Use of Abstraction to Simplify Monitoring

Thomas A. Russ*
USC / Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292
tar@isi.edu

Overview of Approach

I apply existing technology for the construction of monitoring systems to the ICU data set provided by the Symposium organizers. The large volume of data in the ICU set compels any reasoning system to operate efficiently. It is also a good test of scalability of the approach. In order to reach the required efficiency, heavy use must be made of abstraction in order to limit the extent to which new data samples cause reevaluation of derived conclusions.

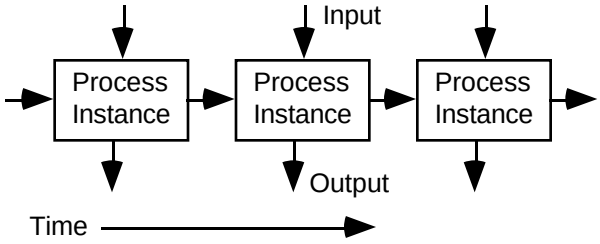
I describe the Temporal Control System (TCS), a programming system designed for building intelligent temporal monitoring programs. Empirical results from the ICU date set validate the scalable design of the TCS. I then focus on the problem of generating interval values from sample points via persistence assumptions. The TCS provides both the framework for the implementation as well as a method of calculating the "cost" of different approaches. In particular, I show that limiting the time span of a persistent interval can be very costly and then suggest how the application of symbolic abstraction can help. Further performance improvements come from the development of additional temporal abstraction techniques.

Description of Temporal Control System

The Temporal Control System [Russ91] is a programming framework designed to facilitate the construction of monitoring applications. It uses dependency-directed updating to allow temporal information to be entered in any order while assuring that all affected calculations are redone. TCS uses data dependency declarations made during the building of an application to monitor information in temporal variables. As the information in those variables changes over time, all calculations which depend on that information are automatically recalculated to bring the system up to date.

* Supported by the Advanced Research Projects Agency under contract MDA903–87–C–0641.

A TCS application consists of *modules* which implement arbitrary reasoning strategies, and temporal *variables* which hold data. Each module's dependency on temporal variables is declared in advance. TCS performs the bookkeeping tasks needed to assure that information is propagated to the appropriate modules and that the reasoning in the system is kept up-to-date. The data dependency algorithm does not depend on the particular form of reasoning used in a module. Module functions are treated as black boxes whose inputs and outputs are monitored. Whenever the input data to a module changes, the TCS schedules a *process instance* to execute. As input values change, this results in a chain of process instances along the time line.



If the output value changes, this change is propagated in turn. When there are no more changes, propagation stops and the application is up-to-date. Since the TCS model has a fundamental unit of computation, the process instance, counting the number of invocations of process instances for a particular module gives a hardware independent method of assessing the amount of computation involved in processing temporal data. I will use this measure in the experiments described below.

This method of propagating information and using the automatic updating facilities has been applied in the areas of cardiac intensive care and the management of acute diabetic ketoacidosis. The TCS is specifically designed to allow information to arrive out of order, and for previous information to be changed [Russ90]. These particular attributes are not needed for processing the Symposium data set,

since all of the information arrives quickly and in order.

The TCS approach differs from other signal processing approaches such as VM [Fagan80] and blackboard architectures [Nii82] in that the data dependencies allow the system itself to handle the overhead of retraction and new calculation.

ICU Data Set

I have combined the ICU data sets into one master file so that the information can be processed in temporal sequence. There is a large amount of data that needs to be processed. 8,050 data points are available at 2,623 separate times (average of 3.06 items per session). The minimum time separating any of the data points is one second. The maximum time separating the data points is three hours and forty-seven minutes. The average elapsed time between data points is 25 seconds. Once monitoring starts, the average elapsed time between data points is less than 20 seconds (17.95).

The implications of these figures is that any reasoning system that hopes to operate in real time must be able to process a large amount of information quickly. On average, the system must be able to handle around 10 data items per minute. Note that this includes the processing of the raw information as well as the higher level deliberation of the system. In this paper I will restrict my attention to the low-level processing of the raw data.

Although there is a lot of data to be processed, most it does not add information to the existing description of the patient's state. In order to not have the system collapse under the weight of the new data, the redundant information must be eliminated from consideration as soon as possible.

Use of Abstraction to Simplify Problem

The standard approach to eliminating unimportant details is to use abstraction of the detailed information. In addition to the TCS, recent medical AI work [Haimowitz93, Kahn90] has examined abstraction methods for handling temporal data. By suppressing the unimportant details, a reasoner can focus on the important principles. This is particularly important when there is a lot of automated monitoring. Such systems can produce data much faster than the raw information can be assimilated. There are several levels at which this abstraction can occur:

Symbolic Abstraction to Cut-Off Propagation

The first and most common approach is to abstract raw numeric data into symbolic (or even numeric) ranges. This serves two purposes: 1) It reduces the reliance of higher level reasoning on the specific numeric value of the underlying data. The same reasoning can then be used in different situations just by varying the function that is responsible for abstracting the numbers into symbols. 2) By mapping many numeric values into fewer symbolic values, one can use a simpler test to determine whether the information needs to be propagated. In particular, by testing for symbolic equivalence, the a system such as TCS can operate without the need to have domain-specific knowledge in its propagation algorithm. This makes it easier to use the same tool in more than one application.

Time Scale Abstraction to Reduce Computation

In addition to the benefits above, symbolic abstraction typically has another benefit that serves to reduce the amount of computation needed to process the data. In addition to summarizing multiple numeric results with the same label, the act of summarization will generally cover a larger time period. This means that the same decision is valid not only for several different numerical values, but it remains stable for a much longer period. In a system that is organized around the propagation of changes, this results in less computation.

Temporal Granularity to Reduce Computation

Unfortunately, these approaches are not sufficient in and of themselves. The data set for the ICU data is so large that new techniques need to be invented. The fatal flaw lies in the interaction between the use of time-limited persistence and the inexorable advance of the time of information. If one simply preserves the values for a fixed period of time, then each new data point will extend the derived persistent interval by the amount of time since the last data sample. With data samples arriving faster than two per minute this can become overwhelming.

The solution to this is the use of temporal granularity to limit the amount of computation. For example, if persistence were eight hours with a granularity of 15 minutes, then all data points within a fifteen minute window would not affect the length of time of the resulting persistent abstraction. This will eliminate the need to gratuitously re plan the future based on another 20 seconds of projected data values. This technique is an abstraction along the temporal dimension. The introduction of ranges for

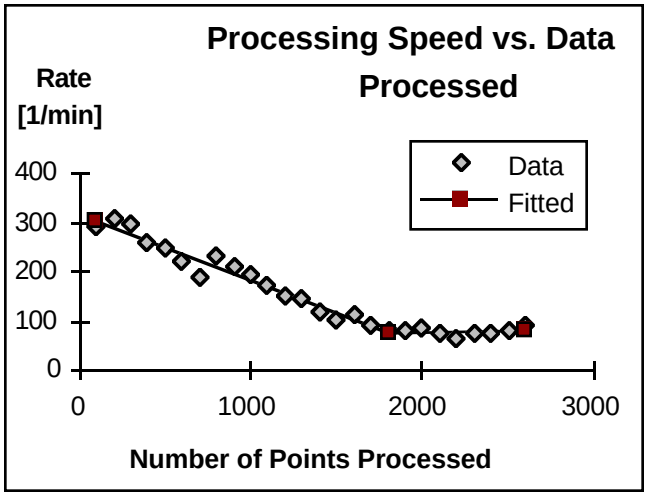
the temporal values is analogous to the use of numeric ranges for data values.

Description of Experiments

In the experiments described here, a sample TCS application was built that performed the abstraction of data samples into interval values. A system covering all of the data was used for the scalability experiment. A much simpler system was used to investigate the interaction of different types of abstraction with different persistence assumptions. In the scalability test, the most expensive method was used.

Scalability Test

The large amount of data available in the ICU data set means that any practical scheme must have run time characteristics that are essentially independent of the amount of data that must be processed. To demonstrate this feature, I created a TCS system that performed time limited persistence of raw data values. As I show in the next section, this particular type of persistence is actually quite expensive computationally. The test used a Macintosh Quadra 800 running this test system on the entire data set of 2622 data entry points. The results are shown below. The processing speed is the number of data entry points processed per minute. The data values are the average results of two tests.



There is an initial slowdown in processing as more data is encountered, but processing then reaches a steady-state plateau. The graph was divided into two regions (0–1800 and 1800–2600) for least squares regression fitting. The first region was well described by a line with slope -0.13 (R^2=0.96). The second region is described by a straight line (slope = 0.006) and shows no correlation (R^2=0.04) with the number of data points processed. The steady-state performance of the TCS is therefore independent of amount of data previously processed.

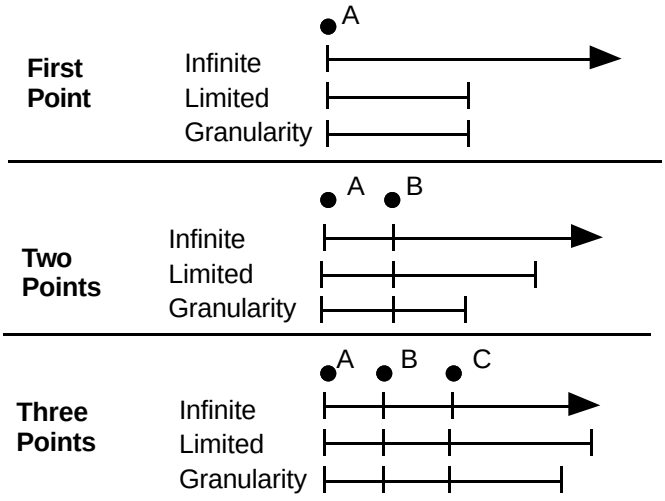
The initial slowdown is due to the effects of building up a set of processing intervals that become fragmented as more data is added. When the first datum is encountered, the system is able to project its value forward up to the pre-set time limit. This involves reexecuting the persistence process over the time that has been changed. It also means that any derivative calculations must be reexamined for changes. Since the TCS takes a black box approach to the actual reasoning processes, it is necessary to re-run processes to cover the time of the change. Since the length of an interval value is also available for reasoning, it is also necessary to re-run processes for intervals whose values haven't changed, but whose duration has been shortened. Since the reasoning is a black box, it is not possible to know whether a change in the duration of the interval would affect the outcome of the decision. If there is no change, then propagation ceases. The effects in terms of excessive calculation suggest that making more information available to the scheduling algorithm is a fruitful avenue for future work.

Since not all of the data is collected with the same frequency, it takes longer for some of the persistence calculations to saturate and reach steady state conditions. Once this condition is achieved, though, the time to process the information does not increase. As long as the steady-state processing is fast enough to handle the data, a TCS system will scale to meet the challenge.

Persistence Tests

Having passed the first qualifying hurdle for use in a monitoring setting, we can now turn our attention to the problem of increasing the efficiency of the reasoning. I begin by considering three forms of persistence calculation: infinite, time limited and time-limited with granularity. Infinite persistence means that a data value is retained until it is replaced with a newer reading. This has the advantage of simplicity, but it allows the use of potentially outdated information in clinical reasoning. Since the abstraction process moves the focus of the data from the actual time-stamped value into a state abstraction, there is no easy way for subsequent reasoning modules to know how old the data is. Time limited persistence solves this problem by associating a maximum allowable persistence for any measurement. If no new information is obtained before this deadline expires, then the state

value reverts to "unknown." This provides a simple method to eliminate outdated information. Limiting the persistence with granularity means that the endpoints chosen for persistence values are constrained to be even multiples of the grain size. This modification was developed to increase the efficiency of time limited performance, particularly when combined with value abstraction. Schematically, the three abstraction methods look like this:



Note in particular that the granularity endpoint for the "Two Points" case is in the same place as "First Point".

To allow deeper analysis I performed several experiments using just one of the measured parameters—heart rate as measured by the EKG. This provided 627 samples of mostly noise-free data for analysis. I then examined the effects of using different persistence schemes on derivative calculations. The test system was a simple TCS system consisting of just two modules and two variables. The first module took the raw point data as input and produced interval values as output. This is a temporal abstraction operation. The second module took the interval value as its input and served as a vehicle for assessing the affect of different choices of temporal abstraction on subsequent system processing. In other words, the second module recorded how much data was being passed on for further processing by what would be the higher level intelligence of the monitoring and control system.

Unfortunately, this is more much more expensive than using the infinite persistence model. Using the TCS scheduling algorithm, infinite persistence caused 1,254 invocations of the secondary reasoning module (or 2 per data point). This is expected from the analysis of the black box scheduling assumption. Using a time limited persistence of 240 seconds, the number of processes scheduled was 4,372 (about three and a half times more). Part of the problem is that the period of time covered by the persistence extends each time a new data point is encountered. For example, at time 0 the persistence extends until time 240. New data at time 10 cause a reexamination of the period from 10 to 240, as well as extending the persistent interval until time 250. More data at time 15 forces reexamination from 15 to 250 and extends the persistence from 250 to 255 as well, etc.

It is unlikely that the incremental extensions of the persistence are clinically significant. One way to limit this is to set up a "granularity" in the persistence assumption. If the persistence granularity were set to 60 seconds, then only changes in input data more than 60 seconds apart would cause the extent of the persistence to change. A change in value would still cause recalculation within the existing persistence window.. Using such a granularity scheme improves the performance figures for the heart rate data by about 30%, reducing the number of secondary processes run from 4,372 to 3,097. This is still 2 and a half times as many as for infinite persistence, so there is still a significant price to pay for limited persistence. (The process scheduling heuristics are not well designed for this particular case. By changing the heuristics to ones tuned for this type of reasoning I was able to reduce the overhead to only a 50% increase for the time-limited persistence and further reduce it to 30% when granularity was added.)

The desire to limit the persistence time forces us to pay a computational price in order to avoid reasoning with stale data. Fortunately, there are other techniques that can be applied which simplify the reasoning and let us reclaim the lost performance.

Abstraction Tests

A careful examination of the heart rate data indicates that most of it has the following form:

```
"8/16/93 23:24:44" 176
"8/16/93 23:26:04" 175
"8/16/93 23:26:24" 176
"8/16/93 23:26:44" 175
"8/16/93 23:27:02" 176
```

In other words, most of the changes from one reading to the next were insignificant. A more rigorous analysis of the heart rate data indicates that there were no two consecutive values that were the same, but that only 60 out of 627 samples changed

by more than one from the previous value. Only 30 samples showed a sample-to-sample change of 5 or more. It is also the case that only gross changes in heart rate are clinically meaningful. If we can introduce such a distinction early in the data processing, then we can save subsequent reasoning tasks the trouble of determining that the information is about the same. To that end we introduce a value abstraction transformation. The following table shows three different abstraction functions that were used in the experiments. The first two break the numeric scale into ranges of 5 and 10 and only use the range of the answer as the persistent value. If two values occur that are in the same range, then that is treated as no change in the data value. (Any temporal persistence is still used, though). "Stratify" divides the numeric range into three categories.

Heart Rate	Abstract 5	Abstract 10	Stratify
< 160	10	10	10
160–165	176		
165–170	120	296	359
170–175	63		
175–180	226	289	
180–185	20		258
185–190	5	25	
> 190	7	7	

By changing the value of the persistent information from a fluctuating numeric measure to an abstract value we can insulate higher level reasoning functions from this fluctuation. The effects of using this abstraction technique are dramatic. We will first consider the effects with infinite persistence. In this case, no transformation caused 1,254 invocations of the higher level calculation. With the 5 unit abstraction, this dropped to 192, with 10 unit abstraction to 126 and with stratify 132. For a real application the size of the abstraction bands and the breakpoints for doing a symbolic stratification will be determined by the medical requirements. The experimental results also show significant, though not as large, reductions in the number of secondary calculations used by limited persistence and limited persistence with granularity.

	Infinite	Limited	w / Granularity
No Abstraction	1254	4372	3097
Abstract 5	192	1795	1159
Abstract 10	126	1664	1052
Stratify	132	1643	1052

Conclusion

The TCS methodology is shown to scale well for problems with a large amount of data that needs to be processed, since the running time is fundamentally independent of the amount of data entered into the system. I describe a method for producing abstract interval descriptions from point data. Varying the parameters of this transformation allows us to see the performance characteristics of various options. The use of abstraction provides a big reduction in the number of subsequent computations needed. Adding time limited persistence has the effect of greatly increasing the number of computations. Adding granularity to time limited persistence makes a big improvement.

References

[Haimowitz93] IJ Haimowitz and IS Kohane, "An Epistemology for Clinically Significant Trends", Proceedings of the Eleventh National Conference on Artificial Intelligence, pp. 176–181, 1993.

[Russ91] TA Russ, *Reasoning with Time Dependent Data*, PhD thesis, MIT, 1991.

[Russ90] TA Russ, "Using Hindsight in Medical Decision Making," *Computer Methods and Programs in Biomedicine*, 32(1):81–90, 1990.

[Kahn90] MG Kahn, CA Abrams, et al., "Automated Interpretation of Diabetes Patient Data: Detecting Temporal Changes in Insulin Therapy," *Symposium on Computer Applications in Medical Care*, pp. 569–573, 1990.

[Fagan80] LM Fagan, *VM: Representing Time-Dependent Relations in a Medical Setting*, PhD thesis, Stanford, 1980.

[Nii82] HP Nii, EA Feigenbaum, et al., "Signal to Signal Transformation: HASP/SIAP Case Study," *AI Magazine*, 3(2):23–35, 1982.