4-1-1998

Resource-Usage Prediction for Demand-Based Network-Computing

Nirav H. Kapadia
Purdue University School of Electrical and Computer Engineering

Carla E. Brodley
Purdue University School of Electrical and Computer Engineering

José A. B. Fortes
Purdue University School of Electrical and Computer Engineering

Mark S. Lundstrom
Purdue University School of Electrical and Computer Engineering

Resource-Usage Prediction for Demand-Based Network-Computing

Nirav H. Kapadia
Carla E. Brodley
José A. B. Fortes
Mark S. Lundstrom

TR-ECE 98-9
April 1998

School of Electrical
and Computer Engineering
Purdue University
West Lafayette, Indiana 47907-1285

Resource-Usage Prediction for Demand-Based Network-Computing

Nirav H. Kapadia

Carla E. Brodley

José A. B. Fortes

Mark S. Lundstrom

School of Electrical and Computer Engineering
1285 Electrical Engineering Building
Purdue University
West Lafayette, IN 47907-1285

This work was partially funded by the National Science Foundation under grants CDA-9617372, EEC-9700762, and MIPS-9500673. The feature-vectors for T-Suprem3 and Minimos were identified with the help of Mike Young and Steven Bourland, respectively.

TABLE OF CONTENTS

ABSTRACT

This document reports on an application of artificial intelligence to achieve demand-based scheduling within the context of a network-computing infrastructure. The described AI sub-system uses tool-specific, run-time input to predict the resource-usage characteristics of runs. Instance-based learning with locally weighted polynornial regression is employed because of the need to simultaneously learn multiple polynomial concepts and the fact that knowledge is acquired incrementally in this dornain. An innovative combination of a two-level knowledge base, and age and usage statistics are used to: a) detect inadequate and noisy feature-vectors, b) account for short-term variations in compute-server and network performance, and c) exploit temporal and spatial locality of runs. Modifications to the basic learning algorithm allow the approach to be computationally feasible for extended use and noise tolerant by selectively adding feature-vectors into the knowledge base and discarding feature-vectors that consistently result in inaccurate predictions, respectively. The learning system was tested on three semiconductor simulation tools during normal use of the Purdue University Network Computing Hub during Fall 1997, and on four synthetic data-sets. Results indicate that the described instance-based learning technique using locally weighted regression with a locally linear model works well for this domain.

1. Introduction

There is increasing evidence to support the view that, in the future, computing will be network-based and service-oriented. Desktop computers will be able to reach out across the network and obtain whatever software and hardware resources the current application needs (Smarr and Catlett, 1992). For example, the proposed Network Computers and Net PCs (e.g., Comerford, 1997) will download the required software from the network, and supplement their computing power by that of network-accessible servers. This view implicitly assumes the existence of an underlying infrastructure capable of supporting network-accessible, *demand-based computing.*

A demand-based computing system can be characterized by its universal accessibility and its ability to make automatic cost/performance tradeoff decisions at run-time. Universal accessibility can be provided via a widely-used networked interface such as the world-wide web. Run-time cost/performance tradeoff decisions require that the infrastructure be able to decide how (which implementation - e.g., sequential versus parallel) and where (which platform) to execute a tool. In contrast, with conventional computing systems, user-commands are implicitly tied to specific - and typically local - implementations and machines. This report presents an application of artificial intelligence to achieve *demand-based scheduling* within the context of a network-computing infrastructure (the Purdue University Network Computing Hub, or PUNCH) that allows users to access and run existing software tools via standard world-wide web browsers such as Netscape.

Cost/performance tradeoff decisions are based on scalability and portability information. Scalability information includes run-specific resource-usage in terms of CPU time, network data-transfer time, memory usage, and disk-space requirements. Portability information consists of a list of the available implementations (e.g., sequential versus parallel) for a given tool, and the architectures on which they are supported.

While portability information is usually available a priori, scalability information is generally dependent on the mn-time input. Although it may be possible to obtain analytical expression:; that describe the relationship between the run-time input and the corresponding resource-usage (e.g., matrix-manipulation codes), in general, tools tend to exhibit complex behavior that make such analytical expressions nearly impossible. Even when it is possible to determine an analytical expression, the resource-usage characteristics cannot be computed from an expression that simply describes the computational complexity of the algorithm; the appropriate: architecture-specific constants must also be determined.

To our knowledge, no other system has used tool-specific, run-time inputs to predict the resource-usage characteristics of runs. Other work aimed at predicting resource-usage (e.g., Goswami, Devarakonda, and Iyer, 1993; Svensson, 1990; Wang, Krueger, and Liu, 1993) utilizes tool-specific analytical expressions or statistical data obtained from past runs to predict the resource-usage characteristics of future runs. For example, Devarakonda and Iyer (1989) use the identity of a tool and its execution history to identify high-density clusters in the space defined by the resource-usage parameters. Results show that even these simple heuristics allow for significantly better scheduling (Goswami, Devarakonda, and Iyer, 1993). This approach was not used here because the resource-usage characteristics of the tools in our domain tend to be highly dependent on run-specific parameters (e.g., a Monte Carlo simulation can require anywhere from a few minutes to several days of CPU time, depending on the problem being solved).

The goal of the AI sub-system described here is to assist the network-computing infrastructure in emulating an *ideal user* in terms of resource-management and usage policies. For the purposes of this work, an ideal user is one who: a) can predict the resource-requirements of each run that he/she initiates, b) preferentially uses the most plentiful resources that support the requirements of the given run, and c) voluntarily relinquishes resources to higher-priority users when necessary.

The AI sub-system uses instance-based learning to predict the CPU time and the network data-transfer time for a given run on the basis of the associated run-time *input.*[1] The prediction is then used to adapt the network-computing infrastructure's resource-allocation policy. The choice of instance-based learning was driven by the need to capture polynomial concepts and the fact that knowledge is acquired incrementally in this domain. Note that each tool has its own knowledge base. The following learning issues were addressed by using an innovative combination of a two-level knowledge base, and age and usage statistics: a) detection of inadequate feature-vectors, b) short-term variations in compute-server or network performance, c) noisy features, and d) scalability of the knowledge base for extended use. The learning system was tested on three semiconductor simulation tools during normal use of PUNCH in Fall 1997, and on four synthetic data-sets (off-line). Locally weighted polynomial regression with a locally linear model was found to perform well for all the data-sets tested.

The report is organized as follows. Chapter 2 provides a brief overview of PUNCH and on-demand network-computing. Chapter 3 discusses the domain characteristics that affect the selection of the learning algorithm. Chapter 4 introduces the AI sub-system and Chapter 5 presents the experimental evaluation and results. Finally, the conclusions of this work and a discussion of on-going work are presented in Chapter 6 and Chapter 7, respectively.

---

1. Memory and disk-space requirements are not predicted. This is due to the lack of a monitoring system that allows these parameters to be measured accurately, and not a limitation of the AI system.

2. The Purdue University Network Computing Hub

The Purdue University Network Computing Hub (PUNCH)[1] is a WWW-accessible collection of simulation tools and related information. Functionally, it allows users to: a) upload and manipulate input-files, b) run programs, and c) view and download output - all via standard WWW browsers. For a detailed description of PUNCH, see Kapadia, Fortes, and Lundstrom (1997).

PUNCH can be logically divided into multiple discipline-specific "hubs" (see Figure 2.1). Currently, PUNCH consists of four hubs that contain tools from semiconductor technology, VLSI design, computer architecture, and parallel processing. A fifth hub is devoted to tools that were developed with support from the Semiconductor Research Corporation (SRC). These hubs contain over thirty tools from five universities and serve more than 500 users from within Purdue, across the US, and in Europe.

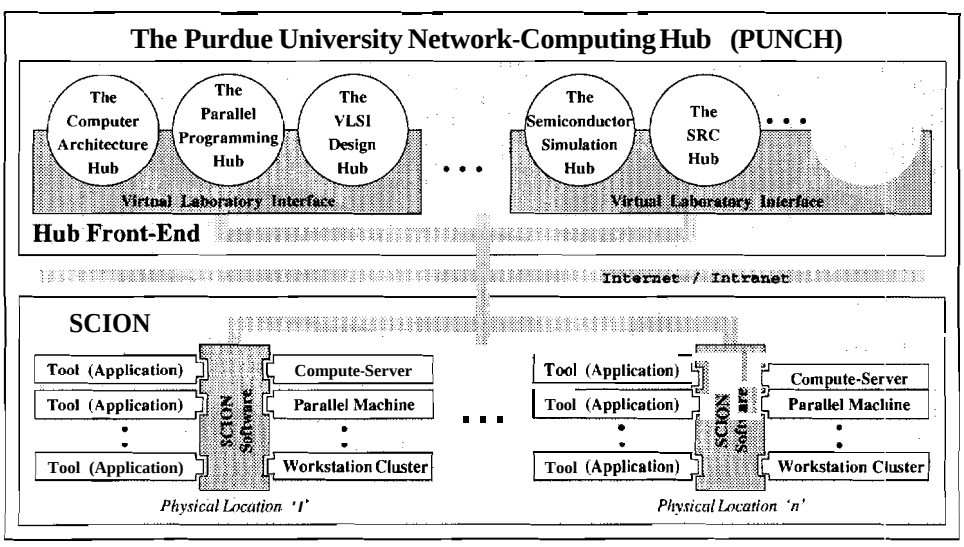

Figure 2.1: A conceptual view of the Purdue University Network-Computing Hub.

Running a simulation on the hub is a three-step process. The first step involves the creation of the input file(s) required for the relevant simulation. In the second step, users define the input parameters (e.g., command-line arguments, etc.) for the program and start the simulation.

1. The Purdue University Network Computing Hub can be accessed at "http://www.ecn.purdue.edu/labs/punch/". Courtesy accounts with access to a limited number of tools are available.

Finally, after the simulation is complete, the user can see and download the results via the hub's output interface.

PUNCIH can be viewed as an operating-system for the world-wide web (see Figure 2.1). Users communicate with the PUNCH infrastructure via a front end (equivalent to an operating-system shell) that allows them to access and use distributed resources in a location-transparent manner. The front end processes user requests by way of a distributed engine (akin to an OS kernel) that can access and control local and remote hardware and software resources. Hardware resources can include arbitrary platforms and software resources include any arbitrary program (the current implementation provides limited support for GUI-based programs). Resources can be located at any network-accessible site, and can be dynamically added or removed from the infrastructure.

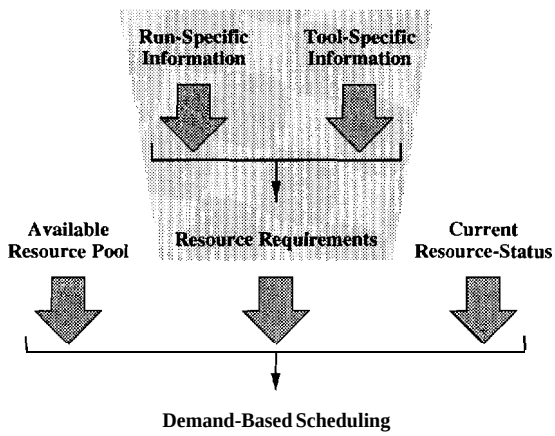

Figure 2.2: A flow-chart depicting the components required to perform on-demand scheduling. The shaded area represents the scope of the work discussed in this report.

PUNCH allows on-demand management of existing software and hardware resources by delaying the binding of a user's command to a specific implementation and machine until run-time, at which point the requirements of the given run can be analyzed (see Figure 2.2). The resource-requirements of a particular run are determined by PUNCH's AI sub-system, which qualifies the user-supplied tool-input with available tool-specific scalability and portability information. The output of the AI system is then used to match a user's request with the underlying network-accessible tools and resources.

3. Domain Characterization

3.1 Introduction

The domain-imposed constraints that determine the selection of the learning algorithm can be divided into two categories. The constraints in the first category are a result of the diversity of the hub tools and users, while those in the second set are a consequence of the nature of the run-time environment associated with a network-computing infrastructure.

3.2 Tool Characteristics

The tools available on PUNCH come from a wide variety of environment:; and disciplines. Each tool requires its own set of features and a separate knowledge base. When possible, the list of relevant features for predicting the resource-usage characteristics of a tool is obtained by consulting the appropriate authors. Otherwise, the list is compiled with the help of an expert in the field. In general, establishing the correct (i.e., relevant) features for a given tool is a difficult problem. Authors are often not accessible, and realistic tools tend to use sophisticated algorithms whose behavior cannot be easily correlated to the user-supplied input values. Another problem associated with the feature-vector is that the range of values that a given feature can assume is generally not known a priori, particularly in a research environment. Even when such information is available, the limits (e.g., how small a semiconductor device can be) tend to be technology-dependent. In terms of the artificial intelligence system, these issues require that the system be able to: a) ignore irrelevant features, b) detect inadequate feature-vectors, and c) work with unscaled ***features.***[1]

The relationship between the 'n' inputs supplied to a program and the corresponding resource-usage characteristics is defined by a set of polynomials in n-dimensional *space*.[2] Thus, the learning algorithm used for this domain must be able to capture concepts described by (possibly multiple) polynomial functions. Moreover, this relationship often has a non-deterministic component with respect to the available inputs. For example, the convergence rate of an iterative matrix-manipulation algorithm is likely to depend on the distribution of the eigenvalues of that matrix, which are difficult to compute in advance. This effectively implies

1. Note that "unscaled" in this context implies that the system cannot use a constant scaling factor that has been determined a priori; it can still scale features on the fly.
2. Recall that any function can be represented as a polynomial, although this may not be the most concise representation.

that the learning algorithm will have to work with an incomplete or noisy description of the features that determine the resource-usage characteristics of the program.

3.3 Run-Time Environment

When a request for a run is received by PUNCH, it extracts the values of the administrator-specified features from the user-supplied input and uses thern to predict the resource-usage characteristics. The prediction is then used to determine how and where to schedule the request. After the run completes, PUNCH provides the true resource-usage characteristics to the artificial intelligence system, allowing the learning algorithm to incorporate the new information into its knowledge base. Because this process happens in real-time and during norrnal use of the system, an incremental learning approach is needed.

The run-time environment is also interactive, which requires the predictions to be made in real-time. This in turn implies that the resources used by the artificial intelligence system cannot grow monotonically with time.

The final issue that affects learning is short-term variations in the performance of computer systems. Short-term variations in performance can occur due to unpredictable events such as a file-server or network router becoming overloaded. While these short-term anomalies essentially amount to noise in the long run, they tend to have a significant impact on run-time when they do exist. Thus, the learning algorithm must be able to quickly tailor its predictions to such short-term variations without being unduly affected by them in the longer term.

4. The Artificial Intelligence System

4.1 Introduction

The learning mechanism of our approach is based on locally weighted regression (LWR). In this chapter, we first present the rationale for the selection of this particular instance-based learning method. We then discuss the learning issues that are specific to this domain and the modifications that we made to LWR to handle these issues.

4.2 Algorithm Selection

Of the requirements presented in Chapter 3, the following are central to the process of selecting a learning algorithm: a) an ability to learn sets of polynomial functions, b) incremental learning, and c) support for irrelevant and unscaled features. These requirements directly contribute to the applicability of local learning algorithms (specifically, instance-based learning algorithms).

Global parametric learning algorithms (Schaal, 1994) such as neural networks attempt to establish an input-output mapping via a single function $y = f(x, \theta)$, where θ is a finite-length parameter .vector. While these methods can theoretically approximate any continuous function (Funahashi, 1989; Hecht-Nielson, 1989; Skapura, 1996), they may not be appropriate for all tools. For example, semiconductor device simulation tools typically allow users to simulate a device in one, two, or three dimensions. In general, different solution techniques are used for each of these cases, implying that the input-output mapping for such tools will consist of three distinct functions. This is likely to cause problems for learning algorithms that attempt to capture concepts at a global level.

Local parametric algorithms attempt to overcome some of the proiblems of global parametric learning by dividing the input space into many partitions (Atkeson, Schaal, and Moore, 1997; Schaal, 1994). Each partition 'i' is now approximated by an independent function $y_i = f_i(x, \theta_i)$; the functions f_i are kept as simple as possible. The problem now shifts to the selection of appropriate partitions for the learning system (Schaal and Atkeson, 1994). Non-parametric algorithms (e.g., Atkeson, Schaal, and Moore, 1997; Moore, Schneider, and Deng, 1997; Deng and Moore, 1995) address this issue by allowing the number of partitions (and consequently the number of parameters) to change dynamically. Instance-based learning (IBL) algorithms achieve this by recomputing a fixed set of parameters as a function of the query point and do not require an explicit training phase (Deng and Moore, 1995). Moreover, because of their localized nature, IBL algorithms are relatively insensitive to the structural complexity of

the function to be learned and are not affected by catastrophic interference (Schaal, 1994). This makes them an ideal choice for this domain.

There are many instance-based learning algorithms, including nearest neighbor, weighted average (kernel regression), and locally weighted regression techniques (e.g., Atkeson, Moore, and Schaal, 1996). Nearest neighbor algorithms use the output-value(s) of the closest available instance(s) to make a prediction. Weighted average algorithms predict the output as a weighted average of the output-values of nearby instances; the weight of an instance is an inverse function of its distance from the query point. Locally weighted regression (LWR) fits a surface to nearby points, typically via a locally linear or quadratic model.[1] With a linear (quadratic) model, the target concept is locally approximated by a linear (quadratic) surface.

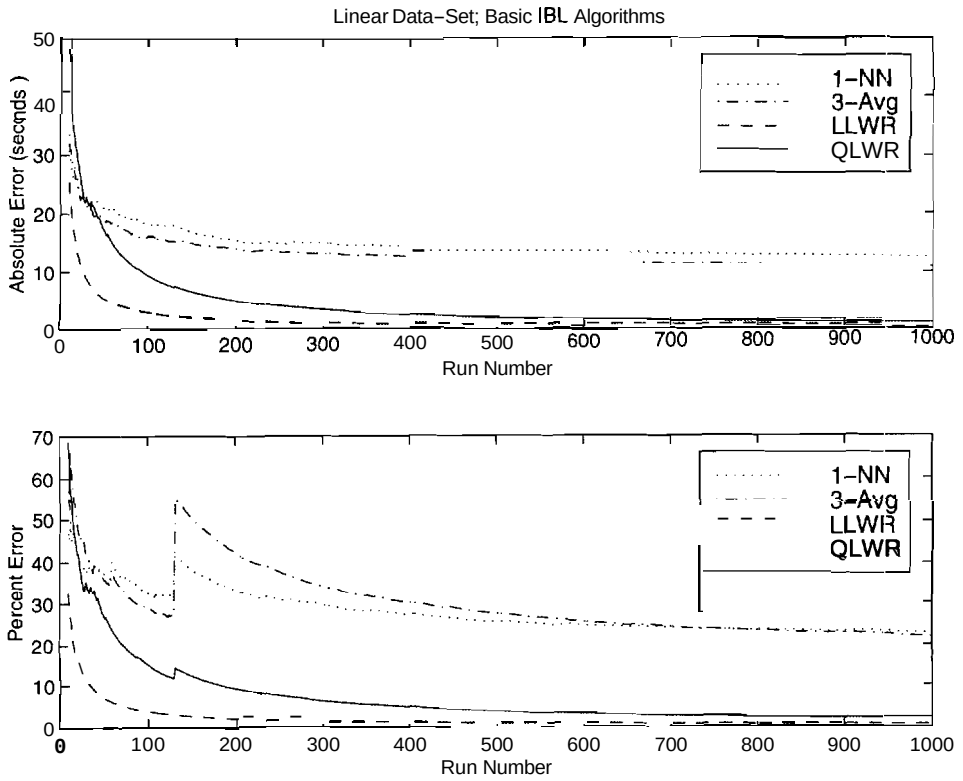

Figure 4.1: Prediction errors for instance-based learning techniques on instances (from a synthetic data-set) whose output-values are linearly-dependent on the feature-vector. Note that the the first few runs have been omitted from the plots for improved readability.

The nearest neighbor and weighted average techniques are not suitable for this domain because of their inability to track (even linear) polynomial surfaces without error (Cleveland, Devlin, and Grosse, 1988). This is illustrated in Figure 4.1, which shows the prediction errors for the one-nearest neighbor (1-NN), three-point weighted average (3-Avg), locally linear LWR (LLWR), and locally quadratic LWR (QLWR) algorithms on a synthetic data-set. The data-set

1. Higher-orcler local models are generally not used because of the associated computational cost.

was made up of 1,000 instances with randomly-generated feature-vectors. Each feature-vector contained seven features. The "measured" resource-usage characteristics (the figure shows the simulated CPU time) were linear functions of the corresponding feature-vectors. The coefficients for the linear functions were also chosen randomly.

In addition to being able to reproduce linear surfaces without error, locally weighted regression algorithms (e.g., Cleveland and Devlin, 1988; Moore, 1991; Atkeson, 1992; Schaal and Atkeson, 1994) can reproduce peaks and are insensitive to unsymmetrically distributed data (Cleveland. Devlin, and Grosse, 1988; Grosse, 1989; Schaal, 1994). This makes locally weighted regression an ideal choice for the given domain. The locally linear model is chosen over the locally quadratic model for two reasons: a) it learns faster (for a locally linear surface; see Figure 4.1), and b) it requires less time to make a prediction. Both are consequences of the fact that, for a feature-vector of length 'n', the LLWR algorithm requires only O(n) parameters to make a prediction as opposed to the $O(n^2)$ for the QLWR algorithm.

4.3 Learning Issues

The basic LLWR learning algorithm addresses the following issues: a) learning sets of polynomial functions, b) incremental learning, and c) support for irrelevant and unscaled features. Modifications are required to address: a) detection of inadequate feature-vectors, b) short-term variations, c) noisy features, and d) scalability of the knowledge base during extended use.

Of the listed issues, the last one is the most critical because the basic IBL algorithms do not scale well-enough for extended use in the PUNCH environment. This is exemplified by Figure 4.2, which shows the monotonically increasing nature of the instance-base size and the average per-prediction lookup time.

The subsequent sections present solutions for each of the problems mentioned here. Detection of inadequate feature-vectors is addressed by storing appropriate meta-information about the instances in the knowledge-base. Sensitivity to short-term variations without an associated loss in longer-term performance is obtained by using a two-level knowledge base, which alsc helps the IBL algorithms scale better. Finally, scalability and noise issues are addressed by: a) not adding all instances to the knowledge-base, and b) allowing instances to be discarded from the knowledge-base.

4.4 Knowledge Representation

In a realistic computing environment, multiple runs with identical feat-ure-vectors often exhibit different resource-usage characteristics. This is caused by noise in the computing environment, inadequate feature-vectors, or a combination of the two. The noise perceived by the learning algorithm is generated by: a) inaccuracies in the system used to monitor resource-usage, and b) indirect (via the computing environment) interactions between the different processes running on a machine. These considerations were the driving factors for the design of

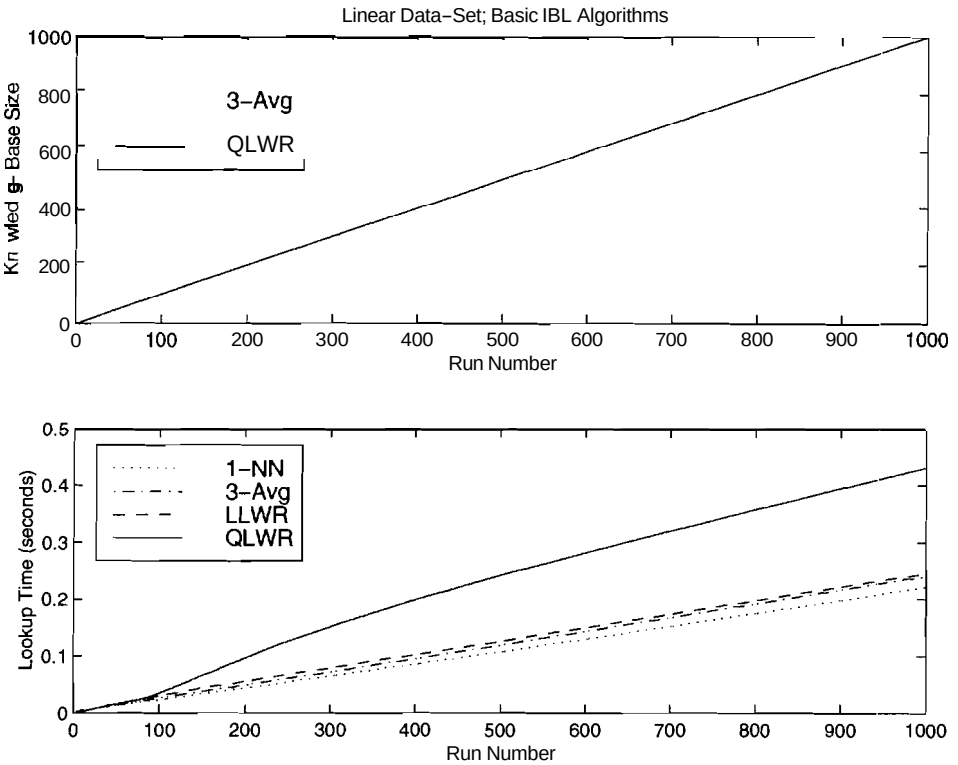

Figure 4.2: Knowledge-base growth and corresponding per-prediction lookup time for instance-based learning techniques on instances whose output-values are linearly-dependent on the feature-vector. The feature-vector contains seven features. Note that the size of the knowledge-base is the same for all four methods.

the chosen knowledge representation.

The information available to the AI system at the completion of a run consists of: a) the feature-vector, b) the measured resource-usage characteristics, and c) the start-time for the run. If the feature-vector is adequate, the measured resource-usage characteristics will generally not vary outside of a small range for multiple occurrences. For these situations, we can coalesce (multiple) observations within a given range (±10%, say) into a single averaged observation with an associated ''use count''. The averaged observation is defined as an ***experience.***

The knowledge associated with a given set of experiences with identical feature-vectors is represented as shown in Figure 4.3. Observe that the knowledge is keyed to the feature-vector. As explained above, each experience contains usage statistics and the associated (average) resource-usage characteristics. The multiple use-counts allow concept drift (Schlimmer and Fisher, 1986) to be detected (see our discussion of future work in Chapter 6). The resource-usage prediction for a given feature-vector is precomputed by calculating the weighted average of the resource-usage characteristics of the set of experiences associated with that feature-vector. Note that infrequently observed experiences can be treated as outliers and excluded from the average. The age associated with the feature-vector is the age of its earliest experience. The usage statistics consist of: a) the number of times the feature-vector was observed, b) the number

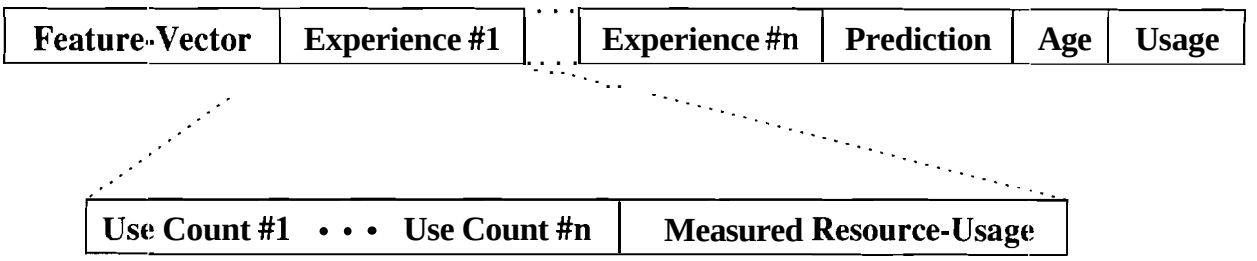

Figure 4.3: Each feature-vector is associated with a set of experiences, the corresponding average resource-usage, and age and usage information. An experience contains knowledge associated with one or more instances.

of times the feature-vector was used for an interpolated query, and c) the number of times the use of the feature-vector resulted in an accurate prediction. Together, the age and usage statistics allow the AI system to detect and discard noisy feature-vectors from the knowledge-base. (The specific manner in which this information is used to achieve noise-tolerance is described after the discuss .on of knowledge-base organization and knowledge retrieval.)

Inadequate feature-vectors can be detected by analyzing the distribution of the associated experience!;. A highly-peaked distribution indicates (with high probability) a good correlation between the feature-vector and the corresponding resource usage. On the other hand, a multi-modal distribution is a definite indication of an inadequate feature-vector. Currently, we do not utilize this information, but anticipate that, in the future, we will use it to trigger a second knowledge acquisition phase in which we query the expert for additional features.

4.5 Knowledge-Base Organization

The organization of the knowledge-base was driven by two considerations: a) short-term variations in the behavior of computing resources, and b) temporal and spatial. locality of runs. For this domain, the principle of temporal locality can be stated as follows. If a run with a given feature-vector is invoked at some time 't', it is likely to be invoked again at some time 't + At'. This is especially true in an academic environment, where a relatively large number of students tend to work concurrently on any given assignment. Similarly, the principle of spatial locality can be stated as follows: if a run with a given feature-vector is invoked at some time 't', runs with similar feature-vectors are likely to be invoked in the near future. This assumption applies to users who, for example, need to characterize a system by perturbing a few parameters at a time (characteristic of a research environment).

Based. on these considerations, a two-level knowledge-base was selected. The first level of the knowledge-base acts as a fixed-size cache, representing the short-term *memory* of the system. When an instance is encountered, it is first stored in the cache. The instance-filtering algorithms described later are not applied to the cache. This allows the learning algorithm to memorize instances in the short-term. During the process of making a prediction, the cache is searched first, which allows the learning algorithm to bias its predictions toward recent information. When

the cache overflows, the least-recently-used instances in the cache are either discarded or incorporated into the long-term memory, based on a specific policy described after the discussion on knowledge retrieval.

4.6 Knowledge Retrieval

In order to retrieve the resource-usage characteristics of a given feature-vector, the learning algorithm scans the two-level knowledge-base in the following manner. It first looks in the cache for an exact match to the query in terms of the feature-vector. If a match is found, the algorithm makes its prediction on the basis of the precomputed characteristics associated with that feature-vector. If a match is not found, the process is repeated with the second level of the knowledge-base. The time-associated performance advantages of the two-level organization1 are based on the supposition that a match will be found in the cache for a significant number of requests.

If an exact match is not found in either level of the knowledge-base, the learning algorithm retrieves the $2 \times (n + 1)$ feature-vectors that are closest to the query. Here, 'n' is the length of the feature-vector. Recall that a linear polynomial with 'n' unknowns contains 'n + 1' terms. This implies that, at a minimum, 'n + 1' feature-vectors are required to obtain a unique solution to the query (with LLWR). The learning algorithm uses 'n + 1' additional feature-vectors to compensate for noisy and "non-independent" vectors (e.g., feature-vectors that have identical values in a given dimension do not provide any information about that dimension).

4.7 Knowledge Management Policies

This section focuses on modifications to the basic IBL algorithms designed to address scalability and noise issues. Basic instance-based learning techniques incorporate all instances into the knowledge-base. As shown earlier (Figure 4.2), this results in a monotonically increasing knowledge-base, making the AI system unscalable. A survey of techniques that address scalability and noise issues can be found in Wilson and Martinez (1997).

Given the characteristics of this domain and the knowledge representation discussed earlier, there are two ways to address this problem. The first option is to selectively incorporate only incorrectly-predicted feature-vectors into the knowledge base. The second option is to discard knowledge associated with feature-vectors that have been consistently used to make incorrect predictions.

The first option can be expected to work well in situations where the learning algorithm is able to capture the target concept. This is exemplified by Figure 4.4. The LLWR and QLWR algorithms are able to learn linear concepts, which results in a self-bounded knowledge base. Note that the concept only has to be locally linear (quadratic) for LLWR (QLFVR) to capture it; its global structure can be much more complex. The 1-NN and 3-Avg algorithms, on the other hand, are less dramatically affected (compare the Y-Axis range with that of Figure 4.2) because they are not able to learn the concept with a finite number of instances. Observe that selectively incorporating knowledge could have a negative impact on the learning rate of the system. For

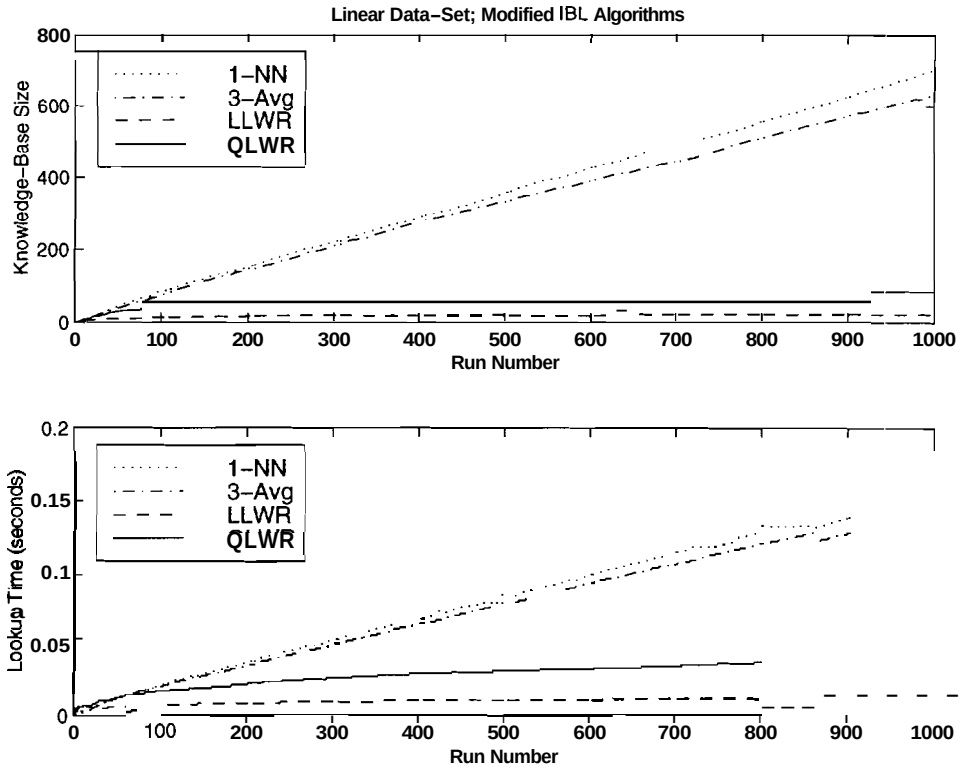

Figure 4.4: Instance-base growth and corresponding per-prediction lookup time for instance-based learning techniques on instances whose output-values are linearly-dependent on the feature-vector. The feature-vector contains seven features.

example, in some cases, an instance that was discarded in the past could have resulted in a better prediction for the current query. While this problem cannot be completely eliminated, it is partially addressed by the two-level knowledge base. All instances are incorporated into the cache regardless of the current policy. When the cache overflows, instances are incorporated into the second level according to the current policy.

With the second option, knowledge associated with feature-vectors that have consistently been used to make incorrect predictions is discarded.[2] The keepldiscard decisions are made periodically; feature-vectors that do not have adequate use statistics associated with them are allowed to retain their history for the next time-frame (the history is reset once a keepldiscard decision is made). In practice, the described policy is not enforced until after a certain number of runs have been observed, to allow for errors before the concept is learned. Note that this approach is similar to the technique used by Aha and Kibler (1989).

Finally, to account for situations in which these two heuristics fail, the size of the knowledge base has a hard upper bound associated with it. When the size exceeds a specified

2. For the results presented here, "consistent" was arbitrarily defined as an error-rate of more than 50% in a 50-run time-frame.

threshold, a LRU policy (e.g., Hennessy and Patterson, 1996) is used to discard feature-vectors.

4.8 Implementation Issues

To ensure numerical stability, the regression matrices were solved by using singular value decomposiiion (Press, Teukolsky, Vetterling, and Flannery, 1992). When a unique solution was not possible, the solution that minimizes the norm of the vector formed by the polynomial coefficients, (Press, Teukolsky, Vetterling, and Flannery, 1992) was chosen. Finally, the kernel width (Atkeson, Moore, and Schaal, 1997) used by the learning algorithm was dynamically adjusted so that it stayed flat until a distance equal to that of the nearest-neighbor (it was Gaussian after that). This ensured that at least one set of experiences was always available for prediction.

5. Experimental Evaluation and Results

5.1 Introduction

In this application, there are three performance criteria: a) the prediction error, b) the time required for prediction, and c) the growth-rate of the knowledge-base.

When minimizing the prediction error, it is more important to reduce errors for runs that make heavy use of resources. For example, a 100% prediction error for a run that uses only a few seconds of CPU time is relatively insignificant, whereas the same error for a run that uses several thousand CPU seconds is likely to have a significant impact on the effectiveness of the scheduler. 'This distinction can be accounted for by tracking both the absolute and the percent prediction error. For a given absolute error, a higher percent error indicates that the prediction errors are biased towards the shorter runs.

Because the predictions must be made in real-time, it is important to place an upper bound on the prediction time. Ideally, this time should be significantly smaller than the shortest runs invoked by users. A related criterion is to ensure that the growth-rate of the knowledge-base eventually decreases to zero. This requirement does not impose a specific upper-bound on the size of the Itnowledge base, but does require that the knowledge-base not grow indefinitely.

5.2 Data-Sets

The learning system was tested on three semiconductor simulation tools (T-Suprem3, Minimos, and S-Demon) during normal use of the hub in Fall 1997. Runs consisted of simulations for class projects and homework assignments.

T-Suprem3 is a commercial package (from Technology Modeling Assocsiates, Inc.) that simulates the processing steps used in the manufacture of silicon integrated circuits and discrete devices. Minimos simulates semiconductor field-effect transistors in two and three dimensions (developed at the Technical University in Vienna). S-Demon uses the Monte Carlo technique to simulate electron transport through one-dimensional silicon devices (developed at Purdue University). The learning instances collected for T-Suprem3, Minimos, and S-Demon comprised of 3398,966, and 131 runs, respectively.

The system was also tested with four synthetic data-sets. Each of these data-sets consisted of 1,000 instances with *randomly generated* feature-vectors. In the first two data-sets, the resource-usage characteristics were linear and quadratic functions of the feature-vector, respectively-. The weights for these functions were selected randomly. The thircl and fourth data-

sets were equivalent to the first two (linear and quadratic, respectively), except that randomly generated noise was injected into the feature-values and the resource-usage characteristics. The noise perturbed the measured characteristics for the linear (quadratic) data-set by 10% (37%) on an average.

This report presents detailed results for T-Suprem3; results for the other data-sets showed similar trends. The features associated with T-Suprem3 characterize aspects of the fabrication process of a semiconductor device. Specifically, the feature-vector was made up of the following: a) number of grid points, b) total diffusion time, c) cumulative epitaxial growth (in terms of thickness), d) minimum implant energy, e) number of deposit steps, f) number of etch steps, and g) number of implant steps.

5.3 Results

The results in this section focus on the errors associated with the prediction of CPU time because of' its importance in terms of scheduling. For convenience, the different policies described earlier are named as follows. The basic IBL approach using LLWR is called 'basic'. The policy that does not add accurately predicted feature-vectors into the knowledge base is called 'noadd'. The policy that deletes feature-vectors that consistently result in bad predictions from the knowledge base is called 'noisetol'. Finally, the combined application of 'noadd' and 'noisetol' is called 'combined'.

Prediction Error. The cumulative prediction error plots associated with the: basic policy (Figure 5.1) confirm that the AI system is able to learn the relationship between the run-time inputs and the resource-usage characteristics of T-Suprem3. Error prediction plots for the other policies show similar trends; the final cumulative errors for all policies are shown in Table 5.1. Observe that discarding feature-vectors that consistently result in incorrect predictions (noisetol and combined policies) considerably improves the prediction accuracy. Also note that the prediction error associated with the noadd policy is essentially the same as that of the basic policy, indicating that discarding feature-vectors did not have a negative impact on the system's overall ability to learn. Finally, Table 5.1 shows that caching reduces the prediction error for the noadd and combined policies, which do not add (to the knowledge base) feature-vectors whose resource-characteristics can be predicted accurately. This is especially true for short runs, as indicated by the larger change in the percent error (versus the absolute error).

Long-Tern1 Scalability. The first plot in Figure 5.2 shows the growth rate of the knowledge base for the basic policy. As expected, the knowledge base grows monotonically. The lookup time in the second plot is a function of the size of the knowledge base, and consequently also increases monotonically. The data in the second plot also indicates the beneficial effects of the two-level knowledge base. A cache size of five reduces the average lookup time by more than a factor of two. Increasing the cache size further does not significantly affect the lookup time,

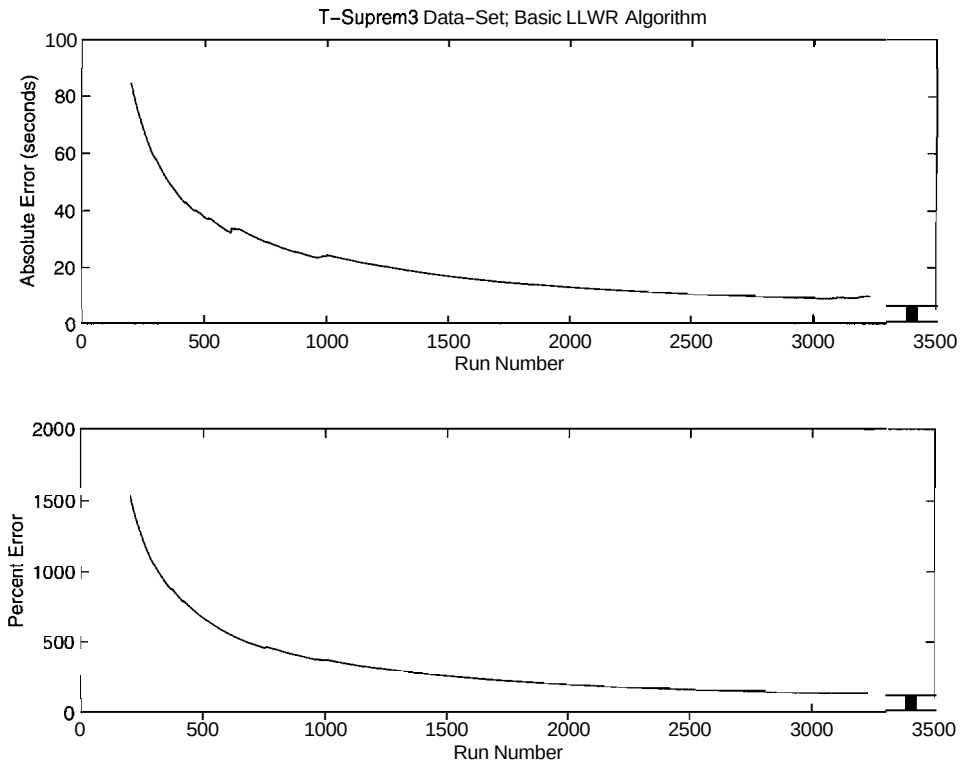

Figure 5.1: Cumulative prediction error plots for T-Suprem3 with the `basic` policy. Note that the the first 200 runs have been omitted from the plots for improved readability.

Table 5.1: Cumulative error statistics for T-Suprem3. For each policy, the absolute and percent errors (for two cache sizes) are presented. The absolute error is in 'seconds'.

Cumulative Prediction Error				
Policy	**Cache Size = 0**		**Cache Size = 20**	
	Abs	Percent	Abs	Percent
`basic`	9.77	131.97	9.77	131.97
`noadd`	9.36	126.93	9.06	119.68
`noisetol`	5.78	46.58	5.72	45.01
`combined`	6.37	62.46	6.26	50.77

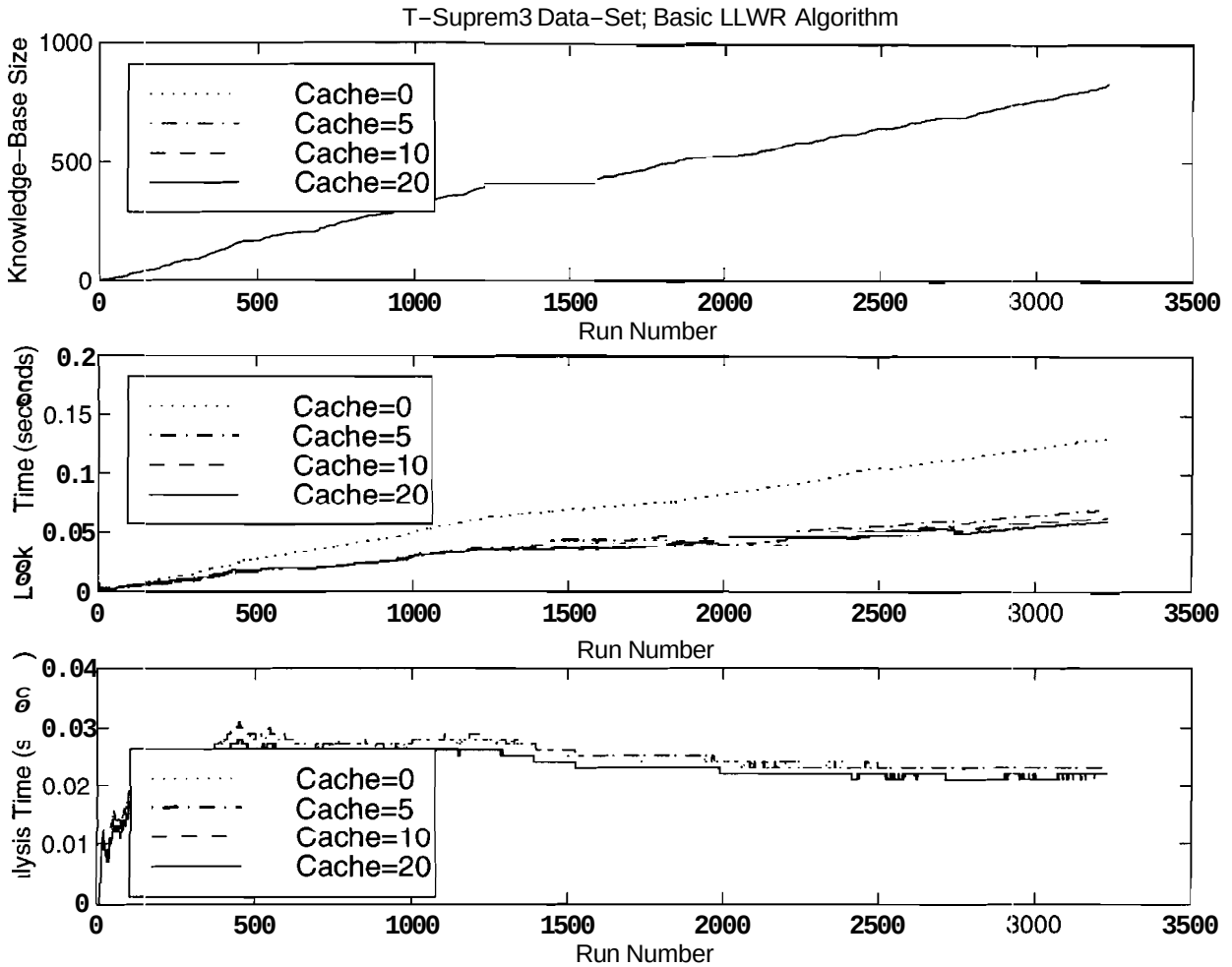

Figure 5.2: Scalability-related information for the basic policy. The first two plots show the number of feature-vectors in the knowledge-base and the average per-prediction lookup time for the LLWR algorithm on the T-Suprem3 data-set. The last plot shows the average analysis time, as explained in the text.

indicating that a short-term memory of only five runs was adequate to capture the spatial and temporal locality discussed in Chapter 4. The analysis time shown in the third plot is equal to the time required to compute the regression matrices if a matching feature-vector is not found in the knowledge base. If a match is found, the analysis time is zero. The average analysis time is only dependent on the number of feature-vectors used for regression, which is a bounded value. Consequently, once an adequate number of feature-vectors are available, the analysis time is approximately constant. Note that the analysis time subsequently decreases sornewhat as the AI system starts memorizing feature-vectors. Corresponding results for the `noadd` policy show a similar trend, indicating that the AI system was not able to learn the input-output mapping completely. This implies that the concept is not locally linear and/or has a lot of noise associated with it.

The same plots for the `combined` policy are shown in Figure 5.3. Note the self-bounded nature of the knowledge base in the first plot. The oscillations in the size of the knowledge base are caused by the periodic deletion of noisy feature-vectors from the knowledge base. Also observe that the size of the knowledge base is dramatically smaller than that for the `basic` policy (compare the range of values on the Y-Axes of Figures 5.2 and 5.3) and that the short-term memory does not significantly affect the overall size of the knowledge base. The second plot shows the lookup time, which, as expected, is bounded. As before, caching helps lower the lookup time. The third plot is similar to that of the `basic` policy, except that caching helps reduce the analysis time. The lower analysis time is a consequence of the smaller number of interpolated queries with higher cache sizes, as shown in Table 5.2. This also lowers the overall prediction error because the resource-usage characteristics for exact matches can be determined more accurately. (A reduction in the number of interpolated queries is accompanied by a corresponding increase in the number of exact matches.)

Finally, the scalability information for the different policies is summarized in Table 5.3. Observe the dramatic reduction in the size of the knowledge base for the `noisetol` and `combined` policies. Corresponding numbers in Table 5.1 show that this reduction is obtained without any loss in prediction accuracy (prediction accuracy actually increases). The table also indicates the relatively modest effects of caching on lookup time because of the small size of the knowledge base (recall that it still helps improve the analysis time, which is not dependent on the size of the knowledge base).

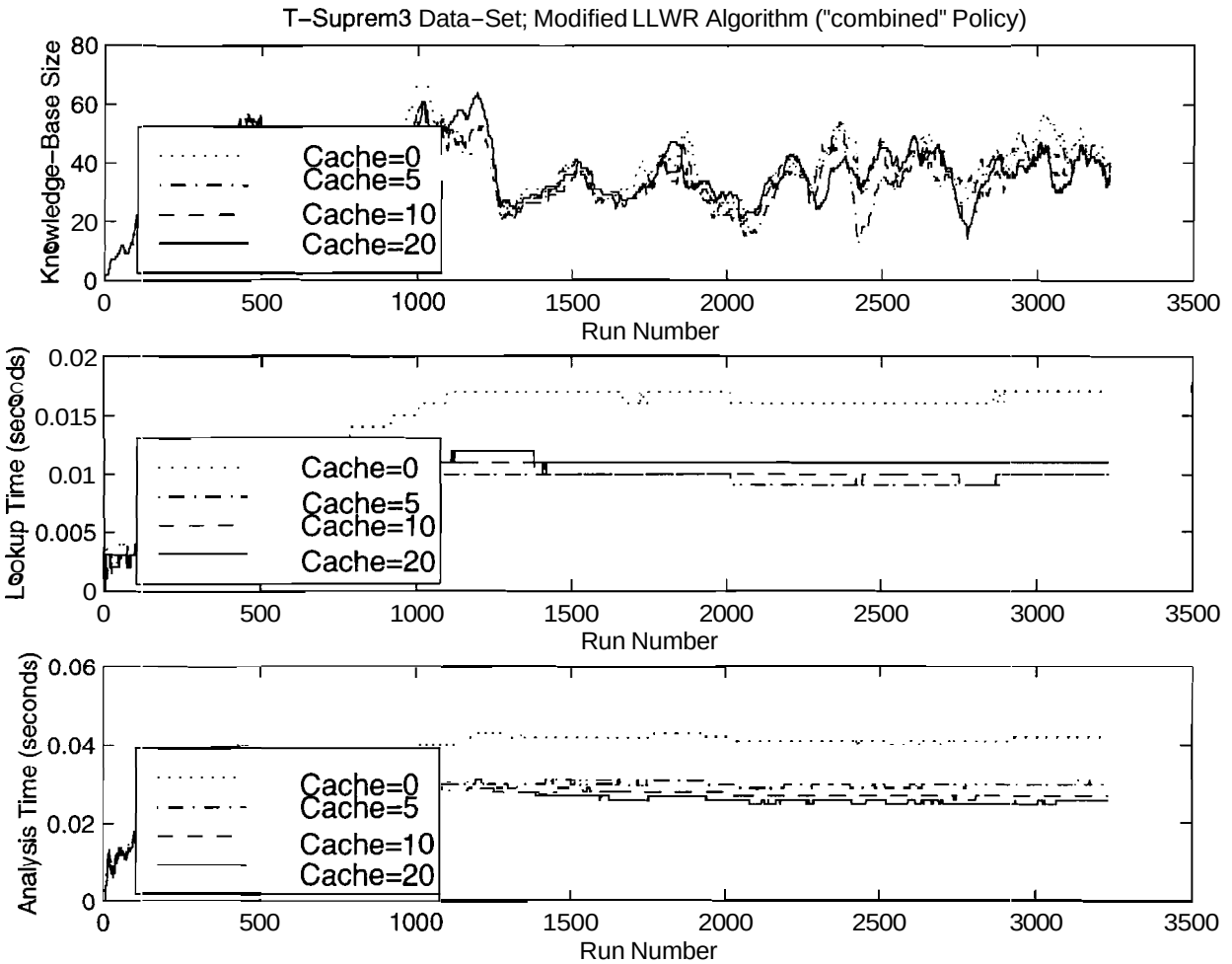

Figure 5.3: Scalability-related information for the `combined` policy. The first two plots show the number of feature-vectors in the knowledge-base and the average per-prediction lookup time for the LLWR algorithm on the T-Suprem3 data-set. The last plot shows the average analysis time, as explained in the text.

Table 5.2: Predictions can be based on exact matches or interpolated queries. This figure shows number of interpolated queries for T-Suprem3. Observe how the number increases with policies that limit the number of feature-vectors in the knowledge base (and then decreases with caching).

Number of Interpolated Queries				
Policy	**Cache Size**			
	0	5	10	20
`basic`	822	823	822	823
`noadd`	1504	1038	946	908
`noisetol`	986	990	993	993
`combined`	1626	1157	1030	1009

Table 5.3: Scalability statistics for T-Suprem3. Observe how caching helps reduce the lookup time.

Scalability Statistics				
Policy	**Cache Size = 0**		**Cache Size = 20**	
	#F-Vect	t_{lookup}	#F-Vect	t_{lookup}
`basic`	839	0.130	839	0.060
`noadd`	595	0.120	531	0.046
`noisetol`	39	0.017	39	0.013
`combined`	39	0.017	35	0.011

Table 5.2: Predictions can be based on exact matches or interpolated queries. This figure shows number of interpolated queries for T-Suprem3. Observe how the number increases with policies that limit the number of feature-vectors in the knowledge base (and then decreases with caching).

Number of Interpolated Queries				
Policy	**Cache Size**			
	0	5	10	20
`basic`	822	823	822	823
`noadd`	1504	1038	946	908
`noisetol`	986	990	993	993
`combined`	1626	1157	1030	1009

Table 5.3: Scalability statistics for T-Suprem3. Observe how caching helps reduce the lookup time.

Scalability Statistics				
Policy	**Cache Size = 0**		**Cache Size = 20**	
	#F-Vect	t_{lookup}	#F-Vect	t_{lookup}
`basic`	839	0.130	839	0.060
`noadd`	595	0.120	531	0.046
`noisetol`	39	0.017	39	0.013
`combined`	39	0.017	35	0.011

6. Conclusions and Future Work

6.1 Conclusions

Our results indicate that the described instance-based learning approach using locally weighted regression works well for this domain. Selectively adding feature-vectors into the knowledge base and discarding feature-vectors that consistently result in inaccurate predictions make the described instance-based learning approach scalable and tolerant to noise.

A two-level knowledge base: a) accounts for short-term variations in compute-server and network performance, and b) exploits temporal and spatial locality of runs. The chosen knowledge representation allows inadequate feature-vectors to be detected.

Experimental data collected during normal use of the Purdue University Network Computing Hub validates the assumptions of temporal and spatial locality. The use of a two-level knowledge base, which exploits these assumptions, results in reduced prediction error, faster retrieval of feature-vectors, and smaller (average) analysis time. It should be mentioned that the performance benefits of caching observed here are likely to be an upper bound. This is because of the fact that the conditions in an academic environment are particularly conducive to the locality suppositions. To some extent, however, an increased cache size will compensate for a reduction in locality. Further benefits may be possible if a three or higher level knowledge base is used.

6.2 Future Work

The logical extension to the described work is to apply it to a larger number of tools. In addition, there are five domain-related issues which are being addressed.

The AI system described here predicts the CPU time for a given run. From a user's perspective, it is more desirable to minimize the ***response time*** (i.e., elapsed time), which depends on the current load on the target machine. This requires the machine load to be incorporated into the feature-vector.

The current AI system does not consider the heterogeneity of hardware resources. In order to account for this, the AI system must learn scaling factors for each architecture, in addition to the tool-specific resource-usage characteristics.

Computing systems exhibit long term variations in characteristics that are typically the result of an upgrade of one or more components of the system (e.g., a new version of the operating system or compiler). In order to adapt to such changes, the learning algorithm must

account for concept drift. A related issue is the ability to track rapidly-changing concepts (e.g., network traffic). It should be possible to do this by heavily biasing the learning algorithm towards more recent observations and discarding older observations.

The final issue is to exploit the predictability of long-term resource-usage trends. Demands on computational resources tend to follow patterns that can be learned (certain resources are generally over-loaded during the late-afternoon hours, for example). An AI-based approach to resource-allocation should be able to exploit them to learn an anticipatory scheduling policy.

References

Aha, D. PI., and Kibler, D. 1989. Noise-Tolerant Instance-Based Learning Algorithms. In Proceedings of the 11th International Joint Conference on Artificial Intelligence, 794-799. Morgan Kaufmann.

Atkeson, C.G., Schaal, S. A., and Moore, A. W. 1997. Locally Weighted Learning. *AI Review* 11:11-73. Kluwer Publishers.

Cleveland, W. S., and Devlin, S. J., 1988. Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting. *Journal of the American Statistical Association* 83(403):596-610.

Cleveland, W. S., Devlin, S. J., and Grosse, E. 1988. Regression by Local Fitting: Methods, Properties, and Computational Algorithms. *Journal of Econometrics* 37:87-114.

Comerford, R. 1997. The Battle for the Desktop. *IEEE Spectrum* 34(5):21-28.

Deng, K., and Moore, A. W. 1995. Multiresolution Instance-Based Learning, In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI).

Devarakonda, M. V., and Iyer, R. K. 1989. Predictability of Process Resource Usage: A Measurement-Based Study on UNIX. *IEEE Transactions on Software Engineering* 15(12):1579-1586.

Funahashi, K. 1989. On the Approximate Realization of Continuous Mappings by Neural Networks, *Neural Networks* 2:183-192.

Goswami, K. K., Devarakonda, M., and Iyer, R. K. 1993. Prediction-Based Dynamic Load-Sharing Heuristics. *IEEE Transactions on Parallel and Distributed Systems* 4(6):638-648.

Grosse, E. 1989. LOESS: Multivariate Smoothing by Moving Least Squares. In Approximation Theory VI: Volume I, 299-302. Chui, C. K., and Ward, J. D. eds. Academic Press.

Hecht-Nielson, R. 1989. Theory of the Backpropagation Neural Network, In Proceedings of the International Joint Conference on Neural Networks, 593-611. Washington.

Hennessy, J., and Patterson, D. A. 1996. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers.

Kapadia, N. H.; Fortes, J. A. B; and Lundstrom, M. S. 1997. The Semiconductor Simulation Hub: A Network-Based Microelectronics Simulation Laboratory. In Proceedings of the 12th Biennial IEEE University Government Industry Microelectronics Symposium, 72-77. Rochester, New York: IEEE.

Moore, A. W., Schneider, J., and Deng, K. 1997. Efficient Locally Weighted Polynomial Regression Predictions, In Proceedings of the 1997 International Machine Learning Conference.

Press, W. W., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. 1992. *Numerical Recipes in C*. 2nd Edition.

Schaal, S. 1994. Nonparametric Regression for Learning. In Proceedings of the Conference on Adaptive Behavior and Learning. Center for Interdisciplinary Research, University of Bielefeld, Germany. Also technical report TR-H-098, ATR Human Information Processing Research Laboratories.

Schaal, S., and Atkeson, C. G. 1994. Assessing the Quality of Learned Local Models. In *Advances in Neural Information Processing Systems 6*, Cowan, J., Tesauro, G., and Alspector J., eds. Morgan Kaufmann.

Schlirnrner, J. C., and Fisher, D. 1986. A Case Study of Incremental Concept Induction. In Proceedings of the National Conference on Artificial Intelligence, 496-501. AAAI.

Skapura, D. M. 1996. *Building Neural Networks*. ACM Press.

Smarr, L., and Catlett, C. E. 1992. Metacomputing. *Communications of the ACM* 35(6):45-52.

Svensson, A. 1990. History, an Intelligent Load Sharing Fiter. In Proceedings of the 10th International Conference on Distributed Computing Systems, 546-552.

Wang, C. J., Krueger, P., and Liu, M. T. 1993. Intelligent Job Selection for Distributed Scheduling. In Proceedings of the 13th IEEE International Conference on Distributed Computing Systems, 517-524.

Wilson, D. R., and Martinez, T. R. 1997. Reduction Techniques for Exemplar-Based Learning Algorithms,.Forthcoming.