Crossroads
The ACM Student Magazine

Reasoning about Computational Resource Allocation

An introduction to anytime algorithms

by ***Joshua Grass***

Anytime Algorithms are algorithms that exchange execution time for quality of results. Since many computational tasks are too complicated to be completed at real-time speeds, anytime algorithms allow systems to intelligently allocate computational time resources in the most effective way, depending on the current environment and the system's goals. This article briefly covers the motivations for creating anytime algorithms, the history of their development, a definition of anytime algorithms, and current research involving anytime algorithms.

Introduction

For the past several years, the computational demands placed on computers by the programs they execute have drastically increased. Not only has the amount of information processed increased, but the manipulations these programs perform have become more complicated. The two main approaches to dealing with increases in computational demands have been to increase the speed of processors, or increase the number of processors working on the problem. Unfortunately, both of these solutions require more expensive computer systems, and in some cases this is not a feasible solution. In this article I will give an overview of a third approach to dealing with large computational tasks, that neither requires faster processors nor more of them, but in exchange returns partial or estimated results. This class of algorithms can also reason about the quality of the results, given the computational time resources allocated for the algorithm. These types of algorithms are called **anytime** because of the ability to trade off time for quality, and to execute for any amount of time.

Anytime algorithms are important in Artificial Intelligence (AI) research for two reasons. The first reason is that although many AI algorithms can take a long time to generate complete results, they often can generate very good partial results in a much shorter time period. Having the ability to reason about how much time is needed to receive a result that is adequate can make many AI systems more adaptive in complex and changing environments. The second reason is that a technique for reasoning about allocating time isn't limited to the internal workings of an agent, it may also be applied to other agents working cooperatively. Intelligent agents must be able to make decisions based on how fast they and

other agents can manipulate their environment, and many of the ideas of anytime algorithms can be expanded to include time-based planning.

In the next section, I give a brief history of anytime algorithms. Then I describe the structure of anytime algorithms and the components used in their creation. Next, I present an example of an anytime algorithm, followed by a description of some of the current research involving anytime algorithms. Finally, I conclude the article and summarize some of the future work that may be done with anytime algorithms.

History

The term ``Anytime algorithm'' was first used in the late 1980's by Dean and Boddy [**1**, **2**, **3**] in their work on time-dependent planning. In 1987, a related idea, *flexible computation* was introduced by Horvitz [**7**, **8**] for solving time-critical decision problems. Since then, the context in which anytime algorithms have been applied has broadened from planning and decision making to include problems from sensor interpretation to database manipulation, and the methods for utilizing anytime techniques have become more powerful. In 1991, S.J. Russel and S. Zilberstein completed work on composing anytime algorithms into more complicated systems [**10**, **13**, **16**]. By proving composition could be optimally performed (i.e. using information about the algorithm's performance to determine the best way to divide the time between the components), it became possible to build complex anytime systems by combining simple anytime components. Currently, many systems use anytime algorithms in the following areas: Sensor interpretation and path planning [**15**], model-based diagnosis [**9**], and evaluation of belief networks [**8**, **12**]. There is also interest in applying anytime techniques to information gathering on the world wide web, although no papers have been published on the subject yet.

Anytime Algorithms

Goals

Anytime algorithms iteratively work on problems to provide an interruptable way of producing results that improve in quality over time. Many current algorithms can be converted into anytime algorithms with little trouble. Anytime algorithms can be executed in two ways: either by being given a **contract time** (a set amount of time to execute), or an **interruptable method** (the anytime algorithm may be halted at any time). Anytime algorithms can also be continued once they have halted (either from the contract time being up, or an interruption) and continue to improve the solution. We would like the following qualities from the anytime algorithms we build:

Quality measure.
: Instead of a binary notion of correctness, an anytime algorithm returns a result with a measure of its quality. Quality measures how useful this result will be to other systems that depend on the information and/or how close the result is to the result that would be returned if the algorithm ran until completion. This may be in terms of a group of possible answers, ranges for answers, or other measurements.

Predictability.
: Anytime algorithms also contain statistical information about the output quality given a certain amount of time and information about the data it will process. This can be used for meta-reasoning about computational time resources to give an anytime algorithm.

Interruptibility and Continuation.

Anytime algorithms can be interrupted and return the partial results they have calculated up to that point or continued past the contract time they are given. This allows systems that use anytime algorithms to change the computational allocation to an anytime algorithm as it is executing.

Monotonicity.

Anytime Algorithms always improve or maintain the output quality of the data they work on as they are given more time. In certain cases this is done by maintaining the best results so far if the result for each iteration is not always improving in quality.

Anytime algorithms correlate output quality with time in a graph called a **performance profile**. This graph is constructed from (time, quality) pairs. Figure 1 is an example of a set of time/quality pairs forming a performance profile.

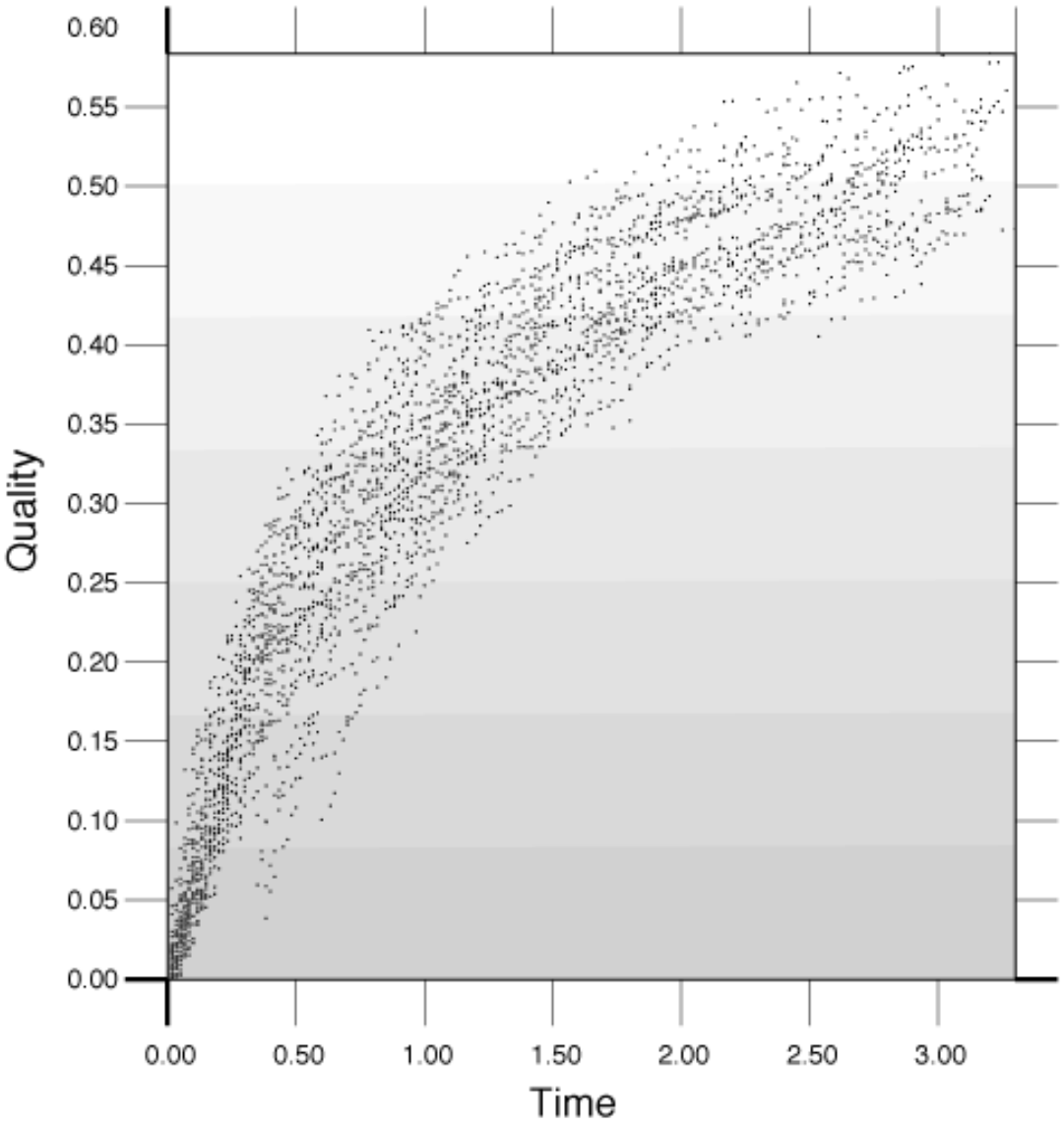


Figure 1 -- A set of time/quality pairs from an anytime problem

Overview

Anytime algorithms are currently being implemented in a variety of ways, but most anytime algorithms can be divided into three components that work together to produce a complete anytime algorithm. These components are:

An iterative improvement function.
: This function improves the quality of the result during each execution. Many algorithms can easily be converted into an iterative improvement function (e.g. iterative deepening search, Monte Carlo algorithms, sorting algorithms) by removing looping structures so that the function represents one step in the process of completing the algorithm.

A result evaluator.
: This function evaluates the result and the iterative improvement function to return information about the quality of the result.

A performance profile.
: This function returns an estimate of the quality given the amount of time that the anytime algorithm will be executed and information about the input data. This information about the input data is termed the **input quality**, but could be any type of meta-information about the initial data set, from the size to distribution information. If the performance profile uses the quality of the input data it may also be called the **conditional performance profile**.

The rest of this section gives a longer description of each of these three components.

The iterative improvement function

The iterative improvement function is repeatedly executed on the current result. In each iteration the function should be improving the quality of the data. This may be by producing a more accurate result, reducing the set of possibilities, or exploring more of the search space. The iterative improvement function should never reduce the quality of the information and should always work on a copy of the data so that if the anytime function is interrupted the current result can be returned immediately without being corrupted.

The result evaluator

Anytime algorithms differ from other algorithms in the results which they return. The result does not only contain the functional result, but also a measurement of the quality of the result. This quality measurement may come in a variety of forms, some of the most common being:

Accuracy.
: This measurement tells the system how close the result returned is guaranteed to be to the actual answer. Much like many numerical approaches to approximating transcendental functions with infinite series, a plus or minus error range is returned along with the result.

Coverage Area.
: This measurement tells the system how much of the potential search space or database has been explored.

Certainty.

This measurement states the correctness of the results returned by the algorithm: it can be a set of possible results, a probability distribution, or other approaches.

Resolution.

This measure returns the level of detail. For example, in a vision algorithm, this could represent the dimensions of the image which have been processed. The whole image is processed, but the dimensions may increase in size over time until the dimensions are equal to the image resolution.

The performance profile

The performance profile is the third and perhaps most important component of an anytime algorithm. The performance profile is the component that estimates the quality of the results depending on the input data and the amount of execution time that will be allocated to the algorithm. Planning algorithms will use this estimation to determine the best amount of time to allocate to an anytime algorithm depending on the current environment. The more accurate the estimate of the output quality, the better the system will be at planning time allocations. Several approaches are used for estimating the quality of the results given the initial data and the execution time.

Statistical estimates.

Statistical performance profiles are the easiest to construct but take the most storage space and the longest amount of time to instantiate. A statistical profile uses a large number of samples to create a database of (initial-quality, time, output quality) entries. With this information the database can then be used to make predictions about the expected quality given time and input quality information [**4**, **5**]. In more advanced databases, the result from a statistic database query is a probability distribution of the expected output quality. This allows the system to make more accurate decisions. Figure 2 shows an example of a statistical performance profile with distributions. The percentages in each box represent the chance of the output quality being in that range at a certain time. This probabilistic spread of output qualities gives a more accurate model than a performance profile that contained only an average output quality at a specific execution time.

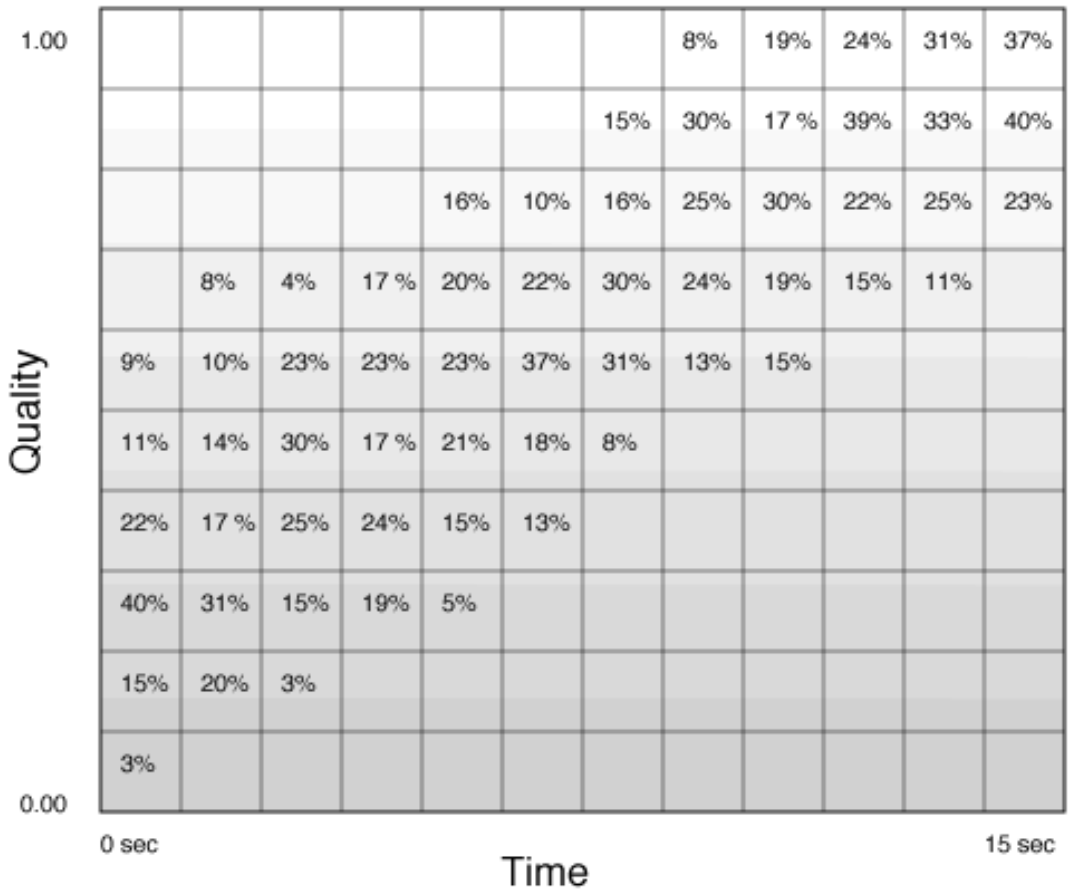


Figure 2 -- Table representation of a performance profile

Trajectory estimates.

Trajectory profiles use the current trajectory of the performance profile and previous trajectories to estimate the future quality. In a sense, we are using the derivative of the quality graph for the current execution to calculate the rest of the path. Figure 3 shows a trajectory profile.

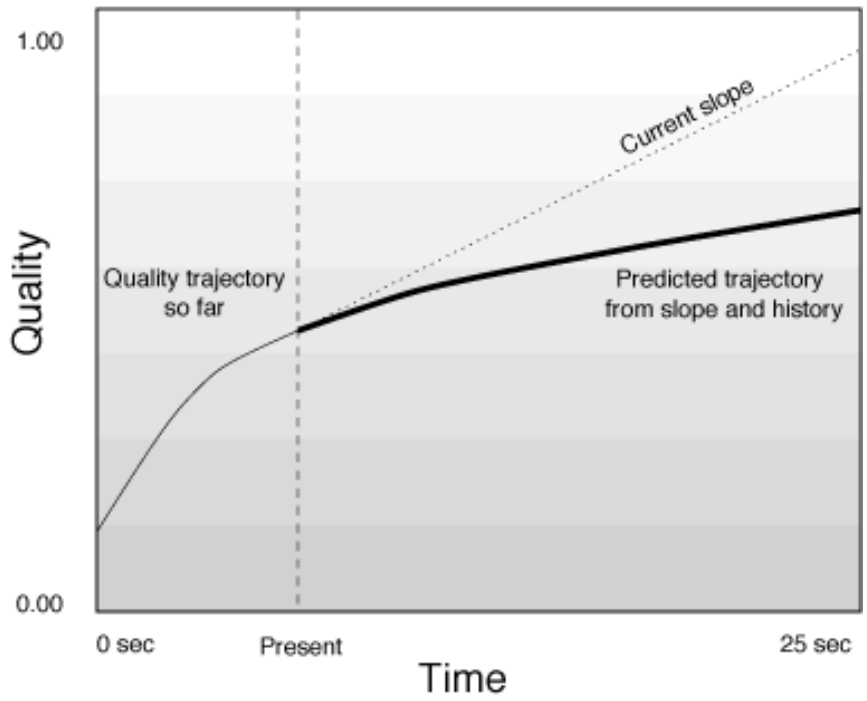


Figure 3 -- Trajectory estimate of performance profile

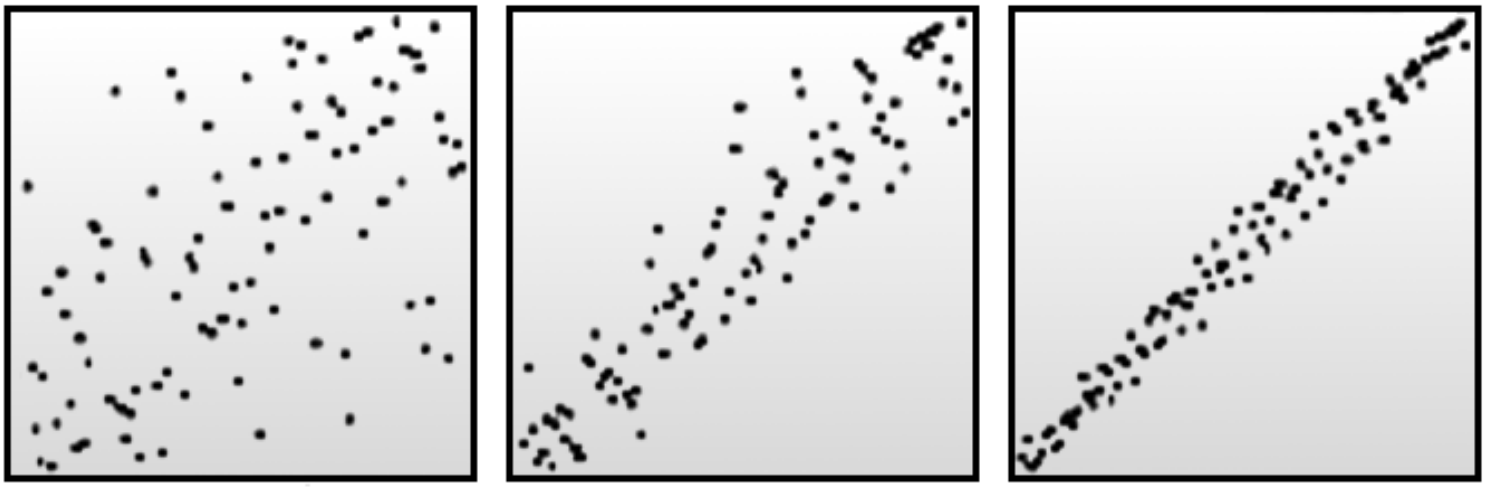
Algorithmic estimates.

In certain cases the structure of the algorithm is simple enough that a mathematic formula can be derived from the code that accurately estimates the quality given the execution time and the initial quality. In these cases only a little run-time information is needed to accurately fill in a few constants (e.g. time to complete an iteration depending on the CPU) for the formula.

An Example

An example of an algorithm that could be converted into an anytime algorithm is shellsort. Shellsort is a method of sorting lists that is very easy to implement and can sort any list in N^1.5 time [**11**], where N is the number of elements. The algorithm works in a manner much like an insertion-sort, except that the function begins by exchanging items that are a large distance away, and slowly decreases the distance. The effect is that the list as a whole becomes more sorted over time, as shown in Figure 4.

Value

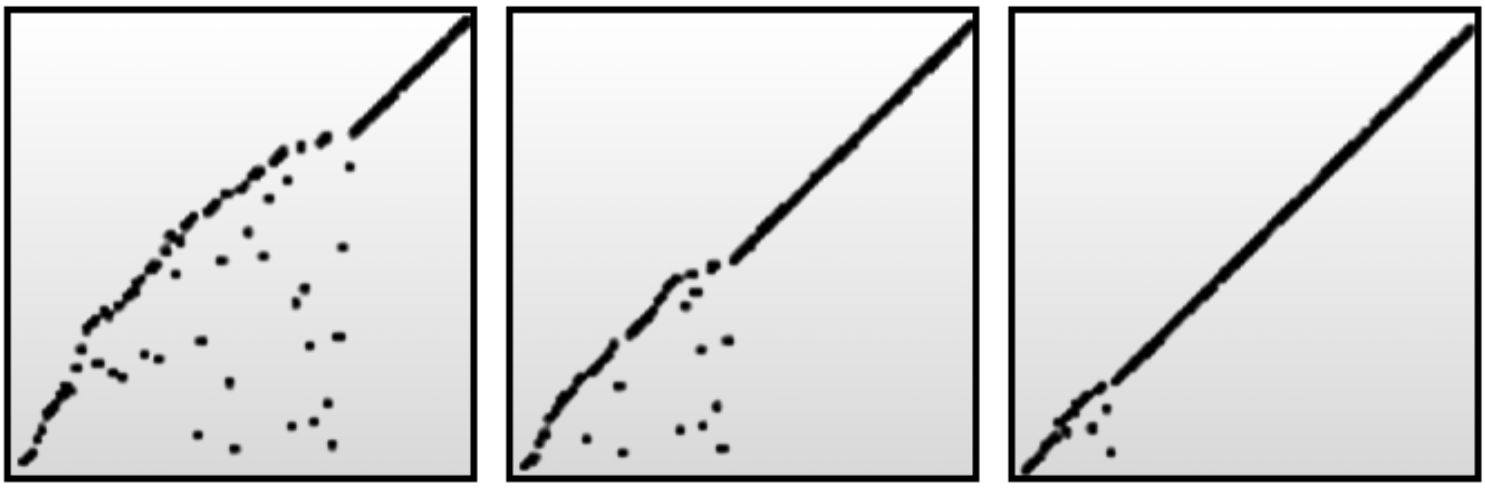


List location

Figure 4 -- Shellsort

In Figure 4, the points on the graph represent the value of entrees in the list. A sorted graph would be represented by a straight diagonal line, a random list would be represented by a scattering of points (see Figure 4, left). As the shellsort algorithm progresses, individual items do not move from un-sorted to sorted, which would be represented by a sorted diagonal line of increasing length surrounded by the unsorted scattered points (see Figure 5), but instead the entire list becomes more sorted.

Value



List location

Figure 5 -- Bubblesort

If we measure the quality of the output as the average distance between each entry and its correct sorted location, shellsort produces a graph much like the one in Figure 1, where the quality of the output increases monotonically over time and has diminishing returns. A function like bubblesort is not a good candidate for conversion to an anytime algorithm because it does not improve the quality of the data globally, it just solves the problem for each entry. In the case of bubblesort, the quality measure instead

becomes a measure only of the amount of work done, which is a trivial straight-line performance profile and doesn't tell us much about the actual quality of the result if it is used by another function.

Current Research

Composition

Composition of anytime algorithms allows complex anytime systems to be created from a number of simpler anytime components. Assuming that each of the components is a well-behaved anytime algorithm, meaning that the performance profile is monotonic and has diminishing returns (the improvement in quality is larger at the early stages of the computation), then composition is possible. Anytime algorithms may be composed linearly or as trees. Figure 6 shows the two composition structures.

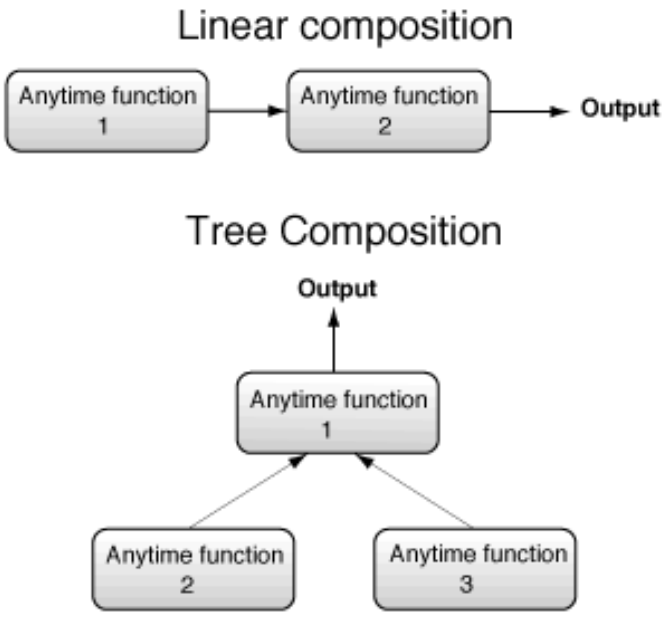


Figure 6 -- Linear and tree composition models

In order for composition to be optimal, anytime algorithms that are receiving input from another anytime algorithm must have a conditional performance profile. A conditional performance profile uses information about the input quality of the information and the time to determine the output quality.

In both the linear and the tree composition, the best time allocation is selected by searching through all possible time allocations to each component and evaluating the expected output quality of the result. By using the performance profiles, it is possible to build a function that optimally allocates the time between the sub-functions of an anytime algorithm so that the output quality is maximized for the total time allocated.

For example, if we are linearly composing two functions A and B in the form output=B(A(input)), we estimate the output quality from the performance profile of A and the conditional performance profile of B. We look at the output quality of A at time t1, and use that as the input quality for the conditional performance profile of function B and determine the final output quality at t2. By looking at all the

possible ways to divide the total time (t1+t2) between the two functions and the final output quality, we can find the best time allocation to the two functions given any amount of total time.

Currently, work is being done on building libraries of anytime functions, creating more complicated anytime structures and extending the scope of composition, and creating systems that intelligently compose anytime components to create complex anytime functions to solve specific problems. For instance, given a particular planning problem, how could an anytime system create a network of anytime sub-functions so that the unit as a whole can solve the problem in an anytime manner? Can this be done quickly and effectively?

Monitoring

Monitoring anytime systems allows a system to alter the amount of time allocated to an anytime algorithm while executing the algorithm. Monitoring can change the amount of time allocated to an anytime algorithm because of changes in the environment and unexpected intermediate qualities. For example, how should the system react if it gets particularly lucky while executing an anytime algorithm (e.g. the predicted output quality at time t is 0.3 and the actual output quality is much higher)? In certain cases that time could be reallocated to different anytime algorithms, a new task could be spawned, or it might be advantageous to allow the anytime algorithm to continue with the hopes to ending with a higher than expected output quality, or perhaps give it even more time. The same questions can be asked if the anytime algorithm returns unexpectedly low intermediate results. There are also situations where the environment demands a reallocation of time resources, for instance long-term planning is not important if an immediate threat appears.

Adding monitoring to anytime functions involves the creation of a new function that uses the current quality and the execution time to return a utility. This function is called the **time dependent utility function** and is represented as: **U(qi, tk)**, the utility of quality qi, at time tk. What could be a good result if returned quickly, may be of no value if it is returned after a long period of time. For example, the utility of a low quality result may be different if the result is return in 5 seconds or 30 seconds.

E. Hansen and S. Zilberstein have worked on developing a monitoring policy [**6**, **14**] based on the dynamic performance profile for the anytime algorithm and the time-dependent utility function. The **dynamic performance profile** is described as: **Pr(qj|qi, dt)**. This function denotes the probability of reaching an output quality of qj by resuming the anytime algorithm for time period dt when the current solution has quality qi.

Using both of these functions, a monitoring policy can be created. The **monitoring policy** is a mapping from a time and a quality into a decision to execute for **x** amount more time and whether to monitor again at the end of this time or not. Once this table is built, it allows the system to quickly make monitoring decisions that optimize the expected utility of the anytime algorithm. Since this table is based on the time-dependent utility function and the dynamic performance profile, it only needs to be calculated once so long as the time-dependent utility function does not change (the dynamic performance profile is based on the machine being used, so it should almost never change).

Anytime monitoring still has many questions to solve: How to incorporate the environment into a utility function? How to balance utility values of competing anytime algorithms so that the system intelligently

allocates time between them? And how often to perform monitoring due to environmental changes?

Programming environments

Another area of current research is creating programming environments that simplify and standardize the process of creating anytime algorithms [4, 5]. This is done by creating standard extensions to current languages in the form of development tools, software to automate many of the time-consuming aspects of creating anytime algorithms, libraries of anytime functions that can be integrated into projects, and a standard interface for anytime functions to communicate with each other.

Currently, most people who develop systems with anytime algorithms have to build the entire anytime environment from scratch, which represents a great deal of repeated effort. By creating standard extensions to many of the most common languages (C, C++, Lisp, JAVA) anytime development will be significantly simplified. Many research questions are still unanswered, such as: How strict to make the communication standard between anytime components? How to store information used by the performance profile in the least amount of space? What specific tools would help debug anytime functions? What should be the structure of the three anytime functions used in an anytime algorithm? Much like the parallel programming extensions added to C++ and other languages, there are many different approaches to adding anytime extensions.

Conclusion

Anytime algorithms offer a technique for controlling execution time at the expense of the quality of the result. Reasoning about computational time allocation to anytime algorithms can make a system more effective at dealing with complex environments where fully computing the sensing, planning and decision-making algorithms would slow the system down to the point of being ineffective. In many cases, a partial or low-quality result is sufficient for intelligently manipulating the environment.

Since the area of anytime algorithms is only about a decade old, there is still a great deal of potential research to be investigated in this field. Some potential topics include: developing extensions to existing languages so that anytime algorithms may be easily implemented, creating anytime systems that react intelligently to a changing environment, and creating anytime systems that intelligently build complex anytime algorithms from simple anytime components to reach particular goals. Anytime algorithms are an excellent solution to real-time problems, and as computers are expected to interact more with the rich, complex and ever changing real world, anytime algorithms will become more important.

References

1

Boddy, M. and Dean, T.L. Solving time-dependent planning problems. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pp. 979-984, Detroit, Michigan, 1989.

2

Dean, T.L. Intractability and time-dependent planning. In *Proceedings of the 1986 Workshop on Reasoning about Actions and Plans*, M.P. Georgeff and A.L. Lansky (eds.), Los Altos, California: Morgan Kaufmann, 1987.

3

Dean, T.L. and Boddy, M. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 49-54, Minneapolis, Minnesota, 1988.

4

Grass, J. and Zilberstein, S. Programming with anytime algorithms. In *Proceedings of the IJCAI-95 Workshop on Anytime Algorithms and Deliberation Scheduling*, Montreal, Canada, 1995.

5

Grass, J. and Zilberstein, S. *Anytime Algorithm Development Tools*. Technical report 95-94, University of Massachusetts at Amherst, 1995.

6

Hansen, E. and Zilberstein, S. Monitoring the Progress of Anytime Problem-solving.

7

Horvitz, E.J. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the 1987 Workshop on Uncertainty in Artificial Intelligence*, Seattle, Washington, 1987.

8

Horvitz, E.J., Suermondt, H.J., and Cooper, G.F. Bounded Conditioning: Flexible inference for decision under scarce resources. In *Proceedings of the 1989 Workshop on Uncertainty in Artificial Intelligence*, pp. 182-193, Windsor, Ontario, 1989.

9

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

10

Russel, S.J. and Zilberstein, S. Composing real-time systems. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pp. 212-217, Sydney, Australia, 1991.

11

Sedgewick, Robert. ***Algorithms in C*** . Reading, Massachusetts: Addison-Wesley Publishing Co., 1990.

12

Wellman, M.P. and Liu, C.L. State-Space Abstraction for Anytime Evaluation of Probabilistic Networks. In *Proceedings of the 10th Conference on Uncertainty in AI*, Seattle, WA, 1994.

13

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

14

Zilberstein, S. and Hansen, E. Modeling Performance Improvement using Markov Processes. *Private Communication*, 1995.

15

Zilberstein, S. and Russel, 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.

16

Zilberstein, S. and Russel, S.J. Optimal Composition of Real-Time Systems. *Artificial Intelligence* , forthcoming, 1995.