualizing Text Data Sets. *IEEE Computing in Science and Engineering*, Vol. 1(4):pp. 26–34, July/August 1999.

[2] S. Chakrabarti, B.E. Dom, D. Gibson, J. Kleinberg, R. Kumar, P. Raghavan, S. Rajagoplan, and A. Tomkins. Mining the Link Structure of the World Wide Web. *IEEE Computer*, pages 60–67, August 1999.

[3] D. Florescu, A. Levy, and A. Mendelzon. Database Techniques for the World-Wide Web: A Survey. *SIGMOD Record*, Vol. 27(3), September 1998.

[4] E.N. Houstis, J.R. Rice, A.C. Catlin, V.S. Verykios, N. Ramakrishnan, and C.E. Houstis. PYTHIA II: A Knowledge/Database System for Managing Performance Data and Recommending Scientific Software. *ACM Transactions on Mathematical Software*, 2000. to appear.

[5] N.D. Jones. An Introduction to Partial Evaluation. *ACM Computing Surveys*, Vol. 28(3):pp. 480–503, September 1996.

[6] S. Lawrence and C. Lee Giles. Searching the World Wide Web. *Science*, Vol. 280(5360):pp. 98–100, 1998.

[7] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting Schema from Semistructured Data. *Proc. ACM International Conf. on Management of Data (SIGMOD'98)*, 1998.

[8] N. Ramakrishnan. *Recommender Systems for Problem Solving Environments.* PhD thesis, Dept. of Computer Sciences, Purdue University, 1997.

[9] N. Ramakrishnan, J.R. Rice, and E.N. Houstis. GAUSS: An Online Algorithm Recommender System for One-Dimensional Numerical Quadrature. *ACM Transactions on Mathematical Software*, 2000. to appear.

[10] C. Shapiro and H. Varian. *Information Rules: A Strategic Guide to the Network Economy.* Harvard Business School Press, November 1998.