Functional Knowledge Representation in AI Applications for Scientific Computing

Michael Lucks
Space Telescope Science Institute*
3700 San Martin Drive
Baltimore, MD 21218

Ian Gladwell
Department of Mathematics
Southern Methodist University
Dallas, TX 75275

Abstract

We describe a knowledge representation scheme in which expertise is encoded via expert-supplied mappings, or *knowledge functions*. This functional representation technique was originally developed for the Selection Advisor for Initial Value Software (SAIVS), a prototype system for recommending ordinary differential equation software from numerical subroutine libraries. We discuss the deficiencies in previous knowledge representation schemes that motivated the development of functional scheme, and then present the method. We propose other classes of mathematical software to which the existing SAIVS shell may be applied. Recently, the representation has been adapted for use in the Parallel Object Matching System (POMS), an operational system for scheduling parallel scientific observations on the Hubble Space Telescope. Our experience with the scheme in these two very different areas suggests that it is a generally useful method with potentially widespread scientific applications. We discuss the technique's advantages and limitations, as observed in SAIVS and POMS.

Representational problems

Users of numerical subroutine libraries are frequently confused by the large number of software options available for solving a particular class of problems. Use of an inappropriate code may increase the time required for a solution by many orders of magnitude. Selection of "best" software requires a rare combination of expertise: an understanding of both the mathematics of problem domain as well as the relative behavior of the alternative software modules (possibly from different sources). In 1988, the authors began research toward an automated intelligent system for mathematical software selection. Ordinary differential equation initial value problem (IVP) software was adopted as a test domain. IVP software is characterized by the property that most codes may be applied to most problem instances, but with significant differences in efficiency. At the time, the most robust comparable system [Addison et al., 1991] employed a decision tree model that was quite difficult to modify and was also subject to occasional gross errors. It was assumed by us and others [Schulze and Cryer, 1988; Kamel and Enright, 1992] that selection expertise could be encoded using some conventional symbolic knowledge representation, e.g. production rules. Although a rule-based scheme was clearly a more flexible vehicle than the decision tree approach, it soon became clear that both approaches (and, in fact, any purely symbolic representation) suffer from the same propensity for serious errors when selecting initial value problem software, because they lack the ability to evaluate competitive tradeoffs among the selection criteria. The discretization of quantitative criteria (e.g. system-size, stiffness, sparsity, expense of evaluation, etc.) by symbolic representations results in a significant loss of information that greatly complicates the assessment of such criteria. (For instance, a non-stiff code is generally not recommended for stiff problems. However, it may be be preferable to a stiff code for a very stiff system of ODEs, if the system is also very large. The choice depends on precisely "how stiff" and "how large" -- and the recommendation is critical because these are very expensive problems to solve.)

Faced with the inadequacy of symbolic representations, we examined [Lucks, 1990] various quantitative alternatives. Probability theory and its popular AI variants (e.g. inference networks, belief networks, Dempster-Shafer theory, influence diagrams, etc.) impose a statistical framework that is poorly matched to the software selection problem. (We are typically interested in the strength with which a particular selection criterion is evidenced in a user's problem -- not the probability that the criterion is present.) Numerical extensions to rule-based systems (e.g. certainty factors) are incapable of expressing nonlinear relationships between the strength of a criterion and the suitability of a code. (Using the EMYCIN certainty factor model, for instance, it is impossible to express the $O(n^3)$ effect of increasing system size n, on the performance of certain codes.) Techniques that require some form of machine learning (e.g. neural networks) are impractical due to the expense of generating and executing a sufficiently diverse yet practical set of training problems. A deficiency shared by all of the above approaches (as well as fuzzy logic strategies) is that they are designed primarily for quantifying and aggregating uncertainty. They offer no particular advantage in the selection of IVP software, where it is assumed that the problem description is unambiguous and that the expert's knowledge is precise.

SAIVS knowledge representation

The functional knowledge representation is briefly sketched below. See [Lucks and Gladwell, 1992] for a more detailed description.

Given:

a problem domain P (e.g., IVPs);

a user's problem $p \in P$ (e.g., a particular system of IVP equations);

a set of properties, or features, $F = \{f_1, f_2, ..., f_n\}$ that characterize problems in P (e.g., stiffness, sparsity, accuracy, system size, etc.);

a set of software modules (codes) $S = \{s_1, s_2, ..., s_k\}$ for solving problems in P;

we seek a set of performance functions $H = \{H_1, H_2, ..., H_k\}$, where $H_i : P \times S \rightarrow [0,1]$, such that $Hi(p, s_i)$ quantifies the estimated efficiency of applying code s_i to solve p. Let $f_j(p)$, j = 1,2, ..., n, be an evaluation of feature f_j (e.g if f is system-size, $f_j(p)$ is equal to the number of equations in p). We represent any p by a feature vector $F(p) = <f_1(p), f_2(p), ..., f_n(p)>$. Then we may redefine the domain of H_i as $H_i : F(j) \times S \rightarrow [0, 1]$. The functions in H rank the candidate codes -- the higher the score, the more suitable the code. Values near 1 indicate that the code is highly compatible with the problem features. Scores near 0 imply poor suitability, while scores near 0.5 denote "moderate" compatibility.

The SAIVS knowledge base constructs an H_i for each s_i using a uniform set of mappings -- or knowledge functions -- provided by the domain expert. There are four types of knowledge functions, each fulfilling a specific representational task:

(1) measurement functions $M_j : p \rightarrow R$, j = 1,2, ..., n, that quantify presence of feature f_j in a problem. These are simply feature evaluations, e.g. system-size as described above.

(2) intensity functions $I_j : R \rightarrow [0,1]$, j = 1,2, ... n, that normalize measurement values onto a uniform scale;

(3) compatibility functions $C_{i,j} : [0,1] \rightarrow [0,1]$, i = 1,2, ..., k, j = 1,2, ...n, that express the compatibility of code s_i with the intensity of feature f_j present in the user's problem; the output of $C_{i,j}$ is a number (*compatibility value*) $c_{i,j} \in [0,1]$;

(4) an aggregation function $A_i : [0,1]^n \rightarrow [0,1]$ that aggregates the n individual compatibilities of the various features into an estimate of the code's overall efficiency.

The definition of H_i via composition of the knowledge functions may be visualized as a network shown in Figure 1. The measurement functions are first applied to the input problem, yielding n measurement values. The measurement values are then input to the corresponding intensity functions. The resulting intensity values are input to the appropriate compatibility functions, yielding n compatibility values, which are then fed to the aggregation function A_i. Finally, the output of Ai is returned as the computed value of H_i, yielding the estimated suitability of s_i for problem p.

Measurement functions are represented procedurally, i.e. as code invoked by the system to analyze the input problem and return a feature value. In SAIVS, these procedures are simply queries to the user, i.e. the user is expected to describe the input problem in terms of its feature values. This is a serious limitation on the practical use of SAIVS. An automated facility to perform the problem analysis (a substantial project) is planned for the future. The POMS system (described below) invokes computational procedures to obtain the feature values.

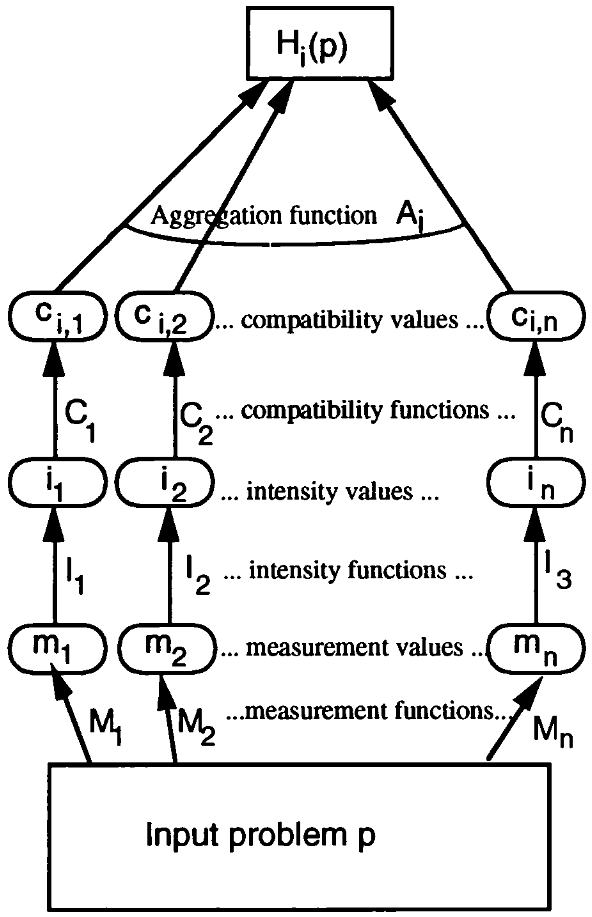
$H_i(p)$
Aggregation function A_i
$c_{i,1}$
$c_{i,2}$
... compatibility values ...
$c_{i,n}$
C_1
C_2
... compatibility functions ...
C_n
i_1
i_2
... intensity values ...
i_n
I_1
I_2
... intensity functions ...
I_3
m_1
m_2
... measurement values ..
m_n
M_1
M_2
...measurement functions...
M_n
Input problem p

Figure 1: Performance function H_i

Intensity and compatibility functions are represented as sets of ordered pairs that generate either piecewise linear functions (for continuous features) or discrete point functions (for qualitative features).

Aggregation functions are constructed as nested compositions of three primitive functional forms: MIN : $[0,1]^2 \rightarrow [0,1]$, MAX : $[0,1]^2 \rightarrow [0,1]$ and a special function $\alpha : [0,1]^2 \longrightarrow [0,1]$. Each A_i may be visualized as a parse tree in which the leaves are input compatibility values and the nodes are aggregation primitives. Figure 2, for example, depicts an aggregation function for code s_1, expressing the combined effects of features f_1, f_2, f_3 and f_4 on the code.

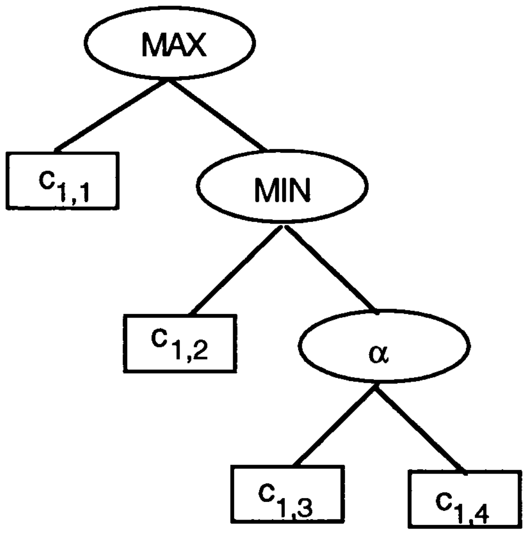

Figure 2: Aggregation function MAX ($c_{1,1}$, (MIN ($c_{1,2}$), α($c_{1,3}$, $c_{1,4}$)))

Each primitive expresses how the combined effect of two features impacts the performance of s_1. MIN and MAX represent the situations where the combined effect two features is dominated by either one or the other, i.e. where there are no tradeoffs. MIN denotes domination by the least compatible feature, while MAX denotes domination by the most compatible feature. For qualitative features, MIN and MAX specialize to the logical operations AND and OR, respectively. α denotes that the combined effect of two features involves tradeoffs. In SAIVS, α is defined as

$$\alpha(x,y) = \frac{xy}{xy + (1-x)(1-y)} \quad .$$

If both features are compatible with the behavior of code, then their combined effect using α is "extra-compatible". Similarly, the combined effect of two incompatible features is less compatible than either one. If one feature is compatible with the code and the other is not, then their combined effect is intermediate

Performance of SAIVS representation

The SAIVS shell, consisting of a control module and an interactive user interface, was originally implemented in Franz Lisp. (Since then, the system has been ported to Allegro Common Lisp.) Using the functional representation, a knowledge base was constructed to evaluate the performance of six IVP codes with respect to eight features (stiffness, accuracy, stability angle, expense of evaluation, system size, number of discontinuities, sparsity and bandwidth ratio). The knowledge base, which reflects the opinions of the domain expert (the second author) regarding the performance of the codes, consists of 8 measurement functions (one per feature), 8 intensity functions (one per feature), 48 compatibility functions (one for each code-feature pair) and 6 aggregation functions (one per code).

SAIVS has demonstrated that the functional scheme is an adequate representation scheme for the IVP problem domain. After an initial round of testing and refinements to the knowledge base, the recommendations of the expert agreed with SAIVS' selections in about 95% of 131 randomly generated problem instances. This is significantly superior to the performance of the decision tree program [Addison et al., 1991]. Among the disagreements, SAIVS' advice was almost always reasonable and there were no instances in which the system recommended a highly inappropriate code. In approximately 11% of the test cases the expert initially disagreed with SAIVS, only to change his mind after a closer inspection of the problem. These problems usually displayed extreme (and unfamiliar) values in multiple features. Hence, the system appears to exceed the ability of its own knowledge source to assess quickly the combined effects of multiple criteria.

The SAIVS knowledge base has proven to be relatively easy to extend and modify. The uniformity of the representation makes knowledge acquisition a relatively mechanical activity, requiring the definition of a known set of functions, instead of the usual unstructured dialogs between the domain expert and the knowledge engineer. Both codes or features may be added or deleted from the system by simply adding or deleting the associated knowledge functions. Small changes to an existing knowledge base are usually made via minor adjustments to compatibility functions. Since the representation is modular and continuous, such local tunings may be made without significantly disturbing the global behavior of the system.

The deficiencies of the functional approach in SAIVS include:

(a) The use of straight line segments for approximation of curves, some of which are really of exponential nature;

(b) The lack of a way of representing "infinities". This is serious only in trade-offs between two or more measurements simultaneously at extreme values;

(c) The "flat" nature of the knowledge base. The "correct" knowledge base (that is the direct representation of the algorithms) would be hierarchical in nature. The use of a one level approximation to a multilevel system may not be as serious as the deficiencies listed above.

Possible remedies, in order, include:

(a') Fitting the data supplied by the expert with a smooth curve, possibly with special properties such as monotonicity and convexity preservation where appropriate;

(b') Approximating the "tails" of the function by exponentials when the functions were originally defined on an infinite range. These exponentials should be connected smoothly to the smooth functions discussed above.

(c') Restructuring the knowledge base in a hierarchical way. This would require a major redesign of the system. Also, it may tend to make the system too problem specific. Only if the system, improved as in the previous paragraphs, proves inadequate in the sense of making incorrect or poor recommendations should this be considered further.

Other mathematical software domains

The SAIVS shell may be used as is to construct software selection advisors for other classes of mathematical problems. The system is most appropriate for problems in which:

(1) The performance of the software depends on an identifiable set of problem features that can be quantified and measured;

(2) Multiple codes exist that solve the entire problem class;

(3) Severe tradeoffs exist among multiple competing performance criteria.

In numeric software, these conditions are satisfied by many problem classes, such as quadrature, linear equation solving and optimization. The performance of software in each of these domains depends on the size and structure of the problem and on various measurable mathematical properties of the input problem. Other potential numeric applications of SAIVS include the choice of approximation algorithms and the selection of discretization strategies for partial differential equations.

A SAIVS-like facility might also be a useful tool to support within a symbolic manipulation system. For example, most computer algebra systems provide several sophisticated algorithms for the computation of multivariate polynomial GCDs. The performance of these algorithms varies greatly with several measurable properties of the inputs (e.g. number of variables, sparsity, degree, and estimated length of primitive remainder sequence). GCD calculation is a fundamental operation that is invoked frequently during intermediate computations required by other procedures (e.g. rational function manipulation or polynomial factorization). Since it is impossible to predict intermediate results in advance, the procedures blindly invoke a fixed default algorithm to perform the GCD, regardless of the input, with potentially disastrous results (e.g. memory exhaustion). With a selection tool available, an "intelligent" generic polynomial GCD algorithm could be written that first runs the selection mechanism, and then invokes the highest ranked algorithm. Since the tool would be internal to the manipulation system, it could access the symbolic software to perform the problem analysis, thereby avoiding the computational limitations of the current SAIVS.

The POMS system

The Hubble Space Telescope (HST) is an orbiting observatory launched in 1990. The presence of six scientific instruments on the telescope provides opportunities for *parallel science*, i.e. the simultaneous use of different instruments to observe different targets, thereby increasing scientific throughput. Parallel science is implemented by partitioning the observation requests into two pools: (1) primary observations (*primaries*), which are scheduled first at times where they best fit; (2) parallel observations (*parallels*), which are of lower priority and are scheduled simultaneously with primaries, provided they do not conflict with the constraints of a primary or impede its execution. For each parallel observation, a search is conducted for compatible primaries. The problem requires finding suitable placements -- i.e. the best matches with primaries -- for as many parallels as possible.

Compatibility depends on a variety of criteria, both qualitative (e.g. both observations may not require the same instrument) and quantitative (e.g. the goodness-of-fit of the observations' timing requirements, and the amount of maneuvering required by the spacecraft in order to bring both targets into view at the same time). Evaluating the compatibility between thousands of primary and parallel observations over a year-long schedule is a complex and time-consuming task, hence an intelligent, automated system was desired to assist HST schedulers. This, in turn, required the selection of a knowledge representation medium to encode schedulers' preferences and spacecraft knowledge.

Although the HST matching problem is dissimilar to IVP software selection, both problems require the comparative assessment and aggregation of multiple quantitative criteria, hence the functional knowledge representation seemed applicable. Other factors that argued for use of the scheme were:

(1) Whatever method was chosen, it had to be implemented from scratch within the existing Spike scheduling system [Johnston, 1990] in order to access the data required for match assessment. A SAIVS-like shell was deemed to be simpler to build than other alternatives, e.g. an embedded rule-based system;

(2) The problem was new and there was no authoritative expertise about what should go into the knowledge base. We anticipated that the contents of the knowledge base would be highly dynamic until the problem was better understood. Hence, the flexibility of the functional representation seemed very desirable.

The Parallel Observation Matching System (POMS) [Lucks, 1992], using the functional representation, was implemented in Allegro Common Lisp and has been an operational part of the HST scheduling system since March, 1992. The system scores the compatibility of each primary-parallel pair based on fifteen problem features, in a manner similar to that depicted in Figure 1. The POMS knowledge base differs from SAIVS in that there is only one aggregation function, as opposed to one per code in SAIVS. This is because in POMS the effects of the problem features are identical for all candidate pairs, whereas in SAIVS the feature effects depend on the particular code under consideration. Also unlike SAIVS, the measurement functions in POMS are actually implemented as computational procedures, as opposed to user queries in SAIVS.

Like SAIVS, POMS has an explanation facility that displays its rankings in an understandable, tabular format.

Performance of representation in POMS

Currently, POMS' decisions are reviewed and either confirmed or disconfirmed by human schedulers. POMS' recommendations have generally been considered acceptable by the schedulers. When a recommended match has been rejected, the reason for the error has usually been revealed promptly using the POMS explanation facility.

As expected, the POMS knowledge base underwent significant modification during its implementation and testing phases, and has still not stabilized completely. The ease and locality with which changes have been effected is a major advantage of the system.

The POMS shell was implemented quickly (about one week) within a large, complex existing software system.

Since certain measurement functions for certain features involve expensive calculations, the exhaustive assessment of all candidate primary-parallel pairs creates an efficiency problem for POMS. The following measures are taken to minimize this problem:

(1) The intensity and compatibility functions for each feature are evaluated immediately after the features measurement function (instead of evaluating all measurement functions first). If any single feature is found to be incompatible for a candidate pair, evaluation of the remaining features for the pair is aborted. This avoids unnecessary measurement function evaluations.

(2) The features are evaluated in order of their computational expense, so that a more expensive feature is never evaluated if a less expensive one is determined to be incompatible;

(3) Preprocessing is performed to weed out infeasible matches prior to submission to POMS.

References

Addison C. A., Enright, W. H., P. W. Gaffney, Gladwell, I. and Hanson, P. M. 1991. A Decision Tree for the Numerical Solution of Initial Value Ordinary Differential Equations, *ACM Transactions on Mathematical Software*, 17:1-25.

Gladwell, I. and Lucks, M, 1992. An Interactive Session with a Knowledge Based System for Numerical Software Selection, to appear in the *Transactions of the IMACS 13th World Congress, AI, Expert Systems and Symbolic Computing for Scientific Computation*, (E. N. Houstis and J. R. Rice, eds.), North-Holland.

Johnston, M., 1990. SPIKE: AI Scheduling for NASA's Hubble Space Telescope. In *Proceedings of the Sixth IEEE Conference on Artificial Intelligence Applications,* Los Alamitos, California, IEEE Computer Society Press, 184-190.

Kamel, M. and Enright, W. H., 1992. ODEXPERT: A Knowledge Based System for Automatic Selection of Initial Value ODE System Solvers, in *Expert Systems for Scientific Computing*, (E.N. Houstis and J. R. Rice, eds.), Elsevier.

Lucks, M., 1990. A Knowledge-Based Framework for the Selection of Mathematical Software, Ph. D. dissertation, Dept. of Computer Science, Southern Methodist University.

Lucks, M, 1992. Detecting Opportunities for Parallel Observations on the Hubble Space Telescope, in *Proceedings of the 1992 Goddard Conference on Space Applications of Artificial Intelligence*, NASA Goddard Space Flight Center, Greenbelt, MD, 29-44.

Lucks, M. and Gladwell, I., 1992. Automated Selection of Mathematical Software, *ACM Transactions on Mathematical Software*, 18:11-34.

Schulze, K. and Cryer, C. W., 1988. NAXPERT: A Prototype Expert System for Numerical Software, *SIAM Journal of Scientific and Statistical Computing*, 9(3):503-515.