Any-Dimension Algorithms

David J. Musliner   Edmund H. Durfee   Kang G. Shin

Computer Science and Engineering Division
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109-2122

djm@eecs.umich.edu
(313) 763-5363

Introduction

In striving to apply computer control systems to domains which require predictable performance, researchers have developed methods of obtaining performance guarantees using limited system resources. These guarantees are usually based on either off-line scheduling techniques that reserve resources (e.g., the Spring kernel [5]), or on-line techniques that dynamically adjust resource usage to meet guarantees. These on-line techniques are exemplified by "any-time" algorithms, which can be interrupted at any time to yield a result, possibly with reduced precision, confidence or completeness [2, 3]. Any-time algorithms provide an on-line, dynamic method for guaranteeing the timeliness of a result. This paper generalizes the any-time technique to provide guarantees on other measures of performance, and examines some of the issues involved with the resulting class of *any-dimension* algorithms.

Any-Dimension Algorithms

We can measure a system's performance by its progress along a set of (not necessarily orthogonal) dimensions that describe its resource usage (e.g., time, data, memory) and the quality of its output (e.g., timeliness,[1] precision, confidence). On-line performance guarantees are often defined by the conditions that determine when a system's control algorithm returns a result. We call methods that halt when they reach a certain threshold along a dimension "any-dimension" algorithms. Any-time algorithms are one type of any-dimension algorithm, terminated when an interrupt indicates that a temporal threshold has been reached. Similarly, "any-precision" algorithms halt when a result with a certain precision has been achieved; many iterative numerical methods [1] are any-precision algorithms. Thus any-dimension algorithms can provide guarantees on either the quality of the final output of a process or the maximum resources it will consume while running.

Because they must be able to halt and produce a result whenever a threshold is passed, any-dimension algorithms must be iterative, incremental computations. That is, they loop around a computation that produces intermediate results, usually in the same form as the final, ideal result. Most existing work on specific any-dimension algorithms focuses on iterative algorithms which produce intermediate results with non-decreasing quality. In most cases, this restriction is really only used to justify further iterations in the hope that better results will be produced. If the results could get worse as more effort is expended, intuition tells us that effort may be wasted. However, no aspect of the any-dimension paradigm *requires* algorithms to have monotonic results. Any-dimension algorithms are characterized solely by their incremental nature and their

The work reported in this paper was supported in part by the National Science Foundation under Grants DMC-8721492 and IRI-9158473, and by a NSF Graduate Fellowship. The opinions, findings, and recommendations expressed in this publication are those of the author, and do not necessarily reflect the views of the NSF.

[1]Note the duality of time as a resource and timeliness as measure of output quality.

```
last_result = initial_guess;
while (quality(last_result) < threshold)
      {
         new_result = incremental_computation();
         if (quality(new_result) > quality(last_result))
            { last_result = new_result; }
      }
return(last_result);
```

Figure 1: A simple shell to conceal lower-quality intermediate results.

termination threshold. This flexibility is an advantage, because non-monotonic results are both common and desirable. For example, many search domains have local minima and maxima that can trap monotonic search algorithms at non-optimal solutions. A non-monotonic algorithm can move out of local extrema in order to continue searching for a globally optimal result.

However, it is not necessary to reveal this non-monotonic nature outside of the any-dimension algorithm. Wrapping the simple shell shown in Figure 1 around an arbitrary incremental computation will prevent lower-quality results from being returned, and thus any incremental algorithm can be made to appear monotonic.

The mechanisms which actually monitor progress along some dimension and terminate an any-dimension algorithm can be synchronous or asynchronous. For synchronous monitoring, the any-dimension algorithm's code, which generates incremental results within a loop, also checks within that loop for the termination condition (as in Figure 1). For asynchronous monitoring, a process external to the any-dimension process monitors the termination condition and sends an interrupt to the any-dimension algorithm when the threshold is reached.

Synchronously-monitored algorithms are most appropriate for thresholds based on the quality of the computed result, since the any-dimension algorithm need only check its threshold condition when a new incremental result has been computed. A continuous asynchronous monitor might check the condition too frequently, wasting effort by repeatedly examining the quality of a result that has not changed. Or, an asynchronous monitor might not run frequently enough, so that some intermediate results would never be checked against the termination threshold. Furthermore, the most logical place to locate the knowledge of a desired result attribute is in the program generating the results, as opposed to some external arbiter.

Asynchronously-monitored algorithms are more appropriate for thresholds on resource dimensions, since the any-dimension process itself may not have sufficiently fine-grained access to monitor its resource usage. Also, if we are monitoring memory usage, for example, we would not want the any-dimension algorithm's code to be responsible for checking a memory limit each time an allocation was performed; that should be done outside the any-dimension process, preferably in operating system code.

Related Work

Liu *et al.* [4] have investigated an approach known as "imprecise computation," which is very similar to an any-time algorithm. They discuss using an incremental algorithm which generates intermediate results with non-decreasing precision. The key difference from pure any-time algorithms is that the authors of [4] also postulate that a known amount of computation time will lead to an acceptably precise result. Given this minimum "mandatory" computation time, they can then build task schedules that guarantee to achieve the minimum required precision by always reserving the

```
xguess = initial_xguess;
while (abs(xnew - xguess) > .01)
      {
          xguess = xnew;
          xnew = xguess - F(xguess) / Fprime(xguess);
      }
```

(a) Newton's method.

F	initial_xguess 1	10	20
x^2	7	10	11
$e^x - 1$	4	13	23
$e^{25x} - 1$	27	252	502

(b) Iterations to achieve .01 precision.

Figure 2: Showing the difficulty of mapping precision to time for Newton's root-finding method.

corresponding minimum required computation time.

Unfortunately, this type of guarantee relies on an accurate mapping between the time an imprecise computation algorithm runs and the precision of its result. In general, such a mapping is <u>not</u> available, because the precision of a result is highly dependent on the particular problem to which the algorithm is being applied. For example, Liu *et al.* [4] describe an any-time implementation of Newton's method for finding the roots of a function F. Unfortunately, as illustrated in Figure 2, the number of iterations this method requires to achieve a result with specified precision is highly dependent on both the function F and the initial guess for the root value. Thus the root-finding computation cannot be cleanly separated into mandatory and optional parts based on time alone; the precision threshold cannot be mapped onto the time dimension.

This example shows that, while various quality and resource dimensions may be related, their relationships are usually not constant functions. Instead, the mappings of response quality to resource usage will be highly dependent on both the domain (F) and on the internal state of the system (initial_xguess). Thus any-dimension algorithms do not rely on such mappings to make guarantees. Instead, any-dimension algorithms retain their thresholds on the original measurement dimensions.

A similar argument applies to Dean and Boddy's work on "deliberation scheduling," deciding how long to run any-time algorithms [2]. Their formulation does indicate that the problem domain can have an effect on the mapping of an algorithm's result utility to running time. However, they do not discuss precisely how that effect can be quantified. Casting iterative computations as any-time algorithms is fairly straightforward, but obtaining a parameterized mapping of result quality to resource usage is not. Even if an accurate mapping function was known, computing the effect of a particular domain on the performance of an algorithm might involve considerable effort.

Combining Dimensions

The termination conditions used by any-dimension algorithms can be combined using conjunction and disjunction to yield more interesting algorithmic behavior. Disjunctive (OR) combinations of any-dimension methods lead to a guarantee that crossing one threshold <u>or</u> the other will yield a result. Thus, combining any-time and any-confidence conditions might be an appropriate method for solving problems in a timed exam or planning parts of a path under time pressure; the resulting algorithm would work on each problem until it either found a result in which it had sufficient confidence, or until the time allotted to that problem expired.

Disjunctive combinations of thresholds are actually quite common. A pure any-dimension algorithm will run until it uses up the specified resources or its result reaches the quality threshold. Thus any-dimension algorithms can fail to terminate if, for example, they never find a result with sufficiently high quality. As a practical matter, most implementations of any-dimension algorithms also include an alternative termination condition, so that they will terminate even if their original dimensional threshold is never reached. For example, an any-time search algorithm might have both a deadline and a termination condition specifying the goal of the search. If the goal is reached before the deadline arrives, the algorithm terminates and returns its highest-quality result, without ever reaching its resource threshold.

Conjunctive (AND) combinations of any-dimension methods lead to guarantees over multiple dimensions. For example, combining any-confidence and any-precision conditions leads to results with guaranteed precision and confidence: the algorithm will continue until both thresholds are reached. However, if we try to conjoin any-time and any-precision algorithms, we will not necessarily obtain guaranteed precision and guaranteed timeliness. What happens if the time threshold (deadline) is reached before the precision threshold? The deadline indicates that all the allocated resource (time) has been consumed, so the algorithm cannot ignore the deadline and continue running. This example illustrates a fundamental restriction on conjunctive combinations: they cannot be applied to resource-monitoring any-dimension algorithms. They can be applied to quality-monitoring methods because the thresholds on those dimensions are minima rather than maxima: while a resource threshold indicates the maximum available resource, a quality threshold indicates the minimum acceptable quality. Going beyond a resource threshold is prohibited, while going further on a quality dimension is generally desirable.

Conclusion

We have described the general class of any-dimension algorithms, which can provide on-line dynamic guarantees for a variety of performance dimensions. Our investigation of any-dimension algorithms is just beginning. This new classification of methods may lend useful insight to software engineers responsible for translating program specifications into algorithms. The any-dimension paradigm clarifies the performance guarantees made by an algorithm, and thus can help in determining appropriate methods for a given problem.

Also, by mapping existing methods into the single classification space of any-dimension algorithms, we may be able to recognize areas of that space that have not yet been explored. For example, many search programs consume memory as a resource, maintaining a growing open-list of nodes yet to be searched. An any-memory search algorithm would terminate when it had used up an allocated portion of memory. This type of algorithm might be appropriate for problems which have no time limit, but do encompass an enormous search space (e.g., the game GO).

By stressing the nature of guarantees made along distinct performance dimensions, the any-dimension concept differs from the imprecise computation paradigm. Rather than trying to map quality thresholds to a time threshold, any-dimension algorithms retain the original quality thresholds and thus provide domain-independent performance guarantees.

References

[1] R. L. Burden and J. D. Faires, *Numerical Analysis*, PWS-KENT Publishing Co., 1989.

[2] T. Dean and M. Boddy, "An Analysis of Time-Dependent Planning," in *Proc. National Conf. on Artificial Intelligence*, pp. 49–54, 1988.

[3] K.-J. Lin, S. Natarajan, and J. W.-S. Liu, "Imprecise Results: Utilizing Partial Computations in Real-Time Systems," in *Proc. Real-Time Systems Symposium*, pp. 210–217, December 1987.

[4] J. W.-S. Liu, K.-J. Lin, and S. Natarajan, "Scheduling Real-Time, Periodic Jobs Using Imprecise Results," in *Proc. Real-Time Systems Symposium*, pp. 252–260, December 1987.

[5] J. A. Stankovic and K. Ramamritham, "The Design of the Spring Kernel," in *Proc. Real-Time Systems Symposium*, pp. 146–157, December 1987.