Efficiency in Pure Blackboard Systems

Rattikorn Hewett
hewett@cse.fau.edu
Computer Science and Engineering Department
Florida Atlantic University, Boca Raton, FL 33431-0991

Micheal Hewett
mhwett@cse.fau.edu

Abstract
The blackboard model is a versatile framework for implementing expert systems. In recent years the speed of blackboard access has been improved but the execution cycle, which manages knowledge source activation and execution, still needs substantial improvement. This paper presents a method of condensing knowledge sources and shows how to efficiently activate and execute a class of blackboard application systems using the condensed representation. We show that the time complexity of the execution cycle of a condensed system is faster than the conventional approach by the ratio of the time required for blackboard retrievals to the time required for vector element retrievals. In practice, this ratio is approximately four orders of magnitude.

1 The Problem
Efficiency is a concern for almost every computer system, including complex AI systems. The blackboard model is a versatile framework for implementing expert systems, providing great flexibility in knowledge representation and problem solving control. As blackboard systems are used for more AI applications, especially real-time applications, their efficiency becomes ever more important.

Every operation in a blackboard system reduces to a storage on or retrieval from the blackboard. Thus, the most obvious place to increase efficiency is at the lowest level—the blackboard mechanism itself. In the GBB blackboard design [4], the user specifies the structure of the blackboard in detail. GBB can then pre-compile access routines for blackboard objects. Pre-compilation greatly increases the speed of atomic blackboard operations. Results show that GBB-style blackboard operations are sufficiently fast for most current blackboard-based expert systems [3].

Several efforts to increase efficiency have focused at a higher level. For example, [13] have developed meta-level frameworks for efficient control. Also, blackboard implementations by [14], [3], [2], and [16] have explored distributed and parallel implementations of blackboard systems. These approaches apply concurrency to blackboard access, agenda maintenance, scheduling, and/or knowledge source execution. In general, these approaches have been encouraging and should yield significant increases in execution speed when the underlying hardware and software environments mature.

We choose to focus at an intermediate level: the execution cycle of sequential blackboard systems. The execution cycle has three main duties: agenda maintenance (including activation), scheduling, and knowledge source execution. In production systems, the RETE network [7] provides an efficient activation mechanism. We are not aware of previous work on a similar mechanism for blackboard systems. In our experience, agenda maintenance and scheduling often consume much more processing time than knowledge source execution. For example, in a blackboard shell we are familiar with, agenda maintenance becomes relatively slow when the agenda contains more than ten items. Since most sequential blackboard systems and shells use a similar agenda and execution mechanism, we consider the question: can the basic execution cycle in blackboard systems be made more efficient?

In this paper we present a method for reducing knowledge sources to a condensed form and show how to efficiently activate and execute the condensed form. In the condensed form, relatively complex blackboard references are replaced by simple vector references, substantially reducing processing time. Our method is applicable to a class of blackboard applications called pure blackboard systems, where all blackboard references are bound at compile time. An extended result applicable to all blackboard systems is given in [12]. The approach described in this paper derives from the BB1 architecture [9]. Although our approach is based on one architecture, it can easily be adapted to other blackboard systems ([6], [15], [1] and [5]) because they all share the same basic execution cycle, with minor variations. BB1 and similar systems will be referred to as conventional blackboard systems. We will now describe our terminology.
2 Condensed representation

A conventional blackboard system has four main components: blackboards, knowledge sources, an agenda, and a scheduler. Blackboards are global data areas that contain objects, the basic unit of representation. Blackboard objects represent domain knowledge, control knowledge, and operational constructs such as knowledge sources. A blackboard is usually partitioned into levels containing related objects. An object may have links to other objects, and attributes with corresponding values. An application may have multiple blackboards, each containing multiple levels, with multiple objects at each level.

1. [ACTIVATE.]
   for every knowledge source KS
   for every Event E of the last cycle
   if KS.triggerConditions are satisfied by E then
   for every context C of KS
   generate a KSI of KS;

2. [ENABLE.]
   for every triggered KSI
   if KSI.preconditions are satisfied
   then make KSI executable;

3. [OBLVIA].
   for every executable KSI
   if KSI.obviationConditions are satisfied
   then discard KSI;

4. [SCHEDULE.]
   For every executable KSI
   assign a rating to KSI based on the current control plan;
   Using the current scheduling rule, select a KSI to execute;

5. [EXECUTE.]
   Execute the scheduled KSI;

6. [LOOP.] Go to Step 1;

Fig. 2.1. The execution cycle in BB1.

A knowledge source is the basic unit of execution. It is described by the following fields:

- **Trigger condition**: Describes blackboard events that activate it.
- **Context**: Describes contexts in which the triggered knowledge source should be instantiated. Instantiation creates a KSI.
- **Precondition**: Describes blackboard states that make the KSI executable.
- **Obviation condition**: Describes blackboard states in which the KSI should be obviated (discarded from the agenda).
- **Action**: Describes changes to the blackboards that the KSI will make when executed. The changes can include creating or deleting blackboard objects, creating links between objects, and changing the values of attributes on objects.

The system routine that manages execution is called the *execution cycle*. At runtime the execution cycle uses the conditions and actions to activate and execute the knowledge sources, as in the BB1 execution cycle shown in Figure 2.1.

2.1 Pure blackboard systems

We define a *pure blackboard system* to have the same blackboard representation and reasoning mechanisms as conventional systems, with the following assumptions:

1. All links are unidirectional.
2. The set of attribute names of any blackboard object can not overlap its set of link names.
3. Knowledge source conditions and actions must reference blackboard objects directly by name. In other words, there can be no anonymous references [4]. This precludes pure knowledge sources from containing local variables, including those normally used in contexts.

The first two assumptions can be made without loss of generality. They are to simplify implementation of the algorithms described in Section 3. The third restriction is the primary difference between pure systems and conventional systems, and is the foundation of the increased efficiency provided by the method described in this paper.

Fig. 2.2. Classifications of Blackboard Applications.

Blackboard applications can be deterministic or non-deterministic. In addition, they can be pure systems or non-pure systems. Figure 2.2 shows the classifications of

---

1. Here, deterministic means a fine-grained determinism. That is, the sequence of low-level actions must be invariant over different runs of the system.

66
blackboard applications. If a sequence of low-level actions at run time of a pure blackboard application is identical from run to run then the application system is deterministic. Otherwise it is non-deterministic. On the other hand, a deterministic system might or might not have anonymous references. However, determinism allows one to trace the execution and replace all local variables by the values obtained at runtime. Hence, any deterministic blackboard application using anonymous references can be converted into a pure blackboard system. Thus, we obtain Proposition 1 and the corresponding fact that any deterministic blackboard application is pure, as illustrated in Figure 2.2.

**Proposition 1.** The class of deterministic blackboard application systems is a subset of a class of pure blackboard systems.

While eliminating contexts and local variables in pure systems may appear to be too restrictive, we have merely eliminated a useful development technique, not a required feature. When an application is finished or when development flexibility can be waived, many applications, including all deterministic systems as described above, can be transformed into pure blackboard systems. During the transformation, conventional knowledge sources that activate in different contexts and whose local variables are assigned different values in different instantiations are mapped into multiple pure knowledge sources.

Knowledge sources in a pure blackboard system are called pure knowledge sources. They are like conventional knowledge sources, except that they have neither contexts nor local variables. As in a conventional blackboard system, the knowledge source’s actions can create new objects, delete existing objects, assign new values to an attribute, or link one object to another.

In this paper we restrict the syntax of condition and action clauses in pure knowledge sources to the language shown below. This is a minimal language that provides the functionality required by most blackboard systems. It is not intended to be the entire set possible in pure systems. This language is defined in more detail in [12].

**ACTIONS**

1. ADD <object-name> TO <level-name>
   [WITH
   [ATTRIBUTES <attribute-value-list>]
   [LINKS <link-object-list>]
   ]
2. DELETE <object-name> FROM <level-name>
3. CHANGE <attribute-name> OF <object-name>
   TO <new-value>
4. LINK <link-name> OF <object-name>
   TO <object2-name>
5. UNLINK <link-name> OF <object1-name>
   TO <object2-name>
6. BIND <variable> <value>
7. EVALUATE <expr>

**EVENT-BASED CONDITIONS**

1. ADDED <object-name> TO <level-name>
2. DELETED <object-name> FROM <level-name>
3. CHANGED <attribute-name> OF <object-name>
   [AT <level-name>]
4. LINKED <link-name> OF <object-name>
   [AT <level-name>]
5. UNLINKED <link-name> OF <object-name>
   [AT <level-name>]

**STATE-BASED CONDITIONS**

1. [NOT] OBJECT-STATE
2. [NOT] LEVEL-STATE <level-name>
   [ATTRIBUTES <variable>]
   [LEVELS <variable>]
   [BLACKBOARD <variable>]
3. [NOT] BLACKBOARD-STATE <blackboard-name>
   [LEVELS <variable>]
   [NUMBER-OF-LEVELS <variable>]
4. BIND <variable> <value>

For a precondition or obviating condition, rel can be any typical relation such as =, \#, <, >, \geq, or \leq. The conditions and actions of knowledge sources are represented by the conjunction of zero or more clauses.

### 2.2 The ESA vector

An Event/State/Action (ESA) vector is a vector representing potential blackboard events, states, and actions of a set of knowledge sources. Each element of an ESA vector is a triple of the form (t, l, v) for some type t, label l and value v. Figure 2.3 illustrates some possible types and corresponding labels and values.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>LABEL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>level-name</td>
<td>object-name or T or $</td>
</tr>
<tr>
<td>DELETE</td>
<td>level-name</td>
<td>object-name or T or $</td>
</tr>
<tr>
<td>CHANGE</td>
<td>level-name</td>
<td>object-name or T or $</td>
</tr>
<tr>
<td>LINK</td>
<td>level-name</td>
<td>object-name or T or $</td>
</tr>
</tbody>
</table>

**Fig. 2.3.** Entries in the ESA vector.

**Fig. 2.4.** Two representations of an ESA vector.

Since, as described in Section 3, the type and label do not change at runtime, we often display an ESA vector using just the value of each element. As an example, Figure 2.4 shows both methods of displaying an example ESA vector X.
3 Condensed knowledge sources

The blackboard reference vector $\mathbf{R}$ is defined to be an ESA vector with entries representing every unique reference to a level name, attribute name, or link name in a system. The contents of $\mathbf{R}$ are determined by statically analyzing the knowledge sources prior to runtime.

1) analyze-fever
   TC: ADDDED Fever TO P.s
   PC: OBJECT-STATE (= P.s.Fever.St NULL)
   OC: <none>
   A: CHANGE St OF P.s.Fever TO present

2) hypothesize-flu
   TC: CHANGED St OF P.s.Fever
   PC: OBJECT-STATE (= P.s.Fever.St present)
       OBJECT-STATE (= P.s.Headache.St present)
       OBJECT-STATE (= P.s.Stomachache.St present)
   OC: OBJECT-STATE (= P.s.Leg-swelling.St present)
   A: CHANGE Hypothesis OF P.Diag.Virus TO flu

Fig. 3.1. Two pure knowledge sources.

<table>
<thead>
<tr>
<th>INDEX</th>
<th>TYPE</th>
<th>LABEL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ADD</td>
<td>P.s</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CHANGE</td>
<td>P.s.Fever.St</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>CHANGE</td>
<td>P.s.Headache.St</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>CHANGE</td>
<td>P.s.Stomachache.St</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>CHANGE</td>
<td>P.s.Leg-swelling.St</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>CHANGE</td>
<td>P.Diag.Virus.Hypothesis</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 3.2. An $\mathbf{R}$ vector.

In the condensed form, each field of a knowledge source is replaced by a single ESA vector. Each field’s vector is based on the $\mathbf{R}$ vector for the set of knowledge sources, so each vector has the same size and the same type and label components, but different values. Figure 3.1 shows two example pure knowledge sources, each containing four fields: Trigger Conditions (TC), Preconditions (PC), Obviation Conditions (OC) and Action (A). The $\mathbf{R}$ vector of the two knowledge sources shown in Figure 3.2 contains six elements representing the one level reference and five attribute references present in the knowledge sources. In Figures 3.1 and 3.2, the symbols P, s and St refer to patient, symptom and status, respectively. Entry 1 is from the trigger condition of analyze-fever and Entry 2 is from its precondition. Entry 2 also represents the action of analyze-fever as well as the trigger condition and part of the precondition of hypothesize-flu. Entries 3 through 5 are from the precondition of hypothesize-flu and Entry 6 is from its action.

Fig. 3.3. Condensed form of the pure KSes.

Figure 3.3 shows the condensed forms of the two sample knowledge sources. The knowledge sources can be condensed by a two pass algorithm. Pass 1 scans the knowledge sources to determine the contents of the $\mathbf{R}$ vector. Pass 2 creates condensed knowledge sources by storing four copies of the $\mathbf{R}$ vector in the four fields of each condensed knowledge source. It then specializes the four field vectors of each knowledge source by scanning the knowledge sources again. For more details see [10].

3.1 Activating condensed KSes

We now use the condensed representation of knowledge sources to speed up the ACTIVATION step of the execution cycle shown in Figure 2.1. Since we know the values of all actions and trigger conditions, we can determine before runtime which knowledge sources will be activated when any given knowledge source is executed. We can build an activation graph by comparing the action of each knowledge source to the trigger condition of every other knowledge source and see whether the action matches the trigger condition. All knowledge sources whose trigger conditions match the action become children of the action’s knowledge source in the activation graph. Figure 3.4a illustrates an example of an activation graph where the action of KS1 creates an event that triggers KS2 and KS3. Similarly, the action of KS2 triggers KS5 and the
Figure 3.3. The action of KS3 triggers KS4 and KS5 whose action in turn triggers KS4. Figure 3.4b shows an activation graph produced from the condensed knowledge sources obtained in Figure 3.3. The action of analyze-fever creates an event noting a change to the status of P. a Fevers which matches the trigger condition of hypothesize-flu.

To use the activation graph we revise Step 1 of the execution cycle from Figure 2.1 and add an initialization step, as shown in Figure 3.5. Section 4.1.2 describes how the modified execution cycle using activation graph improves efficiency of activation in conventional blackboard systems like BB1.

0. [INITIALIZE.] for every knowledge source KS condense-KS(KS);
AG := build-activation-graph;

1. [ACTIVATE.] triggered-KSes := AG.children(last_KSI)

Fig. 3.5. Modified execution cycle.

3.2 Executing condensed KSes

When dealing with condensed knowledge sources, we can express the basic execution cycle more formally and succinctly in terms of set and vector operations. For example, in the conventional execution cycle, the operation of checking a precondition involves executing some code (usually via the LISP interpreter) to perform a blackboard retrieval and then evaluating a predicate. In the condensed version, the same step involves comparing a condensed knowledge source's precondition ESA vector against the global blackboard state vector S. As we show in Section 4, this results in a tremendous reduction in execution time.

We use two vector operations in the revised execution cycle. The first is called match and is represented by the symbol &. Match is a pairwise comparison of the value components of the two vectors; it returns TRUE if they all match (see details in 111). The second is the overlay operation and is represented by the symbol ⊕. It is used to merge the contents of two vectors, as when the actions of a knowledge source are written into the global blackboard state vector S. The overlay operation takes two ESA vectors as arguments and returns a new ESA vector. Every non-∅ element of the first ESA vector is copied to the corresponding element of the second ESA vector. The two vector operations described here can be executed in linear time on a standard computer and in constant time on a vector-based computer.

3.2.1 Algorithm E1: We will now present an algorithm to efficiently execute the condensed pure system.

At runtime we maintain two global ESA vectors, E and S, representing the events of the last execution cycle and the current state of the blackboard, respectively. We also maintain two sets, T and X, representing the set of triggered (but not yet executable) and executable knowledge sources. Figure 3.6 shows Algorithm E1, the execution cycle for condensed knowledge sources. In the algorithm, Steps 3, 4, 5 and 6 use the condensed form to increase their efficiency.

To start the system, at least one knowledge source must have no trigger conditions, i.e. a condensed trigger vector of (∅, ∅, ∅, ..., ∅). However, there are alternative ways to start the system. For example, the user could declare an initial value for KSX and the execution cycle could start at Step 6.

1. [INITIALIZE.] E := {} S := {} T := {} X := {} (KSX.TC ⊕ E) = TRUE

3. [ENABLE.] X := X ⊕ (KSX.TC ⊕ E) = TRUE

5. [SCHEDULE.] --implementation-dependent KSX := schedule-ks(X) X := X ⊕ KSX


Fig. 3.6. Algorithm E1: executing condensed KSes.

3.2.2 Algorithm E2: Using the activation graph described in Section 3.1, algorithm E2 improves Step 2 of Algorithm E1. By building the activation graph at initialization, modifying Step 2 to retrieve triggered knowledge sources from the activation graph, and eliminating the E vector from Step 6, as shown in Figure 3.7.

Analogous to the activation graph, it may be possible to construct an execution graph by computing the transitive closure (or similar transform) of the activation graph. If an execution graph were available, one could optimize steps 3 and 4 of the execution cycle. This work is still in progress at the time of this report.
(Steps 3, 4, 5 and 7 are the same as in Algorithm E1)

1. [INITIALIZE.]
   \[ S := \{\phi, \phi, \phi, \ldots, \phi\} \]
   \[ T := \{\} \]
   \[ X := \{\} \]
   \[ AG := \text{build-activation-graph(KSes)}; \]
   \[ KS_X := \text{root}(AG); \]

2. [ACTIVATE.]
   \[ T := T \cup X \cup AG.\text{activations}(KS_X) \]

3. [EXECUTE.]
   \[ S := S \oplus KS_X.\text{Actions} \]

Fig. 3.7. Algorithm E2: Condensed execution with efficient activation.

The remaining way to increase the efficiency of execution is to optimize Step 5, the scheduling step, of Algorithms E1 and E2. BB1 [9] and many of its derivatives use a scheduler that ranks executable actions by means of rating functions stored on a control blackboard. We believe that a condensed form of this mechanism can be implemented and integrated with the above reasoning cycle, but our work in this area is still in the preliminary stages.

4 Analysis and commentary

To understand how our approach improves the efficiency of activation and execution in a blackboard system, we compare the time complexity of our approach and the conventional execution method.

4.1 Time analysis

<table>
<thead>
<tr>
<th>Variable</th>
<th>Represents</th>
<th>Typical/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)</td>
<td>The number of knowledge sources</td>
<td>20</td>
</tr>
<tr>
<td>(l)</td>
<td>The number of fields in each knowledge source</td>
<td>4</td>
</tr>
<tr>
<td>(k)</td>
<td>The average number of conditions per field</td>
<td>2</td>
</tr>
<tr>
<td>(L)</td>
<td>The length of the ESA vector for a system</td>
<td>150</td>
</tr>
<tr>
<td>(b)</td>
<td>The average time to retrieve a blackboard object</td>
<td>(3 \times 10^4) sec</td>
</tr>
<tr>
<td>(v)</td>
<td>The average time to retrieve a vector element</td>
<td>(3 \times 10^4) sec</td>
</tr>
<tr>
<td>(C)</td>
<td>The number of cycles a system runs</td>
<td>200</td>
</tr>
<tr>
<td>(s)</td>
<td>The scheduling time per cycle</td>
<td>various</td>
</tr>
</tbody>
</table>

Fig. 4.1. Variables for time analysis.

One important distinction between that analysis of the two approaches is that a comparison operation in our system is an operation on vector elements whereas in the conventional system it involves a blackboard retrieval. This makes the comparison operation in our system, in general, much more efficient than the conventional one even though the time complexities of both systems are of the same order. The variables used in our analysis are shown in Figure 4.1.

4.1.1 Condensing KSes: In condensing knowledge sources, the first step is to determine the contents of the \(R\) vector. Each field of each condition of each knowledge source is parsed and the corresponding triple is inserted into the \(R\) vector if it is not already there. By using a hash table implementation for the ESA vector, the insertion takes a constant time and the time complexity of this step is of order \(O(nk)\). The second step is to create a condensed form of each knowledge source. First, copy the \(R\) vector obtained previously into each field (i.e. TC, PC, OC and A) of each condensed knowledge source; this takes time of order \(O(fL)\). Then store the value of each parsed condition or action into the corresponding field vector, which takes time of order \(O(nfk)\). As a consequence, the time complexity of condensing is of order \(O(fnk + L)\). The condensing algorithm and a more detailed analysis can be found in [10].

4.1.2 Activation: The complexity of constructing an activation graph is of order \(O(n^2)\) and retrieving the children of a given node in an activation graph takes constant time. Thus, the time complexity of Step 0 [INITIALIZATION] in our approach is of order \(O(fnk + L + n^2)\).

An activation step in BB1 checks every knowledge source against every event satisfying the trigger conditions and generate a KS1 for every context. Thus for a system with \(n\) knowledge sources, \(C\) execution cycles, and, on the average, \(k\) conditions for each context and trigger condition, the activation time is approximately \(Cbnk^2\). Compared to the time spent in activation in BB1, activation in our approach takes only constant time, and is thus more efficient than BB1 activation. Furthermore, it must be stressed that the time unit of complexity in BB1 is a blackboard access, while in our system it is a vector access. As a result, all of our operations are inherently faster by a large constant factor (at least four orders of magnitude, compared to results in [8]). Thus, even though the initialization step might seem to take more time than BB1, the actual cost is quite small compared to BB1 because of the above factor and the fact that it only needs to be computed once while BB1's cost increases proportionally to \(C\).

4.1.3 Execution: After condensing the knowledge sources, we apply the execution algorithm. The execution algorithm operates on instances of ESA vectors representing different fields in the condensed knowledge sources. Section 3.2 defined two operations on the ESA vector: match and overlay. The time complexity of both operations is of order \(O(p)\) where \(p\) is the maximum size of the vectors being operated on. We will now analyze the
execution algorithms presented in Sections 2 and 3.2.

4.1.3.1 Algorithm E1: In this algorithm, Step 1 takes a constant amount of time. The sets of triggered and executable knowledge sources, T and X, can be implemented efficiently by using bit vectors. Thus, in each execution cycle, the time complexity of maintaining T and X in Steps 2-4 is of order \(O(nL)\). Let the scheduling in Step 5 be of order \(O(n)\). Updating the \(S\) vector in Step 6 is of order \(O(L)\). Thus, for each cycle, the time complexity of the execution algorithm E1 is of order \(O(nL + s)\).

4.1.3.2 Algorithm E2: E2 is analyzed similarly to E1. The computational complexity of the construction of an activation graph is of order \(O(n^2)\) and retrieving the children of a given node in a given graph takes constant time. Thus, Step 1 and Step 2 are of order \(O(n^2)\) and \(O(n)\), respectively. The rest of the steps have the same complexity as analyzed in Section 4.1.1. The computation time for each cycle is of order \(O(nL + s)\) and the overall time complexity of E2 is of order \(O(n^2)\).

4.1.3.3 Execution of conventional systems: At runtime the execution cycle uses the conditions and actions to activate and execute the knowledge sources, as in the BB1 execution cycle shown in Figure 2.1. Consider the time spent on execution for each execution cycle in BB1. The time required for Steps 2 and 3 in Figure 2.1 is approximately \(2nk_b\) where \(b\) is the average time required for a blackboard operation.

The time complexity for scheduling (Step 4) and execution (Step 5) are of order \(O(s)\) and \(O(1)\), respectively. Thus, the total execution time required for BB1 is approximately \((2nk + as + l)bC\), where \(C\) is the number of execution cycles and for some constant \(a\). In Section 4.1.2, we showed that the time required for activation in BB1 is approximately \(Cbnk^2\). Next we will compare the results of our analyses.

4.2 Comparison of execution times

We claim the following:

1. Using the activation graph in a pure system provides more efficient execution than a condensed system without the activation graph.
2. Condensing a pure blackboard system (whether using the activation graph or not) provides more efficient execution than conventional systems.

Figure 4.2 summarizes the results of our analysis. The time complexity of the two pure systems is of the same order. This means that the execution efficiency will differ by only a constant factor. Comparing E1 and E2, for each execution cycle the main difference between E1 and E2 lies in Step 2. The time required for Step 2 is approximately \(2n + vnL\) for Algorithm E1, and \(2n + 1\) for Algorithm E2, where \(v\) is the average time spent on a vector operation. Clearly, E2 provides more efficient execution than E1 and thus proves claim 1.

### Applications

<table>
<thead>
<tr>
<th>Compile time</th>
<th>Runtime(per cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure Systems</td>
<td>(O(knL + L))</td>
</tr>
<tr>
<td>Pure Systems with activation graph</td>
<td>(O(knL + n^2))</td>
</tr>
</tbody>
</table>

**Fig. 4.2. Blackboard execution times.**

To prove claim 2, it suffices to compare E1 to the conventional execution cycle shown in Figure 2.1. In Section 4.2.1, the total execution time required for the conventional system is shown to be approximately \(nk^2 + 2nk + as + l)bC\) where \(b\) is the time required for a blackboard retrieval. Using a similar analysis from Section 4.1.1, the execution time of E1 is approximately \(cnL + as + l)vC\), where \(v\) is as defined above and \(c\) is a constant.

In [8] the typical blackboard retrieval time is shown to be approximately 0.03 seconds for a well-organized blackboard. Thus, \(\delta = 0.03\) in the formulas above. A vector element retrieval is a much simpler operation and should not take more than 10 CPU clock cycles. Assuming a clock rate of 30MHz, this means that \(v\), the average vector element retrieval time, is approximately 0.0000003 seconds. The ratio \(b/v\) is thus \(10^5\), which is very large compared to the other terms in the time analysis.

A comparison of the time complexities of the two execution methods reduces to comparing \(bnk^2\) to \(vnL\). Since \(L\) is bounded by \(nk\), which is approximately \(nk^2\), we can reduce the comparison to a comparison of \(b\) and \(vn\). Since \(vn < b\) we conclude claim 2, that the execution time of a conventional system is much larger than the execution time of a condensed system (by about four orders of magnitude).

5 Summary

We have presented a method for condensing pure knowledge sources, an efficient activation mechanism using condensed knowledge sources, and two algorithms to efficiently execute condensed knowledge sources. The first algorithm executes condensed pure knowledge sources; the second is a modified algorithm that uses the activation graph obtained from the condensed system. We conjecture that
our activation mechanism for pure systems is optimal. Since any deterministic blackboard system can be converted to a pure system, this method is applicable to many existing systems.

The methods in this paper provide a large increase in efficiency while maintaining the blackboard execution model. Often an expert system is reimplemented in another language for efficiency after it is completed. Reimplementation is a long and error-prone task. We described how condensing can eliminate the need to convert a finished blackboard system to another language for "efficiency reasons". Our approach provides efficiency by precompiling the activation and execution parts of applicable sets of knowledge sources. A conservative estimate shows that a speedup of four orders of magnitude is possible by utilizing condensed knowledge sources. This is most likely a greater speedup than is possible through reimplementation.

Our analysis shows that the time complexity of the execution cycle of our approach has improved on the conventional approach by the ratio of the time required for blackboard operations to the time required for vector operations. This ratio is relatively large and thus has a large impact on execution efficiency. Work continues on optimizing the remaining steps of the execution cycle: control components, including rating and scheduling.

Acknowledgements

We thank Mete Kabakcioglu, Olga Mazuera-Gomez, Freeman Rawson III, Clovis Tondo, and anonymous referees for their comments on earlier versions of this paper.

References


