Context-Dependent Computational Components

Thomas L. Dean*
Department of Computer Science
Brown University
tld@cs.brown.edu

Abstract

Many applications can be described in terms of a control system embedded in a dynamic environment. The computational components that comprise such an embedded system often must be responsive to the context in which they are executed. In order to be responsive, they must specify how their performance affects and is affected by context. This paper considers anytime algorithms as a particular sort of context-dependent computational component. Traditional anytime algorithms are passive components that provide performance specifications to external processes and receive allocations of computational resources in return. We also consider context-dependent components that are interactive, probing the environment to establish their value in a given context and competing with other components for computational resources.

1 Introduction

Anytime algorithms were invented as a convenient basis for building systems capable of responding to external events in a timely manner. They borrow from the perspective of feedback control in which the output of a control system depends on external factors that change over time and are only partially influenced by the output of the control system. In the case of time-critical applications, the external events typically require responses that involve significant computations. The basic problem addressed by anytime algorithms is characterized in the following paragraphs.

*This work was supported in part by the Air Force and the Advanced Research Projects Agency of the Department of Defense under grant No. F30602-95-1-0020, by the National Science foundation in conjunction with the Advanced Research Projects Agency of the Department of Defense under grant No. IRI-9312395.

Imagine a control system *embedded* in a changing environment. The system continuously monitors external events in the environment. When an external event is detected, the system generates zero or more decision problems whose solution may be required to respond appropriately to the external event.

We are concerned primarily with the subsystem that receives as input a stream of decision problems and provides as output a stream of solutions to these problems. We refer to the decision problems that are generated in response to external events as *computational challenges*. These challenges typically involve problems that require searching over combinatorial structures or performing iterated numerical calculations. The value of a solution to a given challenge depends on the time the challenge is issued and the time the solution is provided. We generally assume that there is some flexibility in what constitutes a solution. In particular, approximate solutions to hard computational problems are the norm in time-critical applications.

2 Time-Dependent Measures of Performance

We identify two measures of performance: *solution quality*, a time-independent measure of performance, and the *costs due to delay*, a time-dependent measure of performance. The costs due to delay account for the costs associated with missing opportunities in responding to events and the consequences of tying up scarce computational resources. We are particularly interested in problems in which it is possible to trade solution quality against the costs due to delay to improve a comprehensive measure of performance.

Given limited resource computational resources, the computations required to respond to a challenge can introduce lags into the underlying dynamics, and these lags

can significantly affect performance. In the case of hard deadlines, an optimal answer to a computational challenge that arrives after the deadline is no better than no answer at all. The role of anytime algorithms in addressing embedded control problems can be explained as follows.

First, we assume that there is a wide variety of off-the-shelf algorithmic components that can be composed or applied individually to solve a given computational challenge. Second, we assume that these algorithms have the property that they can be interrupted at any point in their execution to return an answer whose solution quality improves monotonically (at least in expectation) as a function of execution time. This is the essential property of an *anytime* algorithm.[1] Many search and estimation algorithms can be used directly or easily modified to satisfy this property. Third, we assume that it is possible to profile the performance of anytime algorithms so that the solution quality for any allocation of execution time (possibly conditioned on some measure of input quality) can be easily computed. This easily computed function is called a *performance profile.*

3 Building Embedded Systems

The engineer building a system for a time-critical application maps computational challenges to anytime algorithms and then designs a scheduler that, given a set of anytime algorithms and their associated performance profiles, allocates computational resources to algorithms, and then interrupts and extracts solutions from these algorithms when appropriate. The process of allocating computational resources to anytime algorithms is called *deliberation scheduling.* That is the basic idea; of course, the devil, as they say, is in the details.

The problem of scheduling anytime algorithms is easy if the computational challenges are independent of one another and there is a single, well-behaved anytime algorithm for each type of challenge. In most real-world problems, however, the computational challenges are interrelated and the solution of any given set of challenges may involve a variety of sorting, searching, and integration algorithms with intermediate results and complicated dependencies. In particular, if the computational challenges are dependent on one another or the solution methods are naturally expressed as a composition of anytime algorithms in which one algorithm takes as input the output of one or more other algorithms, then deliberation scheduling can be quite difficult. Solving real-world transportation planning problems is a good example of a time-critical problem of this sort.

An air transportation planning problem is often thought of in terms of a number of scheduling tasks that include scheduling crews, routine maintenance of aircraft, loading and unloading passengers and cargo, and assigning takeoff and landing runways. These different tasks are dependent on one another. In some cases, however, it is computationally convenient to treat them as if they were independent. Whether or not to treat them independently depends on the time available for computation and possibilities for improving the joint scheduling problem.

Computational challenges may force reconsideration of one or more of the schedules for these tasks. For example, it may be necessary to find an alternative crew when a connecting flight is cancelled, or reschedule the maintenance of an aircraft when a pilot notices a warning light. An embedded system for air transportation scheduling application will require deciding when to revise one or more of the schedules corresponding to these tasks in response to a sequence of such challenges. Deliberation scheduling for the air transportation problem described above is computationally quite complex and will require clever engineering to make practical.

Anytime algorithms are first and foremost an engineering concept. Such algorithms embody a useful property for building embedded control systems. Their utilization stands in stark contrast to the traditional off-line input-output model found in much of computer science. Anytime algorithms address one aspect of the general idea that the basic components used to construct systems should somehow be sensitive to the context in which they are executed. Anytime algorithms together with their associated performance profiles provide information to an embedded control system as to how they might be used in a particular time-critical situation. We might also imagine computational components that assess their own context-dependent value and vie with other components for computational resources.

4 Self-Allocating Algorithmic Components

Anytime algorithms represent one example of a computational component that provides as part of its specification information about how it behaves in various contexts. In the case of anytime algorithms, this information might take the form of conditional performance profiles (*i.e.,* solution quality as a function of execution time conditioned on the quality of the input).

[1] We are actually more interested in the procedures that are used to implement anytime algorithms, but we gloss over the distinction in this paper.

You might also imagine more detailed specifications that specify the side effects of employing a given component. These side effects might include disk usage, memory, special computing resources such as graphics accelerators and networking bandwidth, or even robotic sensors and manipulators.

In the following, we consider two different sorts of context-dependent computational components: passive and interactive. Passive components rely entirely on other processes to allocate them resources and manage their side effects. Traditional anytime algorithms are passive; they rely on a deliberation scheduler to make effective use of their capabilities and account for the context in which they are to be executed. Traditional anytime algorithms make available their time-dependent performance specifications to the outside world.

Interactive components might manage their own resources, establish from context how much time they should be allocated and then compete for the required resources. The advantage of interactive components is that they don't require global deliberation scheduling and are therefore better suited to distributed computing environments. The disadvantage is that such components will tend to be myopic in order not to spend too much time and effort deciding how to perform in a given context.

Consider how interactive components might play a role on the world wide web. Suppose that you want information about a particular subject and you invoke an internet search routine. The search routine might invoke one or more remote procedures implementing anytime algorithms.

In the process of invoking such a procedure, the search routine might ask for a performance profile relating, say the comprehensiveness of the search, to the time spent in performing the search.

Given the performance profile, the search routine may ask for a solution within a given interval of time. In this case, the search routine is using the remote procedure as a passive component, but delegating some flexibility by specifying a range rather than a rigid allocation.

Once given such a range, the remote procedure may decide on the basis of what it discovers about the current environment to terminate its computations early, *e.g.*, in the case where it discovers a crucial server is down or over subscribed. In this case, the procedure is taking initiative and performing as an interactive component. The potential utility of such interactive components should be apparent to anyone who has ever used a web browser

Complex graphical objects might be implemented as interactive components that decide to transfer part or all of their associated data, *e.g.*, polygons, pixels, depending on whether or not they are currently in a viewable portion of the browser's open window or based on more complicated criteria, *e.g.*, a camera mounted on the user's monitor might provide information on which portions of the screen the user is attending to. Similarly, Java programs (called 'applets') could be incrementally compiled on-line based on some measure of how they might improve the display of information in a web browser. Interactive components that implement mixed-initiative remote control (remote teleoperation augmented with "smart" local, real-time execution) over limited bandwidth networks might rely less on remote commands when the network connection exhibits longer lag times.

Building systems that consist of interactive context-dependent components will require that the operating system support the on-line allocation of computational resources. In addition, a protocol will have to be developed so that components can bid on computational resources. A somewhat less ambitious approach might be for components to post their performance specifications in an accessible place and then simply allow algorithms to decide for themselves when to terminate based on these specifications and other contextual information.

5 Conclusion

Context-dependent computational components are computing entities whose behavior is determined not just by their initial input but also by the context in which they are executed. Such entities can provide or make use of information regarding their performance as a function of time, input, or other aspects of the execution environment. They can behave in a passive mode in which they provide their performance specifications and allow external procedures to control their run-time behavior. Alternatively such components can perform in an interactive mode in which they gather information about their external environment and about the availability of computing resources and attempt to control their own behavior.

This paper considers anytime algorithms as a mechanism for supporting one particular sort of context-dependent computational component. Traditional anytime algorithms operate in a passive mode. The performance specifications for anytime algorithms are performance profiles that are are used by a deliberation scheduler to allocate computational resources. My guess is that passive anytime algorithms controlled by global deliberation schedulers, while they may turn out to be useful for highly time-critical applications, will be eclipsed by interactive components perhaps with anytime characteristics. The reason is simply that such components are easier to implement and use in a heterogeneous comput-

ing environment. Researchers in operating systems and program languages will have to develop minimal conventions allowing components in an interactive mode to assess a given context and compete for computational resources and components in a passive mode to provide performance information to other components.

Further Reading

The term "anytime algorithm" was coined by Dean and Boddy DeanandBoddyAAAI-88. The usefulness of the basic anytime property was independently recognized by a number of people. The notion of anytime algorithms developed by Dean and Boddy is closely related to the flexible computations of Horvitz HorvitzAAAI-88 and the imprecise computations of Liu and her students [Shih *et al.*1989]. For more on anytime algorithms in general and the problems of deliberation scheduling in particular see Boddy and Dean BoddyandDeanAIJ-94. For more on the composition of anytime algorithms see Zilberstein and Russell ZilbersteinandRussellAIJ-95. Wegner WegnerIEEE-92 explores notions of reactivity and interactivity in the context of object-oriented programming.

References

[Boddy and Dean1994] Boddy, Mark and Dean, Thomas 1994. Decision-theoretic deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence* 67(2):245–286.

[Dean and Boddy1988] Dean, Thomas and Boddy, Mark 1988. An analysis of time-dependent planning. In *Proceedings AAAI-88.* AAAI. 49–54.

[Horvitz1988] Horvitz, Eric J. 1988. Reasoning under varying and uncertain resource constraints. In *Proceedings AAAI-88.* AAAI. 111–116.

[Shih *et al.*1989] Shih, W-K.; Liu, J. W. S.; and Chung, J-Y. 1989. Fast algorithms for scheduling imprecise computations. In *Proceedings of the Real-Time Systems Symposium.* IEEE. 12–19.

[Wegner1992] Wegner, Peter 1992. Dimensions of object-oriented modeling. *IEEE Computer* 25(10):12–21.

[Zilberstein and Russell1995] Zilberstein, S. and Russell, S. J. 1995. Optimal composition of real-time systems. *Artificial Intelligence* 79(2).