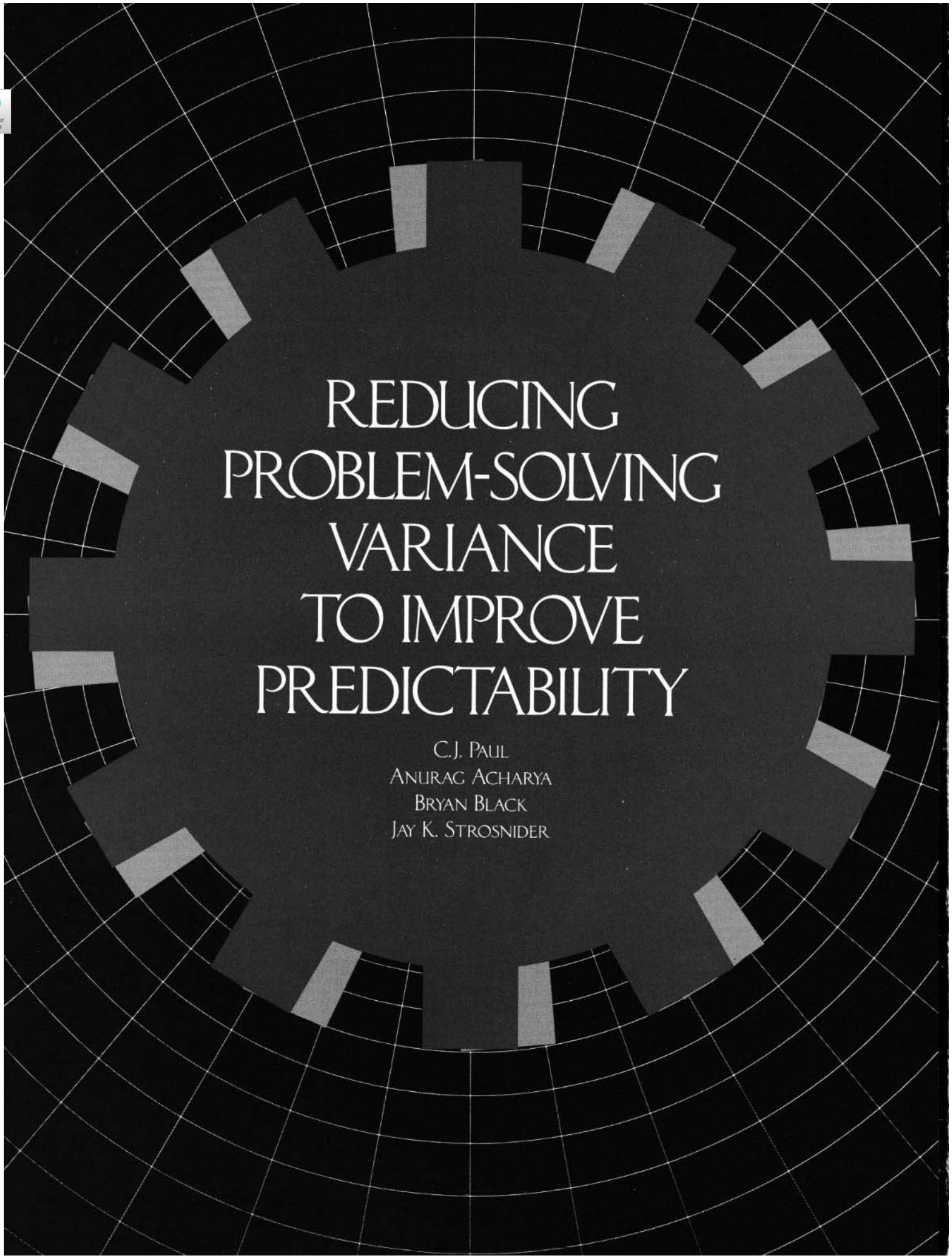
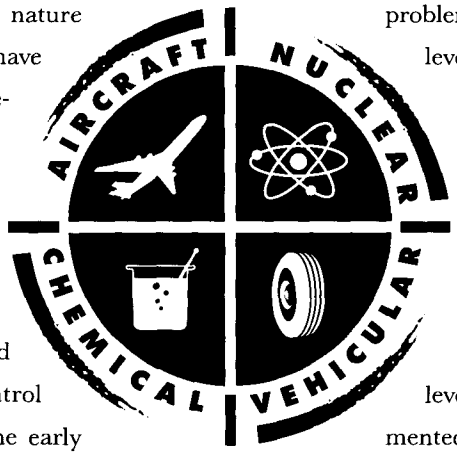


REDUCING PROBLEM-SOLVING VARIANCE TO IMPROVE PREDICTABILITY

C.J. Paul
Anurag Acharya
Bryan Black
Jay K. Strosnider

Real-time systems are playing an increasingly vital role in today's society. Such systems include manufacturing, control, transportation, aerospace, robotics and military systems. No longer are real-time systems limited to low-level control functions. They are now being asked to monitor and control complex, hierarchial systems in dynamic, sometimes hazardous, environments [7, 13]. Furthermore, some real-time systems such as the Mars Rover [2] are being asked to operate with little to no human interaction. Other large real-time systems are required to operate in environments that are not fully characterized [2]. The lack of information and the uncertainty of the environment requires the use of problem-solving techniques. To make things more difficult, real-time systems tend to be critical in nature where the impact of failures can have serious consequences. ✲ Current research in real-time artificial intelligence (AI) is driven by a need to make knowledge-based systems function in real time [12], and a need to integrate knowledge-based approaches to handle non-linearities and problem-solving behavior in control systems [3, 11, 17, 22]. Some of the early attempts to build such systems resulted in coincidentally real-time systems, which were difficult to analyze and predict [12]. At the other extreme, rule-based systems have been subject to exhaustive testing to guarantee they would be able to meet deadlines. Response time analysis is in general undecidable, and is PSPACE-hard in the case where all the variables have finite domains [18], making this technique infeasible for even moderate-sized systems. ✲ We believe execution time variance is the primary problem in providing performance guarantees for real-time problem-solving systems. Previous research in real-time scheduling has addressed some of the issues in integrating tasks with stochastic execution times into real-time systems [5]. However, the problem of taming the variance of problem-solving tasks has not been addressed. In typical real-time systems design, applications are created and worst-case times are calculated. Real-time scheduling is mostly based on worst-case execution times of the tasks in the system. The fundamental problem with problem-solving tasks for real-time applications is that the worst-case execution time is often unknown or orders of magnitude larger than the average case execution time. This results in systems which are either not schedulable or have very low utilization. Furthermore, if the execution time variance of the problem-solving tasks is not constrained, these tasks cannot be integrated into conventional real-time systems since the variance is likely to affect the predictability of the conventional real-time tasks. ✲ The execution time variance of problem-solving tasks manifests itself at two levels: the methodology level and the problem-solving architecture level. To improve predictability, it is necessary to tackle the variance at *both* these levels. We present an approach which integrates problem-solving methodology and architectural primitives to reduce the variance at both levels. We have designed and implemented an architecture, Concurrent Real-Time OPS5 (CROPS5) [20], illustrating these principles. Using this architecture, we demonstrate that problem-solving and real-time tasks can coexist within a readily analyzable framework, that hard deadlines can be guaranteed for critical problem-solving tasks, that soft deadlines for other problem-solving tasks can be provided so "best-effort" solutions are guaranteed within timing constraints. ✲ We begin by discussing the sources of execution time variance and methods to deal with them. We then develop the requirements of a real-time problem-solving architecture, providing not only the mechanisms to tackle the variance, but also the functionality required for integration into real-time environments. Later, we examine these issues in the context of CROPS5. An aircraft collision avoidance system is used as an example of



how problem-solving tasks can coexist with conventional real-time tasks on a common computing platform while maintaining guaranteed response time performance for the conventional real-time tasks. We currently have implementations of CROPS5 running on two real-time operating systems, ARTS [30] and CHIMERA II [24], and are in the process of porting it to Real-Time Mach [29].

Execution-Time Variance: The Sources

We now examine the fundamental differences between conventional real-time tasks and problem-solving tasks, and discuss how these differences affect the execution time variance of these tasks.

First, let us consider the execution time variance of conventional real-time tasks. Typical real-time signal-processing algorithms have little to no variance associated with their execution times because, regardless of the complexity and size of most signal-processing algorithms (eg., FFTs, filters), there are generally no data dependencies which can cause the execution times to vary. The input data is simply processed in a uniform, deterministic fashion. On the other hand, control-oriented real-time tasks often have data dependencies. As the system to be controlled increases in complexity, the number of data dependencies will probably increase, resulting in an increased variance in the execution time of real-time tasks.

Next, let us consider the execution time variance of problem solving. According to the Problem-Space hypothesis advocated by Newell and Simon [19], all goal-oriented symbolic activity occurs in a problem space. Search in a problem space is posited to be a completely general model of intelligence. Search is thus fundamental to all problem-solving processes.

Figure 1 illustrates a continuum between tasks in a knowledge-poor domain and tasks in a knowledge-rich domain. On the far right, knowledge-rich tasks are fully characterized, and there exists an explicit algorithm that transforms a given set of inputs to an appropriate output. There is no notion of search or backtracking at this end of the spectrum. Any variations in execution time are associated solely with data dependencies, as is the case for conventional real-time tasks.

As one moves to the left, either the task characteristics or their interactions with the environment are not completely known. Heuristics are now required to search the state space for an appropriate result. At the far left, there is no knowledge to direct the search; this results in a blind search. In this case, one would expect to have a large variance in execution time. To illustrate, let us consider a simple search tree of arity a, depth d. The total number of nodes in the tree is given by $(a^d - 1)(a - 1)$. In a simplistic sense, the worst-case and average-case execution times can be characterized by:

$$\text{Worst-Case Execution Time} = K(a^d - 1)(a - 1)$$

$$\text{Average-Case Execution Time} = KV(d)$$

where K is the average time to expand and evaluate a single node and $V(d)$ is the multiplier for the number of nodes examined in the average case. Assuming $d = 10$, $a = 3$, $V(d) = 30$, the ratio of the average case to the worst case is 30:29524 ($\approx$ 1:1000).

Given the large variance in the execution time of problem-solving tasks, and the fact that the worst-case execution time is either too large or unknown, traditional methods for the design of real-time systems cannot be directly applied to problem-solving tasks. An attempt to blindly apply these techniques will result in systems which are either not schedulable or are grossly underutilized. For this simple example, the worst-case execution time is almost 1,000 times longer than the average case. A system designed with the worst-case estimate of execution time will have a schedulable utilization of <0.001

Most problem-solving falls midway between the two extremes shown in Figure 1. As one moves back to the right, increasing knowledge may be applied to reduce the variance due to search.

Search is manifested in the two levels of problem solving: the knowledge retrieval level and the knowledge application (problem space) level. Several methods exist for implementing both these levels. We will use the problem space approach mentioned earlier as a basis for developing our arguments. Even though the principles are illustrated in this context, they have wide applicability.

A problem space can be characterized by a set of states and a collection of operators that map states to states. A problem instance consists of a problem space, an initial state and a set of goal states. Problem solving can thus be viewed as finding the sequence of operations that map the initial state to the goal state. When more than one operator is applicable at a state, and there is insufficient information to select between the operators, search is required. A search in which exactly one operator is applicable to each state is often called an algorithm, corresponding to the far right of Figure 1.

At each state, selection of the next operator constitutes knowledge retrieval. Application of this knowledge controls the process of moving from state to state. The problem space search discussed here is at the knowledge application level. In addition, knowledge retrieval involves searching the available body of knowledge for knowledge that is applicable in the

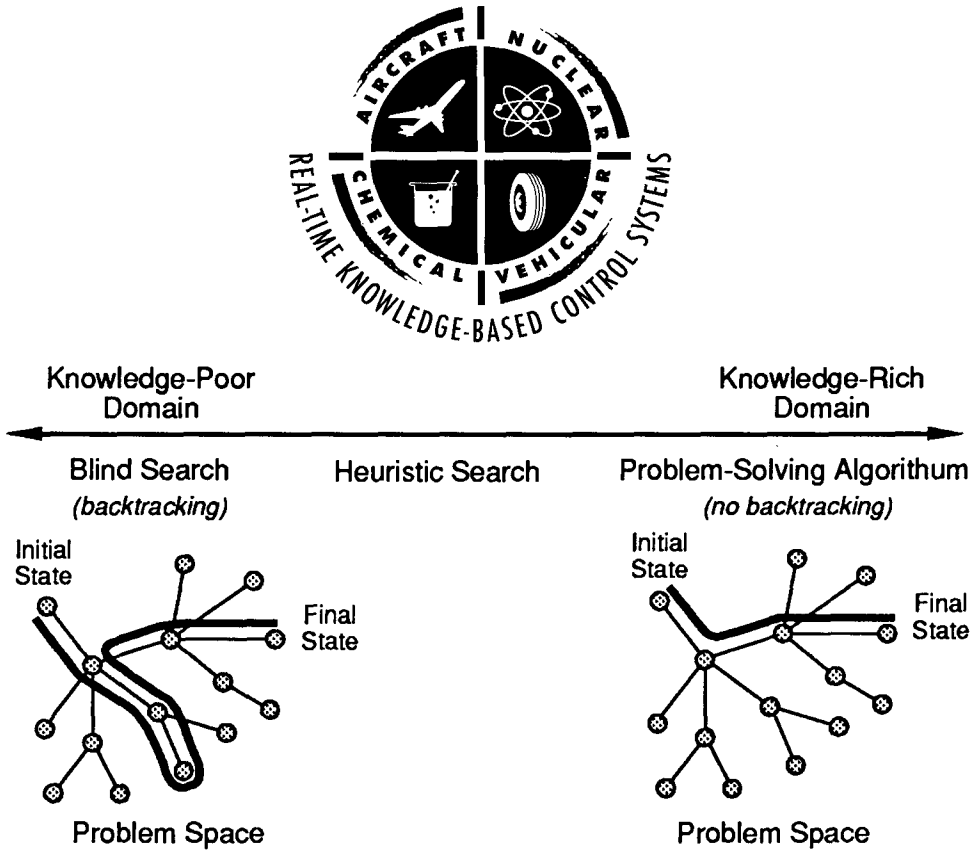


FIGURE 1.
The Search Spectrum

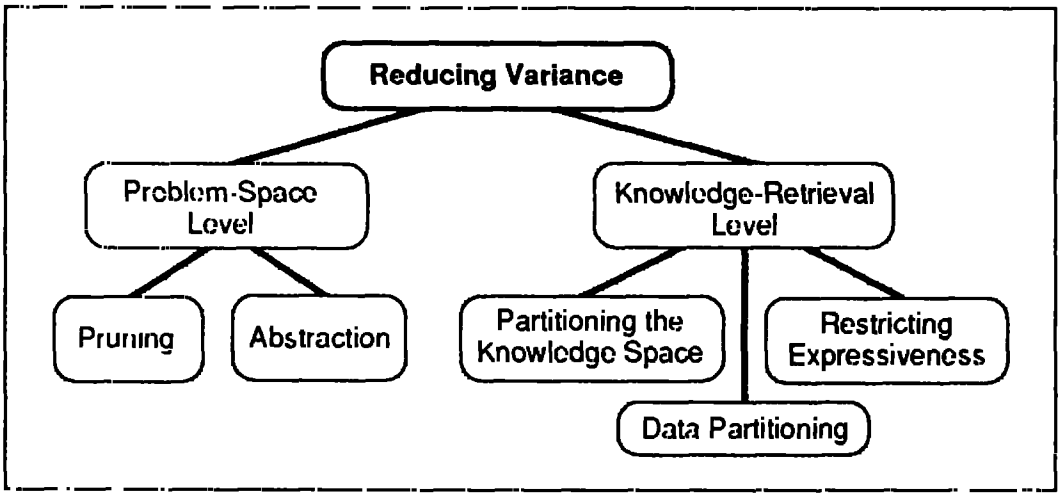


FIGURE 2.
Reducing Execution Time Variance of AI Tasks

THERE IS NO NOTION OF SEARCH OR BACKTRACKING AT THIS END OF THE SPECTRUM

current state. This is referred to as search in the *knowledge space* [27].

There is a fundamental difference between search in the problem space and search in the knowledge space. At the problem space level, the intent is to select the best possible operator applicable to the current state. On the other hand, at the knowledge retrieval level, the intent is to retrieve all knowledge that will influence the selection of the operator. Therefore, knowledge is available to prune and control the search in the problem space, but no comparable knowledge is available to restrict the search in the knowledge space.

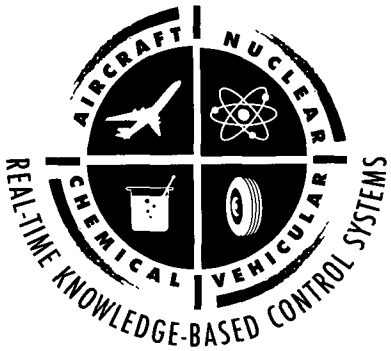
Reducing Variance

The only way to reduce the variance at the problem space level is to reduce the number of states searched. There are two ways to achieve this goal as shown in the left half of Figure 2.

The first technique is to *prune* the search space. This involves looking earlier at the states that are more likely to lie along the solution path. This corresponds to the classical "best-first search" technique. The best-first search technique uses heuristics to achieve the pruning of the search space. The better the heuristic, the lower the variance.

The second technique is *abstraction*. This involves creating an abstract problem space whose states are less detailed than those in the original problem space. A single state in the abstract problem space corresponds to multiple states in the original problem space. Search in the abstract space provides guidance for search in the original problem space, thereby reducing the variance. For example, in planning a route from Pittsburgh to New York, one can work at the abstractions of interstate highways, major roads, or smaller roads. Determining the entry and exit points at the abstraction of interstate highways constrains the number of roads examined at the lower levels.

We will present a sample application illustrating these techniques later.



At the Knowledge Retrieval Level

The amount of processing required in the knowledge retrieval phase depends on the amount of knowledge in the system, the size of the state (data), and the amount of data each piece of knowledge is potentially applicable to. The variance of the knowledge retrieval phase can be reduced by the techniques illustrated in the right half of Figure 2:

Partitioning the Knowledge Space. This allows us to avoid searching sections of the knowledge space that contains knowledge which is *a priori* known to not be applicable to particular pieces of data.

Partitioning the data. Many pieces of knowledge express relations, desired or otherwise, between multiple pieces of data. Partitioning the data allows us to avoid considering sets of data which are *a priori* known not to belong to that relation.

Restricting expressiveness. Highly expressive knowledge representations allow a large number of relations to be represented. When a part of the state changes, a large number of potential relations with the rest of the state have to be checked. Representation formalisms which restrict expressiveness *a priori* restrict the number of relations that need to be checked during every state change.

Next, we will present a problem-solving architecture which not only controls execution time variance at the knowledge retrieval phase, but also provides architectural mechanisms to support a range of problem-solving methodologies to reduce the variance at the problem space level.

CROPS5: An Architecture for Real-Time Problem Solving

We contend that the function of an integrated real-time problem solving architecture, as shown in Figure 2 is to provide *mechanisms* to partition, order and prune the search space; *predictable low-variance primitives* for problem solving; and *features* which facilitate easy integration into real-time operating environments. These three categories represent the mechanisms and features required for implementing the variance reduction methods.

We have designed a real-time problem-solving architecture, CROPS5, in accordance with the broad requirements we have outlined. It is based on the production system model [9], and borrows heavily from OPS5 for its syntax and semantics.

An OPS5 production system is composed of a set of *if-then* productions (rules) that constitute the *production memory* and a set of data items, called the *working memory*. The execution of an OPS5 program can be characterized by a cycle which has three phases: match, resolve, and act. Several efficient match algorithms have been designed for production systems. The best-known match algorithm is Rete [8]. The Rete algorithm performs matching using a special kind of data-flow network compiled from the left-hand side (the *if* part) of productions. This data-flow network passes items called *tokens* across the arcs between its nodes. Tokens are *partial instantiations* of productions. The basic computational step in the algorithm is to determine if the current set of partial instantiations (tokens) can be extended by matching more working memory elements against the productions involved. Checking whether a single token can be thus extended is the smallest logical unit of computation in the algorithm and is referred to as *token-processing* time. This time is typically on the order of 200–300 machine instructions.

There have been several efforts to use OPS5 or OPS5-like languages for real-time AI [6,23]. However, there are several charac-

teristics of OPS5 that limit its utility for real-time applications. Some of these include the following:

- Current OPS5 systems can be interrupted only at rule-firing boundaries. The time period between successive rule-firings is, on the average, large and has a high variance. As a result, the responsiveness and predictability of these systems is severely impacted.
- OPS5 systems consist of a single problem-solving stream, whereas real-time applications typically require multiple streams to be active simultaneously. While it is possible to simulate multiple streams within OPS5 by using a special data item as context identifier, switching between these streams is extremely expensive and has unpredictable processing requirements.

In addition, OPS5 does not have an interface to the external environments. and its pattern-matching time is unpredictable due to the expressive power of the language and the incremental nature of the match algorithm.

CROPS5 Mechanisms

The design of CROPS5 attempts to remove these limitations by explicitly addressing each of these problems. CROPS5 is based on CParaOPS5 [1], a parallel implementation of OPS5 developed at Carnegie Mellon University. CROPS5 consists of an OPS5 to C compiler and a run-time library in C. It is significantly faster than Lisp-based versions of OPS5, and does not suffer from the unpredictability of the garbage collection mechanism in Lisp. The support of match parallelism in CParaOPS5 allows CROPS5 to be easily extended to run on parallel processors for enhanced performance.

Figure 4 shows the architecture of CROPS5 as a task in a real-time system. CROPS5 is shown to coexist on a common computing platform with other hard real-time tasks like engine control and life support systems control. We address the issue of integration in greater detail later. First, we discuss the mechanisms provided by CROPS5 to efficiently partition, order and prune the search space. These mechanisms correspond to those in the leftmost branch of Figure 3.

Concurrent Prioritized Streams. As opposed to the single problem-solving stream in OPS5, CROPS5 [20] supports multiple problem-solving streams. Individual streams have disjoint sets of productions (and hence disjoint Rete nets). Each stream has a *private* working memory. The system uses a global working memory to communicate between streams. The mechanism of multiple streams facilitates knowledge base partitioning and data partitioning to reduce variance.

Associated with each stream is a stack of tokens that are yet to be matched and a buffer of working memory elements yet to be processed. A stream can therefore be characterized by a (Rete net, token-queue, working-memory-buffer) tuple. Fast and predictable switching between streams is achieved by switching between pointers to the corresponding tuples.

Knowledge-Based Scheduling and Context Switching. CROPS5 provides a dispatcher for the streams. The unit of time is the time to process a single token. Token counters keep track of the relative time spent in processing each stream, and can be used to implement a variety of user-defined scheduling policies.

Preemptability. While current OPS5 systems allow preemption only after all match processing is completed, CROPS5 allows preemption of the match process at token-processing boundaries. This ability to interrupt the match at fine-grained intervals not only improves the responsiveness of CROPS5, but also provides a mechanism to guard against excessive data rates and runaway match processing.

These mechanisms allow the user to partition, order and prune the search space.

Predictable Primitives

CROPS5 also improves the predictability of some of the basic problem-solving primitives indicated in the middle branch of Figure 3.

Earlier, we identified context switching and the points of preemptability as having a large variance in OPS5. The CROPS5 primitives for context switching and preemption have much lower variance. Experiments were conducted with CROPS5 running under the CHIMERA II real-time operating system [25] on a VME-based Ironics IV3220 single-board computer with a 68020 CPU running at 20 MHz. Experimental results show that the variance decreases significantly. The rationale for this decrease involves the following factors:

Predictable Context Switching. CROPS5 reduces the variance of context switching by providing an architectural mechanism to switch streams. To perform a context switch in conventional OPS5, the old context element had to be deleted, and the new context element inserted into the working memory. This results in a flurry of match activity. The minimum time required for replacing the context element is a function of the sum of the number of productions in the two contexts. The context switch time is thus highly dependent on the partial state of the match and the uncertainty in the environment, and is on the order of the time required for a match-resolve-act cycle. The numbers presented next are an

order of magnitude measure of the time required for a context switch. Notice that the variance in context-switching time is on the order of thousands of microseconds. In contrast, context switching in CROPS5 is two orders of magnitude faster and more predictable. Since each stream has its own Rete network, the new stream can immediately begin processing its data without having to spend significant time performing bookkeeping on the state of the match algorithm.

—Sample range in OPS5 context switch time: 1800 μseconds–4300 μseconds
—CROPS5 *Dispatcher Statistics:*
  * Context Switch to a different stream: 55 μseconds
  * Continue processing same stream: 9 μseconds

Predictable Preemption Points. Context switching can be done only at preemption points. Since preemption in conventional OPS5 systems is at the rule-firing boundary, there is a large variance in the points of preemption. CROPS5 reduces the variance in the points of preemption by reducing the granularity of preemption from the match-processing level to the token-processing level. Associated with the reduction in granularity is an order of magnitude decrease in variations and an equivalent increase in responsiveness.

—Sample range in OPS5 preemption points: 1800 μseconds–4300 μseconds
—CROPS5 *Granularity of preemption:*
  *Average token firing time: 134 μseconds

The predictability of the points of preemption, and the context switch time allows us to estimate *a priori* the time required to react to different events.

Predictable Match. Preemption points and context switching are not the only sources of variance. Match processing is another. Different techniques have been explored to bound the match processing to make it predictable [28]. These techniques have relied on restricting the expressive power of the language to eliminate match combinatorics. Of these techniques, the unique attribute formalism [28] appears promising. This formalism can be adopted in CROPS5 by restricting the types of productions written, and can be used to provide polynomially bounded match times for CROPS5 applications.

Integrating CROPS5 into Real-Time Systems

CROPS5 was designed for embedded real-time applications. Our approach allows CROPS5 to coexist with other real-time tasks on a common computing platform. Using operating system primitives, CROPS5 can run at any priority level while still guaranteeing deadlines of other tasks in the system. The integration features referred to in the rightmost branch of Figure 3 are addressed as follows:

Encapsulation within an AI Server. To guarantee temporal isolation between conventional real-time tasks and problem-solving tasks, we encapsulate all problem solving within an AI Server. We borrow the Server abstraction directly from the real-time scheduling community [24,26]. Servers have previously been developed to provide highly responsive aperiodic performance in periodic, hard deadline environments.

We utilize this approach to create an AI Server which is a bandwidth-limited task whose schedulability impact can be explicitly evaluated and guaranteed[1]. The AI Server differs from the Deferrable Server [26] and the Sporadic Server [24] in that it services both periodic and aperiodic tasks and has a different replenishment policy [20]. Given a set of conventional real-time tasks with deadlines, one can apply scheduling analytical techniques to determine the maximum possible capacity of the AI Server at any priority level. This technique is illustrated later, using an application example.

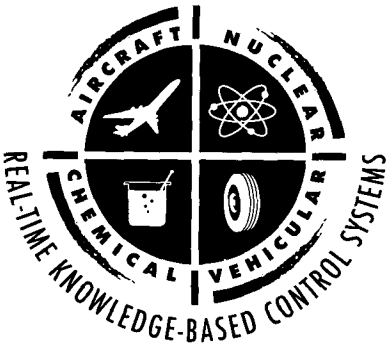
Responsiveness and Preemptability. The CROPS5 production system was specifically designed to be preemptive and priority driven. At the integrated-system level, the preemptability of the AI Server which encapsulates the CROPS5 system is identical to the preemptability of any other real-time task. Thus the schedulability analysis of the real-time task set, including the AI Server, can be performed in a uniform way. CROPS5 also provides a high degree of preemptability of problem solving, by allowing the problem-solving stream to be interrupted at the token-processing granularity. The fine granularity of preemption provides a high degree of responsiveness to the environment.

Environment Interface. The interface to the external world is through a Data Handler. The Data Handler accepts input from the other tasks in the system. This allows CROPS5 to run in embedded applications which process sensor data.

Integration with Conventional Systems: CROPS5 supports mechanisms to facilitate easy integration between the rule-based component and existing procedural software. A C-language interface is provided from the right-hand sides of productions, allowing external C functions to access and modify internal *working memory elements* of the production system.

CROPS5 is portable, and runs on most Unix and Mach machines. CROPS5 also runs on the ARTS [30] and CHIMERA II [25] real-time operating systems and is currently being ported to Real-Time Mach [29].

[1] Schedulability of a system is the level of resource utilization attainable before a deadline is missed.



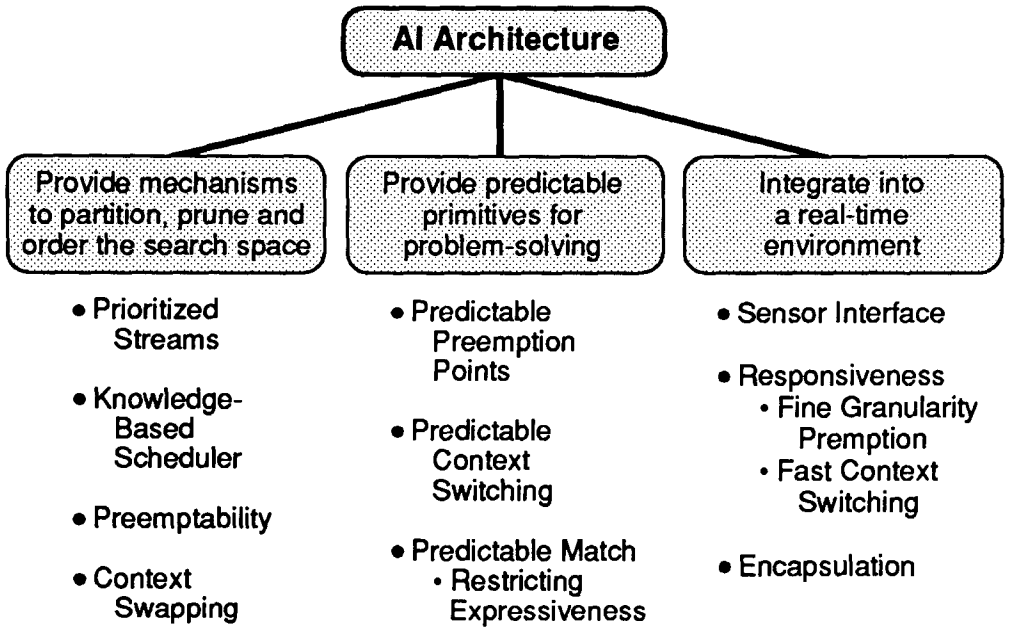


FIGURE 3.
Capabilities Required of Real-Time Problem-Solving Architectures

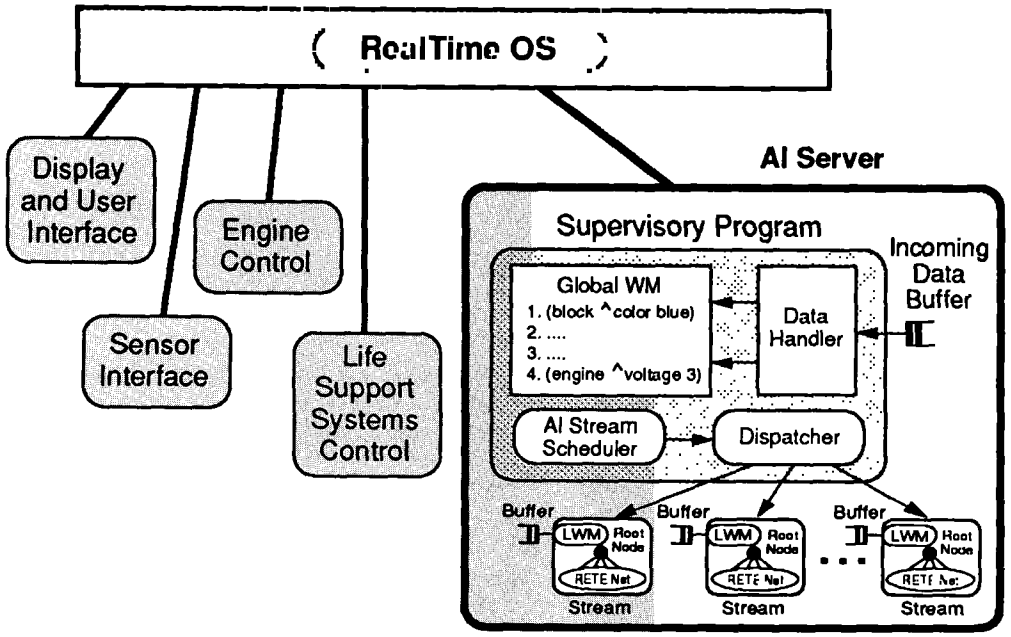


FIGURE 4.
CROPS5 in a Real-Time Environment

Application Example: The Collision Avoidance System

Two systems have been implemented using this architecture: an aircraft collision avoidance system and a dynamic factory-scheduling system [10]. We use the Collision Avoidance application as an illustration of how the techniques to reduce execution time variance improve the predictability of the system. The Collision Avoidance application has been in use as an experimental benchmark over the past few months. In this section, we describe how this system is implemented and substantiate our claims with experimental results.

Specifically, we will demonstrate the use of CROPS5 mechanisms to partition, order and prune the search space. We will compare the CROPS5-based implementation with an implementation using conventional OPS5, and show how the variance has been reduced. The application will be analyzed to determine the evolution of the system in response to changing data rates. We will demonstrate a best effort solution strategy with predictable breakdown points. We will also demonstrate that problem-solving processes can be successfully integrated with conventional hard real-time tasks on a common computing platform, while guaranteeing deadlines of all real-time tasks.

Application Background

The Collision Avoidance System (CAS) consists of an airplane receiver that listens to the signals emitted by radar transponders on other planes in response to interrogation signals from the host plane. By interpreting the transponder return, measuring the time delay of the response, and checking the angle the reply is coming from, the system can determine the altitude, distance and bearing of nearby transponder-equipped aircraft. While displaying the raw information is informative, it still must be processed to determine evasive action. Given the extremely limited response time requirement (< 10 secs.), any decision aid in this situation would be extremely beneficial. The CAS provides decision support capability to the pilot by generating advice to climb, dive, or turn right or left to avoid potential threats.

We implemented this application using the problem space search technique discussed earlier. In this case, the problem space is defined by the dimensions along which advice needs to be generated, and the number and orientation of the target planes around the host plane. The solution space is the set of final advice recommendations to the pilot.

The nature of the application imposes specific real-time performance and resource requirements [20]. Due to limited footprint space on modern aircraft, it is desirable for the collision avoidance system to coexist with conventional real-time tasks on the same computing platform. The collision avoidance system must also share communication networks and system resources with other real-time tasks, without causing them to miss their deadlines.

Demonstration of Variance Reduction

As a first step to solving this problem, we partition the overall functionality among a number of real-time tasks at the system level. The basic tasks are the reading of radar sensors, the knowledge-based processing of the information, and the display of the advice generated. The knowledge-based processing of the information is done using both CROPS5 and OPS5.

We implemented the collision avoidance application in three different systems to illustrate the reduction in problem-solving variance—OPS5 without context elements; OPS5 with context elements; and CROPS5.

In the OPS5 implementation without context elements, a single program handled advice generation for all planes. This program had the following limitations:

- *No preemptability.* Once the program starts execution, it stops only after advice for all planes is generated.
- *Unpredictable match processing.* Since multiple data elements can match the condition elements of rules, data-dependent combinatorics result.

This approach and programming style are not suited to most real-time applications. A real-time system has to be aware of the possible limitations in time. It must order its computations so the most important ones are done first. The nonpreemptability and large execution time variance of the conventional OPS5 implementation makes it difficult to provide performance guarantees for the CAS. We used the variance-reduction techniques discussed previously and the mechanisms provided by CROPS5 to reduce the execution time variance of this application at both the problem space level and the knowledge retrieval level.

Problem Space Level

Pruning: In this application, partitioning is done so that each stream handles the processing for a single plane. Ordering of the search is done by prioritizing the streams based on the degree of the perceived threat. Figure 5 illustrates our problem-solving strategy. In our application, all streams calculate their own priorities at the beginning of every data cycle. This priority is passed to the dispatcher. Advice generation is ordered, starting with the highest-priority threat. Streams handling planes moving away from the host plane are set to the lowest priority. This solution strategy is a best effort strategy, in that if time runs out before all planes are processed, the system would have considered the highest

priority threats and generated some partial advice [4].

Abstraction. For this problem, advice generation can occur at multiple levels of abstraction, each with varying amounts of detail [15]. The limiting cases of advice generation are characterized next, with other situations lying in between. On one hand, if a collision were imminent, advice is generated to immediately swerve to avoid the threat. On the other hand, if more time were available, factors such as weather, the range of available operating altitudes, and the expected intentions of the threat are taken into account before advice is generated.

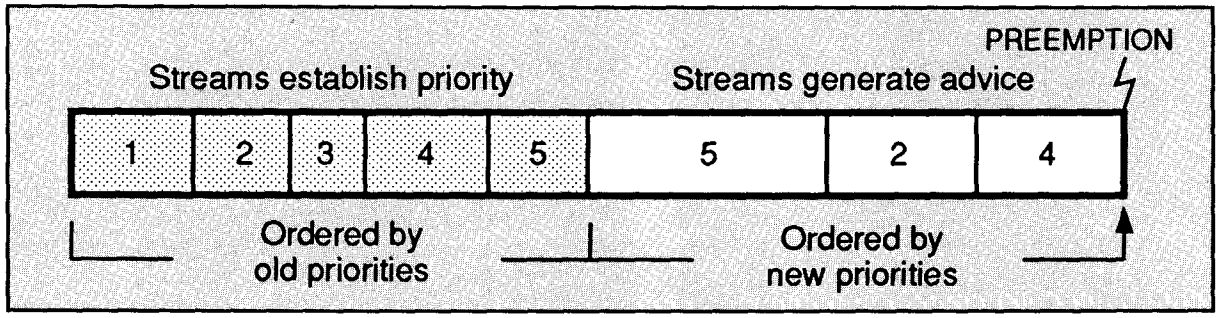
Knowledge Retrieval Level

Knowledge Base Partitioning. Since the knowledge base for this application is relatively small, we did not partition it.

Data Partitioning. The CAS uses data partitioning to ensure the predictability of the match process. In this application, we partition the data so that each stream looks only at the data associated with one plane. This ensures that at most, one working memory element matches any given condition element in the Rete net of the productions, thereby eliminating match combinatorics. The trade-off for limiting the match combinatorics is an increase in the number of streams (increased memory space). The relatively small size of the Rete net per stream allowed us to replicate streams without incurring too much memory cost. The maximum number of planes that can be considered is limited by the maximum number of streams.

Restricting Expressiveness. Since we could make the match predictable by data partitioning only, we did not consider restricting the expressiveness of the language to bound the match time.

FIGURE 5.
Problem-Solving Strategy



Our problem-solving strategy was to partition and order the computation to reduce the execution time variance of the application. A number of experiments were conducted to measure the performance of the application. We present some of these results to illustrate the reduction in variance.

In Table 1, we compare the performance of the OPS5 implementation using context elements, and the CROPS5 implementation using streams. To factor out dependencies due to programming style and implementation, all OPS5 problem solving was programmed using CROPS5—which is upward compatible with OPS5.

This experiment was conducted to determine the variation in the time taken to generate advice for one plane, as a function of the input data. We see that CROPS5 does better in two respects: predictability and efficiency.

The CROPS5-based system is relatively immune to changes in input data, while the OPS5-based system is not similarly immune even though it uses context elements. In this experiment, we see that in generating advice for one plane, OPS5 takes 222 ms (545 tokens) when there is one plane, vs. 274 ms (789 tokens) when there are five. CROPS5, on the other hand, varies only between 148 ms and 154 ms (310 tokens–318 tokens). While both execution times and token numbers are presented, the token numbers are not affected by the limitations of the time measurement process. The net variation is 244 tokens in OPS5 vs. only 8 tokens in CROPS5. This demonstrates an order of magnitude improvement in CROPS5 predictability.

Table 2 isolates the context switch performance of OPS5 and CROPS5 at each of the data points. Notice that CROPS5 context/stream switching is constant in this application, unlike OPS5. The tokens processed for the OPS5 context switch increase in a regular fashion since the code to process a plane is the same across all contexts. The context switch processing is typically unpredictable in OPS5, while remaining constant in CROPS5.

Moreover, we find that CROPS5 primitives speeded up the application over that of OPS5 by a factor of about 2. This is due to the elimination of redundant processing.

We now discuss how the CROPS5 implementation can be analyzed and integrated into a hard real-time system so that performance guarantees can be provided.

TABLE 1.

Comparison of OPS5 and CROPS5 Performance

Number of Planes	Number Prioritized	Number Advice Gen.	Time Taken (msec.)		Tokens Processed	
			OPS5	CROPS5	OPS5	CROPS5
1	1	1	222	148	545	310
2	1	1	233	153	606	312
3	1	1	248	152	669	314
4	1	1	262	154	730	316
5	1	1	274	152	789	318

TABLE 2.

Context Switch Performance

Number of Planes	Number Prioritized	Number Advice Gen.	Tokens Processed	
			OPS5	CROPS5
1	1	1	77	2
2	1	1	87	2
3	1	1	97	2
4	1	1	107	2
5	1	1	117	2

TABLE 3.

Execution Characteristics of the Real-Time Task Set

Periodic Task	Run-Time msec.	Period msec.	Utilization	Rate Monotonic Priority
Engine Control	6.00	50.00	0.120%	1
Sensor Monitoring	36.00	250.00	0.144%	2
AI Task	C_{AI_S}	1000.00	U_{AI_S}	3
Display & User Int.	100.00	1200.00	0.083%	4
Life Support	120.00	1500.00	0.080%	5
Total			$0.427 + U_{AI_S}\%$	

TABLE 4.

Comparison of Predicted vs. Measured Performance

Number of Planes	Number Prioritized	Number Advice Gen.	Predicted Time (msec.)	Measured Time (msec.)
5	5	1	421	380
5	5	2	498	445
5	5	3	574	517
5	5	4	650	583
5	5	5	726	660

Demonstration of System Analyzability

To demonstrate the ability of the problem-solving architecture to coexist with hard real-time tasks on a common computing platform, we consider the experimental task set shown in Table 3. This is the same task set illustrated in Figure 4. In addition to the radar sensor monitoring and display interface tasks required for the collision avoidance application, we have added two critical additional tasks with widely different responsiveness requirements—an engine control task with a very short period of 50 ms, and a Life Support Systems task with a relatively long period 1500 ms.

AI Server Capacity

Previously, we introduced our integrated, real-time problem-solving architecture and provided a qualitative discussion on how one can jointly schedule conventional real-time tasks and CROPS5 using an AI Server. Here, we will demonstrate, via the Collision Avoidance application example, how to assign priority to the AI Server and how to solve for its maximum capacity consistent with the RT tasks' scheduling requirements.

In general, the priority assigned to the AI server is a function of its response time requirements. In the Collision Avoidance application, the AI processing requirements are periodic with a period, T_{AI_S}, of 1000 ms. Since the conventional RT tasks are also periodic with periods summarized in Table 3, the Rate Monotonic scheduling algorithm [16] can readily be applied to evaluate the schedulability of the task set. The Rate Monotonic algorithm has been proven to the optimal fixed-priority scheduling algorithm for periodic tasks. Using this algorithm, the tasks are priority ordered by their rates—the shorter their period, the higher their priority. Note that assignment of priorities is based solely upon response time requirements and does not consider the relative semantic importance of the

tasks. When the schedulability of the entire task set cannot be guaranteed, the relative semantic importance of the tasks comes into play. In such a case semantic-based load shedding is appropriate. Otherwise, the highest schedulable utilization is achieved by assigning priorities solely on response time requirements.

Given the maximum run time, C_i, and period, T_i of each of the conventional real-time tasks, we now solve for the maximum capacity of the AI Server task, C_{AI_S} which will not violate the response time requirements of the lower-priority Display and Life Support System tasks. A tight schedulability bound can be calculated by an exact-case analysis consistent with the rate-monotonic algorithm [14]. This bound expressed as:

$$\forall i,\ 1 \le i \le n,$$

$$\min \sum_{j=1}^{i} C_j \frac{1}{lT_k} \left\lceil \frac{lT_k}{T_j} \right\rceil \le 1 \quad (1)$$

$$(k, l) \in \boldsymbol{R_i}$$

$$R_i = \left\{ (k, l) | 1 \le k \le i,\ l = 1, \ldots, \left\lceil \frac{T_i}{T_k} \right\rceil \right\}$$

takes explicit account of the actual task sets' period ratios and run times. Equation 1 yields a maximum C_{AI_S} of 516 ms which corresponds to a maximum utilization of the AI Server of $U_{AI_S} = C_{AI_S}/T_{AI_S} =$ 0.516 or 51.6%. Adding this to the utilization of the other real-time tasks, we get a total schedulable utilization of 94.3%. The following section provides an analytical treatment to answer whether the AI Server capacity is sufficient to meet the response time requirement of the Collision Avoidance application.

Application Analysis

To analyze the application, we use the following equation:

$$C_{AI} = n[t_{P_w}(s) + 2t_{sw}] + m[t_{lw}(d) + 2t_{sw}] + mt_{gw}(d, m)$$

where

- n maximum number of planes looked at and prioritized;
- m maximum number of planes for which advice is generated;
- $t_{P_w}(s)$ worst-case priority calculation time for a stream, which is a function of the prioritization strategy s. In our example, we use an algorithmic evaluation strategy to calculate the priority of the plane;
- t_{sw} stream-switch overhead;
- $t_{lw}(d)$ worst–case execution time in generating advice for each plane, and is a function of the number of dimensions d considered. In our example, we calculate advice along two dimensions, namely, altitude and turn;
- $t_{gw}(d, n)$ worst-case execution time to resolve advice conflicts at a global level, and is a function of the number of dimensions d and planes n considered.

Given the number of planes (n-5), the worst-case computation requirement would be to generate advice for all planes (n = 5). From Table 4, the predicted value of C_{AI} is 726ms, using the fastest problem-solving strategy available. The server capacity C_{AI_S} is 516 ms. Thus this application cannot generate advice for all planes. The largest predicted time (C_{AI} = 498 ms), which is less than the server capacity is the level of guarantee. In this case, we can guarantee at least five planes will be looked at, and advice generated for the two highest-priority planes.

To verify the validity of our predictions, we ran experiments to determine the execution times. The results comparing our predicted performance and the actual performance are summarized in Table 4. The effectiveness of the techniques to reduce variance, and the relatively small size of the application have allowed us to make useful predictions about the performance of the system.

Predictable "Best Effort" Evolution

In real-world situations, the application can sometimes be subjected to overloads beyond the design limits. Even though the design requirements dictate that there will be less than five planes in the vicinity at any given time, the system must degrade gracefully in cases where this requirement is exceeded. If this happens, the application must continue to prioritize and generate advice starting from the highest-priority plane, until time runs out. We should note that regardless of the strategies available, a minimum amount of execution time is needed before a useful result can be generated. This minimum time is determined by using the fastest problem-solving strategies available. If the minimum time is greater than the server capacity, the system breaks. From an engineering perspective, it is useful to be able to predict the breakdown points of the system.

If the number of planes, n, crosses a threshold, the system spends all the time classifying the data that it has no time remaining to generate useful advice. In our example, the limiting case of the system is reached when it has just enough time to calculate advice for the highest-priority plane. Assuming the worst-case advice calculation time for a single plane, and a zero global-advice conflict resolution time (since advice is generated for only one plane), we estimate:

$$\bar{n}_{\text{breakdown}} = \frac{C_{AI_S} - [t_{lw}(d) + 2t_{sw}]}{[t_{P_w}(s) + 2t_{sw})} \quad (2)$$

Notice that the number of planes

processed in the limiting case is a function of the available computation bandwidth, C_{AI_S}. Substituting values of the variables in equation (2), the system breakdown point is computed to be $\bar{n}_{breakdown} = 7$ planes. With seven planes, we can now generate advice only for the highest-priority threat. If the number of planes goes beyond seven, the system will not have enough computation bandwidth to generate any advice. This provides a design margin of two planes over and above the design requirement of five planes. This analysis allows us to *a priori* predict the effect of changing the number of planes and computational bandwidth on the performance of the application.

Summary

In this article, we argued that large execution time variance is the primary problem in providing practical performance guarantees for integrated, real-time problem-solving systems. We showed that this variance is due to search which is inherent to problem-solving tasks. Search occurs at two levels in a problem-solving task—the problem space level and the knowledge retrieval level. To reduce the execution time variance, it is necessary to reduce the extent of search at both these levels. At the problem space level, the search can be reduced by the application of problem-specific knowledge, whereas at the knowledge retrieval level, no problem-specific knowledge is available. At this level, the search must be reduced by knowledge-lean methods like knowledge and data partitioning, and reducing the expressiveness of the knowledge representation formalism.

To evaluate the effectiveness of these techniques, we implemented CROPS5, a real-time problem-solving architecture. CROPS5 provides predictable primitives at the knowledge retrieval level and supports problem-specific strategies that reduce variance at the problem space level. In addition, CROPS5 has been designed to be easily integrable into conventional real-time systems. Implementations of CROPS5 currently run on two real-time operating systems. CROPS5 has been used to develop a prototype aircraft collision avoidance system and a dynamic factory-scheduling system.

We used the collision avoidance system as a benchmark to compare CROPS5 with OPS5. Results show that the variance of the real-time problem-solving primitives in CROPS5 is significantly lower. Specifically, the variance in context switching is reduced by two orders of magnitude, and the variance in preemption points is reduced by an order of magnitude. Furthermore, these primitives allowed us to eliminate redundant computation, resulting in a speedup of a factor of about 2. We achieved predictable match processing in this application by partitioning data across multiple streams. Overall, we were able to predict application execution times to within 10% of actual measured values in an integrated real-time environment. The AI server was successful in ensuring that conventional real-time tasks on the same computing platform continued to meet their deadlines even after the CAS application was introduced into the system.

Using these techniques, we were able to demonstrate that it is feasible to reduce problem-solving variance and thereby provide practical performance guarantees for time-constrained problem-solving tasks in integrated real-time environments, while maintaining all performance guarantees for the conventional real-time tasks.

Acknowledgments

We thank Herbert Simon for his valuable comments in organizing the arguments in this article. We thank Dorothy Setliff and the other reviewers for their suggestions on improving the clarity of this article. We also thank Hiroshi Arakawa, Dave Stewart, Gary Hildebrand, Stephen Chou and Hide Tokuda for their help at various stages in implementing the experimental testbed.

References

1. Acharya, A. and Kalp, D. Release Notes on ParaOPS5 4.4 and CParaOPS5 5.4, 1989. Available with the CParaOPS5 distribution from the School of Computer Science, Carnegie Mellon University.
2. Bares, J., Hebert, M., Kanade, T., Krotkov, E., Mitchell, T., Simmons, R. and Whittaker, W. Ambler: An autonomous rover for planetary exploration. *IEEE Comput. 22*, 6 (June 1989).
3. Bastani, F.B. and Chen, I-R. The role of artificial intelligence in fault-tolerant process-control systems. In *Proceedings of the First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Vol. 2, June 1988.
4. Boddy, M. and Dean, T. Solving time-dependent planning problems. In *Proceedings Eleventh International Joint Conference on Artificial Intelligence*, Aug. 1989.
5. Chung, Jen-Yao, Liu, J.W.S. and Lin, K.J. Scheduling periodic jobs that allow imprecise results. *IEEE Trans. Comput. 39*, 9 (Sept. 1990).
6. Dickey, F.J. and Toussaint, A.L. ECESIS: An application of expert systems on manned space stations. In *Proceedings of the First Conference on Artificial Intelligence Space Applications* (1984), pp. 483–89.
7. Fishetti, M.A. TMI Plus 5: Nuclear power on the Ropes. *IEEE Spectrum 21*, 4 (1984).
8. Forgy, C.L. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artif. Intell. 19*, 1 (1982), 17–37.
9. Hayes-Roth, F. and Waterman, D.A. Principles of pattern-directed inference systems. *Pattern-directed Inf. Syst.* (1978), 577–601.
10. Holloway, L., Paul, C.J., Strosnider, J. and Krogh, B. Integration of behavioral fault-detection models and an intelligent reactive scheduler. In *Proceedings of the 6th IEEE*

International Symposium on Intelligent Control (Aug. 1991).

11. Kohn, W. Declarative hierarchical controllers. In *Proceedings of the Workshop on Innovation Approaches to Planning, Scheduling and Control* (Nov. 1990).
12. Laffey, T.J., Cox, P.A., Schmidt, J.L., Kao, S.M. and Read, J.Y. Real-time knowledge-based systems. *AI Magazine 9*, 1 (1988), 27–45.
13. Laffey, T., Weitzenkamp, S., Read, J., Kao, S., and Schmidt, J. Intelligent real-time monitoring. In *Proceedings of the AAAI-88 Seventh National Conference on Artificial Intelligence* (Aug. 1988, Lockheed Artificial Intelligence Center), pp. 72–76.
14. Lehoczky, J.P., Sha, L. and Ding, Y. The rate monotonic scheduling algorithm—exact characterization and average case behaviour. In *Proceedings of the IEEE Systems Symposium* (1989).
15. Lesser, V.R., Pavlin, J., and Durfee, E. Approximate processing in real-time problem solving. *AI Mag.* (Spring 1988).
16. Liu, C.L. and Layland, J.W. Scheduling algorithms for multiprogramming in a hard real-time environment. *J. ACM 20*, 1 (1973), 46–61.
17. Meystel, A. Intelligent module for planning/control of master-dependent systems. In *Proceedings of the First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Vol. 1, June 1988.
18. Mok, A.K. Formal analysis of real-time equational rule-based systems. In *Proceedings of the Real-Time Systems Symposium* (Dec. 1989).
19. Newell, A. and Simon, H. Human Problem Solving. Prentice-Hall, Englewood Cliffs, N.J., 1972.
20. Paul, C.J., Acharya, A. and Black, B., Strosnider, J.K. Concurrent real-time ops5: An architecture for real-time problem-solving. Tech. Rep., Carnegie Mellon University, May 1991.
21. Perry, T.S. and Wallich, P. A matter of margins. *IEEE Spectrum 23*, 11 (Nov. 1986).
22. Rao, M., Jiang, T.S. and Tsai, J.J.-P. Integrated environment for intelligent control. In *Proceedings of the First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Vol. 1, June 1988.
23. Skapura, D.M. and Zoch, D.R. A real-time production system for telemetry analysis. In *Proceedings of the 1986 Expert Systems in Government*, (1986), pp. 203–209.
24. Sprunt, B., Sha, L. and Lehoczky, J. A periodic task scheduling for hard real-time systems. *J. Real-Time Syst. 1*, 1 (1989), 27–60.
25. Stewart, D., Schmitz, D.E. and Khosla, P. Implementing real-time robotic sytems using chimera ii. In *Proceedings of the 1990 IEEE International Conference on Robotics And Automation*, (May 1990), pp. 598–603.
26. Strosnider, J.K. Highly responsive real-time token rings. Ph.D. thesis, Carnegie Mellon University, Aug. 1988.
27. Tokuda, H. and Mercer, C. Arts: A distributed real-time kernel. *ACM Oper. Syst. Rev. 23*, 4 (July 1989).
28. Tokuda, H., Nakajima, T. and Rao, P. Real-time Mach: Towards a predictable real-time system. In *Proceedings of the USENIX Mach Workshop* (Oct. 1990).
29. Tambe, M. and Rosenbloom, P. Eliminating expensive chunks by restricting expressiveness. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (Aug. 1989), pp. 731–737.
30. Tambe, M. and Rosenbloom, P. A framework for investigating production system formulations with polynomially bounded match. In *Proceedings of the AAAI-90 Eighth National Conference on Artificial Intelligence*, (Aug. 1990), pp. 693–700.

CR Categories and Subject Descriptors: D.4.7 [Operating Systems]: Organization and Design—*real-time and embedded systems;* I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods and Search—*backtracking, graph and tree strategies, heuristic methods*

Additional Key Words and Phrases: Chimera, decision aids, expert systems, knowledge-based systems, predictability, production systems, real-time systems, search, variance

About the Authors:

C.J. PAUL is a Ph.D student in computer engineering at Carnegie Mellon University. Between 1987 and 1989, he designed and built networked real-time process-control systems at IBM, Research Triangle Park, N.C. His research interests include parallel computer architecture, real-time scheduling theory and artificial intelligence. **Author's Present Address:** Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, cjpaul@ece.cmu.edu.

ANURAG ACHARYA is a graduate student at the School of Computer Science at Carnegie Mellon. His research interests include theory and implementation of programming languages, parallel processing and production systems. **Author's Present Address:** School of Computer Science, Carnegie Mellon University, Pittsburg, PA 15213, acha@cs.cmu.edu.

BRYAN BLACK is currently working at Motorola Semiconductor Products Sector in Austin. His research interests include VLSI design, processor architectures, and real-time systems. **Author's Present Address:**Motorola Semiconductor Products Sector, 505 Barton Springs Road, Suite 400, Austin TX, black@ece.cmu.edu.

JAY K. STROSNIDER is currently an assistant professor of electrical and computer engineering at Carnegie Mellon. He has 10 years of industrial experience developing distributed, real-time systems with IBM. His current research focus is upon integrating technologies within a real-time, scheduling-theoretic framework. **Author's Present Address:** Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, strsnider@ece.cmu.edu.

This research is supported in part by a grant from Northrop Research and Technology Center, by the Office of Naval Research under contract N00014-84-K-0734, and by the Naval Ocean Systems Center under contract N66001-87-C-01155