Mechanisms for Efficiency in Blackboard Systems

Micheal Hewett
Department of Computer Sciences
University of Texas at Austin
Taylor Hall 2.124
Austin, TX 78712-1188
hewett@cs.utexas.edu

Rattikorn Hewett
Computer Science and Engineering Department
Florida Atlantic University
P.O. Box 3091
Boca Raton, FL 33431-0091
hewett@cse.fau.edu

Abstract

The RETE algorithm had a great impact on the development of efficient production systems by providing a fast pattern matching mechanism for activation. No similar mechanism has been available to speed up activation and scheduling in blackboard systems. In this paper we describe efficient, general-purpose efficiency mechanisms that are better suited to blackboard systems than RETE-like networks. We describe a knowledge source compiler that produces match networks and demons for efficient activation and rating while compiling the entire system for increased execution speed. Experiments using the enhancements in a general-purpose blackboard shell illustrate a substantial improvement in run time, including an 80–92% decrease in activation time. The mechanisms we describe are general enough to be used in most existing blackboard systems.

1 Introduction

The blackboard architecture is a flexible framework for solving complex problems. It supports incremental development of solutions, integrated use of different types of knowledge at different levels of abstraction, and opportunistic control of reasoning. This flexibility can lead to complex interrelationships among blackboard states, potential actions, and strategies, causing difficulty in implementing an efficient system. For these reasons, blackboard applications are sometimes perceived as "slow".

The efficiency of a knowledge-based system can be measured at many levels [Carver and Lesser, 1992]. At one level is the *overhead* of activating, selecting, and executing actions. Another measure is the quality of a system's *control knowledge*, which is used to select the best action to perform. In this paper we address the need to reduce the processing overhead of blackboard systems, independent of the quality of knowledge in the system.

The RETE algorithm [Forgy, 1982], a fast pattern matching mechanism, was a major breakthrough in reducing the overhead of production systems. However, there has been no similar breakthrough in blackboard systems; each blackboard system implements different *ad-hoc* efficiency mechanisms for the basic execution cycle.

Earlier work on efficiency of blackboard systems has been at higher or lower levels of the architecture. Some examples include meta-level frameworks for efficient control [Lâasri and Maître, 1989], blackboard structures to optimize storage and retrieval operations [Corkill et al., 1988], and distributed and parallel implementations of blackboard systems [Lesser and Corkill, 1988; Corkill, 1989; Bisiani and Forin, 1989; Rice et al., 1989].

In this paper we will argue why a direct application of the RETE algorithm is not appropriate for blackboard systems. We then describe a set of general-purpose mechanisms that can be used in many blackboard systems. Experiments based on implementing the enhancements in the BB1 architecture illustrate substantial decreases in execution times. Since most blackboard systems share the same basic execution cycle, with minor variations, our results can be applied to most blackboard architectures.

2 Blackboard architecture

Various blackboard architectures have been implemented, including Hearsay-II [Erman and Lesser, 1975], AGE [Nii and Aiello, 1979], BB1 [Hayes-Roth and Hewett, 1988], and ERASMUS [Baum et al., 1989] (see [Engelmore and Morgan, 1988] for detailed descriptions of these and other blackboard architectures). Most implementations of the blackboard architecture provide knowledge sources for execution, a mixture of frame-based and semantic network representation methods, and a general mechanism for control of reasoning. The knowledge representation component in a blackboard system is the *blackboard*. Blackboards contain *objects*, frame-like structures that form the basic unit of representation. A blackboard object has attributes and links to other objects. A blackboard is usually partitioned into *levels* containing related objects.

In a blackboard system the *knowledge source* is the basic unit of execution, similar to productions or rules in a production system. The action of a knowledge source makes one or more changes to the blackboard such as creating or deleting a blackboard object, changing the value of an attribute, or creating a link between two objects. Each change to the blackboard is logged as an *event*.

Each knowledge source is triggered by an event described in its *trigger conditions*. When the described event occurs, the knowledge source is activated and one or more activations are created as appropriate for the context in which the knowledge source was triggered. Each activation, called a KSA, is then placed on the *agenda*.

The agenda has two parts: the *triggered agenda* and the *executable agenda*. Only KSAs on the executable agenda are eligible for execution. A knowledge source contains state-based *preconditions* that determine whether

the KSA is executable. A KSA's state may change from triggered to executable and back as the state of its preconditions varies in response to changes on the blackboard. A knowledge source also has *obviation conditions.* If these become true while the KSA is on the agenda, it is removed from the agenda and permanently discarded.

The component responsible for selecting KSAs for execution is the *scheduler.* It uses control knowledge to select the best KSA available on each cycle.

2.1 The blackboard execution cycle

```
1. [ACTIVATE.]
   for every Event E of the last cycle do
      for every knowledge source KS do
         if KS.triggerConditions are satisfied by E then
            for every context C of KS do generate a KSA;
2. [ENABLE.]
   for every KSA do
      if KSA.preconditions are satisfied then
         place KSA on the executable agenda
      else
         place KSA on the triggered agenda;
3. [OBVIATE.]
   for every KSA do
      if KSA.obviationConditions are satisfied then
         remove KSA from the agenda;
4. [SCHEDULE.]
   rate KSAs and select a KSA to execute;
5. [EXECUTE.]
   execute a KSA, collecting events;
6. [LOOP.] go to Step 1;
```

At runtime the execution cycle activates and executes knowledge sources using their conditions and actions, as in the BB1 execution cycle shown above.

There are several places where a naive implementation can encounter efficiency problems. Steps 1, 2, 3 and 4 all loop through every existing KS or KSA, of which there can be many. In our experience, agenda management (Steps 1 and 2) involves considerable overhead and often consumes more processing time than execution.

3 Efficiency mechanisms

Common techniques for gaining efficiency in AI architectures include compilation, pattern-matching networks, and demons. In our approach we apply all three techniques to construct efficient general mechanisms for use in blackboard systems.

A demon is a small process that is activated by a specific change to working memory. They have been used previously in blackboard systems. `Poligon` [Rice et al., 1989], a parallel, distributed blackboard system, uses demons to directly invoke rules. However `Poligon` was designed to operate without global control and does not have an agenda *per se.* While `Poligon` uses demons to direct execution, we will use demons for agenda maintenance and rating.

Compilation is the main technique for speeding up execution. As is well known, compiled code executes as much as one hundred times faster than interpreted code, so compiling knowledge sources into lower-level functions will increase execution speed. Additionally, compiling a pattern matching network can improve match times by about 15% [Scales, 1986].

3.1 Why not use RETE?

A RETE network is built by parsing the conditions (LHS) of a set of productions and constructing a network with two parts. The *match* part detects when working memory elements (WME) match a single pattern in a condition. The *join* part detects when the entire condition is satisfied. Each terminal node in the join network corresponds to a production. Whenever a WME is added, deleted, or modified, the WME is passed through the network. If a terminal node is activated, its production is placed in the conflict set. Every production system uses some variation of RETE as an efficient activation mechanism.

Our original intent was to construct a version of RETE for use in blackboard systems. However, we now realize that RETE is not an appropriate activation mechanism for blackboard applications. Figure 1 illustrates the difference in activation for production systems and blackboard systems. In a production system, rules (productions) are activated when several WMEs, denoted by the shaded circles, match the patterns in a rule. These patterns may be scattered throughout working memory. In a blackboard system, initial activation (triggering) is caused by a single event, denoted by the unshaded circle in the figure. Full activation, satisfaction of the knowledge source's preconditions, is usually determined by objects related to the triggering object via links or by virtue of being at the same level on the blackboard. These objects are specified by bindings in the preconditions. Thus, there is usually no need to match patterns throughout the entire blackboard—the objects needed to match the patterns can be directly accessed.

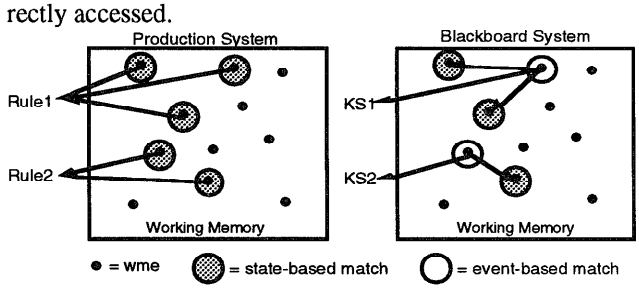

Fig. 1. Different activation mechanisms.

4 Efficient blackboard mechanisms

This section describes how to apply the techniques of the previous section to the major components of blackboard systems. We will describe how to use demons for efficient state-based activation and rating, a discrimination network for efficient event-based activation, and compilation for efficient execution.

4.1 Activation

As described in Section 2, activation in blackboard systems is a two-stage process. The initial *event-based* activation generates activations (KSAs) from knowledge sources. In the second stage, a KSA's *state-based* preconditions must be satisfied for it to be eligible for execution.

4.1.1 Event-based activation

Event-based activation (triggering) involves comparing a number of events against the trigger conditions of each knowledge source on each execution cycle.

In our approach, the trigger conditions are compiled into a discrimination network, much like the match part of a RETE network. We require that trigger conditions bind variables when they are first referenced. This simple restriction eliminates the need for the join part of the RETE network and makes trigger conditions equivalent to a knowledge representation mechanism known as *access paths* [Crawford, 1990]. Access paths provide a well-defined semantics for ordering conjunctive queries so that knowledge base access is contained and controlled, and therefore is more efficient. In our mechanism, the discrimination network uses the event type, event level, attribute and/or link as the match keys at its branch points. The leaf nodes of the network are knowledge sources.

Each cycle the events of the last cycle are passed through the discrimination network. If a knowledge source's node is activated (i.e. an event was accepted by the network), its state-based trigger conditions (if any) are checked and one or more KSAs are generated using the knowledge source's context. The discrimination network improves efficiency by reducing the number of comparisons needed to match events with trigger conditions. For additional speed, our network is compiled into functional form rather than maintained as a data structure.

4.1.2 State-based activation

Like production systems, state-based activation in blackboard systems often consumes much more processing time than knowledge source execution. A typical application has dozens of KSAs on the agenda, each with several preconditions. The task of an efficient architecture is to check a precondition only when its state may have changed. We have determined that a demon-based architecture can provide a very large decrease in activation time while maintaining the generality of the architecture.

In BB1, a KSA is in one of several states: **executable**, **triggered**, or **obviated**. Thus, the Agenda Manager must continually check both preconditions and obviation conditions of many KSAs. The state of each precondition depends on the state of one or more blackboard components (levels, objects, attributes, etc.) which may or may not change state from cycle to cycle. BB1 currently uses some *ad-hoc* mechanisms to improve the efficiency of precondition checking. However, BB1 does not attempt to provide optimal precondition checking and performs very poorly when an agenda contains a large number of executable KSAs [Hewett and Hewett, 1993].

Our implementation uses a demon-based mechanism to indicate which conditions need to be rechecked. A

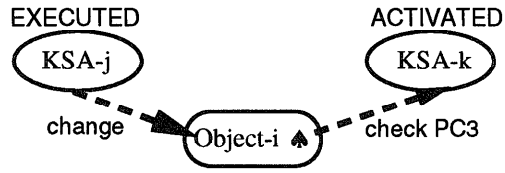

Fig. 2. Demons for state-based activation.

state-based condition must, by definition, reference some item on a blackboard. A condition, then, must be reevaluated when that blackboard item is modified. We place a demon on a blackboard item to note a relationship between the item and a precondition of a KSA, thus providing a way to notify the architecture when a condition must be rechecked. The appropriate location for a demon can be noted by a knowledge source compiler, thus relieving the user of the need to create and place them. A potential disadvantage of this method is the overhead of adding and removing demons as KSAs are created and disposed. However, we will show in Section 5 that demon-based activation produces a large decrease in activation time for every tested application.

The compiler produces demon templates that are instantiated when their corresponding KSAs are instantiated. In the example of Figure 2, when KSA-k is instantiated and one of its local variables is bound to Object-i, the demon ♠ is instantiated and placed on Object-i. When Object-i is modified by the execution of KSA-j, the demon is activated and causes the third precondition of KSA-k to be evaluated. The actual evaluation of the precondition is delayed until the agenda maintenance phase of the execution cycle. The activation signal is also passed to the superior of the demon's location. For example, a change to the value of an attribute is also a change to the object, which is a change to its level, which is a change to the blackboard containing the level. Demons could be activated at any point along the signal's path. Demons are removed whenever a KSA is executed or is obviated. Section 5 shows the time improvements obtained using this mechanism.

4.2 Control

In a blackboard system, control is the process of selecting the next action or actions to execute. In agenda-based systems, control has two parts: rating and scheduling. Rating assigns a priority to each executable KSA. Scheduling selects a KSA and queues it for execution. There are many factors affecting the efficiency of control, including the size of the agenda, the complexity and number of rating functions, and the frequency with which actions must be re-rated. See [Carver and Lesser, 1992] for more details.

4.2.1 Scheduling

The scheduler selects an action from the agenda and queues it for execution. Usually the selected action is the highest-priority action, but with a flexible control module, the actual criteria for selection are user-definable. For

most situations, a sorted agenda would seem to be an appropriate data structure.

However, when we implemented a sorted agenda in BB1, we found that the execution time of the system *increased* by 10% even though the time to retrieve the highest-priority item was significantly reduced. This is because we need not sort *all* of the actions. In most systems only the highest-priority action needs to be identified. The unnecessary sorting of lower-priority actions is simply a waste of time. A simple linear search on an unsorted agenda provides suitable performance for a scheduler.

4.2.2 The BB1 rating mechanism

```
RATE-ALL-KSAs (Operative, Dynamic, Changed)
for every KSA in KSAs-TO-RATE do
    if KSA is newly-executable
        for every R in Operative-criteria do Rate(KSA, R);
    else
        for every R in Rating-criteria do rate(KSA, R);
        for every R in Deactivated-criteria do
            Remove-rating(KSA, R);
    end-if
    Prioritize(KSA)
end-for
```

In BB1, the Rater applies rating functions from active control elements in the control plan to executable KSAs. Control elements can be dynamic or static. A dynamic control element is one whose rating criterion is state-dependent, potentially causing its ratings to change each cycle. During the control phase each control element in the current control plan rates every new KSA, and each dynamic control element re-rates every executable KSA. To rate each executable KSA, the Rater identifies the operative, dynamic, new, and changed control elements. It uses them as shown in the algorithm above.

The Rater frequently rates new executable KSAs, rates KSAs when there is a new control element, re-rates KSAs when a control element is modified, and removes ratings from existing KSAs when a control element is deactivated.

Additionally, the rater re-rates every KSA by dynamic control elements every cycle. There is potentially a lot of redundant computation in rating, especially when dynamic control is used. Our goal is to reduce the number of KSAs that are repeatedly re-rated unnecessarily by using demons to relate control elements to specific KSAs.

4.2.3 Demon-based rating

Similar to the way demons can be used to associate blackboard items with state-based preconditions, we also associate blackboard items with control elements used in rating KSAs. A well-designed control element focuses on a certain part of the solution state. If the relevant part of the solution state changes, KSAs that are related to that part of the solution state will need to be re-rated. These KSAs can be located by following activation demons (♠) from the objects in the solution state. The control elements that need to re-rate these KSAs can be located by following control demons (♥) from the same objects. The Rater is then notified that certain KSAs need to be rated

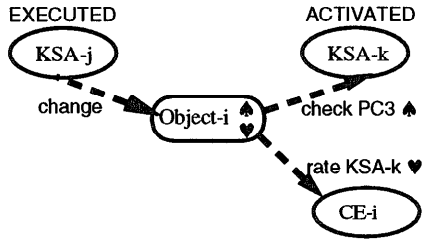

Fig. 3. Using control demons for rating.

by certain control elements.

In the example of Figure 3, Object-i has an activation demon to KSA-k and a control demon to dynamic control element CE-i. If Object-i is modified, the control demon tells CE-i to re-rate KSA-k.

During the rating phase of the execution cycle, the Rater processes activated control demons, much like the demon-based agenda mechanism described above. Rating occurs in the following situations:

1. **New KSA**: When placing new agenda demons, check for any control demons in the same location. If they exist, activate them to rate the new KSA.
2. **New CE**: When placing control demons, check for any agenda demons in the same location. If they exist, activate the new control demons to rate the existing KSAs.
3. **CE deleted**: Remove any associated control demons while checking for agenda demons in the same location. If they exist, activate the control demon one last time to unrate the existing KSAs.
4. **CE modified**: If a CE is modified so that its rating criterion or weight has changed, it will be necessary to re-rate any associated KSAs. This is handled by combining operations 2 and 3 above.
5. **BB modified**: If a blackboard object changes, activate its associated control demons.

Notice that the demon-based rating mechanism depends on the existence of the demon-based activation mechanism. Since the two mechanisms are closely related, the actual implementation of the rating mechanism was very simple. However, while it is fairly easy to write knowledge sources in such a way that the compiler can determine where to place activation demons, it is not as easy to construct "knowledge source independent" control knowledge. The appropriate methods of structuring control knowledge to ensure that it can operate in this manner require further research.

4.3 Execution

We improve the efficiency of executing the actions of a knowledge source by the simple expedient of compiling the actions. The compiler also inserts code into the actions to place and remove activation and control demons. The speed increase from compilation is related to the complexity of the actions. Our test applications, which have fairly trivial actions, do not show a large increase in execution speed. Since demon activation and removal occurs during execution, it is possible for the overhead of these operations to increase the execution time of some systems. In our test systems, the overhead was noticeable, but was

negligible compared to the decrease in activation time.

4.4 Knowledge source compilation

The mechanisms described in the sections above all refer to a knowledge source compiler that produces the discrimination network for triggering, produces the activation demons and control demons, and compiles the actions and conditions of the knowledge sources. In this section we provide an overview of the compiler.

The compiler requires that the knowledge sources be written in a simple blackboard language described in [Hewett and Hewett, 1993]. Some examples of sentences in this language are:

```
1. ADD <object-name> TO <level-name>
     [WITH [ATTRIBUTES <attribute-value-list>]
     [LINKS <link-object-list>]]
2. DELETE <object-name>
3. CHANGE <attribute-name> OF <object-name>
     TO <new-value>
4. LINK <link-name> FROM <object1-name>
     TO <object2-name>
```

The knowledge source compiler processes knowledge sources and compiles their conditions and actions into a lower-level language (e.g. LISP, C or C++). Through an analysis of the conditions, it generates a set of activation demons for each knowledge source. Additional code is added to each compiled KS to instantiate and remove demons at the appropriate times.

At runtime, each KSA has a set of local variable bindings that differentiate it from other instances of the same knowledge source. The conditions and actions of each knowledge source are compiled into functions that accept the set of variable bindings for the particular KSA being evaluated.

5 Experimental results

We used three applications to study the effects of the new activation mechanism in BB1. The first, TSP, is a heuristic solution of a 10-city traveling salesman problem that has previously been used to benchmark BB1. The other two applications, TEST-T and TEST-E, are designed to test the effect of the enhancements on different agenda characteristics. Both applications always have twenty active KSAs. TEST-T has nineteen KSAs on the triggered agenda and one KSA on the executable agenda. TEST-E keeps all twenty KSAs on the executable agenda every cycle. TSP uses a small amount of control knowledge, while the other two applications use no control. Figure 4 summarizes the characteristics of the benchmark applications.

program	exec. agenda	trig. agenda	control know.	number of demons
TSP	*large*	*small*	*small*	*medium*
TEST-T	*small*	*large*	*-none-*	*large*
TEST-E	*large*	*small*	*-none-*	*large*

Fig 4. Characteristics of benchmark programs.

We measured the time spent in the agenda maintenance, control, and execution phases of the execution cycle as well as the total time of run. Included in the execution time is the cost of demon activation and the overhead of instantiating and removing demons. All timed runs were made on a single-user Sun IPC running Lucid Common LISP and BB1 v2.5. The times labeled *eff-1* show the improvement gained by implementing only the efficient activation mechanism. The times labeled *eff-2* include both the efficient activation and the efficient control mechanisms. Figure 5 shows the run times of all three applications.

5.1 Results for TSP

Figure 5a shows the runtime of TSP in standard BB1 and in new implementations using the efficiency mechanisms described in this paper. TSP runs for 17 execution cycles, so the overall runtime is relatively short. Because TSP maintains a large executable agenda and has a lot of movement to and from different parts of the agenda, the large increase in agenda maintenance speed using the demon-based architecture is not a surprise. Notice the very slight increase in execution time when the demon-based control mechanism is used. This is the effect of the demon activation overhead. Overall, the performance gain is approximately 55%.

5.2 Results for TEST-T and TEST-E

Figure 5b shows the runtime of the TEST-T applica-

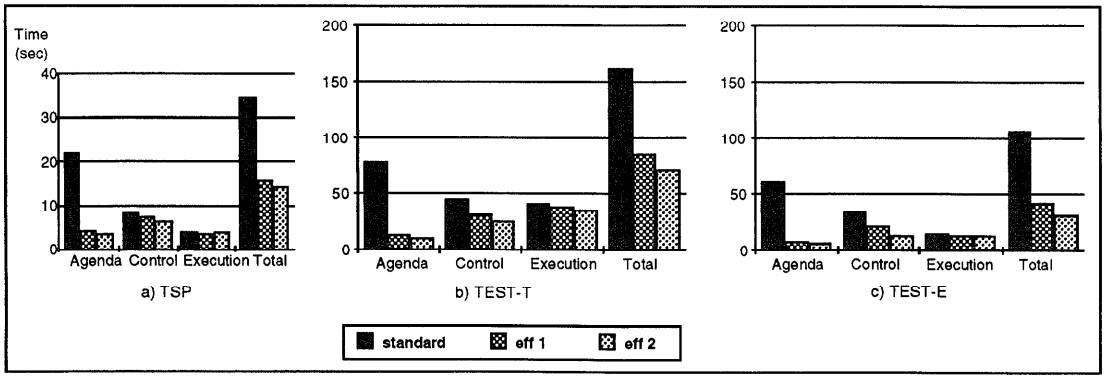

Fig 5. Run times for the test applications.

tion in standard BB1 and the new implementations. The standard BB1 handles TEST-T relatively well, so we should expect our new mechanism to have a relatively smaller impact on the performance of TEST-T than on the performance of the other applications.

Despite this, the performance increase for TEST-T is quite large. Agenda maintenance times are reduced by 80% and the overall runtime is reduced by about 60%. Notice that in standard BB1 agenda maintenance consumes a much larger amount of time than the control and execution phases, while in the new implementation, it consumes much less time.

Figure 5c compares the runtime of the TEST-E application in standard BB1 and in the new implementations. As expected, the results for TEST-E are similar to those of TEST-T, with as good or better improvement in agenda maintenance time.

5.3 Summary of results

Overall, the results show a substantial decrease in agenda maintenance time and significant reductions in other phases. Applications with more complex knowledge source actions will show a larger decrease in execution time, while those with a large amount of complex, dynamic control will show a larger decrease in control time. The overhead of demon processing is not substantial.

A final example demonstrates that the results are consistent for larger applications. As described above, TEST-E maintains an executable agenda of 20 KSAs. We ran TEST-E in modes using 20, 40, and 80 KSAs, producing the total run times shown in Figure 6. The total runtime of the new implementation managing eighty KSAs is approximately two-thirds of the total runtime of the old implementation managing only twenty KSAs.

6 Summary

We have illustrated why a RETE-like pattern-matching network is not suitable for blackboard systems. As an alternative, we presented efficient blackboard activation, execution and rating mechanisms. Our mechanisms combine compilation techniques, a matching network, and demon-based activation and rating to achieve a substantial decreases in runtime for all tested systems.

We have demonstrated that, like production systems, activation in blackboard systems can be a major efficiency problem. Our efficiency mechanisms address this and reduce the time spent in activation to approximately 15% of the total runtime. Further work includes finding suitable formulations of control knowledge that can reap the benefits of the demon-based rating mechanism.

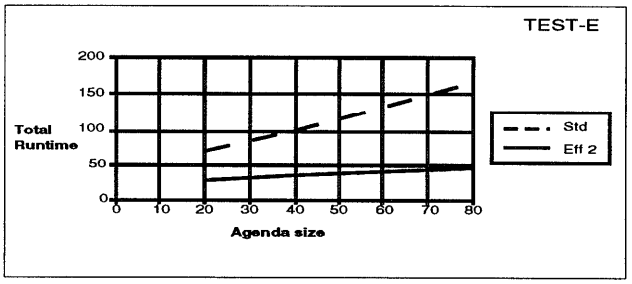

Fig. 6. TEST-E with different agenda sizes.

References

Baum, L.S., Dodhiawala, R.T. and Jagannathan, V. (1989) "The Erasmus System." *Blackboard Architectures and Applications*, Jagannathan, V., R. Dodhiawala and L. S. Baum, editors, pp. 347-370, Academic Press.

Bisiani, R. and Forin, A. (1989) "Parallelization of Blackboard Architectures and the Agora System." *Blackboard Architectures and Applications*, Jagannathan, V., R. Dodhiawala and L. S. Baum, editors, pp. 137-152, Academic Press.

Carver, N. and V. Lesser (1992). *The Evolution of Blackboard Control Architectures.* CMPSCI Technical Report 92-71, University of Massachusetts at Amherst.

Corkill, D.D. (1989) "Design Alternatives for Parallel and Distributed Blackboard Systems." *Blackboard Architectures and Applications*, Jagannathan, V., R. Dodhiawala and L. S. Baum, editors, pp. 99-136, Academic Press.

Corkill, D.D., Gallagher, K.Q. and Murray, K.E. (1988) "GBB: A Generic Blackboard Development System." *Blackboard Systems*, Engelmore, R., and T. Morgan, editors, pp. 503-517, Addison-Wesley.

Crawford, J. (1990). *Access-Limited Logic -- A Language for Knowledge Representation*. PhD Thesis, University of Texas at Austin, Technical Report AI90-141.

Engelmore, R. and Morgan, T., editors (1988). *Blackboard Systems*. Addison-Wesley.

Erman, L.D. and Lesser, V.R. (1975) A multi-level organization for problem-solving using many diverse cooperating sources of knowledge. In: *Proceedings of the Fourth International Joint Conference on Artificial Intelligence* (IJCAI-75), pp. 483-90.

Forgy, C. (1982) RETE: a fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence* **19**(1):17-37.

Hayes-Roth, B. and Hewett, M. (1988) "BB1: An Implementation of the Blackboard Control Architecture." *Blackboard Systems*, Engelmore, R., and T. Morgan, editors, pp. 297-313, Addison-Wesley.

Hewett, M. and Hewett, R. (1993) A Language and Architecture for Efficient Blackboard Systems. In: *Proceedings of the Ninth International IEEE Conference on AI Applications* (CAIA '93), Orlando, Florida.

Lâasri, H. and Maître, B. (1989) "Flexibility and Efficiency in Blackboard Systems: Studies and Achievements in ATOME." *Blackboard Architectures and Applications*, Jagannathan, V., R. Dodhiawala and L. S. Baum, editors, pp. 309-322, Academic Press.

Lesser, V.R. and Corkill, D.D. (1988) "The Distributed Vehicle Monitoring Testbed." *Blackboard Systems*, Engelmore, R., and T. Morgan, editors, pp. 353-386, Addison-Wesley.

Nii, H.P. and Aiello, N. (1979) AGE: A Knowledge-Based Program for Building Knowledge-Based Programs. In: *Proceedings of the Sixth International Joint Conference on Artificial Intelligence* (IJCAI-79), pp. 645-655.

Rice, J., Aiello, N., and Nii, H.P. (1989) "See How They Run...." *Blackboard Architectures and Applications*, Jagannathan, V., R. Dodhiawala and L. S. Baum, editors, pp. 153-178, Academic Press.

Scales, D. J. (1986) *Efficient Matching Algorithms for the SOAR/OPS5 Production System*, Technical Report STAN-CS-86-1124, Stanford University, Stanford, California.