Application of Machine Learning to the Maintenance of Knowledge Base Performance

Lawrence B. Holder
University of Illinois
Department of Computer Science
405 North Mathews, Urbana, IL 61801

(To Appear in the Proceedings of IEA/AIE-90)

Abstract

Integration of machine learning methods into knowledge-based systems requires greater control over the application of the learning methods. Recent research in machine learning has shown that isolated and unconstrained application of learning methods can eventually degrade performance. This paper presents an approach called performance-driven knowledge transformation for controlling the application of learning methods. The primary guidance for the control is performance of the knowledge base. The approach is implemented in the PEAK system. Two experiments with PEAK illustrate how the knowledge base is transformed using different learning methods to maintain performance goals. Results demonstrate the ability of performance-driven knowledge transformation to control the application of learning methods and maintain knowledge base performance.

1 Introduction

Current construction of expert systems begins with the knowledge engineer entering knowledge acquired from a human expert. The knowledge engineer tests the system on a set of test cases and modifies the knowledge until the system achieves a desired level of performance. Machine learning methods have been developed to automate the knowledge acquisition process by reducing the dependence on the quality of the initial knowledge and using information from the test cases to improve the performance of the knowledge base. However, current machine learning methods work in isolation and do not consider the need for multiple learning methods or the possibility of performance degradation after repeated applications of the isolated method.

As expert systems acquire the ability to service requests from multiple problem domains, varying machine learning methods will be necessary to maintain performance in each domain. For instance, if the knowledge in one domain takes the form of examples and the goal is improved accuracy, then the system may choose to invoke an empirical learning method that constructs generalized rules describing the examples. If the knowledge in another domain takes the form of higher-level rules and the goal is improved response time, then the system may choose an analytical learning method to construct a macro from the rule inference chain used to solve the current problem.

Knowledge-based systems must be able to select learning methods appropriate for the desired performance improvement. In addition, application of the learning method must preserve the performance goals of other knowledge used for other tasks. This paper presents a methodology called performance-driven knowledge transformation for integrating multiple machine learning methods in an expert system framework in order to achieve and maintain performance on multiple tasks. This approach controls the application of learning methods based on their ability to achieve desired performance goals.

The next section surveys recent results from machine learning research that indicate the need for more selective application of the particular learning paradigm. Section 3 presents the performance-driven knowledge transformation approach, implemented in the PEAK system. Section 4 demonstrates the use of PEAK for transforming knowledge from a classification task domain and a planning task domain.

2 Related Work

Research in both empirical and analytical learning has uncovered deficiencies in the employed methodologies.

The major deficiencies stem from the naive view that the methodology in question is always applicable to the learning task and therefore should always be applied to the data. In a performance-driven system, one methodology is rarely sufficient to handle the variety of learning tasks.

2.1 Empirical Learning

During experimentation with the AQ system (specifically, AQ15 [Michalski86]), Michalski found that repetitive application of AQ may yield less accurate concepts than a more conservative application strategy combined with a simple inference mechanism [Michalski87]. The AQ methodology finds a conjunctive description that covers as many positive examples as possible without covering any negative examples. Positive examples not covered by the first description are used as input for another execution of AQ. This procedure continues until a concept in disjunctive normal form is produced covering all the positive examples and none of the negative examples. Michalski compared the accuracy of the DNF concept with that of the concept consisting of only the single disjunct covering the most positive examples. Using a simple matching procedure, the truncated concepts out-performed the original concepts in both accuracy and speed. This observation illustrates the need for systems to be more selective in their own behavior when such selectivity is sufficient to achieve the performance goals.

Similar results have been obtained with the decision trees generated by Quinlan's ID3 program [Quinlan86]. Quinlan found that *pruning* the rules extracted from a decision tree can improve the accuracy of the rules on unseen examples [Quinlan87]. ID3 builds decision trees by selecting an attribute from the training examples providing the best split (according to an information theoretic criterion) between positive and negative examples. The program continues by descending each branch and recursively applying itself to the examples satisfying the attribute value for that branch. ID3 halts when all the nodes at the frontier of the tree contain all positive or all negative examples. The pruning stage removes rules from the decision tree until accuracy on the test examples begins to decrease. Compared to the original rules, the pruned rules performed better on the set of unseen test examples. Although the success of pruning is due mainly to the decreased number of examples available at higher depths in the tree, this stage might have been unnecessary if the desired accuracy had been taken into account during the initial generation of the decision rules.

2.2 Analytical Learning

Research on analytical (explanation-based) learning techniques began to focus more attention on performance with the appearance of Keller's work on the definition of operationality [Keller88]. Analytical techniques learn from a single example by proving the example is an instance of the concept to be learned. The proof terminates when the leaves of the proof tree are all operational predicates. The proof tree is then generalized, yielding an operational description of the concept. Earlier work on explanation-based learning defined an operational concept as one whose description is composed from a set of predicates deemed easy to evaluate [DeJong86, Mitchell86]. Keller points out that operationality is more intimately related to the performance element and the desired performance improvement. This view of operationality was used in the METALEX system that learns heuristics for solving calculus problems. METALEX defines an operational concept as one that improves the performance element's (problem solver's) run-time efficiency on a set of benchmark calculus problems, while maintaining effectiveness so that some percentage of the problems are still solved correctly. The increased attention on performance has led to the reevaluation of several analytical learning systems and the observation that performance may degrade with repeated application.

In experimentation with the MORRIS analytical learning system, Minton found that performance degrades as the number of rules grows large (the *utility problem*) [Minton88]. In order to learn a concept, the system acquires several rules whose disjunction forms the system's understanding of the concept. As the number of rules increase, the cost of determining the applicability of a rule may outweigh the benefits of applying, and thus, retaining the rule. Minton attempts to solve the utility problem in the PRODIGY system for learning effective control knowledge. Minton's solution is to maintain empirical estimates of match costs, application savings and frequency of application for each rule. These estimates are used to compute a utility value for the rule. If this value becomes negative, the rule is no longer considered. Minton found that maintenance of a rule's utility value and compression of the rule's conditions result in a substantial performance improvement. These results indicate that a system should be sensitive to the cost and savings of the learned descriptions.

Like empirical learning, analytical learning suffers from degrading performance over time without careful consideration of the function the learned knowledge is to serve. A learning system should be sensitive to the performance goals of the learning task and should consider only the knowledge that provides progress

towards these goals.

The next section presents an approach that applies machine learning methods only when necessary to improve performance, and then, only when the learning method is appropriate for achieving the desired performance goals. Subsequent experimentation demonstrates the ability of the approach to overcome the deficiencies identified with isolated and unconstrained application of machine learning methods.

3 Performance-Driven Knowledge Transformation

The application of machine learning methods is controlled by a process called *performance-driven knowledge transformation*. The process invokes learning methods based on their ability to achieve desired performance goals while preserving the performance on other tasks.

Each task for the knowledge base defines a performance space. The dimensions of the performance space are the performance goals (e.g., completeness, correctness, response time) to be maintained by the knowledge base for that task. The current state of the knowledge base is represented by a point in the performance space for each task. A knowledge transformation can be viewed as a move of the current knowledge base from one point in the performance space of each task to another. Figure 1 shows the performance spaces for two tasks. Task A (Figure 1a) consists of three performance goals G_1, G_2 and G_3. Task B (Figure 1b) consists of two performance goals G_4 and G_5. The location of two knowledge bases K_1 and K_2 are shown for each task.

The desired performance for each task defines a hyper-rectangle in that task's performance space. When the knowledge base moves outside the desired-performance hyper-rectangle in some performance space, performance-driven knowledge transformation selects a learning method to transform the knowledge base so that the corresponding point in the performance space for the current task moves inside the desired-performance hyper-rectangle without moving the point outside the desired hyper-rectangle in the performance spaces for other tasks. Referring to Figure 1, knowledge base K_1 has satisfactory performance for task B, but violates the performance goals of task A. Transforming knowledge base K_1 to K_2 achieves the performance goals of task A and preserves the satisfactory performance for task B.

This research investigates a means-ends approach to performance-driven knowledge transformation. When a performance goal violation is detected while solving a problem from some task, the means-ends approach uses information about the context of the goal violation (e.g., the difference between desired and actual performance) to select a transformation operator for reducing this difference while maintaining other performance levels. Application of the operator yields a new knowledge base. If the new knowledge base achieves the violated performance goals and preserves other performance goals, then the current knowledge base is replaced by the new knowledge base. Otherwise, another transformation operator is selected for application.

In the following discussion, certain assumptions have been made about the knowledge in the knowledge base and the performance element using this knowledge. The knowledge base is a set of Horn clause rules. The performance element is a deductive retriever similar to Prolog. Performance is measured while the performance element attempts to solve a query posed by the user. Attached to the query are the performance goals to be maintained during solution. Performance goal violations occur when the measured performance exceeds the desired thresholds.

3.1 Performance Perspective

Using performance goals as a means of guiding the maintenance and repair of a knowledge base requires a precise definition of performance. The definition of performance depends on the perspective. Four perspectives are applicable for describing the performance of a knowledge base:

- **External** performance is the performance measured from outside the knowledge base, regardless of any internal knowledge transformations.
- **Current** performance is the performance the system currently maintains for the previously seen queries.
- **Expected** performance is the performance the system expects to demonstrate on future queries. Expected performance is usually the same as current performance.
- **Absolute** performance is the performance that the current state of the knowledge would support if given every possible query.

When the user specifies a threshold for some performance measure, the proper perspective must be used to evaluate the performance of the knowledge base. *Absolute* performance is rarely available due to a lack of knowledge about the instance space. *Absolute* performance is inappropriate, because the distribution over the entire instance space may not give equal probability to each instance. *External* performance provides information about the rate of convergence

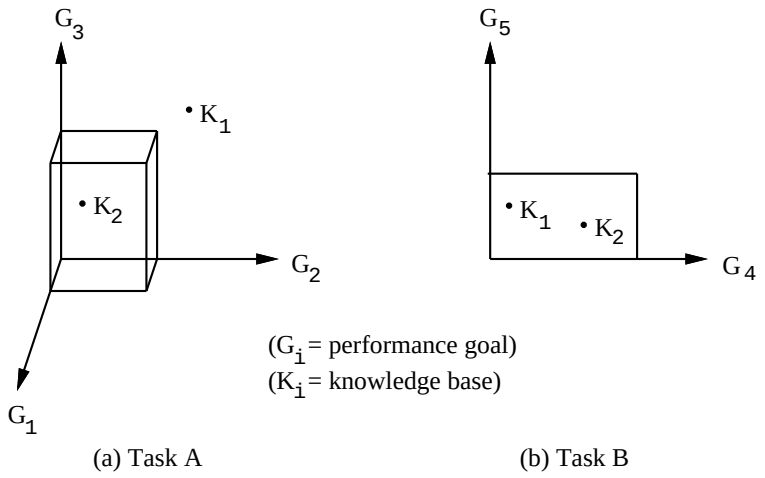

Figure 1: Performance Spaces for Two Knowledge Base Tasks

towards absolute performance. Changes in *external* performance indicate the need for an increase or decrease in the extent of the knowledge transformations. *Current* performance evaluates the knowledge only on previously seen queries. *Expected* performance is the best measure of the current state of the knowledge base, because the objective of the knowledge base is to maintain its expected ability to perform the task within desired thresholds on possibly unseen queries.

Performance-driven knowledge transformation should measure both *expected* and *external* performance. Knowledge transformations are triggered only when *expected* performance falls below desired levels. *External* performance should then be used in the selection of an appropriate transformation operator. The greater the difference between *external* and *expected* performance, the more drastic a transformation operator should be recommended by the system.

3.2 Information on Goal Violations

Once a goal violation has been detected, several pieces of information are available for selecting an appropriate knowledge transformation operator. First, as described in the previous section, the difference between expected and external performance indicates the extent of the necessary transformation.

Second, after the performance element attempts to solve a query, the violated and preserved goals are known. Each goal contains information about the performance measure that this goal constrains, the desired threshold on the measure, the observed value of the measure on previously seen queries (including the query just processed), and the difference between the observed and desired performance (the error). The performance measure constrained by a violated goal is useful for selecting transformation operators capable of improving this performance measure. The magnitude of the error indicates the extent of the transformation. The performance measure constrained by a satisfied goal is useful for selecting transformation operators capable of preserving this performance measure. The magnitude of the error indicates the extent to which the selected operator may degrade performance on the satisfied goals in order to achieve performance on the violated goals.

A third source of information that will be available upon detection of a performance goal violation is the *task history*. Each task known to the knowledge base maintains a task history of previously seen queries from the task. The task history serves two purposes. First, the task history represents an empirical estimate of the distribution over the possible queries of the task. This distribution can be used to verify the achievement of violated performance goals in transformed knowledge. Second, an entry in the task history contains information about the query-solving episode. One useful piece of information about a query-solving episode is the trace of the knowledge accessed during the solution.

The *knowledge trace* is an and/or tree that records the knowledge accessed during the solution of the query and indicates which rules (if any) support the response to the query. Information about the shape of a task's knowledge traces constrains the selection of knowledge transformations. For example, wide, shallow knowledge traces indicate that the knowledge consists of specific instances of the task; whereas narrow, deep knowledge traces indicate a more general set of rules for proving queries from the corresponding task.

Finally, past success of the transformation opera-

tors provides information upon performance goal violation. As the knowledge base transforms to meet performance goals, a record is kept of the old and new knowledge bases along with the operator responsible for the transformation. If the new knowledge base achieves a violated goal while preserving non-violated goals, then the system increases the operators applicability for achieving and preserving the appropriate goals. Over time, collection of this information will allow the system to make a more informed operator selection based on past experience.

3.3 Verification of Knowledge Base

Because no operator application is guaranteed to achieve the desired results, the system must verify that the knowledge base resulting from an operator application achieves the desired performance. Verification can be accomplished by re-solving the queries in the task history. The size of the task history can be changed to tradeoff performance convergence rates for transformation speed. As the system learns operator applicability, there is less chance of multiple verification being necessary to repair one goal violation; thus, the task history size can be increased over time.

4 Experimentation

This section illustrates the application of PEAK on two tasks from diverse domains. The first experiment involves learning to improve response time, completeness and correctness while determining whether to land the space shuttle manually or automatically depending on environmental conditions. The second experiment involves learning to improve response time while constructing plans to build towers in the blocks-world domain. Together, the two experiments demonstrate the ability of performance-drive knowledge transformation to selectively apply appropriate learning methods to achieve desired performance goals.

4.1 Experiment with Shuttle Domain

This experiment executes the PEAK system the shuttle landing control database available from the machine learning databases maintained by University of California at Irvine. The problem is to determine whether to land the shuttle manually or automatically based on environmental attributes. The corresponding task is labeled the *landing* task, and the queries are of the form `landing(ENV,?x)`. The `ENV` in the query represents the environmental situation to be evaluated. The performance element attempts to fill in the `?x` with the recommended landing control: `auto` or `noauto`.

Prior to query answering, the user inputs the performance thresholds to be maintained by the knowledge base while answering *landing* queries using the performance element (a backward-chaining deductive theorem prover for Horn clauses). For this experiment, three performance goals are specified: *correctness*, *completeness* and *response time*. The correctness goal specifies that the answers to queries must be correct 90% of the time. The completeness goal specifies that the query must be answered 95% of the time. That is, the answer should be either `auto` or `noauto` and not *"I don't know"*. The response time goal specifies that the performance element must respond within 10 seconds.

Two knowledge transformation operators are available: rote learning and empirical learning. Application of the rote learning operator asks the user for the correct answer to the query. A new rule is added to the knowledge base having the instantiated query as the consequent, and the facts defined before query execution as the antecedent. The empirical learning operator utilizes the ID3 program to build a decision tree from examples in the knowledge base. The examples are rules such as those learned by the rote operator. Each path in the resulting decision tree is converted to a rule. The examples are replaced by the new rules in the transformed knowledge base.

Starting with an empty knowledge base, PEAK attempts to solve *landing* queries, while maintaining the performance goals. Figure 2 plots the three performance goals for 200 randomly chosen queries from the shuttle landing control domain.

Figure 2a illustrates how PEAK maintains response time performance below 10 seconds. For the first 30 queries, response time increases as the number of rote-learned rules increases. Eventually, the large number of rules in the knowledge base cannot be traversed within the response time threshold.

While processing the 30th query, PEAK was unable to solve the query, generating a completeness failure. PEAK first trys to transform the knowledge base by rote-learning a new rule. However, verification of the new knowledge base uncovers a response time failure. Because the rote learning operator was ineffective, PEAK chose to apply the ID3 operator. ID3 generalized the 29 learned instances into 8 general rules. As Figure 2a indicates, the resulting transformation drastically improves response time performance.

The plot of completeness performance in Figure 2b illustrates how PEAK quickly learns the initial query knowledge. After the ID3 transformation, completeness remained above the 95% threshold for the remainder of the 200 queries.

The correctness plot in Figure 2c shows how per-

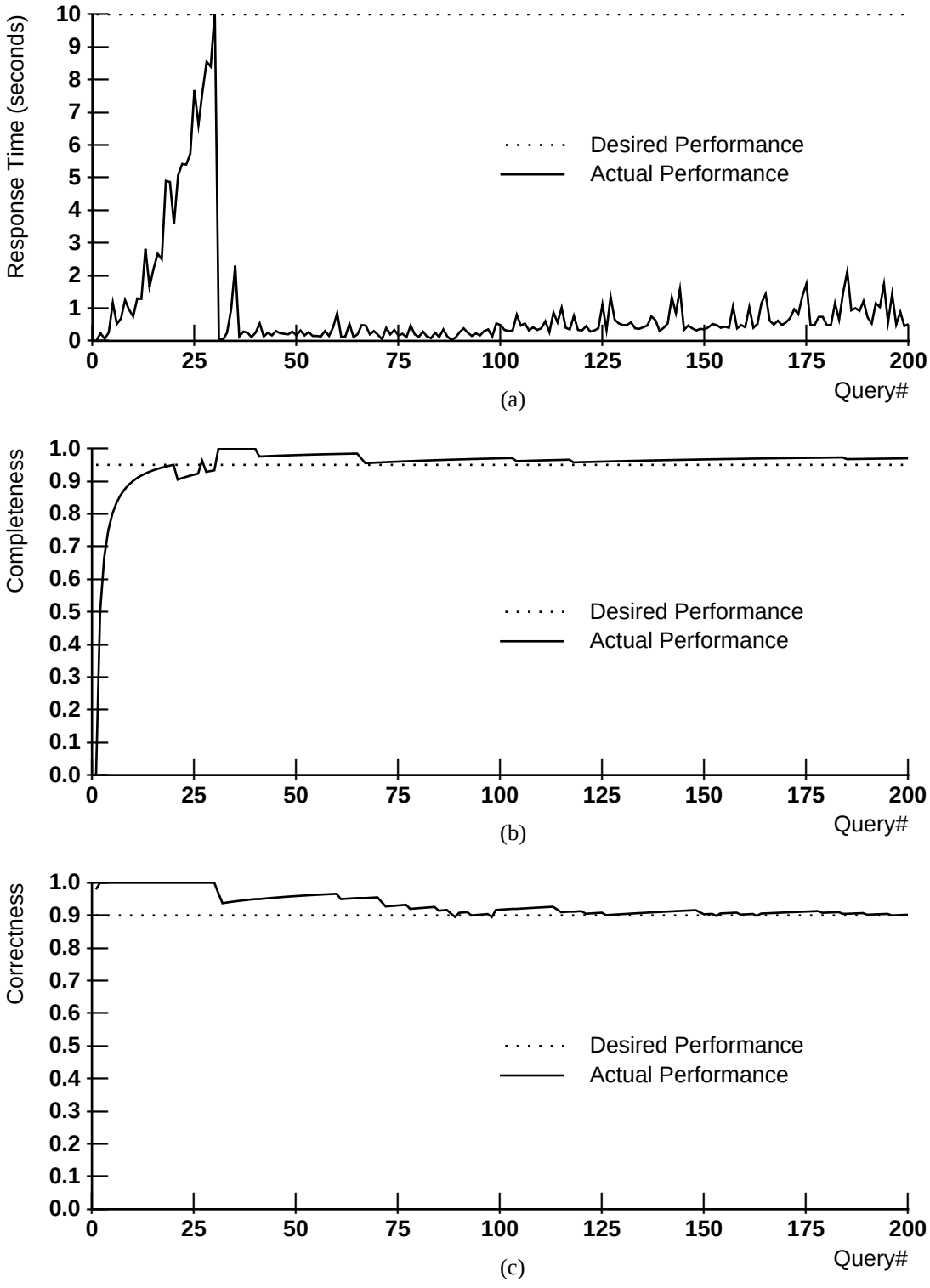

Figure 2: Plots of Performance for Shuttle Domain

formance starts at 100% and converges to the desired 90% threshold. The initial values of 100% for correctness are due to the fact that many of the initial queries could not be answered. Correctness performance only measures the correctness of answered queries. Immediately following the application of ID3, correctness falls to 94% due to the next two queries being incorrectly answered according to the new knowledge base. As query answering continues, the over-generalization in the rules eventually brings correctness down below the 90% threshold. Correctness violations occur at queries 89, 98, 153 and 163. In each case, PEAK uses the rote-learning operator to memorize the incorrectly answered query and restore 90% correctness performance.

The final knowledge base after completion of the 200 queries consists of the 12 rules shown in Figure 3. Rules 5-12 are the general rules learned by ID3. Rules 1-4 are the specific instances learned to repair the over-generalization in ID3's rules. After 200 queries, the knowledge base converged to 8 general rules describing major trends in the shuttle landing domain and four specific rules for special cases not handled correctly by the general rules.

One final observation from Figure 2 is the convergence of the performance towards the desired thresholds and not towards the maximum possible performance. This indicates how performance-driven knowledge transformation utilizes flexibility in one dimension of performance to improve performance in another dimension.

4.2 Experiment with Tower Domain

In the task from the tower domain, the user asks the performance element to construct a plan for building a tower of blocks. The queries are of the form `tower(A B C ?state)`, where `A`, `B` and `C` are blocks, and `?state` is a variable to be instantiated with the plan for achieving the tower.

Prior to query answering, the user inputs the performance thresholds to be maintained by the knowledge base while answering *tower* queries. For this experiment, one performance goal is specified: response-time < 10 seconds. Performance goals for completeness and correctness are inappropriate, because the domain theory is assumed complete and correct.

In addition to the rote learning and ID3 operators used with the first experiment, an explanation-based generalizer, EGGS [Mooney86], is included in the PEAK system. EGGS applies standard explanation-based techniques [DeJong86, Mitchell86] to generalize the proofs obtained by the performance element. When EGGS is applied to a proof, the result is a general rule that is added to the knowledge base.

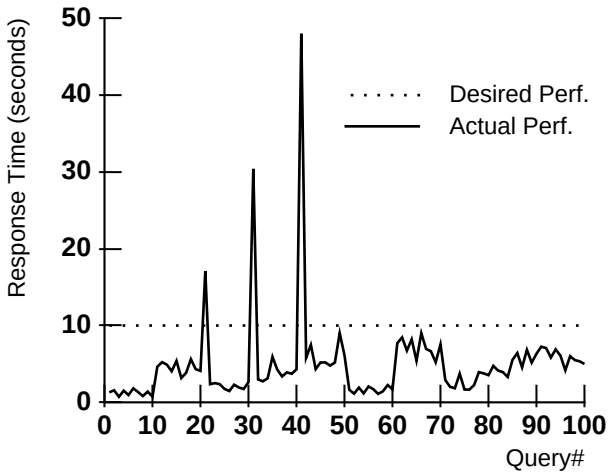

Figure 4: Plot of Response Time for Tower Domain

Starting with the blocks-world domain theory, PEAK attempts to solve *tower* queries, while remaining the response time performance goal. Figure 4 shows the response time obtained by PEAK for 100 semi-randomly chosen *tower* queries. Semi-random means that the first ten queries were all towers of height two, each chosen randomly from among six possible blocks in the initial state. The second ten queries were all towers of height three, and so on for the first 50 queries. The second 50 queries repeat the above sequence to show the effects on response time of the rules learned during the first 50 queries.

As shown in Figure 4, the first 20 queries (towers of height two and three) are solved by the original domain theory within the response time threshold. However, the domain theory is unable to maintain the response time performance goal while processing the 21st query (tower of height three). At this point, ID3 cannot be applied due to the lack of examples in the knowledge base. The EGGS operator is chosen over rote learning due to the knowledge trace for the query. The deep, wide proof tree suggests that EGGS is more likely to succeed than ID3.

Application of EGGS yields a general rule that builds any tower of height three in one step. Thus, the remainder of the *tower* queries for height three are completed within the response time threshold. Similar rules are learned for the 31st query (tower of height four) and 41st query (tower of height five). Figure 4 shows that retrying the towers of heights two through five (queries 50-100) results in no response time performance violations due to the previously learned rules.

1. landing(x,noauto) ← sign(x,nn) & wind(x,head) & stability(x,xstab) & error(x,MM) & magnitude(x,Medium) & visibility(x,yes)
2. landing(x,noauto) ← sign(x,pp) & wind(x,tail) & stability(x,xstab) & error(x,MM) & magnitude(x,Low) & visibility(x,yes)
3. landing(x,noauto) ← sign(x,nn) & wind(x,head) & stability(x,stab) & error(x,MM) & magnitude(x,OutOfRange) & visibility(x,yes)
4. landing(x,noauto) ← sign(x,nn) & wind(x,tail) & stability(x,xstab) & error(x,MM) & magnitude(x,Low) & visibility(x,yes)
5. landing(x,auto) ← error(x,MM) & visibility(x,yes)
6. landing(x,auto) ← stability(x,stab) & error(x,SS) & magnitude(x,Strong) & visibility(x,yes)
7. landing(x,auto) ← visibility(x,no)
8. landing(x,noauto) ← error(x,XL) & visibility(x,yes)
9. landing(x,noauto) ← error(x,LX) & visibility(x,yes)
10. landing(x,noauto) ← stability(x,xstab) & error(x,SS) & magnitude(x,Strong) & visibility(x,yes)
11. landing(x,noauto) ← error(x,SS) & magnitude(x,OutOfRange) & visibility(x,yes)
12. landing(x,noauto) ← error(x,SS) & magnitude(x,Low) & visibility(x,yes)

Figure 3: Shuttle Domain Knowledge Base After 200 Queries

5 Conclusions

The experiments of the previous section demonstrate that different machine learning methods are necessary in different learning contexts. Performance-driven knowledge transformation uses contextual information about the desired performance goals and the knowledge accessed by the problem-solving episodes to select an appropriate learning method. This process can be used to improve and maintain performance for multiple problems in varying domains.

Integrating performance-driven knowledge transformation into knowledge-based systems will reduce dependence on the quality of the initially entered knowledge and allow the knowledge base to adapt to changing performance requirements. Performance-driven knowledge transformation is also able to learn the knowledge necessary to control multiple learning methods, invoking appropriate methods only if necessary to improve performance.

Acknowledgments

I would like to thank Robert Stepp, Diane Cook, Brad Whitehall and Robert Reinke for their helpful suggestions on this work. The ID3 and EGGS operators were adapted from versions written by Ray Mooney. The rules used for the tower domain were adapted from rules written by Jude Shavlik and Ray Mooney.

References

G. F. DeJong and R. J. Mooney, "Explanation-Based Learning: An Alternative View," *Machine Learning 1*, 2 (April 1986), pp. 145-176.

R. M. Keller, "Defining Operationality for Explanation-Based Learning," *Artificial Intelligence 35*, 2 (June 1988), pp. 227-241.

R. S. Michalski, I. Mozetic, J. Hong and N. Lavrac, "The Multi-Purpose Incremental Learning System AQ15 and its Testing Application in Three Medical Domains," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 1041-1047.

R. S. Michalski, "How to Learn Imprecise Concepts: A Method for Employing a Two-Tiered Knowledge Representation in Learning," *Proceedings of the 1987 International Machine Learning Workshop*, Irvine, CA, June 1987, pp. 50-58.

S. Minton, "Quantitative Results Concerning the Utility of Explanation-Based Learning," *Proceedings of the National Conference on Artificial Intelligence*, St. Paul, MN, August 1988, pp. 564-569.

T. M. Mitchell, R. Keller and S. Kedar-Cabelli, "Explanation-Based Generalization: A Unifying View," *Machine Learning 1*, 1 (January 1986), pp. 47-80.

R. J. Mooney and S. W. Bennett, "A Domain Independent Explanation-Based Generalizer," *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA, August 1986, pp. 551-555.

J. R. Quinlan, "Induction of Decision Trees," *Machine Learning 1*, 1 (1986), pp. 81-106.

J. R. Quinlan, "Generating Production Rules from Decision Trees," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987, pp. 304-307.