01 Jan 1994

Parameter Tuning for the MAX Expert System

Christopher J. Merz
Missouri University of Science and Technology, merzc@mst.edu

M. J. Pazzani

Recommended Citation

C. J. Merz and M. J. Pazzani, "Parameter Tuning for the MAX Expert System," *Proceedings of the 6th International Conference on Tools with Artificial Intelligence, 1994*, Institute of Electrical and Electronics Engineers (IEEE), Jan 1994.
The definitive version is available at https://doi.org/10.1109/TAI.1994.346450

Parameter Tuning for the MAX Expert System

Christopher J. Merz, Michael Pazzani
ICS Dept
University of California, Irvine
Irvine, CA 92717
pazzani@ics.uci.edu
(714)856-5888

Abstract

We investigate methods for tuning numeric parameters in Nynex MAX, a telephone trouble screening expert system. Steepest descent, hillclimbing, and simulated annealing parameter adjustment strategies are applied to the problems of maximizing classification accuracy and minimizing misclassification cost. For both of those optimization problems we evaluate each algorithm's ability to tune initial parameters for several situations.

1.0 Introduction

This research was motivated by the problem of learning to troubleshoot a telephone network. NYNEX, the primary local phone company for New York and New England, has implemented a rule-based expert system, MAX ([2]), that is used to determine the location of a malfunction for customer-reported telephone troubles. The task, then, is to predict the location to which a repairman should be dispatched.

Like all expert systems, MAX requires occasional maintenance to its knowledge base. In addition, MAX is used at many different sites in New York and New England and there are small differences in how examples should be classified at each site. The designers of MAX have facilitated this customization by having a set of numeric parameters (e.g., indicating when a voltage is too high) that are set at each site or adjusted periodically to improve its performance. Small increases in the number of correct dispatches via improved parameters settings result is substantial savings in maintenance center operations. The goal of this research is to develop and evaluate strategies for tuning these parameters for optimal performance on a collection of examples.

Several approaches to adjusting the parameters have been investigated based on hillclimbing, steepest descent, and simulated annealing. These algorithms were designed to evaluate different search biases and their ability to optimize performance by escaping plateaus and local minima.

We evaluated each approach using two optimization criterion. The first is to minimize error rate (or to maximize accuracy) and the second is to optimize the misclassification cost by considering some errors to be more costly than others.

Experiments were conducted for three different scenarios evaluating the algorithms' abilities to adapt MAX to a new site, to improve upon the current settings, and to improve upon noisy approximate settings. The experiments reveal that each parameter tuning method offers significant improvement in each scenario and could serve as an alternative to the current manual approach.

In the remainder of this paper, we first discuss MAX in a little more detail. Then, we introduce the optimization criterion for minimizing error rate and misclassification cost. Next, we discuss the algorithms for revising the numeric parameters of the Nynex MAX expert system Then, we evaluate these parameter tuning techniques on several sets of initial parameters. Finally, we discuss the relationship to existing research and describe some limitations and directions for future research.

2.0 The Nynex MAX telephone troubleshooting expert system

Nynex MAX is a telephone network troubleshooting expert system used to screen customer troubles and dispatch them to technicians in the field or the central office. When a customer identifies a number in trouble, an electrical profile of the loop between the customer's telephone and the central office is created. This profile (containing information such as the type of switching equipment and various voltage and resistance readings) is considered by MAX, together with other information such as the weather, to make a screening diagnosis. The location to which a repairperson is dispatched is determined by this diagnosis (e.g., the problem is in the customer's wiring (PDO), the cable facilities (PDF), or the central office (PDI)). In addition, an example may be classified as one in which some additional testing of the customer's wiring must be performed (PDT).

To facilitate knowledge base adjustment and customization, MAX has a set of (fifteen) parameters used as thresholds when evaluating some of the numeric attributes. Figure 1 shows a part of a rule which tests several voltage attributes against some of the parameter

1063-6730/94 $4.00 © 1994 IEEE

settings. The first condition detects voltage readings which are too low, and the last two conditions detect high voltage readings. For each case, a recommended repair action is given.

The remaining sections discuss ways of optimizing the parameter settings for a collection of *examples*[1].

```
If  (craft_dcv_tg < CRAFT_MIN_VOLTAGE) Or
    (craft_dcv_rg < CRAFT_MIN_VOLTAGE)
Then PDT.
Else
  If (highest_dcv >= EXTREMELY_HIGH_VOLTAGE)
PDO.
Else If (highest_dcv >= VERY_HIGH_VOLTAGE)
PDT.
Else PDF.
```

Figure 1. An excerpt from a MAX rule. *Parameters* are capitalized, *attributes* are small letters, and *classifications* are underlined.

3.0 Optimization criteria

A classifier is typically evaluated by estimating its error rate from a sample of test data by finding the proportion of examples that are incorrectly classified. This measurement (which weighs all errors equally) will serve as one optimization criteria used to guide and evaluate the various parameter tuning approaches applied to MAX.

A more general criteria which weighs some errors as more costly than others was also used because in the telephone network troubleshooting problem, some of the classes are easier (i.e., less expensive) to attempt to repair than others. Mistaking a simple repair for a more complex one can be quite expensive (e.g., by dispatching a repair person to the wrong location and incorrectly replacing an expensive functional component) compared to mistaking a complex repair for a simple one. An additional complexity arises because some repairs are similar, so that mistaking one expensive repair for another may not be very expensive (e.g., if both repairs involve dispatching a repair person to the same location). These considerations are incorporated into the second evaluation criteria which measures misclassification cost.

The general form of these criterion is defined in terms of the cost of misclassifying an example which, in turn, is a function of the predicted class and the actual class. We will represent this function as a cost matrix, cost(actual-class, predicted-class). The error rate criteria has a cost matrix with 0's along the diagonal (when actual-class(i) = predicted-class(i)) and 1's otherwise. We'll call such a cost matrix a uniform cost matrix. Table 1 shows the non-uniform cost matrix that we will use for the misclassification cost criteria. Note that the costs represent dollars expended (although the cost matrix shown in Table 1 that we use does not contain the actual costs which are proprietary). The general form of the optimization criterion is defined as follows:

$$\text{average-cost} = \frac{\sum_{i}^{N} \text{cost}(\text{actual-class}(i), \text{predicted-class}(i))}{N}$$

	PDT	PDI	PDO	PDF
PDT	0	126	142	173
PDI	122	0	156	187
PDO	135	153	0	200
PDF	160	178	194	0

Table 1. A hypothetical cost matrix for the NYNEX telephone network troubleshooting problem. The rows are actual classes and the columns are predicted classes. For example, the cost of predicting class PDT when the example actually belongs to class PDI is 126.

4.0 The parameter tuning approaches

We now describe the strategies used to tune MAX's parameters. Basically, each algorithm is given an initial set of parameters, a cost matrix, and a set of training examples. Adjustments are made to the parameters until the convergence criteria is met. The resulting parameters are returned and evaluated on a separate set of test examples to evaluate how well the new parameters would perform in practice. Each adjustment consists of changing a parameter and reevaluating the training examples on the cost matrix. Thus, the amount of work expended by each method is proportional to the total number of adjustments attempted (i.e., it is the number of adjustments attempted multiplied by the number of training examples). An empirical analysis of this work is given in the next section.

The hillclimbing and steepest descent approaches (described below) are biased to make small adjustments first based on the assumption that we are given a reasonable initial approximation of the parameters. The simulated annealing and broad hillclimbing approaches drop this assumption and weigh small and large adjustments equally (i.e., they choose new parameters from a uniform distribution between 0 and twice the current parameter value). This allows us to evaluate the validity of the "good initial approximation" assumption.

[1] An example is collection of the above-mentioned attributes considered by MAX coupled with the correct classification.

Simulated annealing and broad hillclimbing are identical except that simulated annealing retains unproductive parameter adjustments probabilistically (see section 4.3). This allows us to make a direct assessment of the impact of occasional backward steps. The following subsection describe each approach in detail.

4.1 Hillclimbing

The hillclimbing approach adjusts MAX's parameters to reduce the misclassification costs on a set of training data in a greedy fashion. The algorithm is described in Table 2. It cycles through the set of parameters adding or subtracting a small amount from each parameter and as soon as it finds a parameter for which a change reduces cost, it makes that change. In addition, if a change has no effect on cost, it is made with 0.5 probability. If no parameter change results in an improvement, (i.e., a local minimum or plateau in parameter space is reached), it tries making larger changes to the parameter values, giving up after attempting to change parameter values by up to 50% of their value.

```
Given:   Parameters, CostMatrix, and
         (classified) Examples
Produce: (modified) Parameters

Cost = Cost(Examples, CostMatrix, Parameters)
PctChange = 0.025

ReviseLoop: Changed = False
  For Operator in { +, -}
    For each Parameter in Parameters
      OldValue = Parameter
      Parameter = Apply(Operator,
        Parameter,(Parameter * PctChange))
      NewCost = Cost(Examples, CostMatrix,
      Parameters)
      If (NewCost < Cost)
        Then Changed = True
      If    NewCost > Cost or
        (NewCost = Cost & Random(.5) = True))
        Then Parameter = OldValue
        Else Cost = NewCost

If Changed = True
  Then PctChange = 0.025
  Else PctChange = PctChange + 0.025
If Change < .5
  Then GoTo ReviseLoop
  Else Return Parameters
```

Table 2. The hillclimbing procedure for revising numeric parameter values.

4.2 Steepest descent

The steepest descent algorithm (Table 3) is similar to hillclimbing except that only the best change for all of the positive and negative adjustments are kept. If no improvement is made, fifty percent of the time (i.e., by "flipping a coin") we randomly retain one of the

```
Given:   Parameters, CostMatrix, and
         (classified) Examples
Produce: (modified) Parameters

Improvements = Nil
Ties = Nil
Cost = Cost(Examples, CostMatrix,
           Parameters)
PctChange = 0.025

ReviseLoop: Changed = False
  For Operator in { +, -}
    For each Parameter in Parameters
      OldValue = Parameter
      Parameter.Value = Apply(Operator,
        Parameter.Value,
        (Parameter.Value * PctChange))
      NewCost = Cost(Examples,
        CostMatrix, Parameters)
      If NewCost < Cost
        Then Push((Parameter.Value,
          Parameter), Improvements)
      If NewCost = Cost
        Then Push((Parameter,
          ParameterID), Ties)
          Parameter.Value = OldValue

If Improvements ≠ Nil
  Then Retain(BestOf(Improvements))
  Else If (Ties ≠ Nil) & (Random(.5) = True)
    Then
        Retain(ChooseRandom(Ties))
If Changed = True
  Then PctChange = 0.025
  Else PctChange = PctChange + 0.025
If Change < .5
  Then GoTo ReviseLoop
  Else Return Parameters

Macro Retain(Value, Parameter)
  Parameter.Value = Value
  Changed = True
  Cost = NewCost
```

Table 3. The steepest descent procedure for revising numeric parameter values.

adjustments which ties the current values of the evaluation function.

4.3 Simulated annealing

As we have implemented simulated annealing (see Table 4), parameter adjustments are made by choosing a new value randomly from the uniform distribution between 0 and twice the current parameter value. This broader range of possible adjustments departs from the "good approximation" bias of the previous two algorithms and offers a greater range of possible backward steps to be probabilistically taken. As in hillclimbing, improvements are retained as they are found and ties are **occasionally** kept by "flipping a coin". Adjustments that adversely affect the optimization criteria are probabilistically retained as a

```
Given:   Parameters, CostMatrix, and
         (classified) Examples
Produce: (modified) Parameters

Cost = Cost(Examples, CostMatrix, Parameters)
Temperature = MAXTEMP
Non_improvements = 0

ReviseLoop: Changed = False
  For Operator in { +, -}
    For each Parameter in Parameters
      OldValue = Parameter
      Parameter = Apply(Operator,
        Parameter,(Parameter * Uniform(0,1)))
      NewCost = Cost(Examples, CostMatrix,
        Parameters)
      Effect = NewCost - Cost
      ProbKeepAdjustment =
        exp(-Effect/(K*Temperature))
      If (Positive(Effect))
        Then Changed = True
      If (Zero(Effect) & Random(.5) = True))
        Then Parameter = OldValue
        Else Cost = NewCost
      If  (Negative(Effect) &
        Random(ProbKeepAdjustment) = True)
        Then Parameter = OldValue
        Else Cost = NewCost

If Changed = True
  Then Non_improvements = 0
  Else Non_improvements = Non_improvements +1
If   (Non_improvements < MAXWANDER) &
     (Temperature > MINTEMP)
  Then Temperature =
     Temperature - COOLRATE*Temperature
     GoTo ReviseLoop
  Else Return Parameters
```

Table 4. The simulated annealing procedure for revising numeric parameter values. MAXWANDER=12, MAXTEMP=60.0, MINTEMP=1.0, COOLRATE=0.15 and K=10. Random(X) returns True when a generated random number between zero and one is greater than X.

function of the magnitude of the decrease in performance and the current "temperature" of the algorithm. Initially, the "temperature" is quite high allowing adjustments leading to larger decreases in performance to be retained. But as the temperature "cools", even small backward steps are less likely to be kept. This approach was tried in order to avoid the local minima problem which steepest descent and hillclimbing can have.

4.4 Broad hillclimbing

The broad hillclimbing algorithm (given in Table 5) is identical to simulated annealing except no backward step is ever taken. This algorithm was included so we could evaluate the effects of the occasional backward step and the departure from the "good approximation" bias of hillclimbing and steepest descent.

```
Given:   Parameters, CostMatrix, and
         (classified) Examples
Produce: (modified) Parameters

Cost = Cost(Examples, CostMatrix, Parameters)
Temperature = MAXTEMP
Non_improvements = 0

ReviseLoop: Changed = False
  For Operator in { +, -}
    For each Parameter in Parameters
      OldValue = Parameter
      Parameter = Apply(Operator,
        Parameter, (Parameter * Uniform(0,1)))
      NewCost = Cost(Examples, CostMatrix,
                     Parameters)
      Effect = NewCost - Cost
      If (Positive(Effect))
        Then Changed = True
      If (Zero(Effect) & Random(.5) = True))
        Then Parameter = OldValue
        Else Cost = NewCost

If Changed = True
  Then Non_improvements = 0
  Else Non_improvements = Non_improvements +1
If (Non_improvements < MAXWANDER) &
  (Temperature > MINTEMP)
  Then Temperature = Temperature -
       COOLRATE*Temperature
     GoTo ReviseLoop
  Else Return Parameters
```

Table 5. The broad hillclimbing procedure for revising numeric parameter values. MAXWANDER=12, MAXTEMP=60.0, MINTEMP=0.2, COOLRATE=0.15 and K=10. Random(X) returns True when a generated random number between zero and one is greater than X.

5.0 Experimentation and results

Four experiments were run. Classifications for the (training and test) examples of the dataset used in the first three experiments were determined by interpreting the reports of the technician who actually solved the problem in the field. This data is subject to a number of sources of errors such as electronic faults in data collection and reporting devices, and noise in transmission lines ([1]). The fourth experiment evaluates the learning programs on noise-free data where the classifications for the examples are the actual MAX diagnoses with parameter settings chosen by domain experts for the site.

In each experiment, the data were split into twenty random partitions of testing and training data. Learning curves were generated by training on subsets of the training partition (from 100 to 600 examples, increments of 100)

and then evaluating the resulting parameters on the examples of the test partition (of size 554 for the first three experiments, and 294 for the fourth). These results were then averaged to generate the plots in Figures 3 through 10.

For the first three experiments, we started with 3 different sets of initial parameters and trained and tested on data from a single site. The parameters used were:

1. The actual parameters used by MAX at a different site from which the data are collected. We'll call this condition the Different Site setting. This tests the ability to customize MAX to a new site, starting with the parameters of a different site. The initial accuracy of MAX was .316 and the initial cost of MAX was 130.8 in this condition.
2. The actual parameters used by MAX in the same site from which the training and test data are collected. This tests the ability of the algorithms to fine tune MAX in a simulated operational setting. We'll call this condition the Same Site setting. The initial accuracy of MAX was .314 and the initial cost of MAX is 134.6 in this condition. We have access to only a subset of the MAX knowledge-base and the MAX training and test data that is intended to be processed by this subset. The actual MAX accuracy is considerably higher than these figures indicate ([2]).
3. Random values were chosen for each parameter from a uniform distribution in the range of the actual value of parameter minus 25% of its value and the actual value plus 25% of its actual value. This tests the ability to tune the system starting with "reasonable" but erroneous parameter settings. We'll call this condition the Random setting. The initial accuracy of MAX was .312 and the initial cost of MAX was 134.5 in this condition.

The fourth experiment is similar to the first setting, but with the noise-free data.

In Experiment 1 we ran all algorithms twice (once with a uniform cost matrix) and once with the cost matrix starting with Different Site parameters. Figure 2 gives the work expended results for this experiment. The algorithms are listed in decreasing order of the overall amount of work done as measured by the total number of examples tested (i.e., the solid bar labeled "Ex's Tested"). Notice that for every algorithm, optimizing error rate requires less work than optimizing misclassification cost. This clearly shows that the increased cost sensitivity delays convergence to a plateau (i.e., the error surface is not as flat). The white bar labeled "Total Adj's" is the total number of adjustments **kept**, not the total number made, and therefore would not necessarily be expected to correlate directly with the overall amount of work done.

The steepest descent and hillclimbing approaches attempt many more adjustments than broad hillclimbing and simulated annealing. Steepest descent's work level is relatively high because it must evaluate fifteen times as many adjustments (one for each parameter) before deciding which, if any, to retain. Hillclimbing's work level is relatively high due to a considerable amount of "plateau wandering" which is shown in the bar labeled "Coin Flips." This is probably due to its bias to stay close to the good approximation of the initial parameter settings where small parameter deviations are likely to result in performance ties.

For all of the algorithms the majority of adjustments are from coin flips. Simulated annealing has the greatest percentage of improvement adjustments (shown by the bar labeled "Improvements") because it spends more time "recovering" from the occasionally backward steps (shown by the bar labeled "SA Adj's").

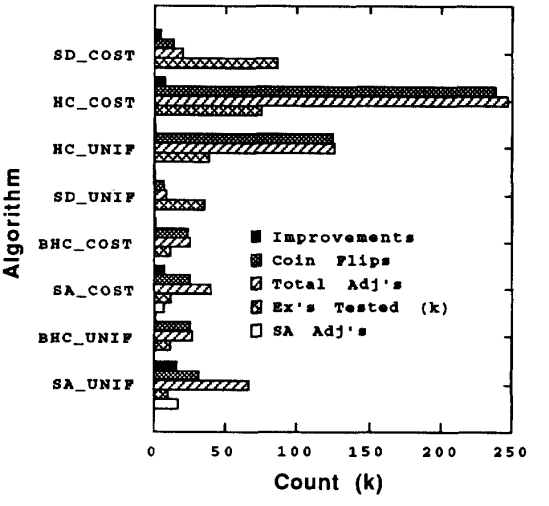

Figure 2. Work expended analysis of hillclimbing (HC), steepest descent (SD), simulated annealing (SA), and broad hillclimbing (BHC) after revising the Different Site numeric parameters of MAX using both the different cost matrices (UNIF and COST) for training.

Figure 3 shows the test data accuracy after optimizing error rate. Both hillclimbing and broad hillclimbing converge to significantly[2] higher test accuracy than simulated annealing and steepest descent. The difference between the two hillclimbing approaches is much smaller, and is not statistically significant.

The effects of minimizing costs on test error rate are illustrated in Figure 4. Once again, broad hillclimbing

[2] All comparisons referred to as "significant" signify a paired t-test with $p<.0001$.

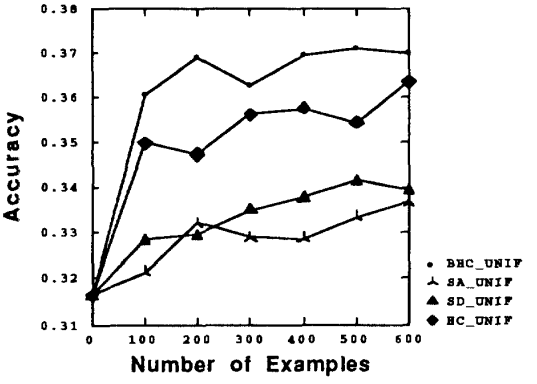

Figure 3. Accuracy of hillclimbing (HC), steepest descent (SD), simulated annealing (SA), and broad hillclimbing (BHC) after revising the Different Site numeric parameters of MAX using the uniform cost matrix for training.

converges to a higher test error rate, but none of the differences between the algorithms is significant.

In comparing Figure's 3 and 4, we see the somewhat surprising result that test error rate is more effectively optimized by algorithms which train optimizing misclassification cost (i.e., using the non-uniform cost matrix), although, the two hillclimbing approaches using the uniform cost matrix were amongst the highest converging algorithms. One possible explanation for this, as alluded to in the analysis of work done, is that the non-uniform cost matrix changes the error surface causing a stronger negative reaction to misclassified examples, thus the basin of "optimal" parameter settings is more V-shaped than U-shaped making it easier to avoid plateaus.

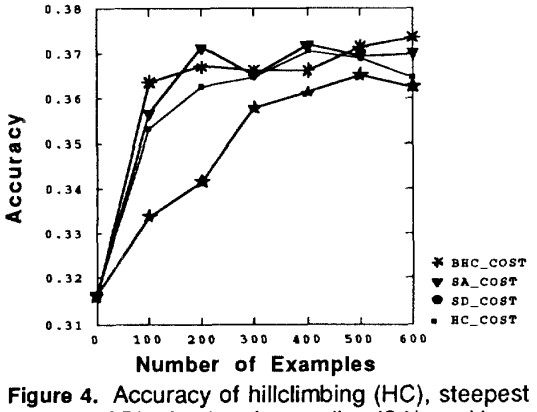

Figure 4. Accuracy of hillclimbing (HC), steepest descent (SD), simulated annealing (SA), and broad hillclimbing (BHC) after revising the Different Site numeric parameters of MAX using a non-uniform cost matrix for training.

Another interesting observation is the disparity between broad hillclimbing and simulated annealing (which is broad hillclimbing with occasional backward steps) when optimizing error rate (Figure 3). This gap closes dramatically when optimizing misclassification cost (Figure 4). This is probably because in the cost-optimizing case, backward steps are measured as more costly and are less likely to be probabilistically chosen, whereas in the uniform cost case, backward steps are measured as less costly and thus are more likely to be probabilistically chosen. In every case, simulated annealing does significantly worse than broad hillclimbing. This suggests that local minima, perhaps caused by interactions between parameters, are not a problem in this search space, and that further cooling rate adjustments may be necessary to better allow the algorithm to settle into a minimum.

Figure 5 shows the test cost performance for each algorithm after optimizing misclassification cost. Broad hillclimbing, simulated annealing, and steepest descent converge to the same point while hillclimbing maintains a slight, but insignificant edge.

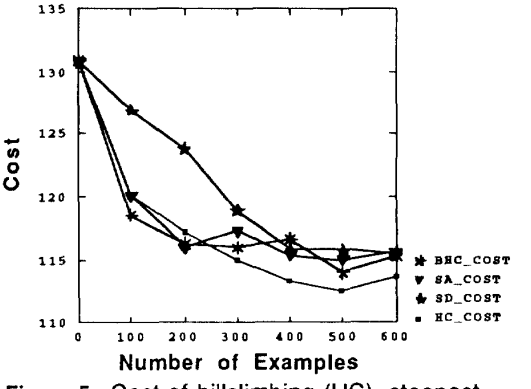

Figure 5. Cost of hillclimbing (HC), steepest descent (SD), simulated annealing (SA), and broad hillclimbing (BHC) after revising the Different Site numeric parameters of MAX using the non-uniform cost matrix for training.

Comparing figures 3, 4, and 5 reveals that steepest descent and hillclimbing tend to perform similar to or better than broad hillclimbing when training to optimize misclassification cost, and worse than broad hillclimbing when training optimize error rate. The "good initial approximation" assumption, which biases steepest descent and hillclimbing to start the search for adjustments close to the initial parameter values, may account for this because a less dramatic error surface exists when optimizing error rate (i.e., with the uniform cost matrix). That is, the flatter

error surface keeps the algorithm from leaving the apparent plateau near which the initial parameters reside.

In Experiment 2, we trained steepest descent and hillclimbing algorithms with both cost matrices starting with Same Site parameters. As in the first experiment, Figure 6 shows significantly higher test accuracy when optimizing cost versus optimizing error rate. Not surprisingly, we see in Figure 7 that test cost is better when optimizing cost rather than accuracy (a similar result was observed, but not mentioned, in the experiment 1). Within each optimization task, hillclimbing maintains a slight, but not significant, test performance edge over steepest descent. In both figures, cost-based optimization leads to similar test performance as with the Different Site parameter optimization.

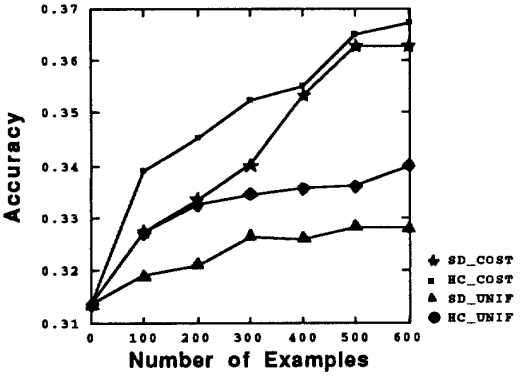

Figure 6. Accuracy of hillclimbing (HC) and steepest descent (SD) after revising the Same Site numeric parameters of MAX using the different cost matrices (UNIF and COST) for training.

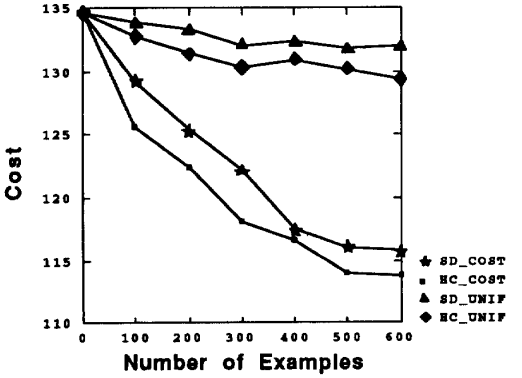

Figure 7. Cost of hillclimbing (HC) and steepest descent (SD) after revising the Same Site numeric parameters of MAX using the different cost matrices (UNIF and COST) for training.

In Experiment 3, we trained steepest descent and hillclimbing algorithms with both cost matrices starting with Random Site parameters. The results in Figures 8 and 9 correspond almost identically with those of experiment 2. That is, cost-based optimization leads to significantly better test performance, and hillclimbing outperforms steepest descent, but not at a significant level. A fourth experiment was conducted to show the performance of the algorithms on noise-free data. That is, rather than using the example classifications determined by interpreting the reports of the technician who actually solved the problem in the field (which are subject to noise (Danyluk, et al., 1993)), the actual MAX classifications are used. The results are more accurate and also more representative of MAX's true performance.

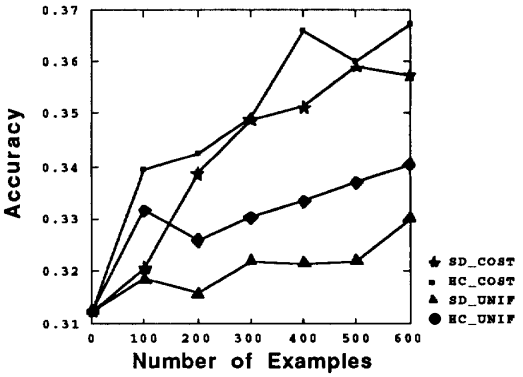

Figure 8. Accuracy of hillclimbing (HC) and steepest descent (SD) after revising the Random Site numeric parameters of MAX using the different cost matrices (UNIF and COST) for training.

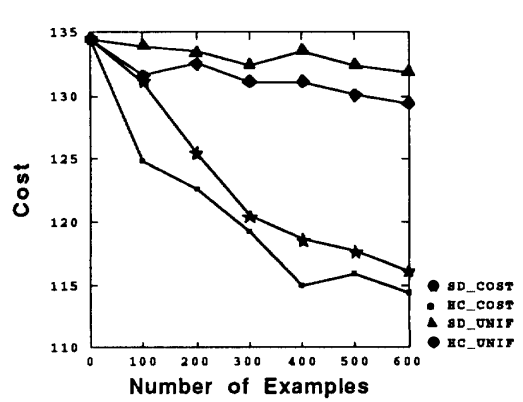

Figure 9. Cost of hillclimbing (HC) and steepest descent (SD) after revising the Random Site numeric parameters of MAX using the different cost matrices (UNIF and COST) for training.

Figure 10 shows the error rate optimization of each of the methods using the Different Site parameters. A similar convergence pattern is observed as in the earlier experiments. It should be noted that the simulated annealing approach required some adjustments in the temperature range for this last experiment (i.e., MINTEMP=0.2) allowing it more time to settle in to a minimum after taking the occasional backward step. In spite of this broader cooling phase, simulated annealing still failed to fall in to a local minimum better than the initial parameters

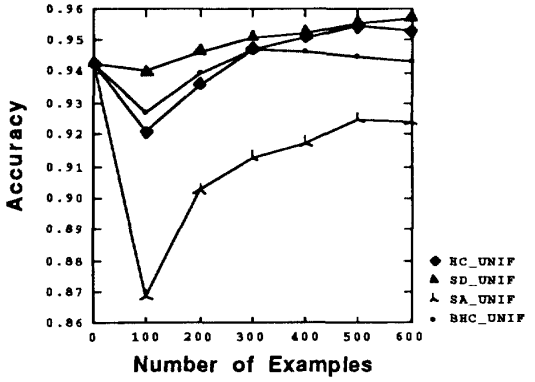

Figure 10. Accuracy of hillclimbing (HC), steepest descent (SD), simulated annealing (SA), and broad hillclimbing (BHC) after revising the Different Site numeric parameters of MAX using the uniform cost matrices for training.

Unlike experiment 1, steepest descent and hillclimbing have significantly better test accuracy than broad hillclimbing. This indicates that the "good initial approximation" bias is less robust in noisy domains. This also demonstrates that broad hillclimbing is more robust in the presence of noise and may need to do more "wandering" as the data contains less noise.

Summary of Experiments. Broad hillclimbing has the lowest work level and the best test error rate performance in the presence of noise. Hillclimbing has the best test cost performance. Steepest descent and hillclimbing have similar performances regardless of the initial parameter scenario, and perform better than broad hillclimbing on noise-free data.

6.0 Future work

Another parameter tuning approach to investigate is genetic algorithms. Such a procedure would "breed" a pool of initial (random) parameter settings retaining and breeding the best of each resultant pool until the optimization criteria stabilizes. The close convergence of the approaches already evaluated leads to the conjecture that a substantial improvement (if any) would be unexpected. We are currently working on this approach.

7.0 Conclusions

We have evaluated four parameter tuning strategies for the Nynex MAX telephone trouble screening expert system. The techniques were used to optimize two objective functions: error rate and misclassification cost. Broad hillclimbing appears to be the algorithm of choice when optimizing error rate, especially with noisy data, and hillclimbing is most effective when optimizing misclassification cost. While the hillclimbing approaches to parameter tuning tend to outperform steepest descent and simulated annealing, all are plausible alternatives to the current manual approach.

References

[1] Danyluk, A. & Provost, F. (1993). *Small disjuncts in action: Learning to diagnose errors in the telephone network local loop.* Machine Learning Conference, pp 81-88.

[2] Rabinowitz, H., Flamholz, J., Wolin, E.,& Euchner, J. (1991). Nynex MAX: A telephone trouble screening expert system. In R. Smith & C. Scott (Ed.) *Innovative applications of artificial intelligence*, 3, 213–230.