Extending a Blackboard Architecture for Approximate Processing

K. S. Decker V. R. Lesser R. C. Whitehair [1]
Computer and Information Science Department
University of Massachusetts

COINS Technical Report 89–115
January 29, 1990

Extending a Blackboard Architecture for Approximate Processing

K. S. Decker V. R. Lesser R. C. Whitehair [1]
Computer and Information Science Department
University of Massachusetts

COINS Technical Report 89–115
January 29, 1990

Abstract

Approximate processing is an approach to real-time AI problem solving systems in domains where there are a range of acceptable answers in terms of certainty, accuracy, and completeness. Such a system needs to evaluate the current situation, make time predictions about the likelihood of achieving current objectives, monitor the processing and pursuit of those objectives, and if necessary choose new objectives and associated processing strategies that are achievable in the available time. In this approach, the system is performing "satisficing" problem solving, in that it is attempting to generate the best possible solutions within available time and computational resource constraints.

Previously published work[1] has dealt with this approach to real-time; however, an important aspect was not fully developed: the problem solver must be very flexible in its ability to represent and efficiently implement a variety of processing strategies. Extensions to the blackboard model of problem solving that facilitate approximate processing are demonstrated for the task of knowledge-based signal interpretation. This is accomplished by extending the blackboard model of problem solving to include data, knowledge, and control approximations. With minimal overhead, the problem solver dynamically responds to the current situation by altering its operators and state space abstraction to produce a range of acceptable answers. Initial experiments with this approach show promising results in both providing a range of processing algorithms and in controlling this dynamic system with low overhead.

[1]This work was partly supported by the Office of Naval Research under a University Research Initiative grant, number N00014-86-K-0764, NSF-CER contract DCR-8500332, and ONR contract N00014-89-J-1877.

1 Introduction

An important property of real-time systems is that task resource requirements be predictable. When complex, cognitive computational tasks are incorporated into traditional real time systems, problems arise because it is difficult to structure the task algorithms so that guarantees can be made *a priori* about the computational requirements of the tasks.[1] One reason it is difficult to put a bound on the amount of time that a complex cognitive algorithm may take is that they are often formulated as search algorithms. These algorithms operate under a large number of situation-dependent constraints, arising from the characteristics of the input data and the problem specification. In a signal interpretation task, for example, it may take a dramatically longer time for the search to be completed in a situation where there are multiple phenomena that need to be interpreted (resulting in a large amount of data) versus a situation with a single phenomenon (and a small amount of data). There is also significant variance in the time needed to reduce the uncertainty in an interpretation of a situation where erroneous input data has coincidentally produced highly plausible false interpretations as opposed to a situation where erroneous input data can be easily discarded. This second scenario is particularly troublesome for time critical situations because processing time can be only partly predicted from the amount or other surface characteristics of the input data. More reliable predictions can be made only after processing has progressed. The lack of fixed time bounds on algorithms can be overcome in domains in which a range of acceptable answers (in terms of certainty, accuracy, and completeness) can be computed (using a range of computational resources). When applied to real-time problems in AI systems, this "satisficing" approach [2,1] can be viewed as an attempt to construct problem-solving systems that produce the best possible answer in a given amount of time.

Our model of real-time AI problem solving is based on the system having explicit goals and plans (strategies to reach these goals), evaluating the current situation, and making predictions about the amount of time tasks will take based on this evaluation. If the predictions indicate the likelihood of not reaching the current goals with the current strategies, then the system modifies or chooses new strategies, pursuing the new tasks that will allow the goals of the system to be achieved within the specified deadline. The progress of these tasks is monitored, and this may trigger re-evaluation of the situation and new predictions, resulting in the choice of new, less acceptable objectives for the system. While previous work [1] has discussed ways of making predictions and monitoring tasks, this work is concerned with the architectural features of a problem solver that permit a smooth integration of a diverse set of task solution methods.

Satisficing problem solving can be accomplished through the use of *approximate processing* techniques. Approximate processing assumes that there are a range of acceptable answers, each requiring a different amount of time to generate — an answer may be more or less certain, precise or imprecise, and complete or partial. Trade-offs along all of these axes are possible and the problem solver must make them in a timely, informed manner, depending on the current situation (such as the amount and characteristics of the arriving data, the results of intermediate processing, or the emerging set of processing goals). For example, it might be more important in a given situation to pinpoint the exact location of some object than to determine the type of the object, even though in the optimal solution both pieces of information are desirable (and, in fact, the type of an object might constrain its location)[1]. Approximate processing techniques can decrease the variability in processing time both by changing the criteria for acceptable solutions so that processing can use simplified algorithms that decrease the overall processing time, and by using algorithms that abstract data to minimize the characteristics of the data that can cause significant processing time variability. By representing both the average processing time and the variance, the system can decide that if the deadline for a task is closer than the average time for a task then approximations must be used, but if the deadline is farther away than the average time (but still within the variance), then the current processing strategy can be continued and monitored closely so that approximate strategies can be exploited if necessary.

Other methods can also be used for satisficing problem solving. Many iterative numerical

[1]The real time response required at the level of these large, complex algorithms is on the order of hundreds of milliseconds to tens of seconds.

approximation algorithms will produce a more precise answer with more time. A class of algorithms called *anytime algorithms*[3,4,5,6], exhibit these characteristics: they can be preemptively scheduled; they can be terminated at any time and produce an answer; and the answers returned improve in some well-behaved manner with respect to time. In contrast, approximate processing algorithms can be terminated at any time *after* a given deadline and will return an answer, and the answer returned improves in a step-wise manner as the initial deadline moves farther away. Anytime algorithms are a subset of approximate processing algorithms, so the techniques discussed here could be applied to them. The deliberation scheduling algorithms that have been developed for anytime algorithms cannot necessarily be easily extended to approximate processing algorithms, but an approximate processing algorithm could be given a short deadline which is increased by some minimum amount so that it could work in an anytime algorithm-based system with an associated loss in performance. In domains where deadlines are likely to be hard to estimate and change often, anytime algorithms may exhibit superior performance. However, in domains where deadlines seldom change or can be effectively estimated, approximate processing algorithms, which may not produce an answer until just before the deadline, may provide better answers (in terms of certainty, precision, and completeness) than anytime algorithms which will always have an answer ready.

We discuss an approach to real-time AI that includes approximate data and intermediate representations, approximate reasoning methodologies, and approximate control strategies. Approximate processing requires that the problem solver must be able to represent and reason with uncertain, imprecise, and incomplete information, and also dynamically alter its representation and reasoning in response to changes in the problem solving situation occurring outside of its control (including changing deadlines). Thus, data, domain knowledge, and control representations are all affected by approximate processing. For example, it is no longer sufficient to represent only individual pieces of data — it must be possible to represent groups of data and to reason collectively about those groups.

The different and complex reasoning methodologies required by approximate processing each carry their own overhead; the problem solver must integrate these methodologies to reduce that overhead. A single control strategy is not sufficient because the order and manner in which processing occurs must be flexible and depend on the current situation. Furthermore, an integrated representation is necessary because the problem solver may develop a partial solution using one method (perhaps an extremely precise solution) only to realize (by its own computation or by a change in the current situation) that a change of approach is needed (to meet some deadline). The problem solver must be able to exploit the existing partial results, rather than throw results away and start from scratch in some new representation.

A signal interpretation task, where groups of agents receive data from sensors detecting moving objects in an environment and try to construct a model of what is occurring in that environment, provides many of the situational constraints discussed above. Data from the sensors may be certain or uncertain, and precise or imprecise in its characterization of a signal's location and class. Constraints in the classes of signals that can occur simultaneously, their spatial separation, and the velocity and acceleration constraints of objects all provide degrees of precision and certainty that can be sacrificed by approximate processing algorithms to produce a timely answer. Ignoring individual pieces or whole classes of data can give rise to only partial (versus complete) answers. Deadlines arise externally, or from high-level considerations, such as the need to identify a hazardous pattern before it becomes unavoidable.

The initial experiments described here center around applying the ideas of approximate processing to such a signal processing task. Since the signal processing task was already implemented using a blackboard problem solving architecture, our focus has been on how to extend such an architecture for approximate processing. This includes defining new data, knowledge, and control representations and defining the types of approximations applicable to signal processing, such as data clustering.

This paper describes approximate processing, how to extend a blackboard system to support approximate processing, how to control these mechanisms, and demonstrates experimentally that approximate processing is a useful technique for decreasing the processing time of an algorithm while generating acceptable (but less certain, less precise, or less complete) solutions. Section 2 explains the general principles behind approximate processing and our model of real-time control. It discusses types of approximations independent of any domain. Section 3 introduces the example signal

interpretation domain, distributed vehicle monitoring, and explains how the ideas in approximate processing can be applied to this domain in particular. Section 4 discusses an architecture based on the blackboard problem solving paradigm that supports all forms of approximate processing. It shows how the approximate processing ideas can be implemented and how the resulting architecture can be successfully controlled. Section 5 discusses how approximate processing affects the results of problem solving, and gives a detailed example of the action of this architecture on a representative domain problem. Finally, Section 6 describes what has been learned from these initial experiments and what extensions are needed.

2 Overview of Approximate Processing

Approximate processing requires that a problem solver reason about its objectives, its problem-solving state, and its plans for achieving its objectives from the current state. Using domain specific knowledge, a problem solver capable of approximate processing builds plans for satisfying an objective. The problem solver then estimates and monitors the feasibility of its plans and modifies them in situations where it predicts that available resources are inadequate. Plan modifications involve making tradeoffs between the resource requirements needed to generate a solution, such as time, and solution characteristics that define a range of potentially acceptable solutions, such as certainty, precision and completeness [1]. The choice of tradeoffs is based on domain specific knowledge of the utility associated with each member of the range of potentially acceptable solutions. Thus, when the problem solver estimates that available resources are inadequate to generate the highest-quality, or optimal, solution, it attempts to make the solution space less costly to search by employing approximate processing in order to generate a satisficing solution. The problem solver's specific choice of approximation techniques is made to maximize the utility of the resulting approximate solution.

Approximate processing should have the following properties:

1. Approximate processing should be *well-defined*. A problem solver should understand the effects an approximation will have on solution quality so that it can determine whether or not the approximation will satisfy an objective. This is in contrast with *ill-defined* approximations that can have unintended effects on solution quality.

2. Approximate processing should be *well-formed*. A problem solver should be able to reason with and combine partial results without regard to their level of approximation. This is required to meet approximate processing objectives – to incorporate minimal approximations into otherwise optimal solutions in order to satisfy time constraints. This implies that in scenarios where there are adequate resources, all solutions generated by the system should be optimal and that in scenarios where additional resources become available after approximate solutions have been formed, the system should improve solution quality by refining approximate solutions. This is in contrast to *ill-formed* approximations that a problem solver may not be able to integrate with existing partial results or may not be able to refine at some later point.

3. Approximate processing should be consistent with processing that would produce an optimal solution. Approximations should never eliminate from consideration a result that would not have been eliminated by optimal processing.

The three general classes of approximation that a problem solver can use are search approximation, data approximation, and knowledge approximation. Approximate search limits the areas of the search space that are explored. For example, a prediction of the utility of potential processing can be used to limit search. A problem solver might eliminate the processing of low-rated input data because it is not expected to lead to any worthwhile solutions. However, limiting search in this way can be an ill-defined approximation technique. This strategy does not take global relationships into consideration and may inadvertently eliminate critical processing, such as processing necessary to

differentiate two mutually exclusive solutions. In a worst case scenario, it could preclude a system from finding a correct solution.

Two general strategies that can minimize these effects of approximate search are the elimination of corroborating support and the elimination competing interpretations. Eliminating corroborating support is a strategy that avoids processing that would otherwise support an existing partial result. If a problem solver can identify processing that does not generate new solutions but only increases the certainty of an existing solution, it might be able to eliminate that work with predictable effects. In so doing, the problem solver must consider the possibility that eliminating corroborating support can reduce the certainty of a solution to the point where inferior alternatives appear more credible.

Eliminating competing alternatives is an approximate search strategy based on first characterizing the relationships between potential solution paths, and then pruning paths that lead to potential solutions that are obviously inferior to a mutually exclusive alternative. If a problem solver has great confidence in its ability to identify work that leads to inferior alternatives, avoiding that work will conserve resources and will have minimal impact on the quality of the correct solution. This is, however, a very difficult determination for a problem solver to make. There is usually some risk that an eliminated alternative is the correct solution. As a consequence, eliminating competing alternatives decreases the certainty that a given solution is the correct alternative.

Data approximation is a strategy that attempts to efficiently apply constraints to aggregations of data and it is particularly useful in domains with large quantities of noise. Instead of processing each individual piece of data, a problem solver can aggregate data into an abstract unit and then process the unit as a whole. For example, if the initial input data is very noisy, and if the noise and the correct data have very similar characteristics, they can be clustered into a single datum with characteristics encompassing both.

Data approximations reduce solution quality in two ways. First, when data is combined into a single unit, the problem solver will have to ignore some attribute of the solution or use a more abstract representation, such as a range of values. This will reduce the precision of the solution. Second, data approximations can lead to ambiguity. Since the data aggregation will be treated as a single unit, it will not be clear which individual elements were actually used to generate the solution. This will increase the uncertainty associated with the solution.

Knowledge approximations simplify or eliminate the constraints used by a problem solver. For example, a problem solver may have a very accurate algorithm for determining some solution attribute. However, this algorithm may be computationally expensive. If the problem solver can predict the effects on solution quality of using a less costly, under constrained algorithm, it can appropriately substitute this algorithm when required by resource constraints. In this case, certainty is reduced because the problem solver does not know what effect the full application of constraints would have had.

By incorporating well-defined and well-formed approximations into modified plans, a problem solver can improve its ability to meet real-time deadlines by reducing the estimated resources required to generate a solution and by improving the accuracy of its predictions about the time resources required to generate "acceptable" solutions.

3 The Domain and an Approximate Processing Example

3.1 The DVMT - An Example Domain

In order to understand the impact of approximate processing on the structure of a problem solver, an abstracted version of the task of knowledge-based signal interpretation, as implemented in the Distributed Vehicle Monitoring Testbed (DVMT), has been studied. The domain of the DVMT is the monitoring of acoustic signals generated by moving vehicles and detected by acoustic sensors. In the simulated domain environment, a vehicle generates sounds that are described in terms of signal groups, which are then described in terms of actual detectable signals. A vehicle may be more likely to generate some signal groups than others, and signal groups may have multiple or uncertain definitions. Signal strength is related to the vehicle loudness and velocity. Signals generated by vehicles may degrade over distance, or be reflected and cause ghosting or acoustic masking by

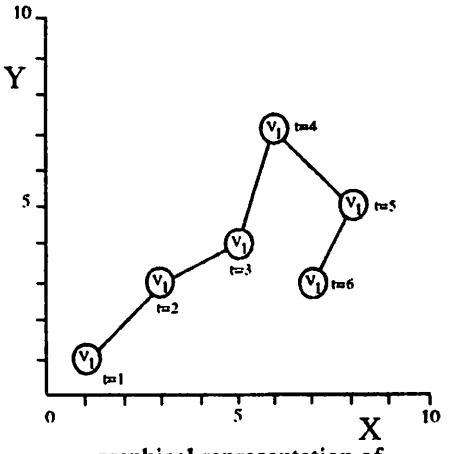
Example Hypothesis:

blackboard level = vehicle

event-class = automobile

belief = 0.9

time-locations = ((1(1,1) (2(3,3)) (3(5,4))
(4(6,7)) (5 (8,5)) (6(6,3)))



graphical representation of example hypothesis

Figure 1: Example of DVMT Hypothesis

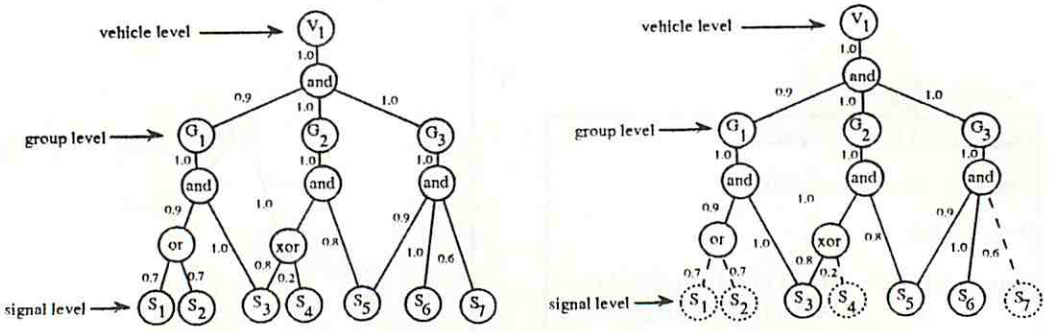
atmospheric effects. The simulated sensors may have different accuracies for different ranges of signals, exhibit directional sensitivity, or introduce random or correlated errors in the incoming signals.

The DVMT itself simulates a network of vehicle monitoring nodes, where each node is a problem solver that analyzes acoustically sensed data in an attempt to identify, locate and track patterns of vehicles moving through a two-dimensional space. A problem solving node is responsible for a specific area and attempts to recognize and eliminate erroneous sensor data as it integrates the correct data into an answer map. Each problem solver has a blackboard architecture with blackboard levels and knowledge sources appropriate for vehicle monitoring. The four principal blackboard levels are: *signal* (for low-level analysis of signal data), *group* (for collections of signals produced by some part of a vehicle), *vehicle* (for collections of groups that correspond to single vehicles), and *pattern* (for collections of vehicles acting in a coordinated manner, such as refueling). Knowledge sources perform the basic problem solving tasks of extending and refining partial solutions, or *hypotheses*.

A hypothesis is characterized by an *event class* (classifying the frequency or vehicle type), by a *blackboard level* (corresponding to the level of data abstraction/processing), by one or more *time-locations* (where a vehicle was at discrete sensed times), and by a *belief* (confidence in the accuracy of the hypothesis). An example hypothesis is shown in Fig. 1. The dark line connecting each of the points in the track indicates the implied path of the vehicle in a two dimensional space during times 1 through 6.

DVMT knowledge sources (KSs) can be divided into three basic classes, two associated with domain problem solving, synthesis and track extension, and one used for communication. *Synthesis* involves combining one or more related hypotheses at one level of the blackboard into a new hypothesis at the next higher level of the blackboard. Knowledge required to generate hypotheses at a given level of the blackboard is stored in a grammar that summarizes knowledge about how different distributions of frequencies are indicative of certain domain events. Figure 2 part (a) is an example of a grammar used in the DVMT. The grammar is represented as a modified AND-OR-XOR tree with a domain specific belief combination function associated with each AND-OR-XOR node[2] [7] and with a domain event associated with each leaf node. A *necessity* metric (with range $0 \leq necessity \leq 1$) is associated with each arc in the tree and the function $f_n(subtree, necessity)$ represents the degree to which the subtree defined by the arc

[2]To enhance the DVMT's utility as a testbed, belief combination functions have been implemented in a modular fashion to allow maximum flexibility in tailoring the DVMT for a specific experimental purpose. As a result, it is necessary to refer to belief combination functions abstractly as f_{and}, f_{or}, and f_{xor}. These functions correspond to the functions represented by AND, OR and XOR nodes in the grammar trees.



(a) Grammar A, an example of actual grammar used to synthesize location hypotheses from the signal level to the group level and from the group level to the vehicle level

(b) effects on grammar A of eliminating corroborating support with necessity ≤ 0.7

Figure 2: Example of a Grammar and the Effects of Approximate Search

is necessary for the existence of the subtree's parent. Based on Grammar A, given overlapping signal level hypotheses with event classes of (1 and 3) or (2 and 3), a synthesis knowledge source could generate a hypothesis with event class 1 at the group level of the blackboard. The belief of G_1 would be given by $f_{and}(f_n(f_{or}(f_n(S_1, 0.7), f_n(S_2, 0.7)), 0.9), f_n(S_3, 1.0))$. Similarly, overlapping hypotheses G_1, G_2, and G_3 could be used to generate a V_1 hypothesis with belief = $f_{and}(f_n(G_1, 1.0), f_n(G_2, 1.0), f_n(G_3, 1.0))$.

Track extension KSs output track hypotheses — a track is represented by a list of sequential time-locations that obey the velocity and acceleration constraints for a vehicle type. Combined with a vehicle's maximum velocity, the acceleration constraint limits the distance a vehicle can travel and the vehicle's turning radius. There are three types of extension KSs: *creation*, where location hypotheses are merged into a track; *forward* and *backward extension*, where a track is extended forward or backward in time by a vehicle location; and *merging*, where two tracks are merged into a single, longer track.

Every knowledge source has a *precondition* that is a relatively inexpensive function that determines the costs and benefits of running a particular KS given a context (triggering hypothesis). In the original DVMT, only benefits are estimated (specifically, the estimated belief in the hypotheses that would be output if the KS were to run).

In the DVMT, control and problem solving are interleaved. The DVMT low-level control loop can be summarized as:

1. Deciding what potential work can be done (hypothesis-to-goal mapping),
2. Relating potential work to existing goals (goal merging and subgoaling),
3. Deciding how to go about achieving them (goal-to-KSI mapping),
4. Choosing which of these potential activities to execute (managing the agenda).

When a hypothesis is formed, goals are created to represent the possible extensions and abstractions of that hypothesis[8]. The goals are stored on a separate goal blackboard with levels that mirror those on the hypothesis blackboard, and each goal is given a rating. The *goal processor* steps through the possible KSs that might satisfy each new goal and checks their preconditions to find the KS that can best satisfy the goal (if any). A *knowledge source instantiation* (KSI) is then created from the KS, the goal, and the hypothesis that triggered the formation of the goal[3]. Each KSI is assigned a rating based on the results of its precondition and the rating of the goal it is expected to satisfy, and is placed in the KSI agenda. The highest rated KSI is then executed.

[3]This is a simplistic explanation of goal processing in the DVMT. For more details on subgoaling and goal merging, see [8].

3.2 Approximate Processing in the DVMT

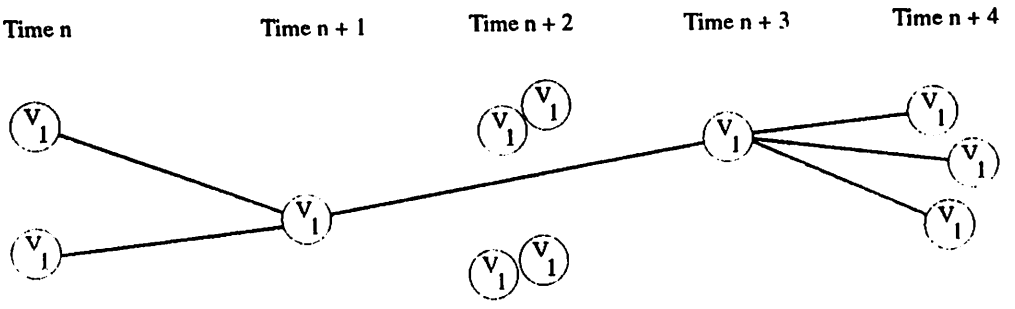
Approximate processing strategies can be used to reduce the expected cost of a problem-solving activity, μ_c, and to bound problem-solving resource requirements by limiting the variance, σ_c^2, of μ_c. Specifically, in the DVMT, a problem solver can trade solution quality for time by eliminating corroborating support, by clustering data, and by simplifying domain knowledge in ways that will reduce μ_c and σ_c^2.

Eliminating corroborating support naturally decomposes into two general strategies, one for synthesis operations and one for extending partially formed tracks. For a synthesis knowledge source, μ_c and σ_c^2 are functions of the number of input hypotheses. Eliminating some inputs will reduce both μ_c and σ_c^2. Furthermore, this approximation can be done in a well-defined manner. When a knowledge source performs a synthesis operation, it can determine the significance that supporting data will have on the synthesized result from the *necessity* of the supporting data. Supporting data with a relatively high *necessity* will be very significant and data with relatively low *necessity* will have little impact. Thus, a synthesis knowledge source can use its understanding of *necessity* to make general predictions about the effects on solution quality of omitting the search for corroborating data with arbitrarily low *necessity*. For example, using domain specific knowledge, a problem solver might determine that it needs to use approximate processing and it might predict that eliminating corroborating support with $necessity \leq 0.7$ will produce an approximate solution satisfying its objectives with the maximum expected utility of any approximation. Figure 2 part (b) represents the effects of eliminating corroborating support with $necessity \leq 0.7$: arcs with $necessity \leq 0.7$ are assumed to be "true" and the belief combination function for G_1 is simplified to $f_{and}(f_n(1.0, 0.9), f_n(S_3, 1.0))$.

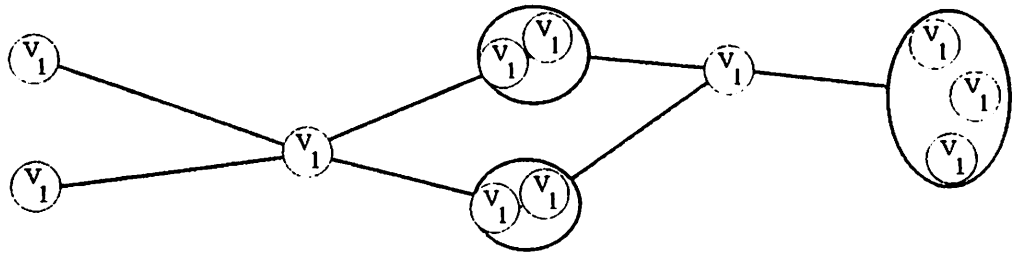
Extension knowledge sources form track hypotheses by constructing a consecutive sequence of time-locations. Each time-location provides corroborating support for the resulting track hypothesis. A problem solver can reduce μ_c and σ_c^2 by limiting the amount of time it expends searching for each point in the sequence. For example, a problem solver could form tracks by skipping some time-locations. This will reduce the certainty and precision in the final track solution, but it is a well-defined approximation. The certainty will be reduced because the problem solver is making an assumption that might not be true – that vehicle data exists for the skipped time-location. The precision will be reduced because the problem solver will not possess an accurate model of the vehicle's position in the skipped region. However, since extension knowledge sources have some understanding of the contribution each time-location makes to the certainty of the final solution and since the knowledge source can use velocity and acceleration constraints to limit the size of the region where supporting vehicle data could exist in the skipped time-location, this is a well-defined approximation. This strategy can be particularly successful in situations where a problem solver is attempting to extend a partial track through a noisy region to another partial track. By "skipping over" the noisy region, precision and certainty will be reduced, but the combinatorial explosion of potential tracks will be avoided, thus reducing μ_c and σ_c^2.

In general, a large amount of noise in a DVMT scenario leads to a combinatorial explosion of potential interpretations. This makes it very expensive to obtain optimal solutions because each piece of data must be individually analyzed. Furthermore, it is nearly impossible to bound σ_c^2 because a problem solver cannot know how many plausible interpretations exist without processing the data. For example, in some instances, 20 individual hypotheses might lead to over 1,000 different interpretations and in other instances, 1,000 individual hypotheses might result in only one plausible interpretation. By clustering data into larger units with attributes defined as ranges for type or location, a problem solver can reduce μ_c and σ_c^2 by treating the aggregation as a single unit. Although this technique reduces the certainty and precision of subsequently generated solutions, it is a well-defined approximation since an understanding of the scope of a cluster can be formulated from its components. (More extensive discussions of this consideration are included in subsequent sections.)

The example shown in Fig. 3 demonstrates how approximate processing can be used in the DVMT to moderate the combinatorial explosion of possible interpretations. In this scenario, there is vehicle data for five sensed times. Due to noise in the environment, there are multiple vehicle



(a) tracks generated by using approximate search to "skip over" vehicle data at Time n + 2



(b) tracks generated by using clustered approximations of data at Time n + 2 and Time n + 4

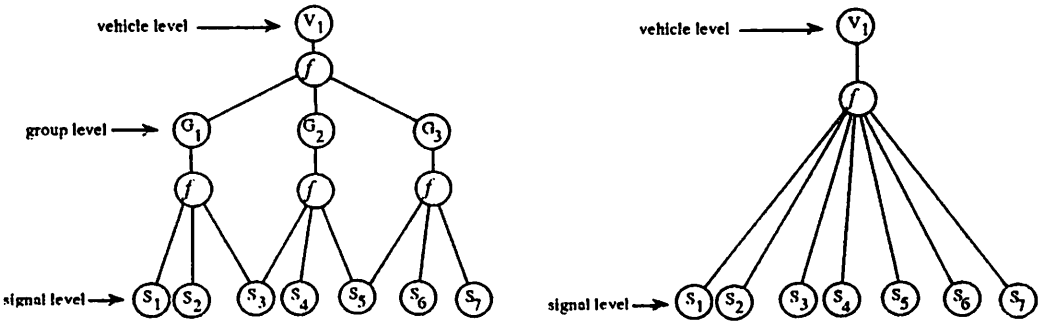
Figure 3: Example of Search and Data Approximations in the DVMT

hypotheses at Time n, Time $n + 2$ and at Time $n + 4$. If exhaustive search were used to generate exact tracks through Time $n + 4$, a problem solver would have to interpret and differentiate (as much as possible) 24 vehicle tracks.

Part (a) shows the tracks formed by skipping over the vehicle data from Time $n + 2$. This search approximation drops the number of generated solution tracks to six. Since the problem solver cannot be sure that vehicle data exists for the skipped time, Time $n + 2$, the approximated tracks have lower certainty than exact tracks. Furthermore, since the problem solver does not know where the track could have been extended to at Time $n + 2$, the approximated tracks have lower precision than exact tracks.

Part (b) shows the tracks formed by first clustering the data. This approximation drops the number of solution tracks to four. Since the problem solver cannot be sure that any individual member of a given cluster satisfies track extension constraints, the approximated tracks have lower certainty than exact tracks. And again, the problem solver does not know which of the clustered locations the track traverses, so the approximated tracks have lower precision than exact tracks.

Knowledge approximations can reduce μ_c and σ_c^2 by ignoring or simplifying constraints used by DVMT knowledge sources. As with approximate search, the use of approximate knowledge naturally falls into two categories, approximate synthesis and approximate track extension. Synthesis knowledge sources use proximity constraints to form groups of mutually consistent hypotheses that can potentially be used to support the synthesis of the hypothesis that triggered the knowledge source instantiation. The knowledge source then applies constraints and belief combination functions specified in the grammar to each group of potentially supporting hypotheses. Simplifying belief combination functions or ignoring constraints will save time and, as a result, reduce μ_c. Furthermore, ignoring constraints will also reduce σ_c^2. The cost of a synthesis operation is proportional to the number of groups of potentially supporting hypotheses and, in a noisy environment, a knowledge source cannot determine a priori how many of these groups it will form. The number can vary from one or two (in situations where the noise is not correlated to correct data), to several hundred (in



(a) modified Grammar A used by approximate synthesis knowledge source

(b) modified Grammar A used by level-hopping knowledge source

Figure 4: Example of Synthesis Knowledge Approximations

situations where the noise is highly correlated to correct data). Consequently, $\mu_{r'}$ can vary greatly and by ignoring proximity constraints, only one group of potentially supporting hypotheses will be formed and $\sigma^2_{r'}$ will be reduced.

In addition, synthesis knowledge sources can perform "level-hopping". Constraints that would normally span two levels of the blackboard, and would thus require the application of two synthesis knowledge sources, can be simplified and "compressed" to allow synthesis of events at a level of the blackboard two levels higher than the input level. Level-hopping gives a knowledge source the ability to alter which level of the blackboard it uses as input in order to choose the most effective method of achieving a goal. For example, given a goal to generate an hypothesis at the vehicle level of the blackboard, a knowledge source capable of level-hopping could choose to use input data from the group level of the blackboard or it could choose to use input data from the signal level of the blackboard if the necessary group level results had not yet been generated and if the situation required approximate processing.

Figure 4 shows the effects of knowledge approximation on the grammar shown in Fig. 2, part (a). In (a), all AND-OR-XOR nodes have been removed and the corresponding belief combination functions have been replaced with a simplified version, f. In (b), the grammar has been simplified further by removing the group level of abstraction.

In general, ignoring proximity and linguistic constraints or simplifying belief combination functions does not reduce the precision of a synthesized result. This is because the characteristics of the synthesized result are taken from the hypothesis that triggered the knowledge source, and the use of approximate knowledge does not alter these characteristics. However, the use of approximate knowledge does result in the generation of inconsistent results that would *not* have been generated by a knowledge source using a complete set of constraints. Thus, the benefits of using approximate synthesis must be weighed against the risk of generating so many inconsistent intermediate results that the net effect on subsequent problem solving is an increase in $\mu_{r'}$ and/or $\sigma^2_{r'}$.

A track extension knowledge source finds all the hypotheses that can potentially be used to extend a given hypothesis, and then applies velocity and acceleration constraints to determine which extensions are consistent. For track extension knowledge sources, $\mu_{r'}$ is a function of the number of potential extensions and, to a lesser extent, the number of resulting consistent extensions. Since a knowledge source cannot determine a priori how many consistent extensions it will form, $\mu_{r'}$ can vary widely. Thus, simplifying or ignoring the constraints can reduce $\mu_{r'}$ and $\sigma^2_{r'}$. This can, however, result in the generation of inconsistent tracks that would not have been generated with the use of full constraints. Since the problem solver will not be able to differentiate between consistent and inconsistent tracks without additional processing, the certainty of all tracks generated with approximate processing will be reduced.

As with approximate synthesis, an approximate track extension knowledge source does not reduce the precision of the extended hypotheses. This is because the velocity and acceleration

constraints are not used to generate the characteristics of extension candidates, but to eliminate candidates that are inconsistent. Consequently, the benefits of using approximate track extension must be weighed against the risk of generating so many inconsistent intermediate results that the net effect on subsequent problem solving is an increase in μ_r and/or σ_r^2.

The choice of appropriate approximations is situation dependent, warranting a control component integrated with the approximate processing mechanisms. A number of different knowledge, data, and control approximations must be coordinated with the computational demands induced by the current situation. In the DVMT, for example, when there is a large amount of initial signal data that does not show much spatial separation, the control component could use a clustering data approximation strategy to quickly arrive at a set of possible solutions in spite of the large amount of noise that is apparently present. To do this, it uses data approximation to cluster the initial signal data, and then processes this clustered data with approximate knowledge sources that synthesize this data into higher blackboard-level clustered hypotheses. This strategy results in a quick approximation of a vehicle track and a set of possible vehicle classes while giving up a precise accounting of the vehicle location and a precise identification of its class.

4 Extending a Blackboard Architecture for Approximate Processing

In order to extend the DVMT, significant changes were made to the problem-solving architecture. The resulting architecture allows a range of approximate problem-solving strategies to be efficiently integrated. The implementation of these extensions required that the following questions be resolved.

1. How should intermediate results of problem solving be represented so that precise and approximate intermediate results can be combined in further processing?
2. How should a knowledge source be structured so that it can exploit intermediate results of varying levels of approximation?
3. How should approximate knowledge sources of different types be organized and controlled?
4. How should uncertainty caused by the use of approximate search, data, and knowledge be represented?
5. How should the control architecture be structured so that it can smoothly integrate different types of approximate processing strategies?
6. How should the control architecture be implemented to minimize overhead when the current approximate processing strategy uses only a subset of the partial results and a subset of the applicable knowledge?

4.1 Data Representation

Data approximations are created by expanding the representation of precise data along one or more dimensions. The semantics associated with approximate data should be consistent with those associated with precise data so that meaningful combinations of the two can be constructed. If approximate data has a significantly different semantic interpretation than precise data, it might not be possible to incorporate both into a single partial result and this might severely limit any advantages offered by approximate processing.

For example, the DVMT's single valued location attribute can be expanded to cover a range of locations. Several acoustic signals might be combined, or *clustered*, into a single partial result that encompasses not only the area of the sensed data, but areas where no data was sensed as well. The new data cluster has several interpretations. It can take on an *existential* interpretation, such as "there is some support for a signal source *somewhere* in this area", or it can take on a *universal* interpretation, such as "there is some support for a signal source at *every* point in this

area." If some knowledge source activity involves checking for physical overlap among data, then the existential and universal interpretations will produce distinct results. In particular, determining whether or not an existentially interpreted datum and a universally interpreted datum overlap might be prohibitively expensive or impossible. On the other hand, if precise and approximate data are readily interchangeable, then a system has tremendous flexibility in deciding when to produce approximate data, how to incorporate it into the solution, and how much of it to use.

In order to combine both precise and approximate intermediate results, the DVMT was modified to represent *all* partial results as clusters of data with characteristics defined by ranges of values. Specifically, each partial result is now represented by a set of event classes, E, and a location range, R. In addition, partial result attributes were extended with the addition of a domain specific *precision*, or "level of approximation", statistic. A partial result's precision is a function of the size of R, the size of E, and the variance of the partial result's likely location within R. This extension effectively expands the dimensionality of the blackboard along an axis corresponding to precision.

Using this representation, an existential interpretation of data has been adopted. An *exact* hypothesis is represented with a range attribute specified by a single point, an event class set with cardinality one, and a precision of zero[4]. A *cluster* hypothesis has a range, R, defined as a convex region encompassing all the component locations, an event class $E = \cup$(component event classes), and a precision = f_p(size of R, $\mid E \mid$, f_v), where f_v is the variance of the clustered hypotheses' locations within R.

Figure 5 shows several examples of approximated data. Part (a) is an exact location hypothesis with precision zero: the size of the range spanned is zero and the vehicle's variance within that range is also zero. The location cluster shown in (b) is less precise than the exact location hypothesis because nine distinct vehicle hypotheses have been aggregated into one. The result is a cluster spanning a larger range and having a non-zero variance within that range. Even though they have similar sized ranges, (b) is more precise than the location cluster shown in (c) because the variance of vehicle locations within (c) is much greater than in (b). (d) highlights the adverse impact wide distributions of data can have on a cluster's precision. Despite having a slightly smaller range,(d) is less precise than either (b) or (c) because of the large variance of vehicle locations it encompasses. (e) shows the adverse effects of clustering multiple event classes. (e) is similar in size and variance to (c) but it is much less precise because it encompasses multiple event classes. (f) and (g) show how quickly precision is reduced when widely distributed data with multiple event classes is clustered.

4.2 Knowledge Organization

The introduction of approximate knowledge sources leads to a range of options for organizing a problem solver's knowledge. At one end of the spectrum, there is a specific knowledge source for each form and level of approximation. For example, in the DVMT, there could be a specific synthesis knowledge source for eliminating corroborating support of $necessity \leq 0.1$, a specific track extension knowledge source for extending data with precision ≤ 5, and so forth. At the opposite end, there is a single, general knowledge source corresponding to each of the original knowledge sources that encompasses the full range of exact and approximate processing strategies.

The approach taken in modifying the DVMT was at the second end of the spectrum. In order to organize and control the the different types of approximate knowledge sources and to enable a knowledge source to exploit intermediate results of varying levels of approximation, the original knowledge sources were parameterized and restructured so that they are now capable of working with any level of approximate data. Belief combination functions and other domain problem-solving functions were modified to take into account the new interpretation of data described in section 4.1. For example, synthesis knowledge sources now use the precision statistic to reason about the probability that location hypotheses overlap. Similarly, extension knowledge sources use the precision statistic to reason about the probability of a hypothesis satisfying velocity and acceleration constraints. With these modifications, DVMT knowledge sources process exact and approximate

[4]Because the precision measure is unbound, the DVMT uses an inverted precision scale. Thus, exact data has precision zero, and greater values of precision indicate less precise data.

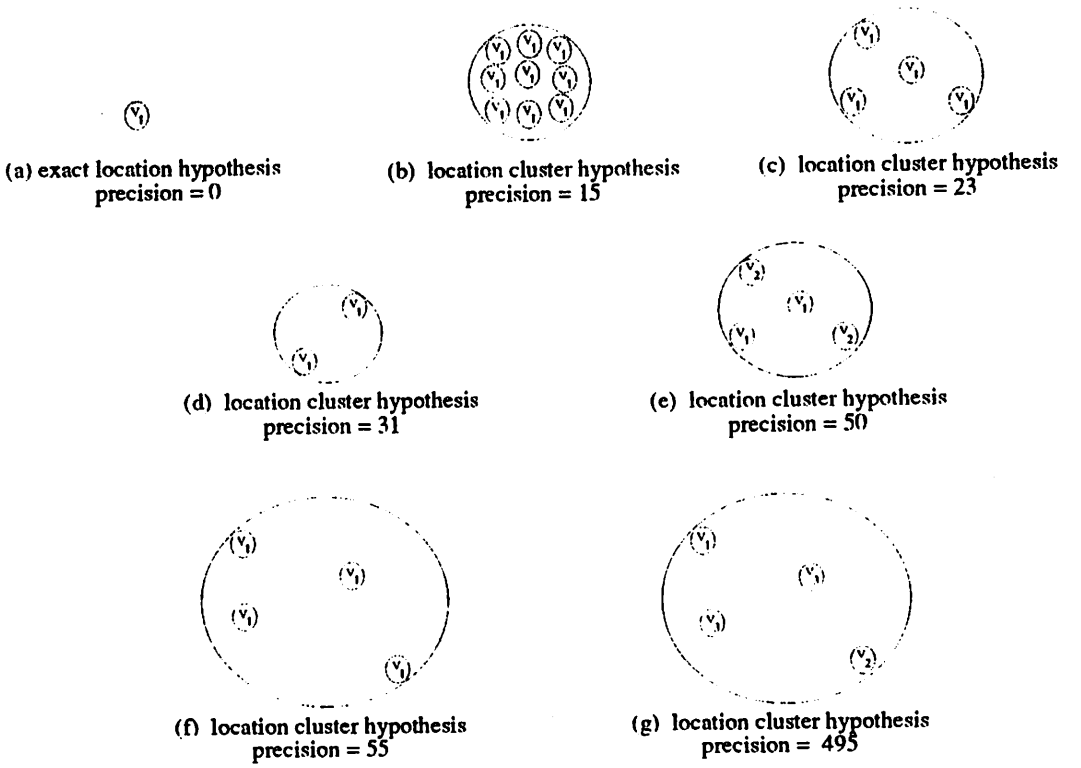


Figure 5: Examples of Precision and Approximate Data

data identically and problem solvers can use exact and approximate processing interchangeably.

In addition, a *clustering* knowledge source for aggregating data was added. The precision of the data approximations produced by the clustering knowledge source is controlled by two parameters, C, the number of clusters to form, and I, the *information loss threshold*. To combine individual partial results, the clustering mechanism creates a new hypothesis with characteristics subsuming all of the clustered hypotheses. The clustering mechanism cycles through a set of input data forming clusters until a cluster is generated with a precision $\geq I$ or until the input data has been combined into C or fewer clusters, at which point it halts and outputs the generated clusters.

In order to control the different types of approximation, an *approximation control block* (*ACB*) was added to each knowledge source instantiation. After determining that it needs to use approximate processing to meet some objective, a specific approximation strategy is implemented by adjusting the *ACBs* of the appropriate knowledge source instantiations. When it is invoked, a knowledge source internally chooses from its repertoire of approximations the best way to satisfy the specification defined in the control block. As will be shown later, this method is extremely flexible and allows for a wide range of approximation strategies. The approximation control block contains the following slots.

Input Rating Threshold (IRT): The IRT limits the number of partial results used as input by a knowledge source. A knowledge source ignores all input data with a rating $< IRT$. This approximation is based on a non-global evaluation of the input data and is therefore ill-defined. In situations where a problem solver predicts that it cannot form an acceptable answer using well-defined approximation techniques, it can risk using ill-defined approximations that may lead to incorrect solutions, or no solutions at all.

Output Rating Threshold (ORT): The ORT limits the number of partial results generated as output by a knowledge source. A knowledge source discards all partial results it forms with rating $< ORT$. This approximation is based on a non-global evaluation of the partial results formed by a knowledge source and is therefore ill-defined.

Input Precision Filter (IPF): The IPF indicates the level of approximate data the knowledge source retrieves from the blackboard. A knowledge source will not consider potential input data with precision $> IPF$.

Work Level Precision Filter (WPF): The WPF indicates the level of data approximation the knowledge source actually processes. It has the form (N, C, I), where N = maximum number of partial results to use as input, C = number of clusters to form, and I = the *information loss threshold*. A knowledge source uses these values to cluster input data to the desired level of approximation. By manipulating the WPF, a problem solver can specify the quantity of the highest rated input hypotheses to process, the number of clusters to form, and the amount of precision it is willing to sacrifice.

Output Level Precision Filter (OPF): The OPF determines the level of data approximation of the knowledge source outputs and has the form (N, C, I), where N, C, and I have the previously defined interpretations.

Search Approximation Level (SAL): The SAL indicates the knowledge source's level of approximate search. The SAL has the form (SL), where SL specifies the approximate search strategy to use. Specifically, for synthesis, $SL = n$ indicates that the knowledge source should assume the existence of any subtree of the grammar with $necessity < n$, and for extension, SL indicates the number of time frames it should skip when extending a track.

Knowledge Approximation Level (KAL): The KAL indicates the level of approximate knowledge the knowledge source should use and has the form (KL), where KL specifies which knowledge approximations to use. Different values of KL indicate whether a synthesis knowledge source should ignore constraints or perform level-hopping. Similarly, KL specifies whether or not an extension knowledge source should ignore velocity or acceleration constraints.

By manipulating combinations of parameters in the approximation control block, a problem solver has great flexibility in its use of approximate processing. It can adopt well-defined strategies that are either local or global in extent. For example, it can tailor the precision of a specific area of the solution by setting the appropriate approximation control block parameters in a few specific knowledge source instantiations. Alternatively, it can implement a strategy where all knowledge sources work at a given level of precision, P, by first grouping all data to level P with the clustering knowledge source, then setting every knowledge source's IPF to P. Furthermore, when the situation warrants, a problem solver can also use ill-defined approximation strategies. This can be done with the IRT, the ORT and the N component of the precision filters. Finally, a problem solver can combine thresholding filters, such as the IRT and the ORT, with clustering to moderate the effects of ill-defined approximations.

4.3 Belief Representation and Uncertainty

Uncertainty arises in any system from the *reliability* of the initial data, the *imprecision* of that data or the language used to represent it, the *incompleteness* of the data, and the *aggregation* of the data from data approximations or multiple sources[9]. Additional uncertainty is introduced with the use of approximate search, data and knowledge. For example, the credibility of a partial result constructed by ignoring potentially corroborating data must be distinguishable from the credibility of a similar partial result constructed using all available data. In the first case, further processing could raise the credibility of the partial result, but in the second case, no amount of processing will increase the credibility of the partial result because all corroborating data has already been considered.

To represent uncertainty caused by the use of approximate search, data, and knowledge, the DVMT has expanded its representation of belief to a four-valued system. The new belief system was derived from evidential reasoning [10] and is similar to that in RUM [7]. The belief in a hypothesis is now represented by a measure of positive belief (certainty) and of negative belief (refutation). To represent the completeness of the solution, the positive and negative beliefs are further divided

into upper and lower bounds. Belief in a hypotheses therefore is summarized with four values, *(certainty, plausibility, refutation, doubt)*, where:

certainty - The lower bound of belief in a partial result. This is a measure of the amount of irrefutable evidence supporting the partial result.

plausibility - The upper bound of belief in a partial result. This is a measure of the degree to which available evidence does not refute the partial result. No amount of consistent processing will raise a partial result's certainty above its plausibility.

refutation - The lower bound of disbelief associated with a partial result. This value is analogous to certainty.

doubt - The upper bound of disbelief associated with a partial result. Similar to plausibility, doubt reflects the degree to which available evidence does not support a partial result. No amount of consistent processing will raise a partial result's refutation above its doubt.

The need for four values can be demonstrated with a simple example. Approximate search is implemented by allowing a knowledge source to make an *assumption* that some relevant supporting data exists. After this assumption is made, the system needs to represent *certainty* so it can determine if an approximate solution meets the satisficing criteria. However, since some supporting data is not used, *certainty* will be less than it would have been using exact processing. Therefore, *plausibility* needs to be represented so that the problem solver does not prematurely eliminate from consideration a result that could be improved with additional processing. Furthermore, since problem solving control is based partially on the credibility associated with partial results, it is necessary to temper overly optimistic assumptions by representing the associated *doubt* that the assumption might be false. Finally, it is necessary to represent *refutation* in order to differentiate assumptions that are found to be false from assumptions that are merely unconfirmed. Figure 6 introduces a graphic representation of the four-valued belief system that will be used in subsequent sections to present experimental results.

Furthermore, second-order relationships between the elements of the belief system can also be used to control problem solving. Specifically, various measures of *ignorance*[10] and conflict can be computed. Ignorance measures, such as $(plausibility - certainty)$, $(doubt - refutation)$, $(1 - plausibility)$, $(1 - doubt)$ and $(1 - (certainty + refutation))$, indicate the amount of useful work that can be done to refine the belief in a hypothesis. Conflict measures, such as $(1 - (certainty + refutation))$, indicate the consistency of the processing that generated a hypothesis.

4.4 Control

In order to effectively exploit the range of approximate processing strategies that are made possible through approximate knowledge sources, the basic blackboard control loop was modified. These modifications provide the flexibility to dynamically reconfigure the system for different control regimes and to exploit approximate data and knowledge for real-time processing, as well as enable the reduction of low-level control processing overhead. Although these mechanisms have not yet been used as a part of the real-time model of evaluating, predicting, and monitoring control, they have been used to react dynamically to the current problem solving situation.

The mechanisms can be divided into three classes. *Filters* limit the amount of data being considered to reduce overhead or distraction. *Mappings* control the general character of problem solving. *Mergings* control the granularity or specificity of problem solving activity. The new DVMT low-level control loop can be characterized as evaluating the blackboard to decide (see Figure 7):

1. *What information to exclude from any further processing (hypothesis filtering).*
2. *What potential work can be done (hypothesis-to-goal mapping).* The goals that result from this mapping are called *data-directed* goals.

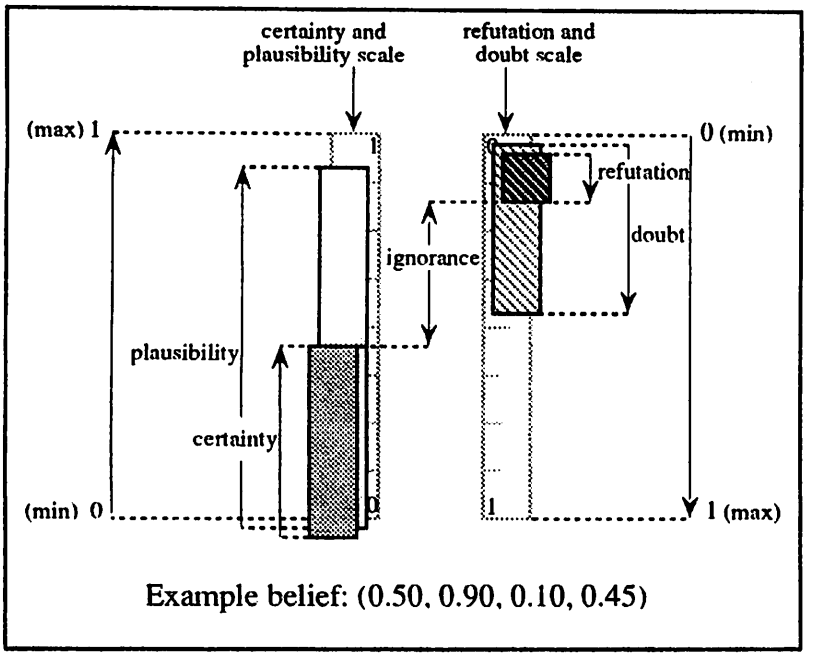


Figure 6: Graphic Belief Representation Key

3. *Relating potential work to existing goals (goal merging and subgoaling).* Three types of goals are merged: data-directed goals from the hyp-to-goal mapping, *received* goals from communication with other nodes, and *goal-directed* goals from subgoaling.

4. *Determining what goals are important to achieve (goal filtering)*

5. *Deciding how to go about achieving them (goal-to-KS mapping)* This produces a set of triggered Kss that may accomplish a given goal. The preconditions of the KSs are run, which results in a set of costs (such as estimated time) and benefits (such as an estimated output set) for each triggered KS. One KS is chosen based on this data and its instantiation is merged into the runnable KSI queue.

6. *Choosing which one of these potential activities to execute (managing the agenda).* This includes both rating the KSIs and merging KSIs.

The above steps ensure that all of the KSIs that appear on the KSI agenda are potentially *useful*[5] to run (as opposed to merely *runnable*). Allowing parameterized control over all of these phases makes high-level control strategies clearer and easier to implement by reducing the amount and type of domain data and knowledge that must be considered at each point. Each of these steps is parameterized so that the system can react flexibly to changes in the current situation, which is necessary for our model of real-time problem solving. In this initial implementation, the low-level loop is controlled by a BB1-style meta-control mechanism[11], modified so that all aspects of the low-level control loop can be controlled — what activities are placed on the agenda, why they get there, and the amount of effort involved in making these decisions. The ability to dynamically modify the low-level control loop is an extension of ideas developed originally in BB1 for dynamically specifying the predicates used to evaluate activities on the agenda in order to impose different high-level strategies. In more recent work, B. Hayes-Roth has also proposed extensions similar in character to some developed here for controlling other aspects of agenda maintenance[12].

Here is a summary of a few of the major modifiable control parameters that were used in the experiments detailed in Section 5:

[5]Applicable to the current system goals and strategies for achieving those goals.

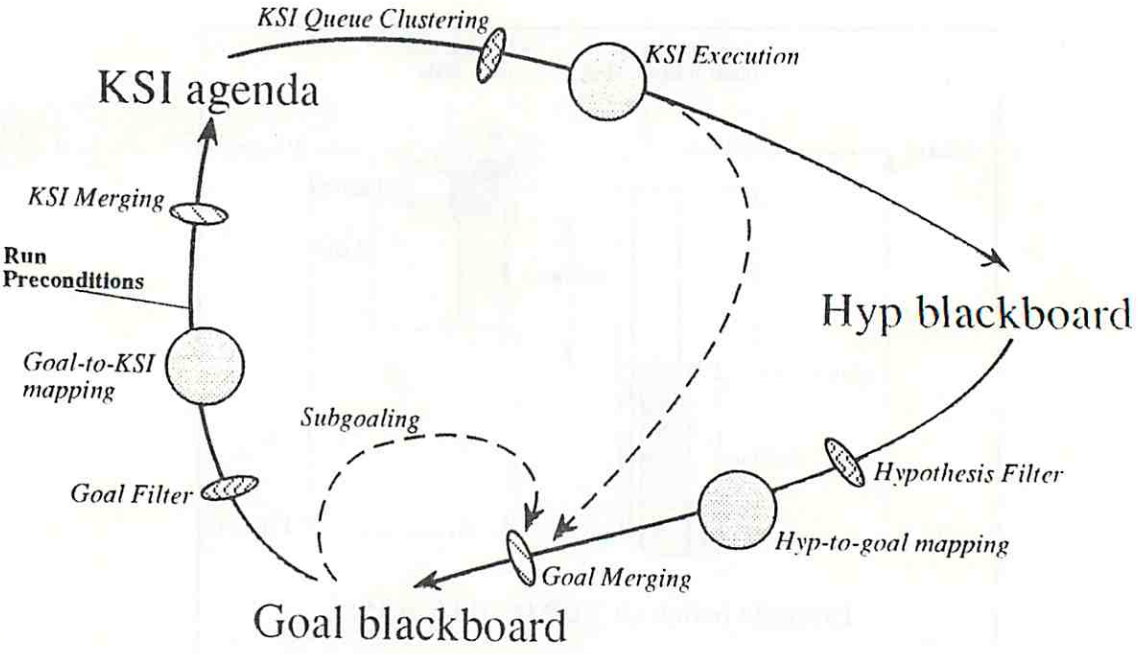


Figure 7: The New Low-level DVMT Control Loop

Hypothesis Filter: The hypothesis filter takes as input the hypotheses created by a KSI execution and outputs hypotheses to be processed by the hyp-to-goal mapping. Hypotheses can be filtered by time, region, event class, blackboard level, or belief. Filtering a hypothesis means that it will not create data-directed goals, and therefore will not stimulate any KSs. For example, if a DVMT node decides it will only track objects of class $\mathcal{A}$ the hypothesis filter can be set so that only hypotheses that have class $\mathcal{A}$ event classes are processed by the hyp-to-goal mapping. Hypothesis filtering allows the reduction of low-level control processing overhead since the filtered hypotheses do not create goals, KSIs, or cause any of the processing associated with goals and KSIs. In general, the hypothesis filter is used to prune the raw data by class, time, or location. Filtered hypotheses (those blocked by the filter) are stored and can be re-filtered if necessary. All hypotheses, filtered or not, are stored on the hypothesis blackboard and are available for supporting KSI executions or for examination by a planner.

Hyp-to-Goal Mapping: The hyp-to-goal mapping takes as input a hypothesis that passed the hypothesis filter and outputs goals to be merged into the existing goals on the goal blackboard. The mapping is specified by a set of *goal templates* that specify a pattern that a hypothesis must match and a transformation of that hypothesis into a goal. The hyp-to-goal mapping allows the control of what types of problem solving tasks the system should be concerned with by controlling what goals are created, and therefore what KSs are potentially triggered. For example, if only hypotheses at time n need to be clustered, then only hypotheses at the signal level at time n will generate a cluster goal (all hypotheses at the same time will then be merged into a single cluster goal for that time). Hyp-to-goal mapping is used to control the rough character of problem solving (types of methods applied to the data) by the creation of the proper data-directed goals.

Goal Filter: The goal filter takes newly created or updated (merged) goals of any type and outputs goals that will be used to trigger KSs. Goals can be filtered by time, region, event class, blackboard level, belief, or type. The goal filter reduces the number of goals that might trigger KSs, and therefore the number of KS preconditions that must be run. For example, if the control strategy is to produce vehicle level data before creating vehicle tracks, vehicle track

goals that are created (by the hyp-to-goal mapping, subgoaling, a planner, or received from other nodes) can be filtered so that none of them trigger (stimulate) track creation/extension KSs (and do not subsequently run KS preconditions or create KSIs). The goal filter is used to avoid triggering and running the preconditions for a class of KSs that is not currently desired but will be in the future (otherwise the goals would not have been created in the first place).

Goal-to-KS Mapping: The goal-to-KS mapping takes goals and produces a set of KSs that may satisfy those goals. It is specified as a table that matches goal types to KSs that can satisfy those types of goals. The preconditions of these triggered (or stimulated) KSs are then run, and the best KS is chosen based on the costs and benefits that the precondition returns (see the KSI precondition choice parameter below). This KS is then instantiated with its context, including the stimulating goal. The goal-to-KS mapping is used to fine-tune the precise method or algorithm used to satisfy a goal. In the DVMT, the goal-to-KS mapping is rather coarse, because KSs tend to be written in a general manner rather than tailored for a very specific situation. For example, one may satisfy a vehicle level goal by either the synthesis KS from the group level to the vehicle level, or a level-hopping KS that synthesizes data directly from the signal level to the vehicle level (skipping any intermediate processing). The goal-to-KS mapping tends to be simple in the DVMT because we have only a few classes of KSs.

Additional parameters involve the control of goal processing and agenda management. These parameters determine the character of the search process in terms of the granularity of the operations and the overhead involved in choosing the most appropriate KSI to execute on the agenda. The complete details of these parameters can be found in [13].

After changing a parameter, the meta-controller may re-run the low-level control loop from any point — usually from just before the point that was changed. The meta-controller has the option to reintroduce filtered data at this time as well, from either the hypothesis or goal filter or both. Various schemes have been discussed for the storage of blocked hypotheses and goals that would make the re-filtering very efficient, but none have been implemented (blocked data is simply re-run through the filter). For example, blocked objects can be divided into classes, i.e., objects to be saved and objects to be permanently removed from consideration. Blocked objects can be stored according to how they were blocked, so that when a filter changes the data that needs to be refiltered (or that passes the new filter) can be retrieved efficiently.

4.4.1 Meta-level control

A BB1-style meta-controller with appropriate control knowledge sources and a control blackboard was used to dynamically set the parameters in the low-level control loop. Control knowledge source preconditions examined the current state of the low-level control loop, usually the contents of the agenda and the blackboards, for events of interest (such as an empty KSI agenda), or for the performance of the low-level loop (such as the average number of goals being created). BB1 prescription KSs [11] are used as control plans that indicated when to change focus, and strategy and focus goals also examined the DVMT agenda and blackboards. Heuristic control KSs are free to modify any of the low-level control loop parameters, not just the agenda rating mechanism.

The meta-controller currently runs synchronously with the low-level control loop, so that it could, for example, examine the KSI queue after a KSI is chosen but before it is run and execute a different KSI instead [14]. It is postulated (and currently being implemented) that the low-level control mechanism can run asynchronously with respect to the meta-controller.

Examples The real-time DVMT meta-controller was developed to control experiments in "soft" real-time approximate processing — using approximate knowledge and data effectively. Eventually, our full real-time model will use time predictions and deadlines to choose appropriate strategies; in this initial implementation, strategies are chosen based on the characteristics of the current situation (but not on how much time they will take). Two BB1-style strategies were developed: a goal-directed strategy and a data-directed clustering strategy. A simple control plan was used

to move between foci when the goal of a focus was satisfied. A pictorial representation of the goal-directed strategy in shown in Figure 8. A brief description of the goal directed strategy is given below.

System initialization sets all filters to open, the hyp-to-goal mapping to normal (signal level to group level to vehicle level to vehicle tracks to pattern tracks), the goal-to-KS mapping to normal (the regular, non-approximate KSs), subgoaling off, and KSI rating tied to the rating of the triggering goals and hypotheses.

Goal-directed Strategy This strategy is invoked when its precondition determines that there are multiple vehicles and that there is good sensor data. This is determined by testing that the initial data at time 1 is spatially separated. This strategy tries to determine what might be out there, whether it is important to track, and then tracks only what it finds to be important. It has three foci: *find initial vehicles*, *approximate short tracks*, and *pattern directed processing*.

Find Initial Vehicles: This focus concentrates on the careful (non-approximate) data-directed analysis of initial data. It consists of two heuristics. This focus is over when the KSI agenda is empty.

- **Consider only time 1 hypotheses:** This heuristic sets the hypothesis filter to allow only hypotheses from time 1 through and to block all others. Only time 1 hypotheses, then, pass through to the hyp-to-goal mapping.
- **Create no tracks nor patterns:** This heuristic sets the goal filter to block any track (ST, GT, VT, PT) and pattern (PL, PT) level hypotheses. Thus the system will only work up to the vehicle level.

Approximate Short Tracks: This focus concentrates on quickly building up an idea of what possible vehicle patterns are present in the system. It does this so that our process resources can be concentrated on what are viewed as important vehicle patterns. This focus is completed when all pattern track hypotheses have reached a minimum length at which a decision about their importance can be made.

- **Consider early signals:** This heuristic sets the hypothesis filter to only pass hypotheses from time 1 through 5 (**minimum-pattern-length**). Hypotheses previously blocked are refiltered. Thus only early hypotheses are considered for further processing.
- **Use level-hopping:** This heuristic changes several parameters to set the system up for level-hopping. Level-hopping is used to approximate vehicle level data directly from signal level data by compression of the signal/group/vehicle grammar. The hyp-to-goal mapping is set to create vehicle level goals from signal level data, subgoaling is turned off, and the goal filter is set to block signal and group goals (since the creation of vehicle level hypotheses is desired).

Pattern Directed Processing: This focus concentrates on developing 'important' tracks at the expense of paying less attention to 'unimportant' tracks. The importance of a vehicle pattern is resolved by a pattern track KS that ran at the end of the approximate short tracks focus. The pattern directed processing focus remains active until the end of problem solving. In pattern directed processing, rather than working in a data-directed manner, we configure the system to work in a goal-directed manner.

- **Work on important patterns:** This heuristic sets up several parameters to process important vehicle patterns. The hypothesis filter is set to only allow through vehicle level hypotheses from the "important" patterns, because we will work below the vehicle level only in a goal-directed manner. Subgoaling is invoked on the important patterns to create goals for extending the important pattern and building it up from the signal level (this creates a partially ordered plan for processing the important tracks). The KSI queue cluster width is set to only execute 1 KSI at a time, so that there is precise control over each invocation.

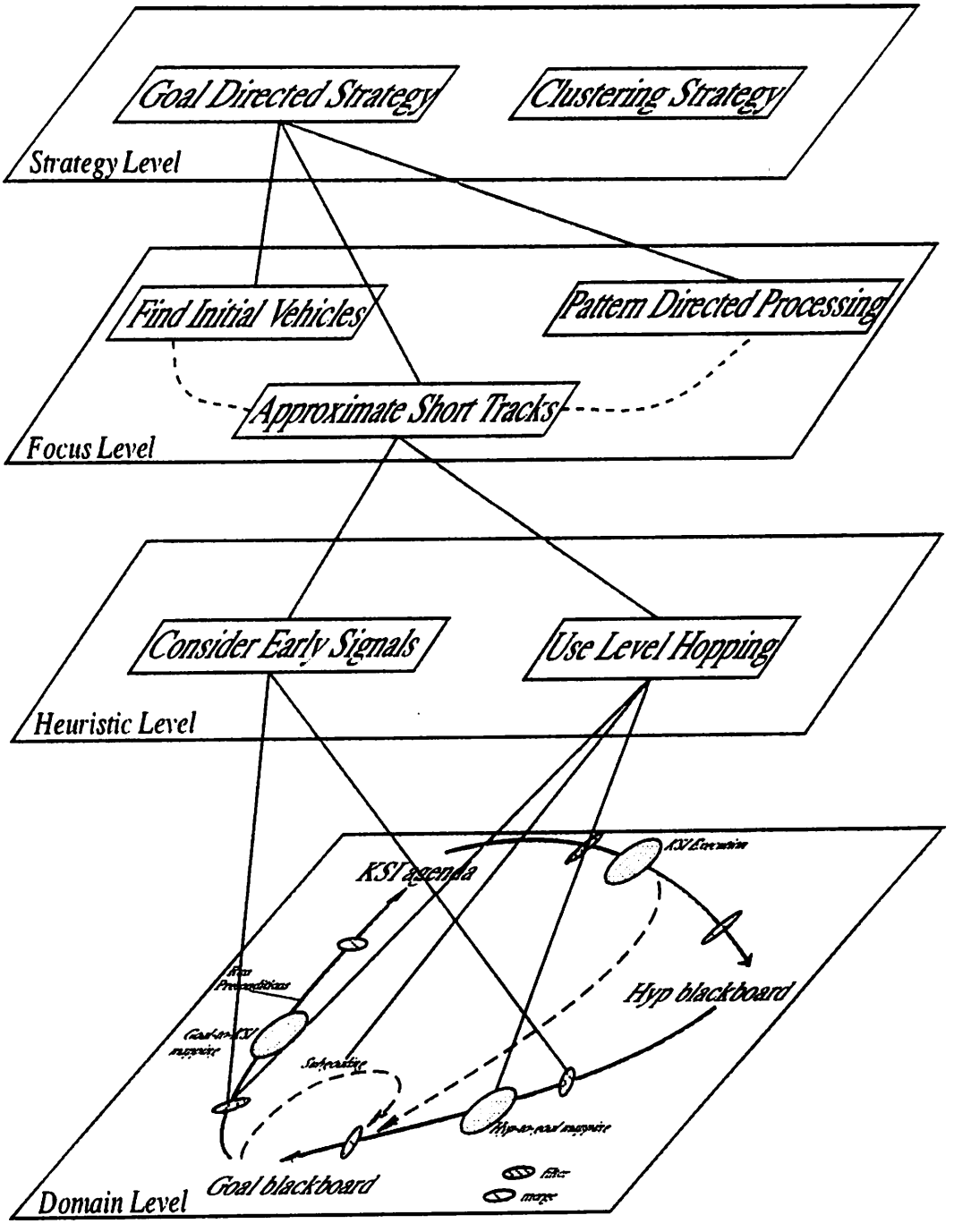


Figure 8: The DVMT Goal-directed Strategy

Approximate unimportant patterns: This heuristic deals with the patterns that are not deemed to be important. The hypothesis filter is set to allow only signal and vehicle level hypotheses through from unimportant vehicle patterns, and the hyp-to-goal mapping is set to create level-hopping goals for approximating these unimportant patterns (in the future, we may use time-frame-skipping, which tracks an object intermittently, instead). The KSI rating is reduced on level-hopping KSs, so the system prefers to work on the important patterns instead. In this manner cheap approximation techniques are used on patterns that are considered less critical.

A cluster-directed processing strategy was also developed that is invoked when its precondition determines that there is highly errorful sensor data: this can be determined by testing if the data at time 1 is not spatially well-separated. This strategy assumes that there is too much data to process individually, and so it clusters data at each time and then processes the clusters in a data-directed manner. For a complete description of both strategies see [13].

5 Experiments

A number of experiments in the Distributed Vehicle Monitoring Testbed (DVMT) have been carried out demonstrating how the extensions proposed to the blackboard model of problem solving permit efficient implementation and integration of a variety of approximate processing strategies. The first five experiments illustrate the effects on solution precision, completeness and certainty resulting from the use of approximate knowledge, approximate search and data approximation. The sixth experiment shows how alternative high-level control strategies that dynamically alter the current set of approximations can be chosen on a situation-dependent basis and also demonstrates the computational benefits of the parameterized low-level control loop.

5.1 The Effect of Approximate Processing on DVMT Solutions

All of the approximation techniques discussed in section 3.2 and the modifications presented in section 4 were implemented in the DVMT. This section illustrates the effects of approximate processing on the time required to generate a solution and the solution's quality.

A series of experiments were conducted, all based on the scenario shown in Fig. 9. Each data point labelled t_i in Fig. 9 represents a noisy signal data corresponding to a vehicle and the X and Y axes represent the signal data's spatial coordinates. In this scenario, a vehicle is moving in a straight line through a noisy environment, from somewhere around location (1,1) at t_1 to somewhere around (16,16) at t_8. The results of experiment 1, the control experiment, are presented in Fig. 10. The result shown is the most consistent interpretation for the movement of the vehicle generated using exhaustive processing and can be used for comparison [6].

The solution generated in experiment 2, shown in Fig. 11, is the result of exhaustive processing that uses approximate data. In the DVMT, this solution is represented as a consecutive sequence of location clusters with relative region sizes corresponding to the shaded boxes shown in the figure. The dark line represents a track formed by connecting the centers of the clusters and can be used to make relative precision comparisons between experiments. To generate this solution, densely packed noise was clustered into less precise hypotheses, then used by exact knowledge sources to generate a less precise and less certain final solution. By aggregating the data, a great deal of time was saved by not exploring every possible combination of partial results. Compared to experiment 1, the use of approximate data required 824 fewer knowledge source executions, a savings of 84%. (See Table 1 for a summary of timing data.) In addition, less overhead was needed to process the reduced number of goals and partial results.

In experiment 3, approximate synthesis knowledge sources used approximate data to produce the solution shown in Fig. 12. As discussed in section 3.2, approximating synthesis knowledge does

[6] A solution in the DVMT is represented as a consecutive sequence of time-locations. Noise was omitted and the time-locations of the solution were connected with a line in Fig. 10 to clarify the presentation.

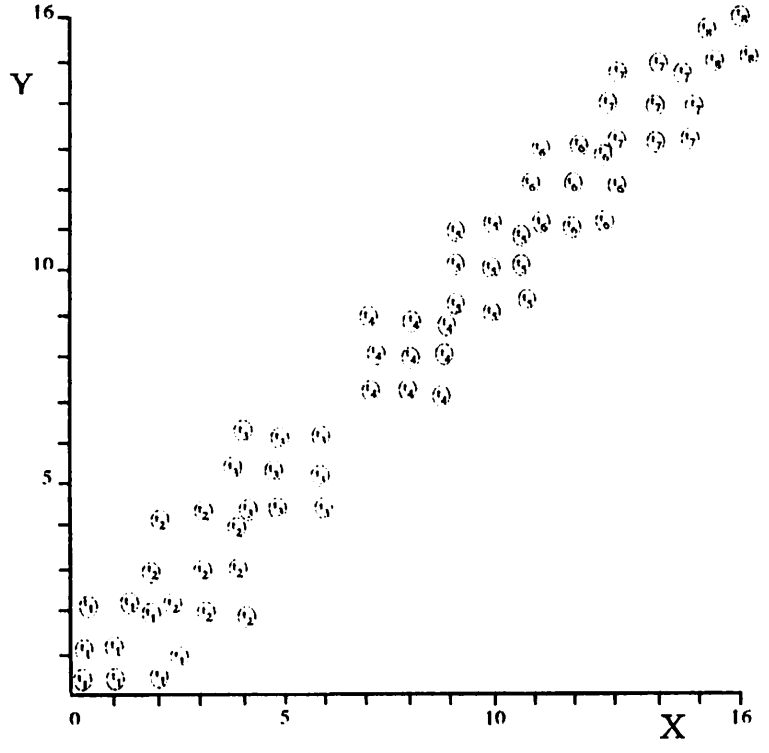


Figure 9: Data used for Experiments 1-5

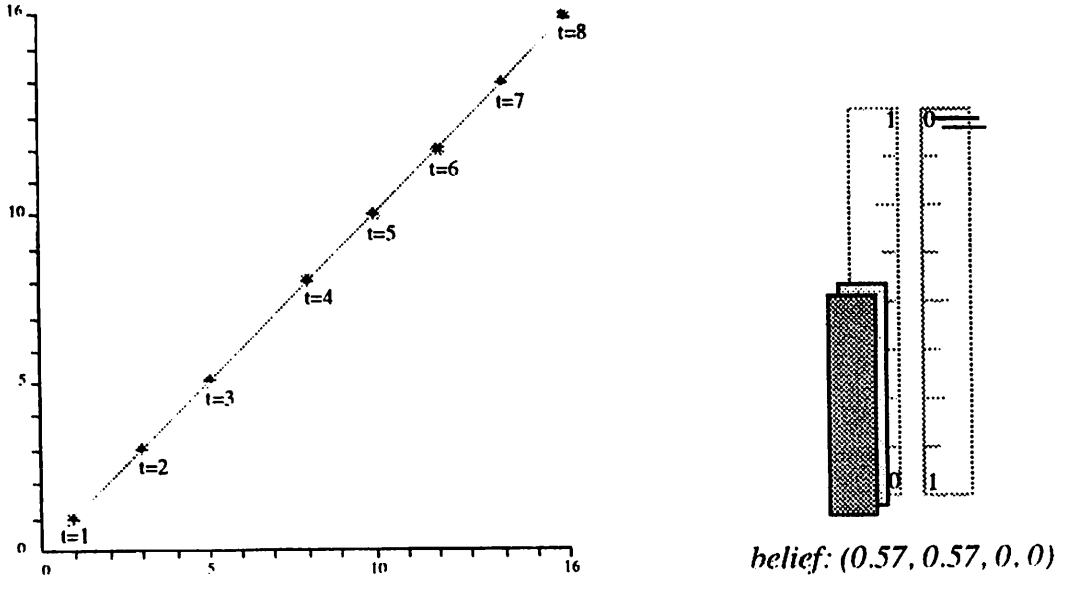


Figure 10: Experiment 1: precise data, full search, complete knowledge

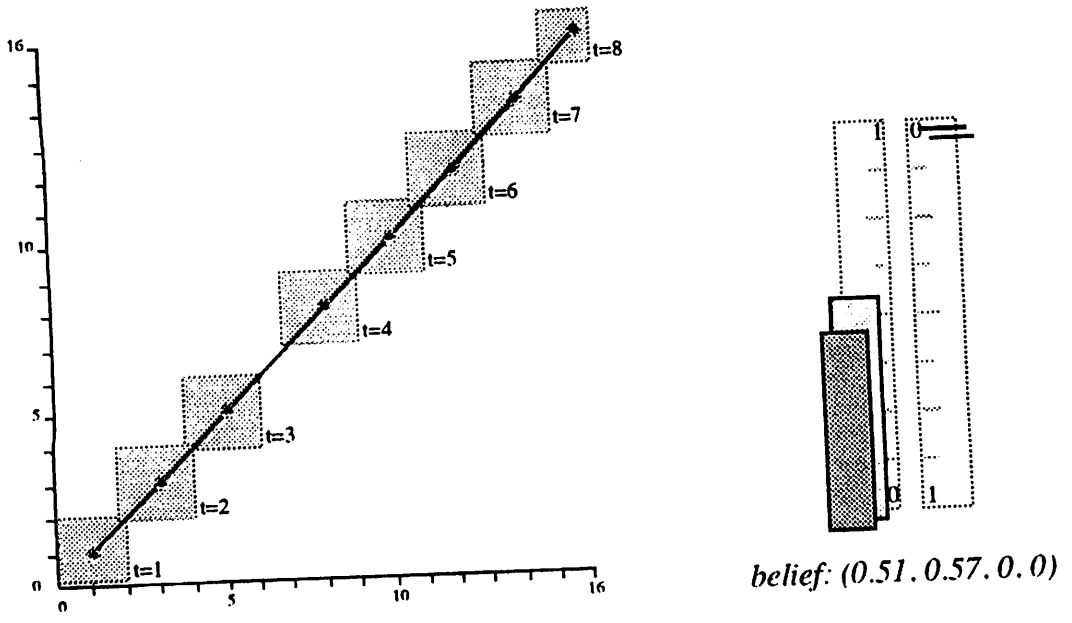


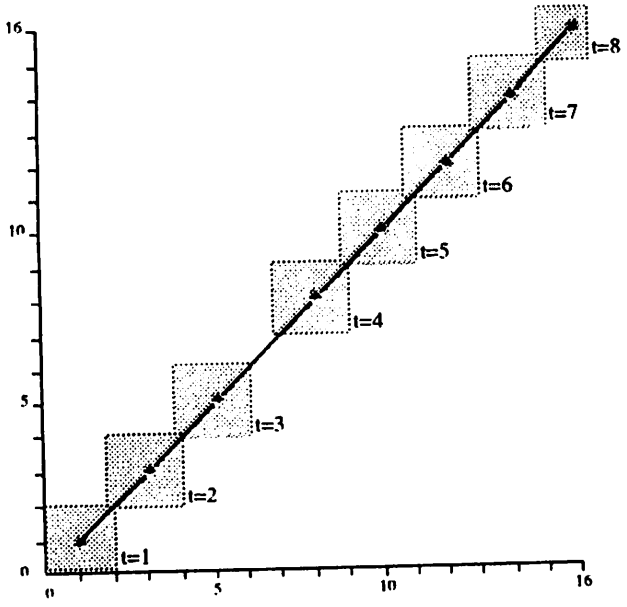
Figure 11: Experiment 2: approximate data, full search, complete knowledge

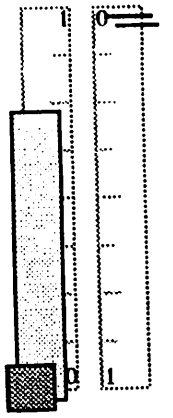
not reduce precision. Consequently, the solution precision is the same as in experiment 2. However, the certainty is reduced to a level that makes no assumptions about the presence of corroborating evidence and the plausibility is increased because fewer constraints have been applied generating the solution. Thus, additional processing could increase the certainty by developing corroborating evidence or it could decrease the plausibility due to failed constraint satisfaction. Compared to experiment 1, 896 fewer knowledge source invocations were required, a 91% savings.

Track extension knowledge sources used approximate search and approximate data in experiment 4, Fig. 13. Times 2,4 and 6 were not searched for corroborating data. Instead, assumptions were made that the required data was present with belief (0, 1, 0, 1). This, along with data clustering, reduced the solution's precision and certainty. The loss of precision is clearly seen when the dark line connecting the centers of clusters in this experiment is compared to the results of previous experiments. Doubt increased to reflect the fact that the assumptions could prove false and plausibility increased because fewer constraints were applied to the solution. Additional search of the areas where the assumptions were made would increase the certainty of the solution if corroborating evidence was found or it would increase the refutation of the solution if corroborating evidence was not found. Compared to experiment 1, 848 fewer knowledge source invocations were required, an 86% savings.

In experiment 5, Fig. 14, synthesis knowledge sources used the same knowledge approximations that were used in experiment 3, and track extension knowledge sources used the same search approximations that were used in experiment 4. The combination of all three approximation techniques reduced the solution's precision and certainty and increased the solution's plausibility and doubt. Again, the loss of precision is clearly seen when the dark line connecting the centers of clusters in this experiment is compared to the results of experiments 1, 2, and 3. Compared to experiment 1, 920 fewer knowledge source invocations were required, a 94% savings.

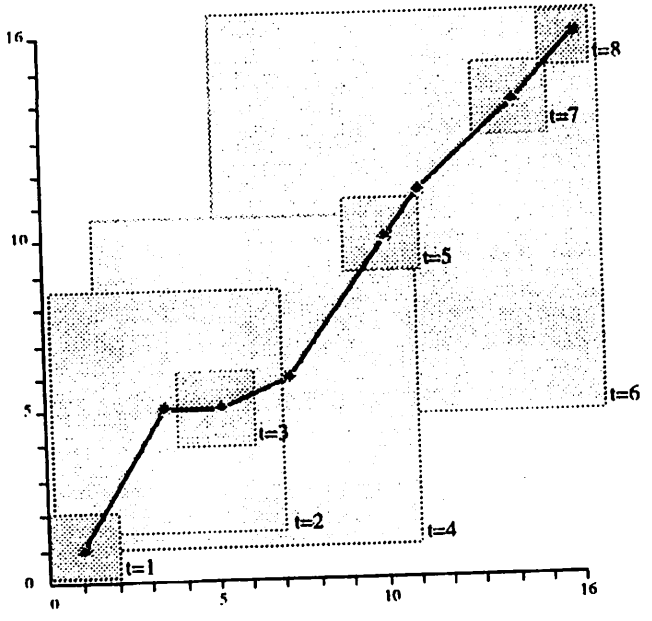
These experiments demonstrate that it is feasible to implement an approximate processing system capable of sacrificing solution quality for processing time. The next section demonstrates the flexibility and efficiency of the extended control architecture.

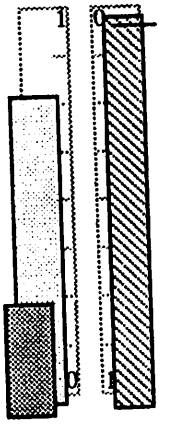




belief: (0.12, 0.75, 0, 0)

Figure 12: Experiment 3: approximate data, full search, approximate knowledge





belief: (0.24, 0.83, 0, 1.0)

Figure 13: Experiment 4: approximate data, approximate search, complete knowledge

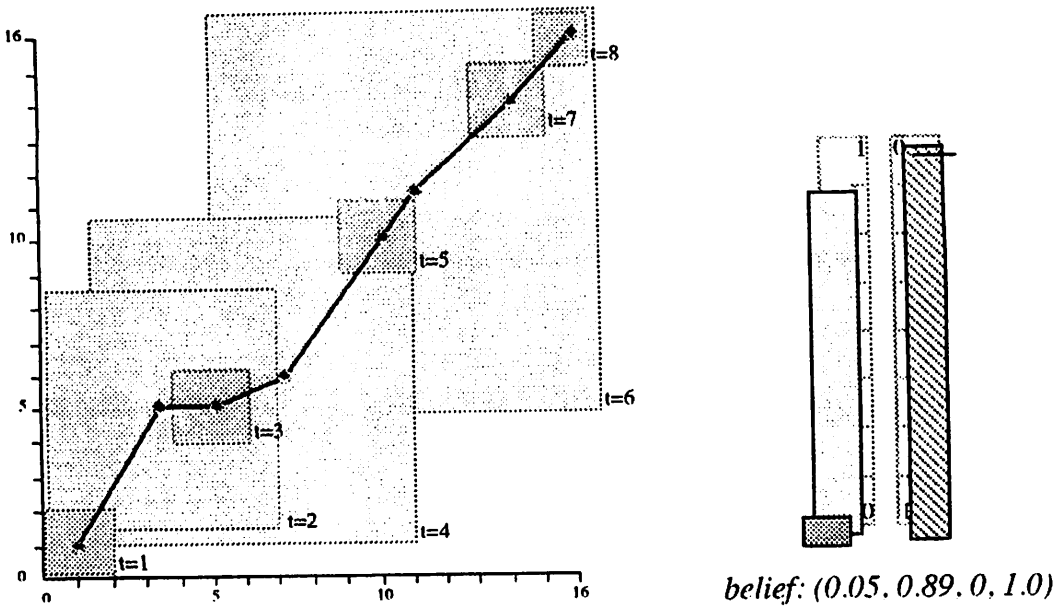


Figure 14: Experiment 5: approximate data, approximate search, approximate knowledge

Table 1: Experiment Summary.

Exp	Data?	Search?	Knowledge?	KS ex	Real Time	Hyps	Goals
E1	precise	full	complete	982	02:40:12	1,298	1,293
E2	approx	full	complete	158	00:13:19	412	198
E3	approx	full	approx	86	00:08:19	388	102
E4	approx	approx	complete	134	00:11:00	398	172
E5	approx	approx	approx	62	00:06:29	374	76

Abbreviations

Exp: experiment
Data?: precision of data used to form solution:
precise = precise data was used
approx = approximate data was used
Search?: search techniques used:
full = all alternative solutions are examined
approx = approximate search was used
Knowledge?: characteristics of knowledge sources used:
complete = knowledge sources used all constraints
approx = knowledge sources ignored or simplified synthesis constraints
KS ex: the number of knowledge source executions required to find the solution(s)
Real Time: the actual cpu time required to generate the solution, in the format hh:mm:ss
Hyps: the number of hyps generated during problem solving
Goals: the number of goals generated during problem solving

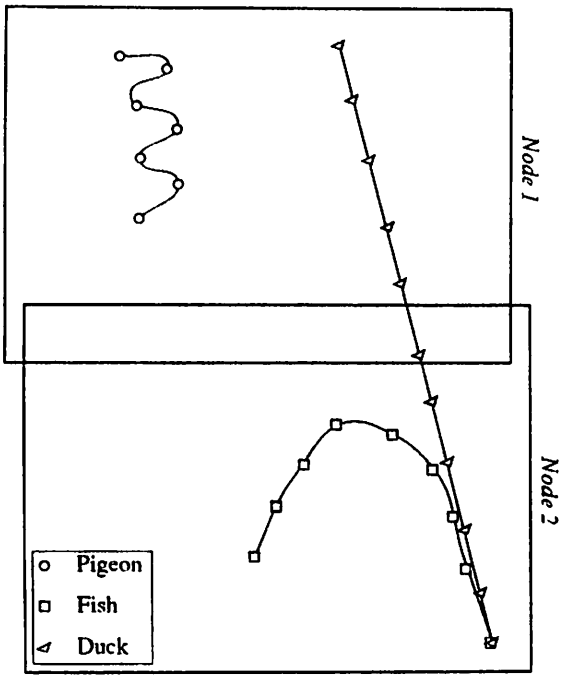


Figure 15: Experiment 6: Controlling Approximate Processing

5.2 Controlling Approximate Processing

Experiment 6 shows an example of how the set of approximations can be altered to respond to the current situation during problem solving. It also demonstrates the reduction in overhead for low-level blackboard control during approximate processing using the new control architecture, compared to just using the approximations alone. Two experimental runs were analyzed; in the first run, the meta-controller and high level strategies detailed in Section 4.4 were used. In the second run, control of the domain processing was simulated using only an elaborate evaluation function with no parameterization of the low-level control loop, i.e., at each point in the second run, the KSI that was executed in the first run at that point was forced to execute. Each system thus ran the same domain KSIs, in the same order. The comparison of the runs (in which the old control system is forced to mimic the new one) illustrates the computational benefits of a parameterized low-level control loop (i.e., the reduction in low-level overhead).

The environment (scenario description) for the run consisted of two nodes, each responsible for a different physical region. Three "vehicles" are present in the situation: a meandering, non-threatening pigeon, a fish, and a fish-eating duck, which attacks the fish during the course of the scenario. Nodes 1 and 2 have the job of notifying any fish of potential bird attacks. Node 1 had good, spatially distributed data, so it chose the goal-directed strategy. Node 2 was given noisy data; thus, it chose the cluster-directed processing strategy. This scenario is depicted in Figure 15. The strategy and focus shifts at each node are summarized in Figure 16.

A summary of our results include:

- Reduction in Overhead: The overhead for low-level control was significantly reduced: in node 1 by 33% and in node 2 by 80%. Node 2 came out especially well because many signals are never processed by the low-level control loop in the parameterized system. Overhead in the low-level control loop is defined as anything except actual KSI execution and time spent in the meta-controller. [7]

[7]Because run 2 *simulated* a BB1-style evaluation function, there was no meaningful way to measure the time that the simulated meta-controller spent in dynamically creating the appropriate evaluation function that would execute the desired KS.

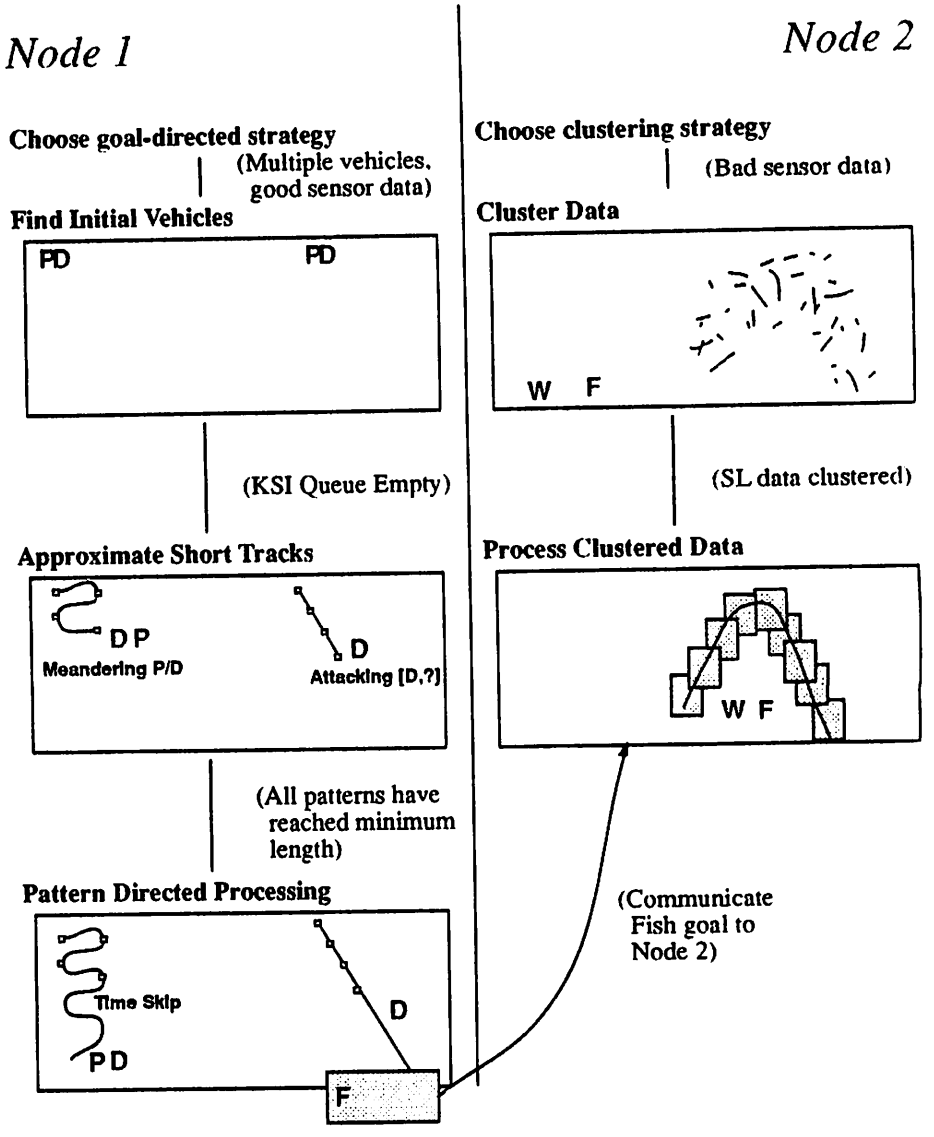


Figure 16: Strategies and Foci at Nodes 1 and 2 in Experiment 6

- Reduction in Access Time: Total blackboard access time was reduced by 15% in the new architecture.
- Reduction in KSIs: Total KSIs created (parameterized / non-parameterized): node 1 – (232/280), node 2 – (222/391).
- Reduction in Goals: Total goals created (parameterized / non-parameterized): node 1 – (221/438), node 2 – (248/831). Node 2 in the non-parameterized system creates more low-level goals and KSIs that are never executed. This is because the unneeded goals (and the KSIs they stimulate) are never created in the parameterized system, which focuses on clustering hypotheses.
- Total executed KSIs in both runs: node 1 – 76, node 2 – 86.
- Filtering hypotheses and goals took $< 1\%$ of the total processing time for each node in the parameterized run. Likewise, passing objects back through their respective filters took $< 1\%$ of the time. Objects were passed back through the filters each time there was a change in focus. This statistic shows that filtering does not add a significant amount of overhead.

In summary, the parameterization of the low-level control loop permits a larger percentage of a node's processing time to be used in executing KSIs. For a more complete description, see [13].

6 Summary and Future Work

We have demonstrated experimentally that approximate processing is a useful technique for decreasing the processing time of an algorithm while generating acceptable (but less certain, less precise, or less complete) solutions. It can be applied to complex cognitive algorithms that operate under situation-dependent constraints in order to form more predictable time bounds.

In order to implement approximate processing significant extensions to the blackboard problem solving architecture were made to answer the following questions:

1. How should intermediate results of problem solving be represented so that precise and approximate intermediate results can be combined in further processing?
2. How should a knowledge source be structured so that it can exploit intermediate results of varying levels of approximation?
3. How should approximate knowledge sources of different types be organized and controlled?
4. How should uncertainty caused by the use of approximate search, data, and knowledge be represented?
5. How should the control architecture be structured so that it can smoothly integrate different types of approximate processing strategies?
6. How should the control architecture be implemented to minimize overhead when the current approximate processing strategy uses only a subset of the partial results and a subset of the applicable knowledge?

Question one brought about a new way of representing approximate hypotheses of varying precision in a consistent and integrated manner. This included an existential data interpretation that indicates support for a hypothesis *somewhere* within a range, and a precision function based on the size of the sensed area, the range of sensed event classes, and the distribution of the data within the sensed area that is used to determine the likelihood that a piece of approximate data meets some constraint.

The answer to the second and third question entailed the creation of generalized knowledge sources that implement a range of approximate processing strategies: the specific strategy to be

used is specified by an approximation control block that is associated with a knowledge source instantiation. The approximation control block contains parameters such as the input and output belief thresholds that limit the data processed and output by a knowledge source: the input, output level, and work level precision filters that indicate the level of data approximation that occurs: and the search and knowledge approximation levels that indicate the amount of grammar compression and constraint relaxation that occurs.

A four-valued belief system was added to capture the notions of approximate and incomplete processing and data for question four. The four values — certainty, plausibility, refutation, and doubt — were used to summarize both the uncertainty and incompleteness of a hypothesis. Measures of ignorance and conflict derived from the belief allow us to represent the incomplete processing of hypotheses (which will have large ignorance measures), the aggregation of hypotheses, the application of approximate knowledge to hypotheses, and the use of approximate search strategies.

A new control component, based on a parameterization of the blackboard control loop, enables the system to control the application of different approximate processing strategies without excessive overhead (questions five and six). It provides the flexibility to dynamically respond to changes in the current situation and exploit approximate data and knowledge. This parameterization includes filters for the hypotheses, goals and KSIs that reduce the amount of data that needs to be processed: mappings from hyps to goals, goals to goals, and goals to KSs that indicate what approximation strategy is being pursued; merging of goals and KSs that indicates the precision of control desired; and precondition evaluation. The system described in the paper is fully implemented (except where indicated) in the DVMT.

Future work will involve the integration of hard deadlines into this framework. This will require us to generate initial time bounds estimations, revise them, and recognize when a deadline will be missed, based on the current processing strategy and set of approximations. A model of plan monitoring and time estimation for the DVMT has already been developed by Durfee[15]. Our first approach to hard deadlines will involve extending this work to the revised blackboard architecture. We must also develop techniques for determining — given a prediction that deadlines will be missed if we continue the current strategy and set of approximations — what the new control strategies and set of approximations are in the current context that will allow the system to meet the deadline [4]. To accomplish this, a more complex and flexible mechanism for expressing control plans is needed, as well as work on how to dynamically move between strategies and build control plans dynamically during problem solving.

In summary, we believe that this work is an important first step in understanding the implications of approximate processing and other satisficing approaches to the architecture of real-time problem-solving systems.

7 Acknowledgment

We would like to acknowledge M. A. Humphrey for his significant contribution to the implementation of this system and the gathering and analysis of the experimental results.

References

[1] V. Lesser, J. Pavlin, and E. Durfee, "Approximate processing in real-time problem solving," *AI Magazine*, vol. 9, pp. 49–61, Spring 1988.

[2] H. A. Simon, *The Sciences of the Artificial*. Cambridge, MA: MIT Press, 1968.

[3] M. Boddy and T. Dean, "Solving time-dependent planning problems," in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Aug. 1989.

[4] K. Kanazawa and T. Dean, "A model for projection and action," in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Aug. 1989.

[5] E. J. Horvitz, "Reasoning under varying and uncertain resource constraints," in *Proceedings of the Seventh National Conference on Artificial Intelligence*, Aug. 1988.

[6] T. Dean and M. Boddy, "An analysis of time-dependent planning," in *Proceedings of the Seventh National Conference on Artificial Intelligence*, Aug. 1988.

[7] P. P. Bonissone, S. S. Gans, and K. S. Decker, "RUM: A layered architecture for reasoning with uncertainty," in *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Aug. 1987.

[8] D. D. Corkill, V. R. Lesser, and E. Hudlická, "Unifying data-directed and goal-directed control: An example and experiments," in *Proceedings of the National Conference on Artificial Intelligence*, (Pittsburgh, Pennsylvania), pp. 143–147, Aug. 1982.

[9] P. P. Bonissone and K. S. Decker, "Selecting uncertainty calculi and granularity: An experiment in trading-off precision and complexity," in *Uncertainty in Artificial Intelligence* (L. N. Kanal and J. F. Lemmer, eds.), North Holland, 1986. Also Technical Report 85-CRD-171, GE Corporate Research and Development, 1985.

[10] J. D. Lowrance and T. D. Garvey, "Evidential reasoning: A developing concept," *IEE 1982 Proceedings of the International Conference on Cybernetics and Society*, pp. 6–9, 1982.

[11] B. Hayes-Roth, "A blackboard architecture for control," *Artificial Intelligence*, vol. 26, pp. 251–321, 1985.

[12] B. Hayes-Roth, "A multi-processor interrupt-driven architecture for adaptive intelligent systems," in *Proceedings of the Third Annual AAAI Workshop on Blackboard Systems*, (Detroit), Aug. 1989. Also KSL-87-31.

[13] K. S. Decker, M. A. Humphrey, and V. R. Lesser, "Experimenting with control in the DVMT," in *Proceedings of the Third Annual AAAI Workshop on Blackboard Systems*, (Detroit), Aug. 1989. Also COINS TR-89-85.

[14] A. Collinot, "Revising the BB1 basic control loop to control the behavior of knowledge sources," in *Blackboard Architectures and Applications* (V. Jagannathan, R. Dodhiawala, and L. S. Baum, eds.), pp. 27–43, Academic Press, Inc., 1989.

[15] E. H. Durfee and V. R. Lesser, "Incremental planning to control a time-constrained, blackboard-based problem solver," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 24, Sept. 1988.

List of Figures

List of Tables