Case-Based Recommender Components for Scientific Problem-Solving Environments*

David C. Wilson David B. Leake Randall Bramley†

Abstract

Component-based problem-solving environments (PSEs) provide scientists and engineers with a framework of integrated problem-solving tools and resources that they can easily compose and apply in their particular task domains. Developing effective solution strategies within these environments depends on making good choices about the selection, parameterization, and organization of component tools and resources. Because making good choices may require considerable effort and expertise, designing "intelligent" components that can make informed recommendations about solution development will play a valuable role in realizing the full potential of PSEs. As part of an overall effort in software component systems and PSEs for scientific computing at Indiana University, the CBMatrix project is developing "intelligent recommender components" that use case-based reasoning (CBR) methods to assist in selection, organization, and application of scientific PSE tools and resources. This paper gives an overview of the CBMatrix project, the issues involved, initial results, and the recommender components under development.

Key words: recommender systems, case-based reasoning, problem-solving environments, numerical linear algebra.

AMS subject classifications: 68T05, 68T20, 68T35, 65F50, 65F99, 65Y99.

1 Introduction

Scientific problem-solving environments (PSEs) provide scientists and engineers with a framework of integrated problem-solving tools and resources that they can easily compose and apply in their particular task domains (e.g., [12, 14]). Increasingly, PSEs are being developed as applications of component architectures [3]. Component architectures simplify and expedite the solution design process by encapsulating functional units of executable code with high-level interface specifications for composition; by facilitating the design and execution of distributed applications, enabling them to be composed from standard component services and resources; and by supporting distributed and collaborative problem-solving. Projects such as the DOE Common Component Architecture (CCA) [3], for example, define specifications enabling scientists and engineers to write software components for high-performance computing that can be reused and composed in a wide range of computing environments. This paradigm of composing functional software units and resources shifts the focus of problem-solving from iteratively building workable system implementations to interactively designing optimized solution strategies from existing high-level components.

Designing effective PSE solution strategies depends on making good choices about the organization and configuration of component tools and resources, and considerable expertise may be needed to achieve full benefit from the tools and resources provided by a PSE. Consequently, artificial intelligence methods to develop "recommender systems" [20] to guide tool selection, organization, and application have a valuable role to play in realizing the full potential of PSEs [1, 11, 26]. In particular, the component paradigm affords significant opportunities for integrating

*This research is supported in part by NASA under award No NCC 2-1035. We would like to thank Vikram Subramaniam and Gary Parker for their contributions to this work.

†Computer Science Department, Indiana University, Bloomington, IN 47405, USA. {davwils,leake,bramley}@cs.indiana.edu.

"recommender components" that users may invoke to aid them in selecting and configuring individual components, composing multiple components, and in selecting, monitoring and managing computing resources.

CBMatrix is an ongoing research project in augmenting scientific PSEs with recommender components to support both novice and expert problem-solving. It is part of an overall effort in software component systems and PSEs for scientific computing at Indiana University (e.g., [28, 12]), and it focuses on applications of case-based reasoning (CBR) [21, 23, 29] as the artificial intelligence methodology for making recommendations. Case-based reasoning systems reason and learn by storing records of specific prior problem-solving and re-applying their lessons in analogous situations. By unobtrusively recording the decisions of experts as they use a PSE to solve problems, and providing those decisions as suggestions to guide new problem-solving, CBR provides a vehicle for capturing and sharing expert knowledge.

This paper describes our perspective on recommender components in scientific PSEs, presents the motivations for a case-based reasoning approach to recommendations, and illustrates this approach with developments in the CBMatrix project. We draw examples from our work with the Linear System Analyzer, a problem-solving environment for developing strategies to manipulate and solve large-scale sparse linear systems of equations [12].

2 Recommender Components for Scientific Problem-Solving

The goal of developing recommender components is to increase the effectiveness of problem-solving activity in PSEs. There are two important and complementary ways to further this goal, and they influence how recommender components are constructed and used. The first, *user support*, deals with helping the user to make decisions more effectively. For example, one way to support a user in setting the parameters for a linear solver might be to invoke a particular visualization tool—helping the user to understand the nature of the matrix in question, in order to select an appropriate parameterization. While the visualization tool itself is not part of the component solution strategy (linear system → parameterized solver → result), it may have played a key role in parameter choice for a similar prior problem-solving episode, and thus a user support recommender component could suggest using the visualization tool as part of current solution setup. The second, *component support*, is aimed directly at optimizing the operation of components and component compositions. For example, a component might directly recommend the best data structure representation for a sparse linear system, based on characteristics of the system, in order to achieve good performance with a given solver.

2.1 Recommendation Types

Recommender components can be useful at all levels of scientific problem-solving, from high-level mathematical modeling to mesh manipulation to (non-)linear algebra to data analysis and visualization. Regardless of the level in question, we can typically divide possible types of recommendations into one of the following categories. Each of the following provides a suggested mapping: *user task + task context → recommendation.*

- **User Support:** Given the description of a user task and intended decision, propose resources that can provide helpful information to better enable the user's decision making. This could be applied as an alternative or addition to any of the component-based recommenders described in the following points.
- **Strategy Selection:** Given a specification of the problem to be solved, select an overall strategy for addressing the problem. For example, given a matrix, select a set of preconditioners and a solver that would give a good solution.
- **Component Selection:** Given the context (e.g., the characteristics of a differential equation or linear system to solve) in which a needed component (e.g., some linear solver) will be executed, recommend the best component for the task (e.g., a particular sparse linear solver). Note that the context could be richer, for example, in selecting a preconditioner for a given linear system *and* solver.
- **Component Parameterization:** Given a component that takes parameters, and a context for that component's execution, recommend the best values for the parameter set. Note that some parameters may be fixed by the user, and would become part of the context. Note also that remaining parameters could be recommended based solely on a partially specified parameter set, a kind of parameter completion recommendation.
- **Resource Selection:** Given a software component that needs to be executed, as well as the parameter settings and input to that component (or a description of the input), recommend a computational resource to use when executing the component. This could involve selecting the initial resources for a run or selecting more appropriate ones during a run, for components to automatically move themselves to more appropriate resources.

Recommenders for PSEs must also be flexible enough to support to users with varying levels of expertise, providing the information they need and shielding them from superfluous information. Recommender components should serve these goals for novices by guiding their decision-making, and for experts by providing them with advice when needed, along with explanations to help them evaluate the advice provided.

2.2 Artificial Intelligence Recommendation Methods

An intelligent component library for scientific computing may include a range of components using different artificial intelligence methods individually or in combination. For tasks that are well understood *a priori* (e.g., selecting direct methods for dense matrices), experts can specify a set of rules that fully cover the range of recommendations associated with varying circumstances. Thus these types of recommenders leverage an existing strong domain theory. Traditional rule-based expert systems have been integrated into a number of scientific systems (e.g., for configuring PDE solver libraries [22], for selecting elliptic PDE solution methods [10], and for selecting ODE numerical solvers [19]). Because the methods rely on static pre-defined knowledge, they are considered *non-learning*.

For tasks without hard-and-fast rules (e.g., selecting preconditioned iterative methods for non-symmetric systems), techniques that learn how to make recommendations by using sets of previous examples are more appropriate. *Eager-learning* methods (e.g., induction of decision trees, backpropagation in neural networks, and inductive logic programming) attempt to make generalizations based on a given set of examples (e.g., by deriving a set of rules). The generalizations are then used to make recommendations. This presumes that there is an implicit and relatively strong domain theory that can be exposed from the given set of examples. Research on agent-based frameworks for distributed, collaborative problem-solving and simulation [17], for example, has extended and integrated earlier research on neuro-fuzzy techniques for categorization as part of addressing the algorithm selection problem for elliptic PDEs [30]. More recently, inductive logic programming techniques have been used to learn rules for tasks such as numerical quadrature [27], and analyzing performance effects in elliptic PDE solvers [14].

For learning tasks in which there may not be a relatively strong domain theory implicit in the working set of examples, *lazy-learning* methods (e.g., instance-based learning, case-based reasoning) that reason from specific examples instead of generalizations are more appropriate. In terms of processing, lazy-learning methods typically incur a greater on-line performance cost, but they have a much lower cost for incremental learning of new examples.

Our research concentrates on applications of case-based reasoning methods. Case-based reasoning is an artificial intelligence methodology for reasoning and learning from stored records of specific experiences with analogous prior problems. Learning is an intrinsic part of the case-based reasoning process, because the solutions to prior problems and their outcomes are saved as cases to extend the reasoner's knowledge. When similar situations arise in the future, successful prior cases are retrieved to suggest useful reasoning to reapply, and failure cases are retrieved to warn about potential problems to avoid.

Because CBR systems can learn from single examples without requiring that those examples be generalized, CBR is an appealing method for automatically capturing information without traditional knowledge engineering. In addition, case-based reasoning can be helpful even if few examples are present. Whenever a relevant case is available, it can be applied; the system can be useful without having cases covering the entire space of potential problems. CBR has been applied to scientific computing tasks such as algorithm selection for solving elliptic PDEs [18] and to guiding settings for mesh generation [16]. Central issues for CBR are how to index cases in memory and how to assess similarity between a new problem situation and problems solved previously, as well as how indexing and similarity criteria change over time.

3 CBMatrix

We are developing a series of case-based recommender components, collectively referred to as CBMatrix. This section describes work in developing and refining one such CBMatrix component for data structure selection, and summarizes our current research directions.

3.1 Data Structure Recommendation

We have constructed a prototype CBMatrix component for recommending data structures to use in solving partitioned matrix blocks from large sparse linear systems. In solving these sizable sub-matrices, efficient data structures must be used to store the individual matrix blocks while allowing standard operations to be applied effectively. The selection of an appropriate data structure for each block can significantly increase the performance of the linear solver system, speeding up the overall problem solving process. Even a small percentage improvement over standard performance could mean a significant reduction in problem solving time.

Because there are no hard-and-fast rules for selecting the best data structure, it is usually chosen based on intuitions about the sparsity pattern of the overall matrix or by simply relying on one standard representation for all problems.

Applying CBR to data structure selection promises three main benefits. First, by automating the choice of appropriate data structures, it enables novices to take advantage of resulting performance gains and relieves the expert of the burden of manual data structure selection (especially when that choice would be made for each block of a partitioned matrix). Second, a CBR system can improve its performance by storing cases corresponding to expert choices and results. Third, cases can be used to inform users (to teach a novice or provide support for an expert), by explaining system recommendations with examples of similar situations.

Given a new matrix, the system recommends the data structures that were most appropriate for similar matrices solved in the past. The similarity judgment is based on easy to compute characteristic features of the matrices (e.g., number of non-zeros, degree of bandedness). The first version of our system used a weighted k-nearest neighbor algorithm (e.g., [29]) to determine its recommendations, selecting a predetermined number of similar situations and using the results from that set to determine which data structure to suggest. Our measure of goodness was performance in flops, and the baseline data structure for performance comparisons was compressed sparse row. Tests with various methods for determining a data structure recommendation from the k-nearest neighbors indicated that the most direct method (selecting the overall closest) produced the best results. The second version of the system used only this method. This produced good results in cross-validation testing (seeding the case-base with a portion the case data and retrieving against the rest), and mixed results for individual selections in completely new situations presented to the linear solver system. The third version implemented a similar algorithm to the second, but performed more data normalization. In cross-validation tests, the system made nearly perfect data structure selections. In informal tests using completely new situations, significant performance increases were found in approximately half of the probes.

3.2 Refining Similarity Criteria

With encouraging results in initial tests, we were interested in how machine learning techniques might be used to refine our similarity metric. In computing similarity, it is possible for certain features to be more predictive of data structure choice (e.g., the relative number of non-zeros in a matrix is a likely candidate). If such features are known, they can be assigned a greater weight in the similarity computation. Likewise, less predictive features can be assigned a lower weight or even dropped entirely, increasing the speed of nearest-neighbor retrieval by decreasing the number of feature comparisons.

We conducted a set of tests that used genetic algorithms (GAs) [13] to automatically determine a good set of feature weightings for matrix characteristics in the data structure selection task. The set of data-structure cases was divided into three distinct sets: a set to use as a reduced case-base, a set to train the GA, and a set to evaluate the weighting scheme learned by the GA. For our randomly chosen test sets, unweighted retrieval accuracy was perfect. This was excellent for the selection task itself, but obviated accuracy as a goal in evolving weight sets. The goal of minimizing the number of required features still remained, however, and the GA evolved a set of weights which preserved perfect accuracy, but reduced the number of features used at all by 63 percent (24 to 9). Taking minimization of the number of features as a new goal, the GA found a weight set that reduced the number of features by 92 percent (24 to 2) with a 7 percent loss in accuracy. In order to test whether the GA could assist in accuracy, we explicitly selected 58 cases that gave some degree of error in the unweighted condition. We performed tests that used the difficult set alone (48 training, 10 testing) and combined that set with additional randomly selected instances (152 training, 20 testing). Though the weightings in this second set of tests could not be taken to apply for the entire population, they did show showed that the GA could both improve accuracy, when gains were to be made, and reduce the number of features used.

3.3 Research Directions

We are currently developing new case-based recommender components in conjunction with the latest component architecture research developments at Indiana University. The Indiana University Extreme! Computing Group has built CCAT [6], an implementation of the Common Component Architecture (CCA) for High Performance Computing specification [7, 4]. CCAT has been used to implement a CCA version of the Linear System Analyzer (LSA), a problem-solving environment for developing strategies to manipulate and solve large-scale sparse linear systems of equations [12, 5]. The LSA provides users with a palette of components that can be selected and wired together to construct complete applications. These components differ from subroutines, libraries, etc., in that component composition involves linking binaries, rather than source code to re-compile, and in that components interact on a peer-to-peer basis without one component designated as the “main” program. We expect experience gained in this framework to facilitate construction of recommender components in other PSEs.

Within the CCAT framework, we are designing components that learn by capturing parameter settings (e.g., for a particular solver), component configurations (e.g., sequences of preconditioners and solvers), and information on resource characteristics (e.g., load patterns on particular machines). Feedback on performance will be gathered both from the user (unobtrusively, for example when the user rejects suggested parameter settings) and from monitoring performance information (e.g., when reasoning from the prior case leads to expectations that conflict with observed performance).

The success of case-based recommender components depends on being able to select useful features for assessing the similarity of scientific computing problems. However, we believe that this burden can be alleviated through the application of machine learning methods to enable automatic refinement of feature weightings (e.g., the GA approach described in Section 3.2). Other research issues include how to make the system adjust its recommendations in response to changes in the external processing environment (e.g., by monitoring and responding to error trends detected over time, as in [25]; how to effectively access cases distributed across case libraries from different component instantiations (e.g., [9]); and how to determine which cases to retain and which to delete, in order to reduce storage requirements as large numbers of problems are solved [24]. We are also investigating machine learning approaches to compacting case bases through both explicit generalization of similar cases (e.g., [8]), and implicit generalization by choosing a smaller representative subset of cases (e.g., [2]).

4 Conclusion

We have described our perspective on recommender components for problem-solving environments in scientific computing, as well as applications and current development of case-based reasoning recommender components in the CBMatrix project. As component-based PSEs for scientific computing continue to develop, we believe that CBR will play an important role in making intelligent recommendations to support component use by experts and novices at all levels of the problem-solving process.

References

[1] Harold Abelson, Michael Eisenberg, Matthew Halfant, Jacob Katzenelson, Elisha Sacks, Gerald Sussman, Jack Wisdom, and Kenneth Yip. Intelligence in scientific computing. *Communications of the ACM*, 32(5):546–562, 1989.

[2] D. Aha, D. Kibler, and M. Albert. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[3] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski. Toward a common component architecture for high-performance scientific computing. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing*, 1999.

[4] Rob Armstrong, Dennis Gannon, Al Geist, Katarzyna Keahey, Scott Kohn, Lois McInnes, Steve Parker, and Brent Smolinski. Towards a common component architecture for high-performance scientific computing. In *Proceedings of the High Performance Distributed Computing Conference*, 1999.

[5] Randall Bramley, Dennis Gannon, Thomas Stuckey, Juan Villacis, Esra Akman, Jayashree Balasubramanian, Fabian Breg, Shridhar Diwan, and Madhusudhan Govindaraju. The linear system analyzer. Technical Report TR511, Indiana University, 1998.

[6] CCAT project. <http://extreme.indiana.edu/ccat/index.html>, February 2000. Accessed February 25 2000.

[7] Common component architecture forum. <http://z.ca.sandia.gov/~cca-forum/>, February 2000. Accessed February 25 2000.

[8] P. Domingos. Rule induction and instance-based learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1226–1232, San Francisco, CA, August 1995. Morgan Kaufmann.

[9] Michelle Doyle and Pádraig Cunningham. On balancing client-server load in intelligent web-based applications involving dialog. Technical Report TCD-CS-1999-25, Trinity College Dublin, 1999.

[10] Wayne R. Dyksen and Carl R. Gritter. Scientific computing and the algorithm selection problem. In Houstis et al. [15], pages 19–32.

[11] Efstratios Gallopoulos, Elias Houstis, and John R. Rice. Computer as thinker/doer: Problem-solving environments for computational science. *IEEE Computational Science & Engineering*, 1(2):11–23, 1994.

[12] D. Gannon, R. Bramley, T. Stuckey, J. Villacis, J. Balasubramanian, E. Akman, F. Breg, S. Diwan, and M. Govindaraju. Component architectures for distributed scientific problem solving. *IEEE Computational Science and Engineering*, 5(2):50–53, 1998.

[13] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*, pages 1–24. Addison-Wesley, 1989.

[14] E.N. Houstis, A. Catlin, J. Rice, V. Verykios, N. Ramakrishnan, and C. Houstis. PYTHIA II: A knowledge/database system for managing performance data and recommending scientific software. *ACM Transactions on Mathematical Software*, in press. http://www.cs.purdue.edu/research/cse/pythia/doc/pythia.pdf.

[15] E.N. Houstis, J.R. Rice, and R. Vichnevetsky, editors. *Expert Systems for Scientific Computing: Proceedings of the Second IMACS International Conference on Expert Systems for Numerical Computing*, Amsterdam, 1992. North-Holland.

[16] Neil Hurley. Evaluating the application of CBR in mesh design for simulation problems. In Manuela Veloso and Agnar Aamodt, editors, *Case-Based Reasoning Research and Development: Proceedings of the First International Conference on Case-Based Reasoning*, pages 193–204, Berlin, 1995. ICCBR, Springer Verlag.

[17] Anupam Joshi, T. Drashanksy, J.R. Rice, S. Weerawarana, and E.N. Houstis. Multiagent simulation of complex heterogeneous models in scientific computing. *IMACS Math. and Comp. in Simulation*, 44:43–59, 1997.

[18] Anupam Joshi, S. Weerawarana, N. Ramakrishnan, E. Houstis, and J.R. Rice. Neuro-fuzzy support for problem solving environments: A step towards automated solution of PDEs. *IEEE Computational Science and Engineering*, 3(1):44–56, 1996.

[19] M.S. Kamel, K.S. Ma, and W.H. Enright. ODEXPERT an expert system to select numerical solvers for initial value ODE systems. In Houstis et al. [15], pages 33–54.

[20] H. Kautz, editor. *Recommender Systems: Papers from the 1998 Workshop*. AAAI Press, Menlo Park, CA, 1998.

[21] J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, San Mateo, CA, 1993.

[22] Patrick Laug. DOMINO: a knowledge-based system for the users of a finite element library. *Mathematics and Computers in Simulation*, 36(4–6):293–301, 1994.

[23] D. Leake, editor. *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. AAAI Press/MIT Press, Menlo Park, CA, 1996.

[24] D. Leake and D. Wilson. Case-base maintenance: Dimensions and directions. In P. Cunningham, B. Smyth, and M. Keane, editors, *Proceedings of the Fourth European Workshop on Case-Based Reasoning*, pages 196–207, Berlin, 1998. Springer Verlag.

[25] D. Leake and D. Wilson. When experience is wrong: Examining cbr for changing tasks and environments. In *Proceedings of the Third International Conference on Case-Based Reasoning*, pages 218–232, Berlin, 1999. Springer Verlag.

[26] Naren Ramakrishnan. *Recommender Systems for Problem Solving Environments*. PhD thesis, Purdue University, Lafayette, IN, 1997.

[27] Naren Ramakrishnan, John R. Rice, and Elias N. Houstis. GAUSS an online algorithm recommender system for one-dimensional numerical quadrature. *ACM Transactions on Mathematical Software*, 2000. To Appear.

[28] Juan Villacis, Madhusudhan Govindaraju, David Stern, Andrew Whitaker, Fabian Breg, Prafulla Deuskar, Benjamin Temko, Dennis Gannon, and Randall Bramley. CAT: A high performance distributed component architecture toolkit for the grid. In *Proceedings of the High Performance Distributed Computing Conference*, 1999.

[29] I. Watson. *Applying Case-Based Reasoning: Techniques for Enterprise Systems*. Morgan Kaufmann, San Mateo, CA, 1997.

[30] S. Weerawarana, E. Houstis, J.R. Rice, Anupam Joshi, and C. Houstis. PYTHIA: A knowledge based system to select scientific algorithms. *ACM Trans. Mathematical Software*, 22(4):447–468, 1997.