Tuning a Blackboard-based Application: A case study using GBB

Daniel D. Corkill
Kevin Q. Gallagher

May 1988
COINS Technical Report 88-37

To be presented at *AAAI-88*
St. Paul, Minnesota, August 1988.

Tuning a Blackboard-based Application: A case study using GBB

Daniel D. Corkill and Kevin Q. Gallagher
Department of Computer and Information Science
University of Massachusetts

Abstract

The run-time performance of a blackboard-based application can be significantly improved by selecting an appropriate blackboard database representation. We present empirical validation of this statement by tuning the representation used in a large, blackboard-based AI application. Dramatic performance gains were obtained without changing *any* problem solving or control activities. The results underscore the importance of efficient blackboard database operations and the benefits of a flexible, instrumented blackboard development environment when tuning the blackboard representation.

This investigation was facilitated by use of the Generic Blackboard Development system (GBB) to construct the application. GBB provides the flexibility to quickly change the database implementation without recoding. Similar performance tuning capabilities are available to any application written using GBB.

1 Introduction

The performance of blackboard-based applications can be significantly enhanced by an appropriate blackboard database implementation. The blackboard paradigm relies heavily on the blackboard for knowledge source (KS) interaction and for holding tentative, partial results until they are needed. Although published measures are nonexistent,[1] the amount of processing time devoted to blackboard interaction is significant—even in applications built with blackboard database machinery that has been customized for speed. Therefore, the runtime performance of a blackboard-based application is strongly influenced by the efficiency of placing and retrieving blackboard objects.

In this paper we present empirical results demonstrating the performance improvements that were obtained by tuning the blackboard database in a large application: the Distributed Vehicle Monitoring Testbed (DVMT) [Lesser and Corkill, 1983; Lesser *et al.*, 1987]. These results are exciting because they were obtained without changing *any* of the problem solving or control activities of the DVMT. Each set of timed experiments executed the same sequence of KSs, created and retrieved the same blackboard objects, and generated the same solution. The *only* difference between each experiment was the processing time required to insert and retrieve blackboard objects.[2]

This investigation was facilitated by use of the Generic Blackboard Development System (GBB) [Corkill *et al.*, 1986] for implementing the DVMT. GBB provides both speed and flexibility in implementing a blackboard-based application as well as efficient execution of the resulting application. The database implementation can be easily changed without recoding (or even recompiling). Such flexibility is important for two reasons. First, the application writer may not initially understand the insertion/retrieval characteristics of the application; so the representation of blackboard objects is subject to change as design intuition evolves into application experience. Second, the insertion/retrieval characteristics may change from those of the prototype as the application is placed into service. This can again require changes to the blackboard representation to maintain high performance under operational conditions.

Before describing how the DVMT's blackboard implementation was tuned using GBB, we present a brief overview of the DVMT's problem-solving architecture, concentrating on its blackboard structure, blackboard objects, and blackboard retrieval characteristics. Next we show how the blackboard representation can be easily varied using GBB. With this background in place, we describe our experiences tuning the DVMT operating on a two relatively small scenarios followed by the results of scaling these results to a larger scenario.

This research was sponsored in part by the Office of Naval Research under a University Research Initiative Grant (Contract N00014-86-K-0764), by donations from Texas Instruments, Inc., by the National Science Foundation under CER Grant DCR-8500332, and by the Defense Advanced Research Projects Agency (monitored by the Office of Naval Research) under Contract N00014-79-C-0439.

[1] An exception is Fennell and Lesser's measurements with an early version of the Hearsay-II speech understanding system which showed a blackboard interaction to KS processing ratio of 10/17 [Fennell and Lesser, 1977]. The blackboard-interaction/processing ratios of the Distributed Vehicle Monitoring Testbed (used in these experiments) range from 8/19-15/3, depending on how efficiently the blackboard is implemented.

[2] Several systems such as Joshua [Rowley *et al.*, 1987], MRS [Russell, 1985], and KEE [Intellicorp, 1987] provide abstraction mechanisms for modifying the representation of data structures without changing rules. However, performance improvements using Joshua often involve reductions in the number and changes in the sequence of rule firings; MRS is tailored to logic programming; and KEE leaves you to write your own procedural storage and retrieval functions.

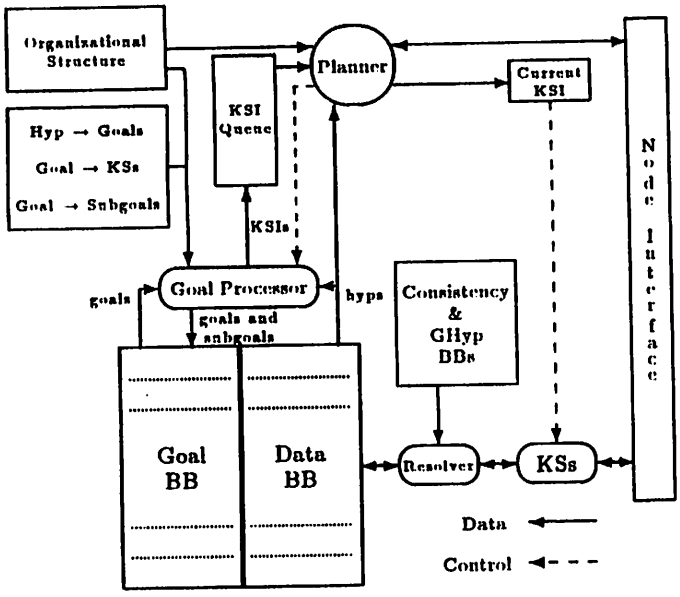


Figure 1: The DVMT Node Architecture.

2 An Overview of the DVMT

The Distributed Vehicle Monitoring Testbed (DVMT) simulates a network of blackboard-based problem solving nodes working on the vehicle monitoring task. The objective of the network is to generate an answer map containing the identity and movement of vehicle patterns based on passively sensed acoustic data. Each network node is a complete Hearsay-II architecture [Erman *et al.*, 1980] with KSs and blackboard levels appropriate for the task of vehicle monitoring. The basic control components of Hearsay-II have been augmented by goal-processing and planning capabilities (Figure 1). In this paper, we concentrate on the major blackboard components: the data, goal, consistency, and ghyp blackboards.

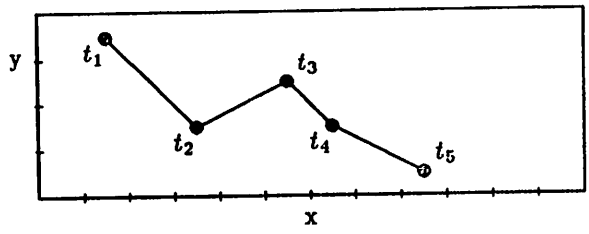
Hypothesized vehicle movements are represented by *hypotheses* placed on the *data blackboard* (D-BB). KSs perform the basic problem solving tasks of abstracting, extending, and refining these hypotheses. The D-BB is partitioned into four data abstraction levels: *signals* (containing minimally-processed sensory data), *groups* (representing harmonically-grouped signal hypotheses), *vehicles* (containing vehicle types hypothesized from related group hypotheses), and *patterns* (containing spatially-related vehicles, such as vehicles moving in formation). Each of these abstraction levels is split into a level for location hypotheses (which have one sensed position) and a level for track hypotheses (which have a compatible sequence of sensed positions over time) for a total of eight D-BB levels: SL, ST, GL, GT, VL, VT, PL, and PT.

KSs combine hypotheses to form more encompassing hypotheses on the same or higher levels. Decisions of which KSs to execute are made using a unified data- and goal-directed framework [Corkill *et al.*, 1982]. The control components of the DVMT (primarily the planner and goal processor) create *goals* on the *goal blackboard* (G-BB), which mirrors the 8-level organization of the D-BB. Each goal represents a request to create a one or more hypotheses on the D-BB within the (corresponding) area covered by the goal. KSs serve as the "actions" for achieving goals on the G-BB.

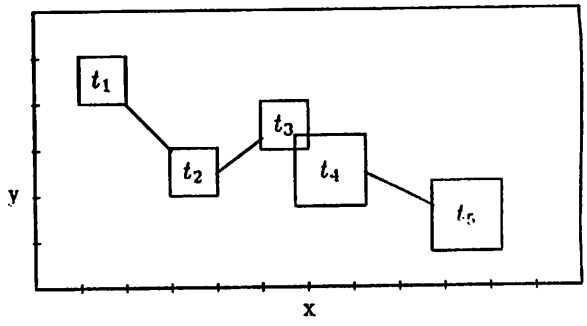
In addition to the D-BB and G-BB, the DVMT includes two "hidden" blackboards for instrumentation. The *consistency blackboard* (C-BB) contains hypotheses representing the correct solution hierarchy as precomputed from the input data. This oracle is invisible to the KSs and control components, but is used to evaluate the developing solution by simulation measurement tools. The *ghyp blackboard* (GH-BB) contains a complete centralized set of the sensory data. Again these hypotheses are only used by measurement tools.

The details of hypotheses and goal objects are also important for tuning the application, and we briefly describe the structure of each.

A hypothesis on the D-BB, C-BB, or GH-BB has the following major attributes: one or more *time-locations* (the vehicle's sensed location at successive points in time), an *event-class* (the frequency classification or vehicle identity information), and a *belief* (the confidence in the accuracy of the hypothesis). The time-location structure of a hypothesis is represented in GBB as a *composite* unit[3] containing series of connected (x,y) points along the *time* dimension:



A goal has the following major attributes: one or more *time-regions* (areas of desired problem solving activity), a set of *event-classes*, and a *rating* (an estimate of the importance of achieving this goal). The time-region structure of a goal is represented as a composite unit containing series of connected (x,y) regions along the *time* dimension:



All DVMT levels are implemented as GBB *spaces* with dimensions *time*, *x*, *y*, and *event-class*. (*Belief* and *rating* would also be useful dimensions but these attributes were not represented as dimensions in the current implementation of the DVMT.) *Space dimensionality* is central to GBB. It provides a *metric* for positioning units on the

[3]GBB's blackboard objects.

blackboard in terms that are natural to the application domain. Units are viewed as occupying some n-dimensional extent within the space's dimensionality. Application code can create and retrieve units according to the dimensions of spaces, without regard to the underlying implementation of the blackboard structure [Corkill *et al.*, 1987]. Dimensional references, however, contain enough information when combined with information about the structure of the blackboard to allow efficient retrieval code to be generated.

3 The Experiments

Access to the DVMT provided the opportunity to empirically evaluate the performance of the DVMT simulator, given differing specifications for the blackboard database implementation. We selected a "typical" single problem-solving node scenario and created three configurations (each one increasingly complex) for experimentation. The first configuration (which will be labeled C1) had a reduced amount of sensory noise and a reduced grammar. The second configuration (C2) had a reduced amount of sensory noise and a full grammar. The third configuration (C3) had all the sensory noise and reduced grammar. The less complicated versions (C1 and C2) required significantly less processor time, and allowed us to run more tuning experiments.

The domain of these experiments was limited to the blackboard implementation strategies provided by GBB, and the performance comparisons are between GBB's various strategies. Considerable effort has been spent optimizing GBB's database machinery, and even GBB's default blackboard implementation strategy results in "reasonable" performance when fewer than 15–20 units reside on a blackboard level.[4]

We emphasize that identical processing occurs in all the experimental tests within an experiment suite. The input data is identical, KSs run in the same sequence, locate the same hypotheses, produce the same output, the control components make the same decisions, and so on. Furthermore, the abstract representation of the blackboard (its decomposition into spaces), the dimensionality of each space, and the unit retrieval pattern specifications remained constant. The *only* variable is the blackboard database machinery used by GBB to store and retrieve blackboard units.

Our experiments concentrated on how hypotheses are stored on the D-BB, C-BB, and GH-BB and how goals are stored on the G-BB. Intuition led us to expect that when a small number of units were to be created on a blackboard level, a simple "push them on a list" implementation would be best due to its low overhead. When a large number of units were created on a space and numerous retrievals were performed on them, a more complex "dimensional-metric-based" implementation was appropriate. Finally, we expected that hypotheses and goals would have different balances in their storage strategies because hypotheses are composites of points while goals tended to be overlapping composites of (x,y) regions. (This expectation proved false.)

[4]Due to its size, it was impractical to recode the DVMT to obtain performance measurements for a non-GBB-based implementation.

We began by analysing each configuration's blackboard interaction statistics. GBB can provide the number and types of units created on each space, the number of insertion and retrieval operations performed on each space, and the time spent on these operations. The numbers of hypotheses and goals created on each blackboard space are as follows:

C1: Number of Blackboard Objects

Level	C-BB	D-BB	G-BB	GH-BB
SL	64	192	0	192
ST	4	0	0	
GL	32	264	96	
GT	2	0	0	
VL	16	44	44	
VT	1	164	362	Total
PL	16	0	0	2132
PT	1	188	450	

C2: Number of Blackboard Objects

Level	C-BB	D-BB	G-BB	GH-BB
SL	64	192	0	192
ST	4	0	0	
GL	32	264	96	
GT	2	0	0	
VL	16	44	44	
VT	1	164	362	Total
PL	16	0	0	2738
PT	1	352	892	

C3: Number of Blackboard Objects

Level	C-BB	D-BB	G-BB	GH-BB
SL	64	2176	0	2176
ST	4	0	0	
GL	32	342	1088	
GT	2	0	0	
VL	16	57	57	
VT	1	234	455	Total
PL	16	0	0	7628
PT	1	297	610	

The number of KS executions required to find the solution in each configuration is:

Configuration	KSIs
C1	743
C2	906
C3	1672

In all configurations, few units are created on the ST, GT, and PL levels. This is because the control components were instructed to restrict the synthesis path to the SL, GL, VL, VT, PT levels.

The number of blackboard unit retrieval operations is also important for tuning. For the each configuration, GBB reports the following operation counts. The tables show the number of retrieval operations for each space followed by the percentage of the total number of retrievals, enclosed in parentheses.

C1: Number of Blackboard Retrievals

Level	C-BB	D-BB	G-BB	GH-BB
SL	680 (3)	1344 (6)	192 (1)	556 (3)
ST	4 (0)	0 (0)	368 (2)	
GL	1184 (6)	800 (4)	456 (2)	
GT	2 (0)	0 (0)	504 (2)	
VL	1204 (6)	338 (2)	348 (2)	
VT	872 (4)	1439 (7)	712 (3)	Total
PL	16 (0)	0 (0)	44 (0)	21,228
PT	5343 (25)	2620 (12)	2202 (10)	

C2: Number of Blackboard Retrievals

Level	C-BB	D-BB	G-BB	GH-BB	
SL	680 (2)	1344 (3)	192 (0)	556 (1)	
ST	4 (0)	0 (0)	368 (1)		
GL	1184 (3)	800 (2)	456 (1)		
GT	2 (0)	0 (0)	504 (1)		
VL	1204 (3)	338 (1)	348 (1)		
VT	872 (2)	2604 (7)	712 (2)	Total	
PL	16 (0)	0 (0)	88 (0)	38,551	
PT	16839 (44)	5134 (13)	4306 (11)		

C3: Number of Blackboard Retrievals

Level	C-BB	D-BB	G-BB	GH-BB	
SL	6624 (5)	16552 (13)	2176 (2)	5512 (4)	
ST	4 (0)	0 (0)	4132 (3)		
GL	18536 (15)	2162 (2)	2530 (2)		
GT	2 (0)	0 (0)	672 (1)		
VL	3310 (3)	484 (0)	430 (0)		
VT	2912 (2)	2392 (2)	994 (1)	Total	
PL	16 (0)	0 (0)	57 (0)	126,873	
PT	45983 (36)	6058 (5)	5335 (4)		

Each retrieval operation involves a composite four-dimensional pattern in *time*, *x*, *y*, and *event-class*. In addition, the DVMT provides additional procedural filtering code to GBB's retrieval process. In our experiments, the time required by these filters is considered part of the retrieval time.

There are two things to note about these numbers. First, almost half the retrieval operations are from the C-BB (used in performance monitoring) but there are relatively few units stored on the C-BB. Therefore, a simple, "low-overhead" strategy is appropriate for representing the C-BB. Second, the distribution of retrieval operations on the D-BB and G-BB shifts dramatically from the filtered case to the complete case. (Surprisingly, we found the retrieval characteristics of hypothesis and goals to be very similar, and the representation strategy that worked well for one also worked well for the other.)

4 Specifying the Blackboard Implementation

In GBB, the implementation strategy for storing units on spaces is specified by defining a *unit-mapping* for each unit to each space in the blackboard. The same unit type can be stored differently on different spaces, and different unit types can be stored differently on the same space. Any unit-mapping can be redefined at any time *before* the specified spaces are instantiated. This means that the implementation strategy can be changed *without* having to recompile unit definitions or application code. The ease of changing the unit-mapping facilitates experimental tuning of the blackboard database implementation strategy.

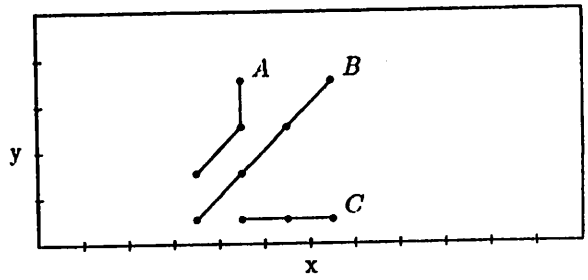
The simplest implementation strategy is to maintain a list of all units of a particular type on a particular space. Retrieval time for this representation is proportional to the number of units on the space.

A more sophisticated strategy is to group units into *buckets* based on their dimensional attributes. This strategy partitions each dimension's range into a number of subranges. Each bucket contains those units which fall within the bounds of the bucket. The number of buckets and their sizes provide a time/space tradeoff for unit insertion and retrieval.

Using more than one dimension for retrieval adds an additional twist to the bucket strategy. One option is to define a multi-dimensional array of buckets. Another option is to define several vectors of buckets and have the retrieval process intersect the result of retrieving in each dimension. A third option is to define one vector and one multi-dimensional array; and so on. Each choice trades off access time for storage space differently.

The retrieval process in GBB can be broken down into four steps: *primary retrival*, *before-pattern* procedural filtering, *pattern-based* filtering, and *after-pattern* procedural filtering [Gallagher and Corkill, 1988]. The primary retrieval step examines the retrieval pattern and determines what buckets must be searched. If more than one array or vector has been defined then an intersection process is performed. The remaining three steps examine the units in the buckets selected by the primary retrieval. The before-pattern and after-pattern filtering steps apply application-specific procedural predicates (if supplied) to units selected by the primary retrieval step or passed by the pattern-based filtering step, respectively. The pattern-based filtering step compares each unit with the entire pattern. This step is necessary because non-conforming units can be retrieved in the primary retrieval step. Pattern-based filtering can be expensive, depending on the complexity of the pattern, so, reducing the number of candidate units in the primary retrieval step can result in significant performance gains.[5]

To illustrate the tradeoffs, consider the three tracks (A, B, and C) depicted below, and suppose the application is looking for tracks which pass through the point (5,3).



If the space is represented simply as a list of units then the primary retrieval step retrieves all three units, which must be compared with the pattern. If the space is represented as two vectors (one for x and one for y), then the primary retrieval step selects all units that occupy the $x = 5$ bucket (in this case B & C). This set is intersected with the set of units occupying the $y = 3$ bucket (A & B), for a primary retrieval result set (B). Pattern-based filtering is then applied to each element of this result set.

5 The Experiments

We began our tuning experiments by running the DVMT on configuration C1 (the simplest scenario) using its "designed" blackboard database implementation: a single vector for *time*. This storage strategy had been intuitively selected (by the first author, based on a pre-GBB implementation of the DVMT) as providing a reasonable balance

[5] All DVMT procedural filters were held constant throughout the experiments reported in this paper.

Experiment	Total Time	BB Time
((t x y))	17:21	10:23 (60)
(t (x y))	17:44	10:48 (61)
(c (t x y))	17:45	10:59 (62)
(t c (x y))	18:00	11:20 (63)
((x y))	18:33	11:43 (63)
(c (x y))	18:48	12:03 (64)
(t x y)	18:56	12:14 (65)
(t x y c)	19:13	12:19 (64)
(x y)	19:52	13:03 (66)
(x y c)	20:06	13:19 (66)
(t)	20:47	14:26 (69)
(t c)	21:28	14:40 (68)
(y)	23:37	17:07 (72)
(x)	23:42	17:08 (72)
(c)	36:19	29:51 (82)
()	36:20	30:01 (83)

Table 1: C1 Configuration Experiments.

Experiment	Total Time	BB Time
(c (t x y))	37:08	18:15 (49)
(t c (x y))	37:56	19:15 (51)
(c (x y))	38:51	20:26 (53)
((t x y))	39:22	20:43 (53)
(t (x y))	40:09	21:56 (55)
(t x y c)	40:11	21:40 (54)
(x y c)	41:14	22:58 (56)
((x y))	41:51	23:27 (56)
(t x y)	43:06	24:55 (58)
(t c)	43:51	25:29 (58)
(x y)	45:11	26:58 (60)
(t)	49:22	31:04 (63)
(y)	54:59	37:05 (67)
(x)	55:11	37:28 (68)
(c)	68:43	51:37 (75)
()	84:40	67:21 (80)

Table 2: C2 Configuration Experiments.

Experiment	Total Time	BB Time
((t x y))	203:18	65:34 (32)
(t (x y))	205:09	68:09 (33)
(c (t x y))	205:42	68:07 (33)
(t c (x y))	207:00	70:35 (34)
(t x y)	207:22	70:43 (34)
(t x y c)	208:52	72:15 (35)
(t)	210:06	89:57 (43)
((x y))	212:46	76:53 (36)
(x y c)	213:38	77:43 (36)
(c (x y))	214:25	78:46 (37)
(x y)	216:11	79:59 (37)
(t c)	218:15	82:37 (38)
(x)	246:54	111:04 (45)
(y)	248:19	112:06 (45)
(c)	353:15	222:41 (63)
()	594:55	493:12 (83)

Table 3: C3 Configuration Experiments.

between retrieval time versus insertion time and storage space. As the experiments demonstrated, this intuition resulted in only mediocre performance—an indication of the importance of database flexibility and performance monitoring tools!

The second experiment ran C1 with the simplest storage strategy, storing all units on a space in a list. As expected, this resulted in even poorer performance. We then tried two vectors, x and y. This gave a dramatic performance improvement, reducing the total execution time by more almost half compared to the baseline "list" strategy. Using three vectors *time*, x, and y resulted in a further 5% reduction in execution time.

We ran many additional experiments (approximately 90) using different strategies for each space and each type of unit. The best strategy was *time*, x, and y as a three dimensional array. The total execution time in the best case was one half that of the worst case (the simple "list" strategy). Even more dramatic is the decrease in the execution time due to blackboard operations. In the best case blackboard operations took only 10:23, whereas in the worst case blackboard operations took 30:01.

Table 1 summarizes the most interesting C1 experiments. Each experiment is identified by the storage strategy used. A list of letters indicates that each dimension is stored in a vector of buckets. An additional level of parentheses indicates that those dimensions are grouped together into a multi-dimensional array of buckets. For example, (t x y c) indicates four vectors (one each for *time*, x, y, and *event-class*) while (t (x y)) indicates one vector for *time*, and one 2-dimensional array for x, and y. In the table, all buckets for the *time* and *event-class* dimensions were unit width; buckets for x and y were of width 5.[6] Furthermore, each space in the C-BB was represented as a simple list (), which was the most effective strategy given its limited number of units.

The processing times are in minutes and seconds from a single run on an 8Mbyte Texas Instruments Explorer II. Differences of 10–20 seconds are insignificant due to timer resolution. The processing time for performing non-blackboard activities in each experiment was approximately 6:40 (ranging from 6:20–6:58). Mean paging time was 9 seconds (8–12 seconds). The last column (in parenthesis) gives the percentage of time spent doing blackboard operations.

Table 2 contains the results with the C2 configuration. Again 10–20 second differences are insignificant. In this set, the processing time for performing non-blackboard activities in each experiment was approximately 18:00 (17:07–18:54). Mean paging time was 36 seconds (34–42 seconds).

The results from C3, the most complex configuration are in Table 3. In this set, the processing time for non-blackboard operations was approximately 135:00 (ranging from 130:33–137:45). Mean paging time was 5:30 (4:50-7:08).

As the three sets of results show, tuning the blackboard representation results in even more dramatic performance improvements as the complexity of the configuration is increased. In C1 and C3 there are only a small number

[6] We experimented with varying bucket sizes, but in these scenarios "reasonable" changes in bucket width had little effect on performance.

of event classes. In C1 the single vector *event-class* unit mapping is no faster then the simple "list" unit mapping. But, in C3, because of the increased amount of sensory noise the (event-class) mapping is 40% faster than the () mapping.

In some cases the overhead of using an additional dimension in the unit mapping is not worth it. For example, in C1 and C3, using *event-class* doesn't improve performance at all. Except for the single vector (event-class) mapping, any mapping that uses *event-class* does worse than the same mapping without *event-class*.

Regardless of the mapping used, the time required to insert units on the blackboard was less then 1% of the total runtime. (The range was ≈0.1% for C3 up to ≈0.9% for C1.) The relatively small insertion cost was surprising, even to the implementers of GBB. Virtually all the blackboard time was spent in retrieval. In fact, 80–90% of the retrieval time was spent in the pattern based filtering step.

The following table illustrates the effect of different mappings on the number of units retrieved by the primary retrieval step. The first column shows the average number of units returned by the primary retrieval and the second column shows the total time spent in the pattern based filtering step. These numbers are for the PT level of the G-BB for configuration C1. At the end of the run there were 892 units on this space. On average 26.25 units passed the pattern based filtering step.

Experiment	Count	Time
(c (t x y))	43.81	7:16
(c (x y))	62.15	8:48
(t)	150.65	14:59
(x)	228.93	19:42
(c)	261.89	26:34

6 Summary

We have provided performance results of tuning the DVMT by matching its blackboard database structure to its blackboard interaction characteristics. We found the improvements to be significant, and the improved performance of the DVMT was worth our tuning investigations.

These results do not suggest that blackboard database optimization should replace the use of superior problem-solving knowledge or control capabilities as a means of enhancing performance. They do demonstrate, however, that improving blackboard interaction efficiency should not be neglected. The potential performance improvements due to the blackboard implementation are proportional to the ratio of blackboard interaction to KS (and control) processing.

Our experiences with the DVMT tuning process demonstrates the importance of obtaining detailed measurements of the insertion/retrieval characteristics of each space (and even within a space). These measurements can significantly augment "intuitive" decisions for blackboard implementation strategies and form an important component of a blackboard development environment.

References

[Corkill *et al.*, 1987] Daniel D. Corkill, Kevin Q. Gallagher, and Philip M. Johnson. Achieving flexibility, efficiency, and generality in blackboard architectures. In *Proceedings of the National Conference on Artificial Intelligence*, pages 18–23, Seattle, Washington, July 1987. (Also to appear in *Readings in Distributed Artificial Intelligence*, Alan Bond and Les Gasser, editors, Morgan Kaufmann, in press, 1988.).

[Corkill *et al.*, 1986] Daniel D. Corkill, Kevin Q. Gallagher, and Kelly E. Murray. GBB: A generic blackboard development system. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1008–1014, Philadelphia, Pennsylvania, August 1986. (Also to appear in *Blackboard Systems*, Robert S. Engelmore and Anthony Morgan, editors, Addison-Wesley, in press, 1988.).

[Corkill *et al.*, 1982] Daniel D. Corkill, Victor R. Lesser, and Eva Hudlická. Unifying data-directed and goal-directed control: An example and experiments. In *Proceedings of the National Conference on Artificial Intelligence*, pages 143–147, Pittsburgh, Pennsylvania, August 1982.

[Erman *et al.*, 1980] Lee D. Erman, Frederick Hayes-Roth, Victor R. Lesser, and D. Raj Reddy. The Hearsay-II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12(2):213–253, June 1980.

[Fennell and Lesser, 1977] Richard D. Fennell and Victor R. Lesser. Parallelism in Artificial Intelligence problem solving: A case study of Hearsay II. *IEEE Transactions on Computers*, C-26(2):98–111, February 1977.

[Gallagher and Corkill, 1988] Kevin Q. Gallagher and Daniel D. Corkill. Blackboard retrieval strategies in GBB. May 1988. (Submitted to the Second AAAI Workshop on Blackboard Systems.).

[Intellicorp, 1987] Intellicorp. *KEE V3.1 Reference Manual.* 1987.

[Lesser and Corkill, 1983] Victor R. Lesser and Daniel D. Corkill. The Distributed Vehicle Monitoring Testbed: A tool for investigating distributed problem solving networks. *AI Magazine*, 4(3):15–33, Fall 1983.

[Lesser *et al.*, 1987] Victor R. Lesser, Daniel D. Corkill, and Edmund H. Durfee. *An Update on the Distributed Vehicle Monitoring Testbed.* Technical Report 87-111, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts 01003, December 1987.

[Rowley *et al.*, 1987] Steve Rowley, Howard Shrobe, and Robert Cassels. Joshua: Uniform access to heterogeneous knowledge structures or why joshing is better than conniving or planning. In *Proceedings of the National Conference on Artificial Intelligence*, pages 48–52, Seattle, Washington, July 1987.

[Russell, 1985] Stuart Russell. *The Compleat Guide to MRS.* Technical Report KSL No. 85-12, Knowledge Systems Laboratory, Computer Science Department, Stanford University, Stanford, California 94305, June 1985.