AI Magazine Volume 17 Number 3 (1996) (© AAAI)

Using Anytime Algorithms in Intelligent Systems

Shlomo Zilberstein

■ Anytime algorithms give intelligent systems the capability to trade deliberation time for quality of results. This capability is essential for successful operation in domains such as signal interpretation, real-time diagnosis and repair, and mobile robot control. What characterizes these domains is that it is not feasible (computationally) or desirable (economically) to compute the optimal answer. This article surveys the main control problems that arise when a system is composed of several anytime algorithms. These problems relate to optimal management of uncertainty and precision. After a brief introduction to anytime computation, I outline a wide range of existing solutions to the metalevel control problem and describe current work that is aimed at increasing the applicability of anytime computation.

Anytime algorithms[1] are algorithms whose quality of results improves gradually as computation time increases. The term *anytime algorithm* was coined by Dean and Boddy in the mid-1980s in the context of their work on time-dependent planning (Dean and Boddy 1988; Dean 1987). Dean and Boddy introduced several deliberation scheduling techniques that make use of performance profiles (PPs) to make time-allocation decisions. A similar technique, termed *flexible computation,* was introduced by Horvitz (1990, 1987) to solve time-critical decision problems. This line of work is also closely related to the notion of limited rationality in automated reasoning and search (Russell and Wefald 1991, 1989; Doyle 1990; D'Ambrosio 1989). Within the systems community, a similar idea termed *imprecise computation* was developed by Jane Liu and others (1991). What is common to these research efforts is the recognition that the computation time needed to compute precise or optimal solutions will typically reduce the overall utility of the system. In addition, the appropriate level of deliberation can be situation dependent. Therefore, it is beneficial to build systems that can trade the quality of results against the cost of computation.

A rapid growth in the development of anytime algorithms in recent years has led to a number of successful applications in such areas as the evaluation of Bayesian networks (Wellman and Liu 1994; Horvitz, Suermondt, and Cooper 1989), model-based diagnosis (Pos 1993), relational database query processing (Vrbsky, Liu, and Smith 1990), constraint-satisfaction problems (Wallace and Freuder 1995), and sensor interpretation and path planning (Zilberstein 1996; Zilberstein and Russell 1993). This article describes the construction, composition, and control of such algorithms.

Anytime Algorithms

Anytime computation extends the traditional notion of a computational procedure by allowing it to return many possible approximate answers to any given input. The notion of approximate processing (Lesser, Pavlin, and Durfee 1988) and the use of satisficing techniques (Simon 1982) have proved useful in AI applications. What is special about anytime algorithms is the use of well-defined quality measures to monitor the progress in problem solving and allocate computational resources effectively. The binary notion of correctness is replaced with a multivalued quality measure associated with each answer. Various metrics have been used to measure the quality of the results produced by anytime algorithms. From a pragmatic point of view, it might seem useful to define a single type of quality measure to be applied to all anytime algorithms. Such a unifying approach can simplify the meta-

Copyright © 1996, American Association for Artificial Intelligence. All rights reserved. 0738-4602-1996 / $2.00

level control problem. However, in practice, different types of anytime algorithms tend to approach the exact result in completely different ways. The following three metrics have proved useful in anytime algorithm construction:

First is *certainty,* a measure of the degree of certainty that the result is correct. The degree of certainty can be expressed using probabilities, fuzzy set membership, or any other approach.

Second is *accuracy,* a measure of the degree of accuracy, or how close the approximate result is to the exact answer. Typically, with such algorithms, high quality guarantees that the error is below a certain upper bound.

Third is *specificity,* a metric of the level of detail of the result. In this case, the anytime algorithm always produces correct results, but the level of detail is increased over time.

> *Anytime computation extends the traditional notion of a computational procedure by allowing it to return many possible approximate answers to any given input.*

Desired Properties of Anytime Algorithms

Many existing programming techniques produce useful anytime algorithms. Examples include iterative deepening search, variable precision logic, and randomized techniques such as Monte Carlo algorithms and fingerprinting algorithms. For a survey of such programming techniques and examples, see Zilberstein (1993). Obviously, not every algorithm that can produce a series of approximate answers is a well-behaved anytime algorithm. Desired properties of anytime algorithms (from the standpoint of metalevel control) include the following features:

First is *measurable quality:* The quality of an approximate result can be determined precisely. For example, when the quality reflects the distance between the approximate result and the correct result, it is measurable as long as the correct result can be determined.

Second is *recognizable quality:* The quality of an approximate result can easily be determined at run time (that is, within a constant time). For example, when solving a combinatorial optimization problem (such as path planning), the quality of a result depends on how close it is to the optimal answer. In such a case, quality can be measurable but not recognizable.

Third is *monotonicity:* The quality of the result is a nondecreasing function of time and input quality. Note that when quality is recognizable, the anytime algorithm can guarantee monotonicity by simply returning the best result generated so far rather than the last generated result.

Fourth is *consistency:* The quality of the result is correlated with computation time and input quality. In general, algorithms do not guarantee a deterministic output quality for a given amount of time, but it is important to have a narrow variance so that quality prediction can be performed.

Fifth is *diminishing returns:* The improvement in solution quality is larger at the early stages of the computation, and it diminishes over time.

Sixth is *interruptibility:* The algorithm can be stopped at any time and provide some answer. Originally, this was the primary characteristic of anytime algorithms, but I show later that noninterruptible anytime algorithms, termed *contract algorithms,* are also useful.

Seventh is *preemptability:* The algorithm can be suspended and resumed with minimal overhead.

The different properties of the elementary anytime algorithms have a major effect on the complexity of composition and monitoring. These properties are typically summarized by the PP of the algorithm.

> **Example:** The *traveling salesman problem* (TSP) is a widely known combinatorial optimization problem where the application of anytime algorithms is useful. The problem involves a salesman that must visit *n* cities. If the problem is modeled as a complete graph with *n* vertexes, the solution becomes a *tour,* or Hamiltonian cycle, visiting each city exactly once, starting and finishing at the same city. The cost function, *Cost*(*i*, *j*), defines the cost of traveling directly from city *i* to city *j*. The problem is to find an *optimal tour,* that is, a tour with minimal total cost. The TSP is known to be NP complete; hence, it is hard to find an optimal tour when the problem includes a large number of cities. Iterative improvement algorithms can find a good approximation to an optimal solution and naturally yield an interruptible anytime algorithm.

Randomized tour improvement is an algorithm that repeatedly tries to make small changes in the current tour to reduce the overall cost (Lawler et al. 1987). In the general case of tour-improvement procedures, *r* edges in a feasible tour are exchanged for *r* edges not in the solution as long as the result remains a tour, and the cost of the tour is less than the cost of the previous tour. Figure 1 demonstrates one step of tour improvement. An existing tour, shown in figure 1a, visits the vertexes in the following order: *a*, *b*, *c*, *d*, *e*, and *f*. The algorithm selects two random edges of the

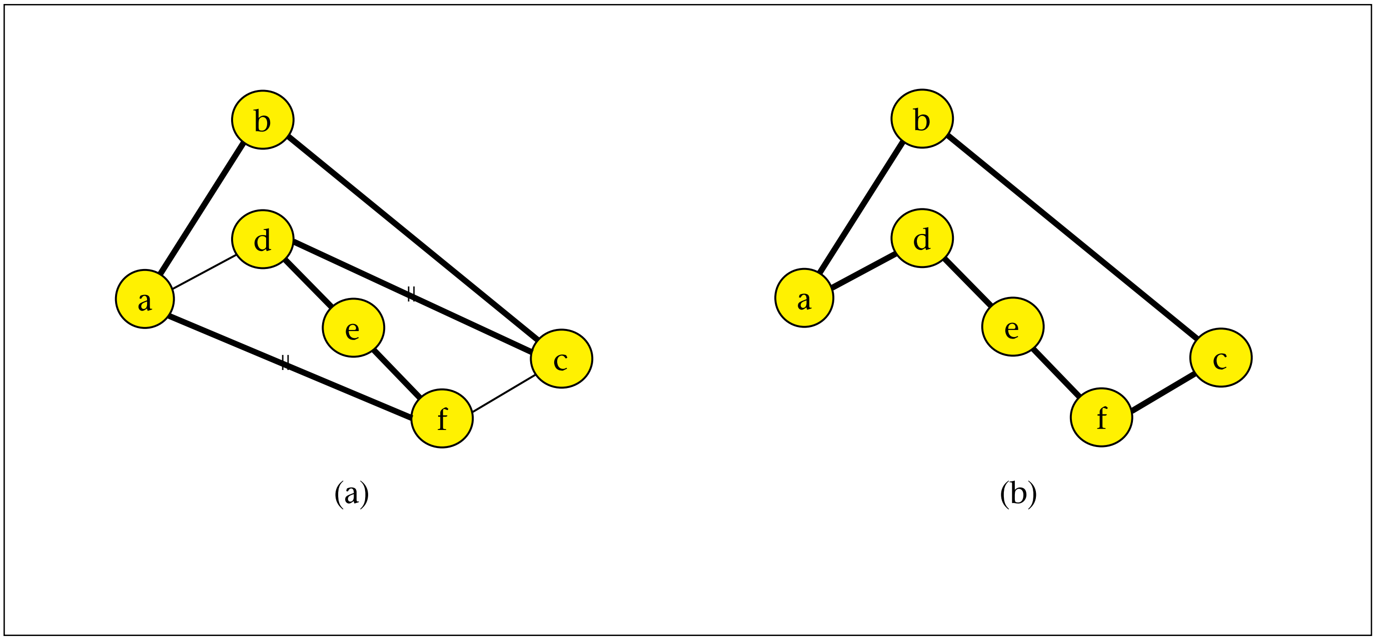

Figure 1. An Example of Tour Improvement.

graph—(c, d) and (f, a)—in this example and checks whether the following condition holds:

$$Cost(c, f) + Cost(d, a) < Cost(c, d) + Cost(f, a) \quad . (1)$$

If this condition holds, the existing tour is replaced by the new tour, shown in figure 1b: a, b, c, f, e, and d. The improvement condition guarantees that the new path has a lower cost. The algorithm starts with a random tour that is generated by simply taking a random ordering of the cities. Then, the algorithm tries to reduce the cost by a sequence of random improvements.

Performance Profiles

To allow for efficient metalevel control of anytime algorithms, their performance improvement over time must be summarized quantitatively. Boddy and Dean (1989) used PPs to characterize the output quality of an anytime path planner. Horvitz (1987) used a similar construct to describe the object-related value of flexible computation. These PPs generally describe the expected output quality as a function of run time.

> **Definition 1:** A PP of an anytime algorithm, Q(t), denotes the expected output quality with execution time t.

PPs are typically constructed empirically by collecting statistics on the performance of an algorithm over many input instances. The raw data that are collected specify the particular quality of results generated at a particular point of time. These data form the quality map of the algorithm.

Figure 2 shows the quality map of the randomized tour-improvement algorithm. It summarizes the results of many activations of the algorithm with randomly generated input instances (including 50 cities). Each point (t, q) represents an instance for which quality q was achieved with run-time t. The quality of results in this experiment measures the percentage of tour-length reduction with respect to the initial tour. These statistics form the basis for the construction of the PP of the algorithm. The resulting expected PP is shown in figure 3.

The basic notion of a PP has been extended in two ways using conditional performance profiles (CPPs) (Zilberstein and Russell 1996; Zilberstein 1993). First, a CPP describes the dependence of output quality on run time as well as on input quality. Second, a CPP specifies the probability distribution of output quality rather than the expected value. Thus, CPPs provide a more informative performance description that facilitates better algorithm composition and monitoring.

> **Definition 2:** A CPP of an anytime algorithm, $Pr(q_{out} \mid q_{in}, t)$, denotes the probability of getting a solution of quality when the algorithm is activated with input of quality q_{out} and execution time t.

Intermediate representations of performance information have also been used in

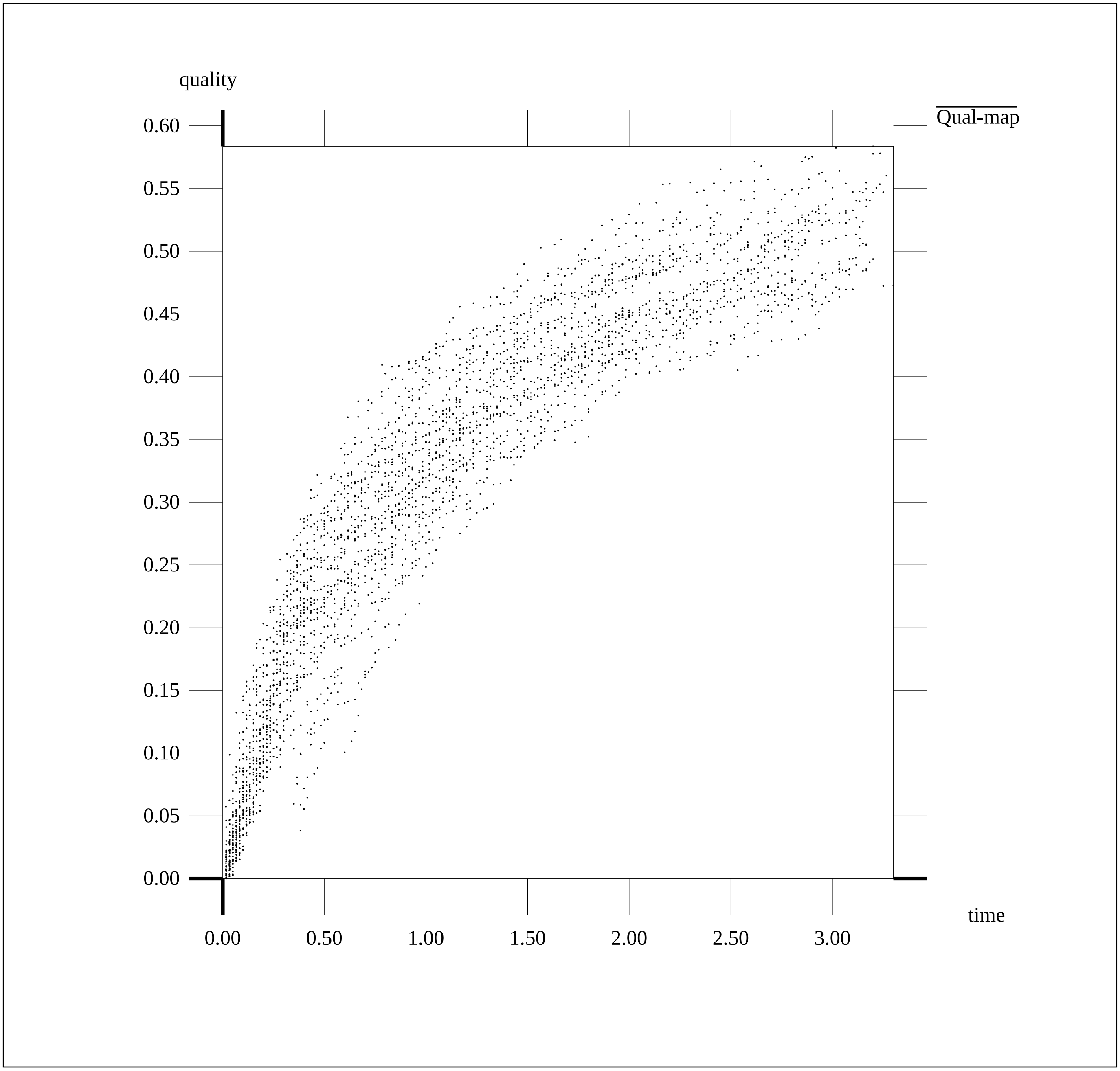

Figure 2. The Quality Map of the Randomized Tour-Improvement Algorithm.

The widespread use of anytime computation depends not only on the computational benefits of such techniques but also on the capability to construct general-purpose, reusable anytime algorithms. A programming environment to support anytime computation must include tools for automatic construction of PPs as well as tools for composition, activation, and monitoring.

applications. For example, Pos (1993) used a construct called a *statistical performance profile* that records both upper and lower bounds and expected output quality for any time allocation. More recently, the notion of a dynamic PP has been developed (Hansen and Zilberstein 1996). It follows the observation that quality improvement can better be predicted at run time when the quality of the currently available result is taken into account. The behavior of the anytime algorithm is modeled as a Markov process in which discrete states are associated with a particular output quality. The quality at time $t + 1$ depends on the quality at time t as well as on the current time. As with CPPs, the probability distribution can be constructed empirically.

> **Definition 3:** A *dynamic performance profile* (DPP) of an anytime algorithm, $Pr(q_j|q_i, \Delta t)$, denotes the probability of getting a solution of quality q_j by resuming the algorithm for time interval Δt when the currently available solution has quality q_i.

PPs are typically monotone functions of time, and they can be approximated using a certain family of functions. Once the quality map is known, the performance information can be derived by various curve-fitting techniques. For example, Boddy and Dean (1989) used the function $Q(t) = 1 - e^{-\lambda t}$ to model the expected performance of their anytime planner. CPPs can be approximated with a similar method by using a certain family of distributions. Another approach is to represent a CPP by a table representing a discrete probability distribution. The size of the table is a system parameter that controls the accuracy of performance information.

Interruptible and Contract Algorithms

A useful distinction has been made between two types of anytime algorithm, namely, *interruptible* and *contract* algorithms (Russell and Zilberstein 1991). An interruptible algorithm can be interrupted at any time to produce results whose quality is described by its PP. A contract algorithm offers a similar trade-off between computation time and quality of results, but the total allocation must be known in advance. If interrupted at any point before the termination of the contract time, it might not yield any useful results. Interruptible algorithms are in many cases more appropriate for the application, but they are also more complicated to construct. It has been shown that a simple, general construction can produce an interruptible version for any given contract algorithm with only a small, constant penalty (Russell and Zilberstein 1991). This theorem allows one to concentrate on the construction of contract algorithms for complex decision-making tasks and then convert them into interruptible algorithms using a standard transformation. It is also important to note that for many applications (that are characterized more precisely in Run-Time Monitoring), contract algorithms are the desired end product.

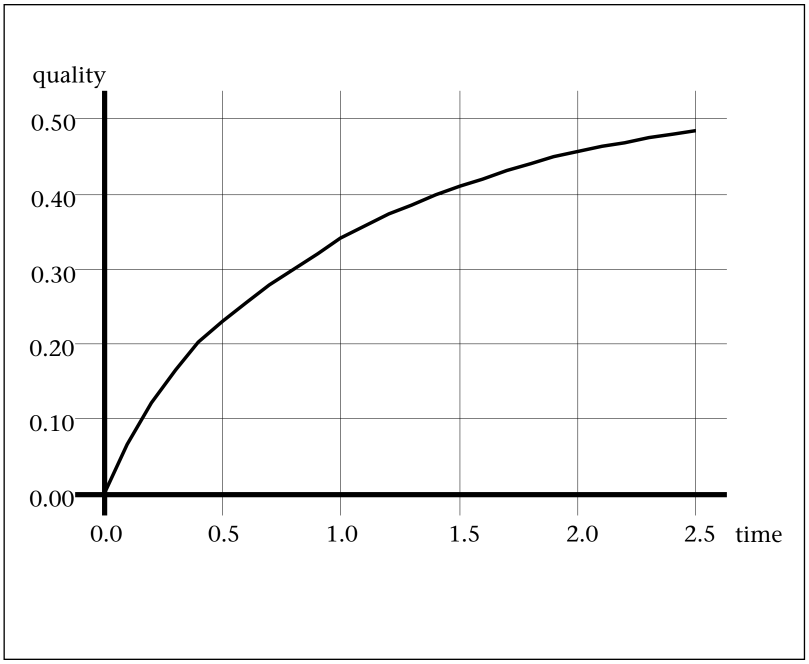

Figure 3. The Expected Performance Profile of the Algorithm.

Programming Tools

The widespread use of anytime computation depends not only on the computational benefits of such techniques but also on the capability to construct general-purpose, reusable anytime algorithms. A programming environment to support anytime computation must include tools for automatic construction of PPs as well as tools for composition, activation, and monitoring. One such effort is reported in Grass and Zilberstein (1996). An anytime library is used to keep the metalevel information that is essential for both composition and monitoring. The construction of a standard anytime package that comes with a library of PPs is an important first step toward the integration of anytime computation with standard software-engineering practices.

Composition of Anytime Algorithms

The use of anytime algorithms as the components of a modular system presents a special type of deliberation scheduling problem. The question is how much time to allocate to each component to maximize the output quality of the complete system. This problem is termed the *anytime algorithm composition problem* (Zilberstein 1993). A large part of the composition problem can be solved offline by analyzing the PPs of the components of the system.

Consider, for example, a speech-recognition system whose structure is shown in figure 4. Each box represents an elementary anytime algorithm whose conditional PP is given. The system is composed of three main components: First, the speaker is classified in terms of gender and accent. Then, a recognition algorithm suggests several possible matching utterances. Finally, the linguistic validity of each possible utterance is determined, and the best interpretation is selected. The composition problem is calculating, for any given total allocation, how much time to allocate to each elementary component of the composite system to maximize the quality of the utterance recognition.

The question is how much time to allocate to each component to maximize the output quality of the complete system. This problem is termed the anytime algorithm composition problem.

Solving the composition problem is important for several reasons: First, 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. Traditional design methodologies require that the performance components meet certain time constraints. In complex intelligent systems, these constraints are hard to derive at design time. The result is a hand-tuning process that might or might not culminate with a working system. Anytime computation offers an alternative to this approach. By developing performance components that are responsive to a wide range of time allocations, one avoids the commitment to a particular performance level that might fail the system.

The second reason relates to the difficulty of programming with anytime algorithms (or approximation techniques in general). To make a composite system optimal (or even executable), one must control the activation and interruption of the components. Therefore, an automated solution of the composition problem simplifies the task of the programmer.

Compilation

One solution to the anytime algorithm composition problem is based on an offline compilation approach (Zilberstein and Russell 1996; Zilberstein 1993). Given a system composed of anytime algorithms, the compilation process is designed to (1) determine the optimal PP of the complete system and (2) prepare any additional control information that would simplify the run-time monitoring task. The solution to this problem and its complexity depends on the following factors:

First is composite program structure: What type of programming operators are used to compose anytime algorithms?

Second is the type of PPs: What kind of PPs are used to characterize elementary anytime algorithms?

Third is the type of anytime algorithms: What type of elementary anytime algorithms are used as input? What type of anytime algorithm should the resulting system be?

Fourth is the type of monitoring: What type of run-time monitoring is used to activate and interrupt the execution of the elementary components?

Fifth is the quality of intermediate results: What access does the monitoring component have to intermediate results? Is the actual quality recognizable?

The Complexity of Compilation

The compilation problem is generally an NP-complete optimization problem that requires finding a schedule of a set of components that yields maximal output quality. To analyze the complexity of compilation of composite expressions, one can consider the decision-problem variant of the problem. Given a composite expression, the conditional PPs of its components, and a total time allocation *B*, the decision problem is whether a schedule of the components exists that yields output quality greater than or equal to *K*. The following result is proved in Zilberstein (1993).

> **Theorem 1:** The compilation of composite expressions is NP-complete in the strong sense.

The proof is based on a reduction from the partially ordered knapsack problem. The meaning of this result is that the application of the compilation technique is limited to small programs. To address this problem, an efficient local compilation technique was developed.

Local Compilation

Local compilation is the process of finding the best PP of a module based on the PPs of its

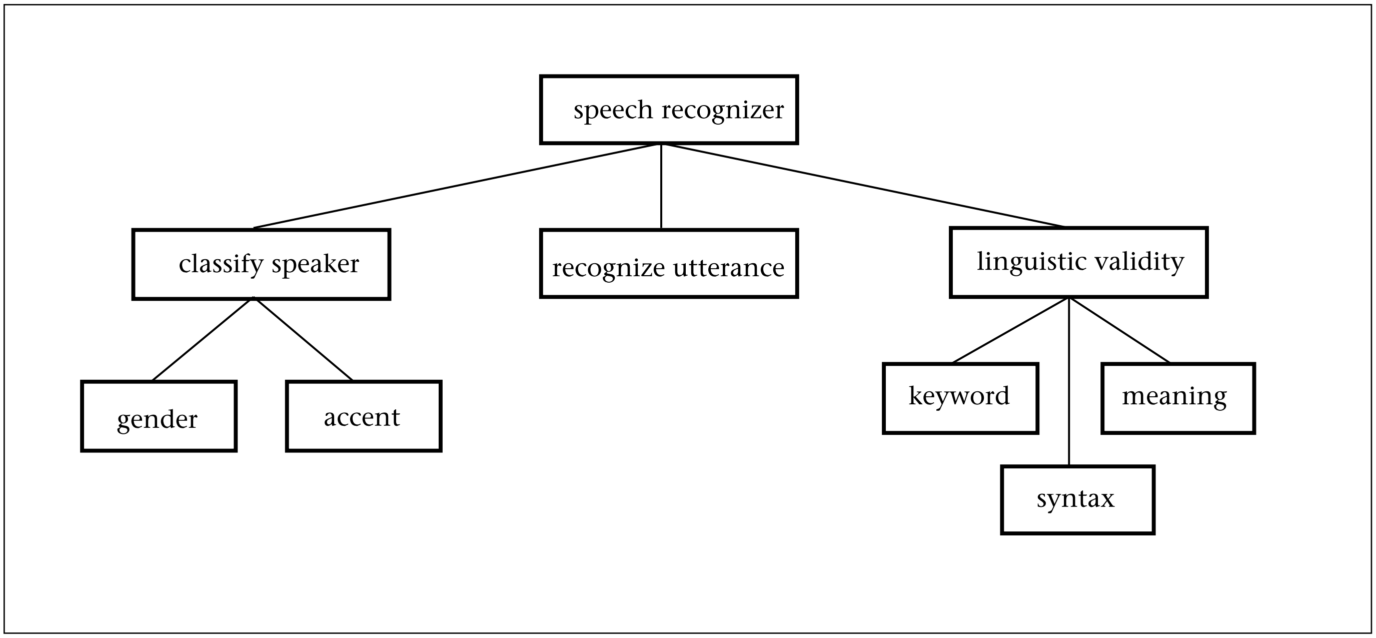

Figure 4. An Example of a Composite Module for Speech Recognition.

immediate components. If those components are not elementary anytime algorithms, then their PPs 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. The following two assumptions are needed for an effective use of local compilation:

> **Assumption 1 (tree structured):** The input expression has no repeated subexpressions; thus, its directed acyclic graph (DAG) representation is a tree.

> **Assumption 2 (bounded degree):** The number of input to each module is bounded by a small constant.

The tree-structured assumption is needed so that local compilation can be applied. The bounded-degree assumption is needed to guarantee the efficiency of local compilation. Under these two assumptions and with a compact tabular representation of PPs, local compilation can be performed in constant time, and the overall complexity of compilation becomes linear in the size of the program (Zilberstein 1993). Moreover, under certain conditions, the result of local compilation is globally optimal.

> **Theorem 2** (optimality of local compilation of deterministic CPPs): Local compilation of an arbitrary composite expression with deterministic PPs is globally optimal when each PP satisfies the input monotonicity assumption.

The input monotonicity assumption requires that the output quality of each module increase when the quality of its input improves. Intuitively, this property is desired of every anytime algorithm. For the more general case of probabilistic PPs, the following result was proved (Zilberstein 1995).

> **Theorem 3** (optimality of local compilation of probabilistic CPPs): Local compilation of an arbitrary composite expression with probabilistic PPs is globally optimal when each PP satisfies the input linearity assumption.

Input linearity is a stronger (and not very realistic) assumption. It requires that output quality improve linearly with input quality. However, a piecewise linear approximation of CPPs is a reasonable approach that allows the application of local compilation and maintains near-optimal results.

Approximate Compilation Techniques

The one assumption that restricts the application of local compilation is the tree-structured assumption, which excludes the possibility of repeated subexpressions. Consider, for example, the expression $F = E(D(B(A(x)), C(A(x))))$, where A, B, C, D, and E represent elementary anytime algorithms. Local compilation is only possible when one can repeatedly break a program into subprograms whose execution intervals are disjoint, so that allocating a certain amount of time to one subprogram does not affect the evaluation and quality of the other subprograms. This property does not hold when subprograms include identical subexpressions. In the example here, *B* and *C* are the two components of *D* whose time allocations cannot be considered independently because they both use the same subexpression, *A*(*x*).

A number of approximate compilation techniques have been developed to address this problem (Zilberstein 1993). Some techniques work efficiently on DAGs and produce near-optimal schedules. Other techniques guarantee optimality when the number of repeated subexpressions is small. Compilation of additional programming constructs such as conditional statements and loops has also been analyzed. To summarize, a wide range of compilation techniques have been developed that can efficiently produce the PP of a composite system based on the PPs of its components.

Run-Time Monitoring

Monitoring plays a central role in anytime computation. The purpose of run-time monitoring is to reduce the effect of uncertainty on the performance of the system. In general, two primary sources of uncertainty affect the operation of intelligent systems. The first source is internal to the system. It is caused by the unpredictable behavior of the system itself that leads to variability in solution quality. The second source is external. It is caused by unpredictable changes in the environment in which the system operates. These two sources of uncertainty are characterized by two separate knowledge sources. Uncertainty regarding the performance of the system is characterized by its CPP. Uncertainty regarding the future state of the environment is characterized by the model of the environment. The appropriate type of monitoring will typically depend on the source of uncertainty and the degree of uncertainty. Previous work on the monitoring problem (Hansen and Zilberstein 1996; Boddy and Dean 1994, 1989; Garvey and Lesser 1994; Zilberstein 1993; Horvitz 1990; Horvitz and Breese 1990) focused on the following questions:

First, how much time should be allocated to each of the components of the system to maximize its overall utility?

Second, how much variance in the performance of the system justifies using run-time monitoring rather than determining a fixed running time when the system is activated?

Third, how should the variance in the performance of the algorithm and the cost of monitoring affect the frequency of monitoring?

Fourth, when solution quality is hard to calculate, what degree of approximation should be used by the monitor? How does approximation of solution quality degrade the effectiveness of monitoring?

Fifth, is it better to monitor periodically or to monitor more frequently toward the algorithm's expected stopping time?

Work on these problems has produced three general strategies for solving the monitoring problem. The remaining three subsections describe these monitoring strategies.

The Fixed-Contract Approach

The *fixed-contract approach* to monitoring is based on calculating the best resource allocation to the system prior to its activation. This approach can be used when the complete system is compiled into a contract algorithm and when the level of variance in the algorithm performance is small.

> **Definition 4:** A domain is said to have predictable utility if the utility of a result can be determined for any future time once the current state of the domain is known.

Predictable utility is a property of the domain that gives monitoring the capability to determine the exact value of results of a particular quality at any future time. In principle, the state of the domain can change (possibly in an unpredictable manner), but utility can still be predictable. The analysis of such domains (Zilberstein 1993) demonstrates theorem 4.

> **Theorem 4** (optimality of monitoring of contract algorithms): The fixed-contract monitoring strategy is optimal when the domain has predictable utility, and the system has a deterministic PP.

Because contract algorithms are easier to construct and monitor, it is useful to use them in domains that have near predictable utility. Two modifications of the fixed-contract approach have been developed to address the uncertainty in such domains (Zilberstein 1993). The first modification involves reallocation of residual time among the remaining anytime algorithms. Suppose that a system, composed of several elementary contract algorithms, is compiled into a contract algorithm. Because the results of the elementary contract algorithms are not available during their execution, the only point of time where monitoring can take place is between activations of the elementary components. Based on the structure of the system, an execution order can be defined for the elementary components. The execution of any elementary component can be viewed as a transformation of a node in the graph representing the program from a computational node to an external input of a certain quality. The quality of the new input is only known when the corresponding elementary component terminates. Based on the actual quality, the remaining time (with respect to the global contract) can be reallocated among the remaining computational components to yield a performance improvement.

The second modification of the fixed-contract approach involves adjustments to the original contract time. As before, once an elementary component terminates, the monitor can consider its output as an input to a smaller residual system composed of the remaining anytime algorithms. By recalculating a new contract time for the residual system, a better contract time can be determined that takes into account the actual quality of the intermediate results generated so far. A similar line of work, *design-to-time scheduling,* is described in Garvey and Lesser (1993).

The Marginal Value of Computation

We turn now to the problem of monitoring interruptible anytime computation. The use of interruptible algorithms is necessary in domains whose utility function is not predictable. Such domains are characterized by nondeterministic rapid change. Medical diagnosis in an intensive care unit, stock market trading, and vehicle control on a highway are examples of such domains. Many possible events can change the state of such domains, and the timing of their occurrence is essentially unpredictable. Consequently, accurate projection into the far future is limited, and the previous contract approach fails.

One solution to this problem is based on calculating the marginal value of computation, which is the expected utility gain as a result of an additional computation step. This approach has been developed by Russell and Wefald (1989) for decision-theoretic control of search. In our case, the monitor must calculate the difference between the expected utility after one time increment (taking into account the improvement in solution quality as well as the change in the state of the environment) and the current expected utility (based on the current solution quality and the current state of the environment). Deliberation is interrupted once this value becomes negative. This myopic approach leads to an optimal stopping time under the following conditions (Zilberstein 1993):

> **Theorem 5** (optimality of monitoring of interruptible algorithms): Monitoring interruptible algorithms using the value of computation criterion is optimal when the PP is monotonically increasing and concave down, and the cost of time is monotonically increasing and concave up.

These conditions amount to an anytime algorithm whose quality improvement diminishes over time and a comprehensive utility function for which the cost of delay in action increases over time.

Monitoring Policies

The previous approach to monitoring based on the value of computation does not take monitoring time into account. This assumption is justified when calculating the marginal value of computation is simple and fast. However, in some situations, monitoring involves complex computation. Estimating both the current state of the environment and the current solution quality can be a complex problem that the monitor must solve. In such cases, one cannot ignore the cost of monitoring.

One approach that takes the cost of monitoring into account is based on modeling the progress of an interruptible anytime algorithm as a Markov process (Hansen and Zilberstein 1996). States of the process are identified with particular solution qualities, and transitions between states correspond to improvement in solution quality as a result of a small increment in computation time. The dynamic PP of the algorithm defines the transition probabilities. This approach to monitoring anytime algorithms allows one to calculate an offline monitoring policy that determines an optimal action for each solution quality. Possible actions are (1) resume the execution of the algorithm for a certain time interval and then monitor again, (2) resume the execution of the algorithm for a certain time interval and then stop, or (3) stop the execution of the algorithm. A simple example of such a monitoring policy is shown in table 1. The table specifies, for any given time step and solution quality, the amount of additional time to be allocated to the algorithm. The letter *M* indicates whether

Current research efforts … are aimed at extending the scope of compilation by studying additional programming structures, producing a large library of reusable anytime algorithms, and developing additional applications that demonstrate the benefits of anytime computation.

Table 1. Optimal Monitoring Policy Based on Actual Solution Quality.

The table specifies, for any time step and solution quality, how much time to be allocated to the algorithm and whether monitoring should be performed afterward.

			time-step						
quality	start	…	5	6	7	8	9	10	11
5			0	0	0	0	0	0	0
4			1M	1M	1M	1M	1M	1	0
3			1M	1M	1M	1M	1M	1	0
2			3M	3M	3M	3M	2	1	0
1			4M	5	4	3	2	1	0
0	5M		6	5	4	3	2	1	0

monitoring should be performed after the time allocation. When solution quality cannot be determined at run time, a similar policy can be developed based on estimated solution quality (Hansen and Zilberstein 1996).

Conclusion

The study of anytime computation is a promising and growing field in AI and real-time systems. This article presented several solutions to the metalevel control problems that arise in anytime computation. The decision-theoretic techniques for composition and monitoring of anytime algorithms offer several important advantages: They simplify the design and implementation of complex intelligent systems by separating the design of the performance components from the optimization of performance, they mechanize and optimize composition and monitoring, and they facilitate the construction of machine-independent intelligent systems that can automatically adjust resource allocation to yield optimal performance. Current research efforts in this field are aimed at extending the scope of compilation by studying additional programming structures, producing a large library of reusable anytime algorithms, and developing additional applications that demonstrate the benefits of anytime computation.

Acknowledgments

My initial work on anytime algorithms was conducted in collaboration with my graduate adviser, Stuart Russell, at the University of California at Berkeley. Tom Dean inspired my initial interest in anytime algorithms and has continued to provide stimulating feedback. Current support for this work is provided in part by the National Science Foundation under grant IRI-9409827 and in part by Rome Laboratory, United States Air Force, under grant F30602-95-1-0012.

Note

[1] A collection of papers on anytime algorithms is available online at http://anytime.cs.umass.edu/~shlomo/.

References

Boddy, M., and Dean T. L. 1994. Deliberation Scheduling for Problem Solving in Time-Constrained Environments. *Artificial Intelligence* 67(2): 245–285.

Boddy, M., and Dean T. L. 1989. Solving Time-Dependent Planning Problems. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, 979–984. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

D'Ambrosio, B. 1989. Resource-Bounded Agents in an Uncertain World. Paper presented at the IJCAI-89 Workshop on Real-Time Artificial Intelligence Problems, 24 August, Sydney, Australia.

Dean, T. L. 1987. Intractability and Time-Dependent Planning. In *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans,* eds. M. P. Georgeff and A. L. Lansky. San Francisco, Calif.: Morgan Kaufmann.

Dean, T. L., and Boddy, M. 1988. An Analysis of Time-Dependent Planning. In Proceedings of the Seventh National Conference on Artificial Intelligence, 49–54. Menlo Park, Calif.: American Association for Artificial Intelligence.

Doyle, J. 1990. Rationality and Its Roles in Reasoning. In Proceedings of the Eighth National Conference on Artificial Intelligence, 1093–1100. Menlo Park, Calif.: American Association for Artificial Intelligence.

Garvey, A., and Lesser, V. 1994. A Survey of Research in Deliberative Real-Time Artificial Intelligence. *Journal of Real-Time Systems* 6(3): 317–347.

Garvey, A., and Lesser, V. 1993. Design-to-Time Real-Time Scheduling. *IEEE Transactions on Systems, Man, and Cybernetics* 23(6): 1491–1502.

Grass, J., and Zilberstein, S. 1996. Anytime Algorithm Development Tools. *SIGART Bulletin* (Special Issue on Anytime Algorithms and Deliberation Scheduling) 7(2). Forthcoming.

Hansen, E. A., and Zilberstein, S. 1996. Monitoring the Progress of Anytime Problem Solving. In Proceedings of the Thirteenth National Conference on Artificial Intelligence. Menlo Park, Calif.: American Association for Artificial Intelligence.

Horvitz, E. J. 1990. Computation and Action under Bounded Resources. Ph.D. diss., Departments of Computer Science and Medicine, Stanford University.

Horvitz, E. J. 1987. Reasoning about Beliefs and Actions under Computational Resource Constraints. Paper presented at the 1987 Workshop on Uncertainty in Artificial Intelligence, 10–12 July, Seattle, Washington.

Horvitz, E. J., and Breese, J. S. 1990. Ideal Partition of Resources for Metareasoning. Technical Report, KSL-90-26, Stanford Knowledge Systems Laboratory, Stanford University.

Horvitz, E. J.; Suermondt, H. J.; and Cooper, G. F. 1989. Bounded Conditioning: Flexible Inference for Decision under Scarce Resources. In *Proceedings of the 1989 Workshop on Uncertainty in Artificial Intelligence,* 182–193. New York: North-Holland.

Lawler, E. L.; Lenstra, J. K.; Rinnooy Kan, A. H. G.; and Shmoys, D. B., eds. 1987. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* New York: Wiley.

Lesser, V.; Pavlin, J.; and Durfee, E. 1988. Approximate Processing in Real-Time Problem Solving. *AI Magazine* 9(1): 49–61.

Liu, J. W. S.; Lin, K. J.; Shih, W. K.; Yu, A. C.; Chung, J. Y.; and Zhao, W. 1991. Algorithms for Scheduling Imprecise Computations. *IEEE Computer* 24:58–68.

Pos, A. 1993. Time-Constrained Model-Based Diagnosis. Master's thesis, Department of Computer Science, University of Twente, The Netherlands.

Russell, S. J., and Wefald, E. H. 1991. *Do the Right Thing: Studies in Limited Rationality.* Cambridge, Mass.: MIT Press.

Russell, S. J., and Wefald, E. H. 1989. Principles of Metareasoning. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning,* eds. R. J. Brachman, H. J. Levesque, and R. Reiter. San Francisco, Calif.: Morgan Kaufmann.

Russell, S. J., and Zilberstein, S. 1991. Composing Real-Time Systems. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, 212–217. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Simon, H. A. 1982. *Models of Bounded Rationality, Volume 2.* Cambridge, Mass.: MIT Press.

Vrbsky, S. V.; Liu, J. W. S.; and Smith, K. P. 1990. An Object-Oriented Query Processor That Returns Monotonically Improving Approximate Answers. Technical Report, UIUCDCS-R-90-1568, University of Illinois at Urbana-Champaign.

Wallace, R., and Freuder, E. 1995. Anytime Algorithms for Constraint Satisfaction and SAT Problems. Paper presented at the IJCAI-95 Workshop on Anytime Algorithms and Deliberation Scheduling, 20 August, Montreal, Canada.

Wellman, M. P., and Liu, C. L. 1994. State-Space Abstraction for Anytime Evaluation of Probabilistic Networks. Paper presented at the Tenth Conference on Uncertainty in Artificial Intelligence, 29–31 July, Seattle, Washington.

Zilberstein, S. 1996. Resource-Bounded Sensing and Planning in Autonomous Systems. *Autonomous Robots* 3:31–48.

Zilberstein, S. 1995. Optimizing Decision Quality with Contract Algorithms. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, 1576–1582. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Zilberstein, S. 1993. Operational Rationality through Compilation of Anytime Algorithms. Ph.D. diss., Computer Science Division, University of California at Berkeley.

Zilberstein, S., and Russell, S. J. 1996. Optimal Composition of Real-Time Systems. *Artificial Intelligence* 82(1–2): 181–213.

Zilberstein, S., and Russell, S. J. 1993. Anytime Sensing, Planning, and Action: A Practical Model for Robot Control. In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, 1402–1407. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Shlomo Zilberstein is an assistant professor of computer science at the University of Massachusetts at Amherst. He received his B.A. in computer science summa cum laude from the Technion, Israel Institute of Technology, and his Ph.D. in computer science from the University of California at Berkeley. Zilberstein's research interests include resource-bounded reasoning, autonomous agent architectures, real-time problem solving, and reasoning under uncertainty.