Knowledge-Based Anytime Computation

Abdel-illah Mouaddib*
IUT de Lens-CRIL/CRIN-INRIA
Rue de l'université, S. P. 16
62300 Lens Cedex France
mouaddib@lens.lifl.fr

Shlomo Zilberstein†
Computer Science Department
University of Massachusetts
Amherst, MA 01003 U.S.A.
shlomo@cs.umass.edu

Abstract

This paper describes a real-time decision-making model that combines the expressiveness and flexibility of knowledge-based systems with the real-time advantages of anytime algorithms. Anytime algorithms offer a simple means by which an intelligent system can trade off computation time for quality of results. Previous attempts to develop knowledge-based anytime algorithms failed to produce consistent, predictable improvement of quality over time. Without *performance profiles*, that describe the output quality as a function of time, it is hard to exploit the flexibility of anytime algorithms. The model of *progressive reasoning* that is presented here is based on a hierarchy of reasoning units that allow for gradual improvement of decision quality in a predictable manner. The result is an important step towards the application of knowledge-based systems in time-critical domains.

1 Introduction

This paper describes a real-time decision-making model that combines the expressiveness and flexibility of knowledge-based systems with the real-time advantages of anytime algorithms. The application of knowledge-based systems to real-time domains such as process control, automated navigation systems, medical monitoring, and robotics is an important problem in artificial intelligence. A major difficulty in solving this problem arises since real-time domains require continuous operation and predictable performance while knowledge-based systems rely on time-consuming algorithms with highly variable performance. Performance variability has limited the application of knowledge-based systems to real-time domains.

To avoid these problems, the AI community has constructed several general paradigms such as anytime algorithms [Dean and Boddy, 1988; Horvitz, 1987], design-to-time [Garvey and Lesser, 1993] and various types of progressive reasoning techniques [Winston, 1984; Mouaddib *et al.*, 1992]. Another approach has been to construct special architectures for particular problem domains such as the Guardian system for monitoring the patient's condition in an intensive care unit [Hayes-Roth *et al.*, 1991]. Among the general paradigms, anytime algorithms in particular are increasingly used in AI applications since they are easy to construct and monitor and since they can be efficiently combined to produce larger real-time systems [Zilberstein, 1993]. But successful control of anytime algorithms require the use of performance profiles that describe the dependency of output quality on computation time. Knowledge-based anytime algorithms that have been proposed in the past do not exhibit high correlation between computation time and output quality. Therefore, their performance profiles can not be constructed and their use as anytime algorithms is very limited.

In this paper, we describe a model of knowledge-based progressive reasoning that meets the requirements of a "well-behaved" anytime algorithm. The model consists of a rule-based language and an associated inference mechanism. Problem solving is performed by an iterative process that produces quickly a first solution and refines it step-by-step until interrupted by the control mechanism. In Section 2, we describe the problem of constructing knowledge-based anytime algorithms and the difficulties with current solutions to the problem. Section 3 describes the model of progressive reasoning in detail. In Section 4, we describe an application of the model to a collision avoidance system. We conclude with a summary of the contribution of this work and future research directions.

2 Anytime computation and knowledge-based systems

This section presents the advantages of anytime algorithms for real-time decision making. The section de-

*This author continues to collaborate with CRIN-INRIA until September 1995 at the address: Bâtiment loria BP 239, 54500 Vandoeuvre, France. e-mail: amouaddi@loria.fr

†Support for this author was provided in part by the National Science Foundation under grant IRI-9409827.

scribes several efforts to develop knowledge-based systems as anytime algorithms and explains the difficulty of combining the two paradigms.

2.1 Anytime algorithms

Anytime algorithms are algorithms whose quality of results improve gradually as computation time increases, hence they offer a tradeoff between resource consumption and output quality. The quality of the results produced by anytime algorithms can be measured by their level of certainty, accuracy, or specificity. A performance profile [Dean and Boddy, 1988] is a probabilistic description of the dependency of output quality on computation time. Zilberstein and Russell generalized this notion capturing also the dependency on input quality using conditional performance profiles. The latter can be used to optimally compose real-time systems using a library of anytime algorithms [Zilberstein, 1993]. To solve the composition problem, an important distinction is made between two types of anytime algorithms, namely interruptible and contract algorithms. An interruptible algorithm can be interrupted at any time to produce results whose quality is described by its performance profile. A contract algorithm offers a similar trade-off between computation time and quality of results, but it must know the total allocation of time in advance. Interruptible algorithms are more flexible, but they are also more complicated to construct. Zilberstein [1993] solved that problem by a general construction that produces an interruptible version for any given contract algorithm and requires only a small, constant penalty. Subsequently, a set of programming tools for composition and monitoring of anytime algorithms have been developed by Grass and Zilberstein [1995].

Since their introduction in the late 1980's, anytime algorithms have been applied to such real-time problems as mobile robot navigation, medical diagnosis and monitoring, information gathering, and model-based diagnosis. In addition, several anytime algorithms have been developed for evaluation of probabilistic networks and for dynamic programming. But the technique has been less successful in the area of knowledge-based systems.

2.2 Real-time knowledge-based systems

In a 1988 comprehensive survey of real-time knowledge-based systems [Laffey *et al.*, 1988], the authors concluded that "Currently, *ad hoc* techniques are used for making a system produce a response within a specified time interval." Unfortunately, not much has been changed since that survey was conducted. The primary method for achieving real-time performance is based in many cases on speeding up individual algorithms in a generate-and-test manner. This method slows down the development of real-time systems and makes them inefficient when operating in dynamic environments.

One approach to deal with the problem has been to develop specialized architectures for particular domains. One successful example is the Guardian system for monitoring the patient's condition in an intensive care unit [Hayes-Roth *et al.*, 1991]. The system integrates perceptual capabilities with real-time reasoning and action. Closer to our approach is the patient monitoring system developed by Ash *et al.* [1993]. The system exhibits an anytime behavior accomplished by organizing actions in a hierarchical structure. The result has been integrated into the Guardian system to provide a response when the slower, deliberative methods cannot complete their tasks. The work described in this paper extends the hierarchical decomposition approach to general knowledge-based systems. Our motivation is best summarized by the conclusion of the above survey:

> "We concluded that one of the main reasons for this situation is that expert systems developers have often tried to apply traditional tools to applications for which they are not well suited. Tools specifically built for real-time monitoring and control applications need to be built. An immediate goal should be the development of high-performance inference engines that can guarantee response times."

2.3 Knowledge-based anytime computation

Knowledge-based systems rely on an inference engine combined with a body of declarative knowledge. Since the amount of relevant knowledge varies from situation to situation, it is hard to predict how problem solving will progress as a function of time. Hence, a naive implementation of progressive reasoning techniques does not lead to "well-behaved" anytime algorithms. In this section we describe two attempts to construct anytime knowledge-based systems and their limitations.

Elkan [1990] presents an abductive strategy for discovering and revising plausible plans. In his approach, candidate plans are found quickly by allowing them to depend on assumptions. His formalism makes explicit which antecedents of rules have the status of default conditions. Candidate plans are refined incrementally by trying to justify the assumptions on which they depend. The implementation of the model replaces the standard depth-first exploration strategy of Prolog with an iterative-deepening version. The result is an anytime algorithm for incremental approximate planning.

As we pointed out earlier, it is hard to find the performance profile of such a planner. Even in the context of particular domain knowledge, the performance of the inference engine (a theorem prover) is going to be highly dependent on the particular query and is hard to predict in advance. Another difficulty is to measure the quality of results in a meaningful way. Our model of progressive reasoning addresses successfully these two issues.

Smith and Liu [1989] propose a monotone query processing algorithm which derives approximate answers directly from relational algebra query expressions. An *ap-*

proximate relation R of a standard relation S is a subset of the Cartesian product of all the domains of S that can be partitioned into two blocks, the certain set C and the possible set P such that: $C \subseteq S$ and $R = C \cup P \supseteq S$. The algorithm assumes that the information stored in the database is complete and that the input data is precise. An incomplete answer to a query is generated when there is not enough time to complete processing the query, or because some relation that must be read is not accessible.

Vrbsky and Liu have implemented the approximate query processing algorithm in a system called APPROXIMATE [Vrbsky and Liu, 1992]. The operation associated with each leaf node of the tree is an approximate-read that returns one segment of the requested relation at a time. Approximate relational algebra is used in order to evaluate the tree. Initially, the certain set is empty for every approximate object and the possible set is the complete range of values for the particular object. After each approximate-read, a better approximate answer to the query is produced. The exact answer is returned if the system is allowed to run to completion.

APPROXIMATE suffers from the same problem as Elkan's approximate planning technique, namely the difficulty to derive the performance profile of the system due to its dependence on the contents of the database and the complexity of the query. It is also hard to evaluate the quality of an approximate relation and represent it quantitatively. To summarize, existing knowledge-based techniques are hard to convert to anytime algorithms due to wide variability in performance improvement over time.

3 Progressive reasoning

Progressive reasoning is an important technique to design knowledge-based systems that exhibit a highly predictable time-quality tradeoff. The technique uses multi-level deliberation in order to gradually transform an approximate solution into a precise one. The mapping from the set of inputs (problem instance) to the set of outputs (solution) is based on progressive exploration of data and knowledge, hence the name progressive reasoning. Progressive exploration is facilitated by using a hierarchical structure of input elements defined by weights that the system's designer attaches to each input according to its importance. Correspondingly, knowledge is also organized in a hierarchical way. This mapping is especially suitable in domains where the reasoner uses abstraction to structure the search space (as in hierarchical planning), and in problems that require the result to be expressed at varying levels of detail (as in model-based diagnosis).

Furthermore, this organization is an important factor in reducing the unpredictability of knowledge-based systems by limiting the amount of knowledge and data that is the focus of the system at each level of the hierarchy. As a result, we can characterize precisely the tradeoff between computation time and quality of results offered by progressive reasoning systems.

This section explains how progressive reasoning works. The two major issues in progressive reasoning are the hierarchical organization of knowledge and the control of the evolution of solution quality. We cover these issues by first describing the conceptual model and then its implementation and properties. The implementation is based on the GREAT (Guaranteed REAsoning Time) model [Mouaddib *et al.*, 1992].

3.1 Conceptual model

The distinctive features of our progressive reasoning approach result from the combination of: (1) A generic knowledge representation language that facilitates progressive problem solving; and (2) A control mechanism that progressively feeds data and knowledge into the inference engine using a preference criterion.

Knowledge is represented using a rule-based language that refers to data by a set of attributes. The progressive problem solving process assumes that there are several solutions with different qualities each of which represents an intermediate (approximate) view of the final solution. The transition from one solution to a more precise one is done by using additional attributes and rules that are more precise than those previously used. Each rule can change the current solution by adding, deleting or modifying the attributes contained in the current solution. This process is shown schematically in Figure 1.

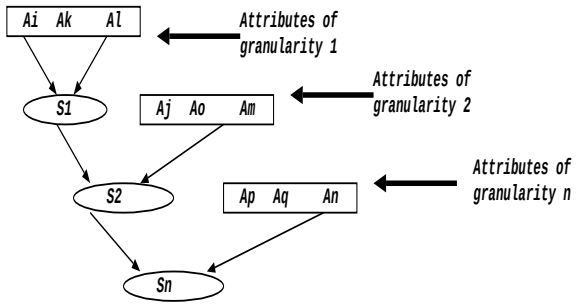


Figure 1: The schematic structure of progressive reasoning.

This organization is defined by a preference criterion representing the accuracy of the attributes specified by the system's designer. The attributes preference criterion allows the solution to be represented at different levels of detail. This preference criterion, named *criticality* by Knoblock *et al.* [1991], is generalized by the notion of *granularity* in the GREAT model. We also use the certainty of attributes to control progressive manipulation of data of the same granularity. Based on the attributes preference criterion, the preference criterion of rules is computed automatically as will be shown in the next section. The preference criterion defines aggregations of attributes and aggregations of rules that are

referred to as *regions* and *packages* respectively. This leads to the definition of a hierarchy of reasoning levels each of which improves the solution quality of the previous one. Reasoning at each level is performed by *execution cycles* that include a reasoning cycle and an evaluation cycle as described below.

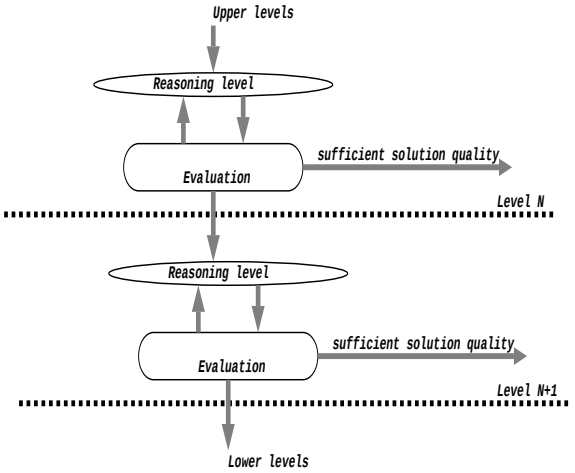


Figure 2: A single execution cycle consists of a reasoning cycle and an evaluation cycle at a distinct level of reasoning.

Reasoning cycles

Each reasoning cycle consists of using one region and one package of rules corresponding to the current level of reasoning. This approach limits the amount of relevant data and knowledge at each level. Reasoning is performed by exploring rules of the current package and attributes of the current region. The cycle of reasoning consists of activating relevant rules. The execution of rules modifies the quality of the solution which is evaluated at the end of each cycle. When the quality is unsatisfactory, another execution cycle is activated. This process is repeated until the quality is satisfactory or the deadline is reached.

Evaluation cycle

At the end of each reasoning cycle, the evaluator judges the quality of the actual solution and decides whether further improvement is needed. When further reasoning is required, the evaluator decides whether it must be pursued at the same level or it is necessary to change the reasoning level and use more precise attributes. Quality is evaluated based on a multi-dimensional criterion that measures the level of completeness, certainty, and precision of the solution. The evaluator checks first whether completeness and certainty are insufficient, in which case another reasoning cycle at the same level is executed. Otherwise, the evaluator verifies the precision quality of the solution. If it is acceptable, the solution is returned and the process terminates. Otherwise, the evaluator indicates that another reasoning level is required. That means that the solution needs additional attributes that are more precise than those in the current level.

3.2 Implementation: The "GREAT" Model

In this section we describe the GREAT model, which is an implementation of real-time progressive reasoning. In particular, we show how to build the hierarchy of reasoning levels, how to use it to perform progressive reasoning, and how this leads to high performance predictability.

Let $P = <D, K>$ be a problem characterized by its domain, D, of data (attributes) and its rule base, K. Let $D = \{d_1, d_2, ..., d_n\}$, where d_i is a data component. Let $K = \{r_1, r_2, ..., r_m\}$, where r_i is a rule of the following form:
r_i : **if** $f(d_i, ..., d_k)$ **then** $g(d_j, ..., d_l)$ where
$f : D^{k-i+1} \rightarrow \{True, False\}$ and $g : D^{l-j+1} \rightarrow D^p$
We now define in more detail several concepts that were introduced earlier.

Context and focus in GREAT A context, C, represents the point of view adopted to handle a given situation. It is a control data structure that makes it possible to adapt a task to a particular situation. The context defines the focus of the task that consists of a subset F of the data in D and a subset K_r of the knowledge (production rules) in K. Consequently, the focus limits the size of data and knowledge used at each cycle.

Granularity of attributes When operating under time pressure, human experts focus their attention on the more relevant data in order to build quickly an approximate solution. In order to formalize this intuitive notion of relevance, we use the concept of *granularity*. As we indicated earlier, the granularity of data components is specified by the system's designer. We represent this information by a function defined as follows:

$$G : C \times F \rightarrow \aleph$$

where C is a set of contexts, F is the focus, and $\aleph$ is the set of natural numbers. In general, $G(c, a) = g$ means that the granularity of the attribute a of the focus in the context c is g, where g is the order of the attribute a in the hierarchy of the current context.

Granularity of rules The progressive use of data throughout the data hierarchy is accompanied by a similar progressive use of knowledge. In order to achieve this, we extend the notion of granularity to rules. The granularity of a rule is computed automatically based on attribute granularity. In our implementation, the granularity of a rule is the maximum of the granularities of the attributes matching its left hand side. This definition means that a rule manipulating attributes at a certain level of accuracy has the same level of accuracy.

$$G' : C \times K_r \rightarrow \aleph$$

$$G'(context, rule) = \max_{c_i \in condition} G(context, c_i)$$

Aggregations and hierarchy reasoning levels The granularities of attributes and rules are used to determine a set of regions, R_i, and a set of packages of rules, P_i, as follows: $R_n = \{A_i \in F | G(context, A_i) = n\}$ and $P_n = \{r_i \in K_r | G(context, r_i) = n\}$. The reasoning levels L_i, represented by (R_i, P_i), consists of the rules of package P_i and the data of region R_i. Finally, the hierarchy, H, is a sequence of reasoning levels, $H = \{L_1, ..., L_n\}$. The length of the sequence, n, is the depth of reasoning.

Quality measures Assigning a precise quality to the approximate results of a knowledge-based system is a hard problem. In the implementation of GREAT, we used a three dimensional quality measure (certainty, completeness and precision) to judge the quality of each solution. The evaluator bases its judgement on a library of evaluation tables. For each context and each level, the evaluator uses a different evaluation table. Each table determines the quality based on the following solution characteristics:

- Completeness is specified by a list of attributes that the solution must contain in order to be complete.
- Precision is determined by the minimal granularity that the attributes of the previous list must have.
- Certainty is determined by the minimal level of certainty that all the attributes of the previous list must have.

Progressive knowledge-based problem solving Once the hierarchy of reasoning levels is created, the progressive reasoning process starts its serie of reasoning cycles followed by evaluation cycles. This process is summarized below:

- $S^0_{res} = \emptyset$ is the initial solution.
- $S^1_{res} = L_1(S^0_{res})$ is the result of activating the first level L_1 and allowing it to add new attributes, delete and modify attributes of S^0_{res}.
- $S^i_{res} = L_i(S^{i-1}_{res})$ is, in general, the result of activating level L_i that receives from the previous level the solution S^{i-1}_{res}. The activation of L_i consists of the use of R_i and P_i. S^i_{res} is built from both attributes of R_i and attributes of S^{i-1}_{res}. We formalize the activation of the level L_i by: $S^i_{res} = \{g_{p_i}(A_i \in (R_i \cup S^{i-1}_{res}))\}$, where g_{p_i} consists of the activation of rules belonging to P_i.

The evaluation cycle is invoked at each level, L_i, to assess the quality of the solution S^i_{res} by checking its completeness and its certainty. If the evaluation fails, processing resumes at level L_i until the evaluation succeeds or the deadline is reached. When the evaluation succeeds, the evaluator invokes the next level in order to improve the *precision* of the solution.

The construction of the reasoning levels (L_i) guarantees the improvement of the solution quality as the system moves from one level to another. This is true since the attributes computed by a new level L_i are always more precise than those contained in the current result. The level L_i deletes, from the current solution, attributes judged incorrect at the current level of granularity (for example, the attribute *time* that contains the value *6h* can be replaced at another level of granularity which takes minutes into account by the value *6h15min*). That is why the quality of the solution S^i_{res} is preferred to that of S^{i-1}_{res}.

4 Application: A collision avoidance system

Consider the problem of collision avoidance in a railway network. Assume that a railway network consists of n horizontal railway tracks and n vertical tracks each of which is used by one train. Each horizontal track intersects all the vertical ones. The main objective is to prevent two trains from colliding with each other at one of the n^2 crossings. The system needs to detect potential collisions and to optimally modify the speeds of the train to avoid any chance of collision. The rest of this section describes how the GREAT model was used to construct this collision avoidance system [Mouaddib, 1993].

4.1 Collision avoidance with progressive reasoning

In our implementation, the collision avoidance task is achieved using different levels of approximation depending on how much time is available. The deadline is defined as the time remaining before a collision may occur. In this domain, there are several different policies to avoid collision. The system can either control the train in a qualitative manner (stop, slow, speed) or it can compute the actual speed at various levels of precision. Different constraints can be taken into account, such as passenger comfort (this means that acceleration or slowing down have to be limited), the time-table of each train, and the priorities or time-dependent utilities of the different trains.

To capture these types of knowledge, we used the components of GREAT as follows:

- The set of data attributes is defined as:
  D = {speed, tendency, nextcross, distance, railpriority, maxspeed, minspeed, timetable}
- The context of a situation is defined as:
  $C = ControlCollision$. In this context the set D is

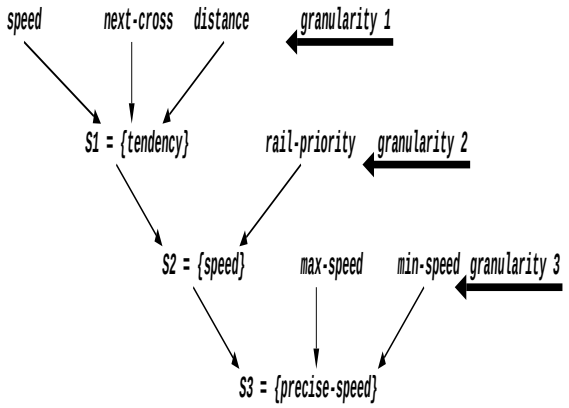


Figure 3: The data hierarchy of the collision avoidance context.

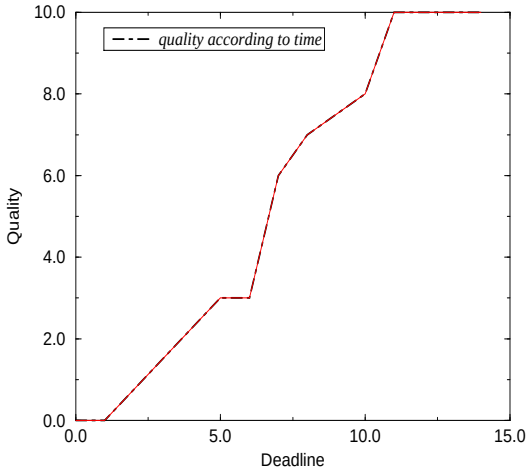
reduced on F:
F ={speed, tendency, nextcross, distance, railpriority, maxspeed, minspeed}
(note that the attribute timetable is omitted).

- The granularities of attributes are shown in Figure 3.
- The granularities of rules are calculated from the granularities of attributes by the formula given in section 3.2. For example, the rule detecting collision is: *r1:* **if** *Collision(speed, distance, nextcross)* **then** *Compute Tendency*. So,
$G'(C, r1) =$
$Max(G(C, speed), G(C, distance), G(C, nextcross))$
$= Max(1, 1, 1) = 1$
- The regions and packages of rules (reasoning levels) are created accordingly.

4.2 Experimental results

The performance of GREAT is summarized by the graphs shown in Figure 4. We examine the performance as a function of two parameters: (1) the deadline, that causes gradual improvement of solution quality as it increases, and (2) the size of the network, that causes gradual degradation of solution quality as it increases. This is because control time increases as a function of the size while the deadline decreases. Measuring the precise "objective" quality of the solution is hard in knowledge-based systems, but the level of reasoning is a good indication of solution quality in our system. Hence we use the ratio between the *number of activated levels* and the *total number of levels* as our overall quality measure. We are currently investigating possible improvements in measuring overall quality, but the above measure is sufficient to show that the system exhibits gradual improvement/degradation of quality as computational time increases/decreases.

To summarize, our experimental results show that GREAT establishes a correlation between solution quality and computation time which can be quantitatively described by a performance profile.



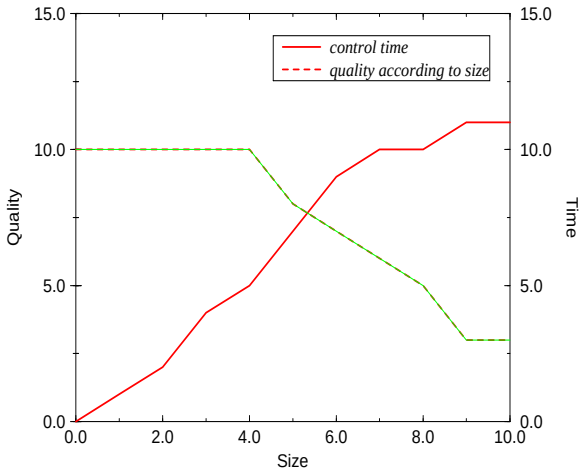


Figure 4: Quality as a function of time and size.

5 Conclusion

We have presented a model of progressive reasoning – a knowledge-based approach to real-time decision making – and its implementation. Our preliminary results show that by structuring the available knowledge in a hierarchical way and by limiting the amount of data and knowledge used at each level, we can construct knowledge-based systems that have all the characteristics of a "well-behaved" anytime algorithm. In particular, our system exhibits gradual improvement/degradation of quality as computation time increases/decreases (or problem size decreases/increases). Moreover, the behavior of the system is consistent and can be characterized by a performance profile, typically used in control of anytime computation. This result is an important step toward the adaptation of knowledge-based systems, that normally

exhibit high variability in performance, to real-time domains where predictability of performance is essential.

Further work is needed on several aspects of the implementation including the designer's task of mapping domain knowledge into a hierarchical structure, the development of more precise quality measures, and the use of a utility-based approach to control the operation of the system. Other research directions include the application of the model to construct a multi-agent system and implementation of a larger application.

References

[Ash *et al.*, 1993] D. Ash, G. Gold, A. Siever and B. Hayes-Roth. Guaranteeing real-time response with limited resources. *Artificial Intelligence in Medicine*, 5(1):49–66, 1993.

[Boddy and Dean, 1989] M. Boddy and T. Dean. Solving time-dependent planning problems. Technical Report CS-89-03, Department of Computer Science, Brown University, Providence, 1989.

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

[Elkan, 1990] C. Elkan. Incremental, approximate planning. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 145–150, Boston, Massachusetts, 1990.

[Garvey and Lesser, 1993] A. Garvey and V. Lesser. Design-to-time real-time scheduling, In *IEEE Transactions on Systems, Man and Cybernetics*, 23(6):1491–1502, 1993.

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

[Hayes-Roth *et al.*, 1991]
B. Hayes-Roth *et al.* Guardian: A Prototype Intelligent Agent for Intensive-Care Monitoring. Technical Report KSL-91-42, Stanford Knowledge Systems Laboratory, Stanford, California, 1991.

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

[Knoblock *et al.*, 1991] C. A. Knoblock, J. D. Tenenberg and Q. Yang. Characterizing abstraction hierarchies for planning. In *Proceedings of the 9th National Conference on Artificial Intelligence*, pages 692–697, Anaheim, California, 1991.

[Laffey *et al.*, 1988] T. J. Laffey, P. A. Cox, J. L. Schmidt, S. M. Kao and J. Y. Read. Real-time knowledge based systems. *AI Magazine* 9(1):27–45, Spring 1988.

[Mouaddib *et al.*, 1992] A. I. Mouaddib, F. Charpillet and J. P. Haton. Progressive reasoning and approximation. In *Proceeding of the AAAI Workshop on Imprecise and Approximate Computation*, San Jose, California, 1992.

[Mouaddib, 1993] A. I. Mouaddib. Contribution au raisonnement progressif et temps réel dans un univers multi-agents. Thèse de doctorat es-science informatique, université de Nancy 1, 1993.

[Mouaddib, 1995] A. I. Mouaddib. Progressive goal-directed reasoning for real-time systems. *International Journal of Engineering Intelligent Systems*, (3)2:67–77, June 1995.

[Smith and Liu, 1989] K. P. Smith and J. W. S. Liu. Monotonically improving approximate answers to relational algebra queries. *COMPSAC '89*, Orlando, Florida, 1989.

[Vrbsky and Liu, 1992] S. V. Vrbsky and J. W. S. Liu. Producing monotonically improving approximate answers to database queries. In *Proceedings of the IEEE Workshop on Imprecise and Approximate Computation*, pages 72–76, Phoenix, Arizona, 1992.

[Winston, 1984] P. H. Winston. Progressive deepening keeps computation within time bounds. *Artificial Intelligence* (2nd ed.), pp. 129–131, Addison-Wesley, Reading, Massachusetts, 1984.

[Zilberstein, 1993] S. Zilberstein. Operational rationality through compilation of anytime algorithms. Ph.D. dissertation, Computer Science Division, University of California at Berkeley, 1993.

[Zilberstein, 1995] S. Zilberstein. On the utility of planning. In M. Pollack (Ed.), *SIGART Bulletin Special Issue on Evaluating Plans, Planners, and Planning Systems*, (6)1:42–47, 1995.

[Zilberstein and Russell, 1993] S. Zilberstein and S. J. Russell. Anytime sensing, planning and action: A practical model for robot control. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1402–1407, Chambery, France, 1993.

[Zilberstein and Russell, 1995] S. Zilberstein and S. J. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, forthcoming, 1995.