Meta-Level Control of Approximate Reasoning: A Decision Theoretic Approach

Shlomo Zilberstein

Computer Science Department, University of Massachusetts, Amherst, MA 01003

Abstract. This paper describes a novel methodology for meta-level control of approximate reasoning. We show that approximate reasoning performed by anytime algorithms offers a simple means by which an intelligent system can trade-off decision quality for deliberation cost. The model exploits probabilistic knowledge about the environment and about the performance of each component in order to optimally manage computational resources. An off-line knowledge compilation technique and a run-time monitoring process guarantee that the system's performance is maximized. The paper concludes with a summary of two applications.

1 Approximate Reasoning in Intelligent Systems

Approximate reasoning techniques, such as abstraction, variable precision logic, and limited horizon search, play an increasing role in intelligent systems. The need to manipulate approximate information stems from various reasons such as an imprecise model of the environment, the presence of stochastic events, limited computational resources, and noisy sensing devices. As a result, complex intelligent systems face a new control problem related to the management of precision.

When a system is composed of a number of modules that produce approximate results, important methodological questions arise regarding the management of uncertainty and precision. How can the performance of the approximate components be described? How does the output quality of a module depend on the precision of the input it receives? How should the execution of a composite system be managed so as to maximize its overall performance? And most importantly, what design methodologies simplify the task of the programmer developing such systems?

We have developed an efficient model that answers these questions. In this model, approximate reasoning is used as a valuable mechanism to trade off decision quality for deliberation costs. This mechanism allows an intelligent agent to control the level of precision of each component and maximize the achievement of its top-level goals.

The basic constructs of our model are anytime algorithms [1, 3] that form a special type of approximate reasoning. They are characterized by the gradual improvement of quality of results as a function of time. Anytime algorithms offer a simple means by which a system can trade-off decision quality for deliberation costs. In addition, the model includes a novel technique to control anytime algorithms using an adaptive, decision-theoretic approach. Efficient control

of computational resources is performed by two major components: an off-line knowledge compilation process and a run-time monitoring process. By mechanizing the control of precision, we take an important step towards the widespread use of approximate reasoning.

The rest of the paper outlines our methodology and its application. Section 2 describes the notion of an anytime algorithm and its attractive properties as an approximate reasoning technique. Section 3 describes a compilation technique to compose anytime algorithms. Sections 4 describes the run-time control mechanisms. In Section 5, we briefly describe two applications. Finally, Section 6 summarizes the benefits of our approach and discusses some directions for further work.

2 Anytime Algorithms

Anytime algorithms are algorithms whose quality of results improves gradually as computation time increases. They offer a tradeoff between resource consumption and output quality. Many existing programming techniques produce useful anytime algorithms. Examples include iterative deepening search, variable precision logic, and randomized techniques such as Monte Carlo algorithms or fingerprinting. For a survey of anytime programming techniques see [8].

Various metrics can be used to measure the quality of a result produced by an anytime algorithm. From a pragmatic point of view, it may seem useful to define a *single* type of quality measure to be applied to all anytime algorithms. But in practice, different types of anytime algorithms approach the exact result in different ways. The following metrics have been proved useful in anytime algorithm construction: certainty – reflecting the degree of certainty that the result is correct, accuracy – reflecting the distance between the approximate result and the exact answer, and specificity – reflecting the level of detail of the result.

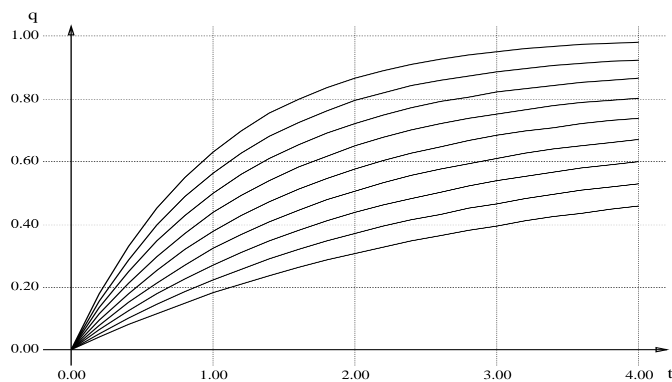

Fig. 1. Graphical representation of a CPP

2.1 Conditional Performance Profiles

To allow for efficient meta-level control of anytime algorithms, we describe their behavior by *conditional performance profiles* (CPP) [7]. A conditional performance profile captures the dependency of output quality on time allocation as well as on input quality. In [8], the reader can find a detailed discussion of various types of conditional performance profiles and their representation. To simplify the discussion of compilation, we will refer only to the *expected* CPP that maps computation time and input quality to the expected output quality.

Definition 1 *The* ***conditional performance profile (CPP)****, of an algorithm* $\mathcal{A}$ *is a function*

$$CPP_{\mathcal{A}} \; : \; \mathcal{Q}_{in} \times \mathcal{R}^{+} \rightarrow \mathcal{Q}_{out}$$

that maps input quality and computation time to the expected quality of the results.

Figure 1 shows a typical CPP. Each curve represents the expected output quality as a function of time for a *given* input quality.

2.2 Interruptible and Contract Algorithms

We distinguish between interruptible and contract algorithms. An interruptible algorithm is an anytime algorithm that can be interrupted at any time. A contract algorithm offers a similar tradeoff between computation time and quality of results, but the total execution time must be known in advance. If interrupted at any point before the termination of the contract time, it may yield no useful results. In many applications, interruptible algorithms are more desirable, but they are also more complicated to construct. In [6] we show that a simple, general construction can produce an interruptible version for any given contract algorithm with only a small, constant penalty. This theorem allows us to concentrate on the construction of contract algorithms for complex decision-making tasks.

2.3 A Library of Anytime Algorithms

Programming with anytime algorithms requires access to their performance profiles. For this purpose, we developed the notion of the *anytime library.* The library stores not only the performance profiles of elementary anytime algorithms, but also the results of the compilation process. The construction of a package of anytime algorithms, accompanied by a library of performance profiles, is an important first step toward the integration of approximate computation with standard software engineering techniques. Behind the anytime library concept lies the vision of a wide-spread use of standard anytime algorithms for essentially every basic computational problem from sorting and searching to complex reasoning tasks. The package of anytime algorithms supplements the compilation and monitoring techniques with flexible building blocks that simplify the

development of complex systems. In current work on the development of an anytime library, we are studying such issues as machine independent representation of performance profiles, standard interface operations, and library maintenance tools.

3 Composition of Anytime Algorithms

Modularity is widely recognized as an important issue in system design and implementation. However, the use of anytime algorithms as the components of a system presents a special type of scheduling problem. The question is how much time to allocate to each component in order to maximize the output quality of the complete system. We refer to this problem as the anytime algorithm *composition problem*.

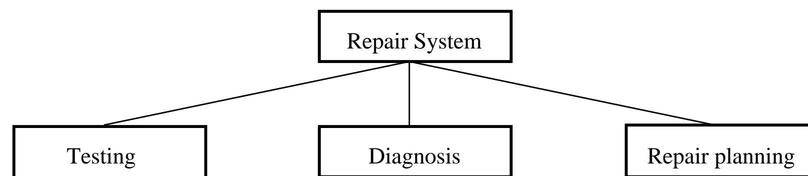

Fig. 2. A composite system for automatic diagnosis and repair

Consider for example an automated diagnosis and repair system whose structure is shown in Figure 2. The system is composed of three anytime modules that perform testing, diagnosis and repair planning. Given the conditional performance profiles of the components and a certain time allocation, the composition problem is to determine the amount of time to be allocated to each component so as to maximize the overall quality.

Solving the composition problem is important because it introduces a new kind of modularity into intelligent system development by allowing for separation between the development of the performance components and the optimization of their performance. In addition, by mechanizing the composition of anytime algorithms, we simplify the programming task.

3.1 The Compilation Problem

Given a system composed of anytime algorithms, the compilation process is designed to: (a) determine the optimal performance profile of the complete system; and (b) insert into the composite module the necessary code to achieve that performance. The precise process definition depends on various factors such as the structure of the composite program, the type of performance profiles and their representation, and the type of elementary anytime algorithms used. Depending on these factors, different types of compilation and monitoring strategies are

needed. To simplify the discussion in this paper, we will consider only the problem of producing contract algorithms when the conditional performance profiles of the components are given. The reader can find a broader analysis of compilation and monitoring in [8].

Let $\mathcal{F}$ be a set of anytime functions. Assume that all function parameters are passed by value and that functions have no side-effects (as in pure functional programming). Let $\mathcal{I}$ be a set of input variables. Then, the notion of a composite expression is defined as follows:

Definition 2 *A* ***composite expression*** *over $\mathcal{F}$ with input $\mathcal{I}$ is:*

1. *An expression $f(i_1, ..., i_n)$ where $f \in \mathcal{F}$ is a function of n arguments and $i_1, ..., i_n \in \mathcal{I}$.*
2. *An expression $f(g_1, ..., g_n)$ where $f \in \mathcal{F}$ is a function of n arguments and each g_i is a composite expression or an input variable.*

For example, the expression $A(B(x), C(D(y)))$ is a composite expression over $\{A, B, C, D\}$ with input $\{x, y\}$. Suppose that each function in $\mathcal{F}$ has a conditional performance profile associated with it that specifies the quality of its output as a function of time allocation and the qualities of its inputs. Given a composite expression of size n, the compiler needs to determine the mapping, $\mathcal{T} : t \rightarrow (t_1, ..., t_n)$, that specifies the optimal time allocation to the components for any given amount of time.

3.2 Global Compilation is Hard

Global compilation of composite expressions (GCCE) refers to solving the compilation problem as a global optimization problem. In [8], we prove the following result:

Theorem 3 *The GCCE problem is NP-complete in the strong sense.*

The proof is based on a reduction from the PARTIALLY ORDERED KNAPSACK problem. This result has led us to search for efficient compilation techniques that can be applied to large programs.

3.3 Local Compilation

Local compilation is the process of finding the best performance profile of a module based on the performance profiles of its *immediate* components. If those components are not elementary anytime algorithms, then their performance profiles are determined using local compilation. Local compilation replaces the global optimization problem with a set of simpler, local optimization problems and thus reduces the complexity of the whole problem. Unfortunately, local compilation cannot be applied to every composite expression. If the expression has repeated subexpressions, then computation time should be allocated only once to evaluate all identical copies. However, the following three assumptions result in an efficient local compilation process that is also optimal [8]:

1. **The tree-structured assumption** – the input composite expression has no repeated subexpressions (thus it can be represented as a directed tree).
2. **The input-monotonicity assumption** – the output quality of each module increases when the quality of the input improves.
3. **The bounded-degree assumption** – the number of inputs to each module is bounded by a constant, b.

The first assumption is needed so that local compilation can be applied. The second assumption is needed to guarantee the optimality of the resulting performance profile. And the third assumption is needed to guarantee the efficiency of local compilation. Using an efficient tabular representation of performance profiles, we can perform local compilation in constant time and reduce the overall complexity of compilation to be linear in the size of the program. We have also developed a number of *approximate* local compilation techniques that work efficiently on DAGs and on a variety of additional programming constructs such as conditional statements and loops.

4 Meta-Level Control

Monitoring plays a central role in anytime computation. We have examined the monitoring problem in two types of domains. One type is characterized by the predictability of utility change over time. High predictability of utility allows an efficient use of contract algorithms modified by various strategies for contract adjustment. The second type of domains is characterized by rapid change and a high level of uncertainty. In such domains, scheduling interruptible algorithms based on the value of computation criterion becomes essential. Due to limited space, we discuss here only the control of contract algorithms.

Suppose that a system composed of anytime algorithms is compiled into a contract algorithm, $\mathcal{A}$. The conditional performance profile of the system is $Q_{\mathcal{A}}(q,t)$ where q is the input quality and t is the time allocation. Assume that $Q_{\mathcal{A}}(q,t)$ represents, in the general case, a probability distribution. When a discrete representation is used, $Q_{\mathcal{A}}(q,t)[q_i]$ denotes the probability of output quality q_i.

Let S_0 be the current state of the domain, let S_t represent the state of the domain at time t, and let q_t represent the quality of the result of the contract anytime algorithm at time t. $U_{\mathcal{A}}(S,t,q)$ represents the utility of a result of quality q in state S at time t. This utility function is given as part of the problem description. The purpose of the monitor is to maximize the expected utility of the result, that is, to find t for which $U_{\mathcal{A}}(S_t,t,q_t)$ is maximal.

The first step is to calculate the initial contract time. Due to the uncertainty concerning the quality of the result of the algorithm, the expected utility of the result at time t is represented by:

$$U'_{\mathcal{A}}(S_t,t) = \sum_i Q_{\mathcal{A}}(q,t)[q_i] U_{\mathcal{A}}(S_t,t,q_i) \quad (1)$$

The probability distribution of future output quality is provided by the performance profile of the algorithm. Hence, an initial contract time, t_c, can be determined before the system is activated by solving the following equation:

$$t_c = arg \max_t \{U'_{\mathcal{A}}(S_t, t)\} \tag{2}$$

One monitoring approach is to allocate the initial contract time to the components based on the compiled performance profile of the system. This approach can be improved by revising the initial allocation based on the *actual* change in the domain and the *actual* quality of partial results.

In some cases, it is possible to separate the value of the results from the time used to generate them. In such cases, one can express the comprehensive utility function, $U_{\mathcal{A}}(S, t, q)$, as the difference between two functions:

$$U_{\mathcal{A}}(S_t, t, q) = V_{\mathcal{A}}(S_0, q) - Cost(S_0, t) \tag{3}$$

where $V_{\mathcal{A}}(S, q)$ is the value of a result of quality q in a particular state S (termed *intrinsic utility* [5]) and $Cost(S, t)$ is the cost of t time units provided that the current state is S. Similar to the expected utility, the expected intrinsic utility for any allocation of time can be calculated using the performance profile of the algorithm:

$$V'_{\mathcal{A}}(S, t) = \sum_i Q_{\mathcal{A}}(q, t)[q_i] V_{\mathcal{A}}(S, q_i) \tag{4}$$

Finally, the initial contract time can be determined by solving the following equation:

$$t_c = arg \max_t \{V'_{\mathcal{A}}(S_0, t) - Cost(S_0, t)\} \tag{5}$$

Once an initial contract time is determined, several monitoring policies can be applied. In particular, we have studied two strategies for contract adjustment. The first strategy re-allocates residual time among the remaining modules once the result of a module becomes available. The second strategy adjusts the original contract time. Both of these methods are activated after the termination of each elementary component. They consider the output of that component as an input to a smaller residual system composed of the remaining anytime algorithms. At that point, a better contract time can be determined that takes into account the actual quality of the intermediate results generated so far. The higher the level of domain uncertainty, the more beneficial is the use of contract adjustments.

5 Applications

The advantages of compilation and monitoring of anytime algorithms have been demonstrated in several domains. In this section we summarize two applications.

5.1 Mobile Robot Navigation

One of the fundamental problems facing any autonomous mobile robot is the capability to plan its own motion using noisy sensory data. We have developed a simulated robot navigation system by composing two anytime modules [9]. The first module, a vision algorithm, creates a local domain description whose quality reflects the probability of correctly identifying each basic position. Each position can be free or blocked by an obstacle. The second module, a hierarchical planning algorithm, creates a path between the current position and the goal position. The quality of a plan reflects the ratio between the shortest path and the path that the robot generates when guided by the plan.

Anytime hierarchical planning is based on performing coarse-to-fine search that allows the algorithm to find quickly a low quality plan and then repeatedly refine it by replanning a segment of the plan in more detail. Hierarchical planning is complemented by an execution architecture that can take advantage of abstract plans. The execution architecture uses plans as advice to direct the base level execution mechanism. In practice, uncertainty makes it hard to use plans except as a guidance mechanism.

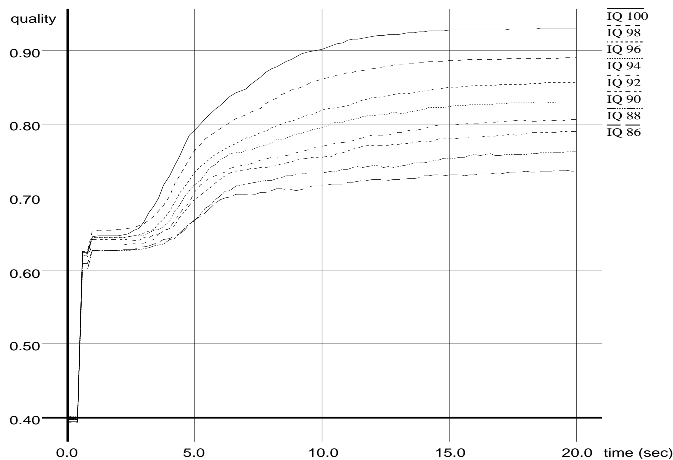

Fig. 3. The CPP of the anytime planner

The conditional performance profile of the hierarchical planner is shown in Figure 3. Each curve shows the expected plan quality as a function of run-time for a particular quality of visual data. An active monitoring scheme has also been developed to use the compiled performance profile of the system and the time-dependent utility function of the robot in order to allocate time to vision and planning so as to maximize overall utility.

An interesting result of this experiment was the fact that the anytime abstract planning algorithm produced high quality plans (approx. 10% longer than the

optimal path) with time allocation that was much shorter (approx. 30%) than the total run-time of a standard search algorithm (A^*). This fact shows that the flexibility of anytime algorithms does not necessarily require a compromise in overall performance.

5.2 Model-Based Diagnosis

Model-based diagnostic methods identify defective components in a system by a series of tests and probes. The goal is to locate the defective components using a small number of tests. The General Diagnostic Engine [2] (GDE) is a basic method for model-based diagnostic reasoning. In GDE, observations and a model of a system are used in order to derive *conflicts* (a conflict is a set of components of which at least one is defective). These conflicts are transformed to *diagnoses* (a diagnosis is a set of defective components that might explain the erroneous behavior of the system). The process of observation, conflict generation, transformation to diagnoses, and generation of probe advice is repeated until the defective components are identified. GDE has a high computational complexity – $O(2^n)$, where n is the number of components. As a result, its applicability is limited to small-scale applications. To overcome this difficulty, Bakker and Bourseau have developed a method, called Pragmatic Diagnostic Engine (PDE), whose computational complexity is $O(n^2)$. PDE is similar to GDE, except for omitting the stage of generating all diagnoses before determining the best measurement-point. Probe advice is given on the basis of the most relevant conflicts, called *obvious* and *semi-obvious* conflicts (an obvious (semi-obvious) conflict is a conflict that is computed using no more than one (two) observed outputs).

Pos [4] has applied our compilation technique to implement the PDE architecture. PDE can be viewed as a composition of two anytime modules. In the first module, a subset of all conflicts is determined. Pos implemented this module by a contract form of breadth-first search. The second module consists of a repeated loop that determines which measurement should be taken next, takes that measurement, and assimilates the new information into the current set of conflicts. Two versions of the diagnostic system have been implemented: one by constructing a contract algorithm and the other by making the contract system interruptible using our reduction technique. The actual slow down factor of the interruptible system was approximately 2, much better than the worst case theoretical ratio of 4.

6 Conclusion

We presented a decision-theoretic model for meta-level control of anytime algorithms. It offers both a methodological and a practical contribution to the field of real-time deliberation in intelligent systems. The main aspects of this contribution include: (1) simplifying the design and implementation of complex intelligent systems by separating the design of the performance components from

the optimization of performance; (2) mechanizing the composition process and the monitoring process; and (3) constructing machine independent intelligent systems that can automatically adjust resource allocation to yield optimal performance.

The study of anytime computation is a promising and growing field in artificial intelligence. Some of the primary research directions in this field include: extending the scope of compilation by studying additional programming structures, producing a large library of reusable anytime algorithms, and developing additional, larger applications that demonstrate the benefits of the methodology.

Acknowledgements

Much of this work was done in collaboration with my graduate advisor, Stuart Russell, when I was a student at U.C. Berkeley. Tom Dean inspired my initial interest in anytime algorithms and has continued to provide important insights and comments. Current support for this project is provided by a Faculty Research Grant from the University of Massachusetts.

References

1. Dean, T. L., Boddy, M.: An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 49–54, Minneapolis, Minnesota, 1988.
2. de Kleer, J., Williams, B. C.: Diagnosing multiple faults. *Artificial Intelligence* **32**:97–130, 1987.
3. Horvitz, E. J., Breese, J. S.: Ideal partition of resources for metareasoning. Technical Report KSL-90-26, Stanford Knowledge Systems Laboratory, Stanford, California, 1990.
4. Pos, A.: *Time-Constrained Model-Based Diagnosis.* Master Thesis, Department of Computer Science, University of Twente, The Netherlands, 1993.
5. Russell, S. J., Wefald, E. H.: Principles of metareasoning. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, R.J. Brachman *et al.* (eds.), San Mateo, California: Morgan Kaufmann, 1989.
6. Russell, S. J., Zilberstein, S.: Composing real-time systems. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pp. 212–217, Sydney, Australia, 1991.
7. Zilberstein, S., Russell, S. J.: Efficient resource-bounded reasoning in AT-RALPH. In *Proceedings of the First International Conference on AI Planning Systems*, pp. 260–266, College Park, Maryland, 1992.
8. Zilberstein, S.: *Operational Rationality through Compilation of Anytime Algorithms.* Ph.D. dissertation, Department of Computer Science, University of California at Berkeley, 1993.
9. Zilberstein, S., Russell, S. J.: Anytime sensing, planning and action: A practical model for robot control. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 1402–1407, Chambery, France, 1993.

This article was processed using the LaTeX macro package with LLNCS style