An Optimized Theory Revision Module*

Russell Greiner, R. Bharat Rao and Glenn Meredith
Siemens Corporate Research, Princeton, NJ 08540
{greiner, bharat, gam}@scr.siemens.com

Abstract

Theory revision systems typically use a set of theory-to-theory transformations $\{\theta_k\}$ to hill-climb from a given initial theory to a new theory whose empirical accuracy, over a given set of labeled training instances $\{c_j\}$, is a local maximum. At the heart of each such process is an "evaluator", which compares the accuracy of the current theory KB with that of each of its "neighbors" $\{\theta_k(KB)\}$, with the goal of determining which neighbor has the highest accuracy. The obvious "wrapper" evaluator simply evaluates each individual neighbor theory $KB_k = \theta_k(KB)$ on each instance c_j. As it can be very expensive to evaluate a single theory on a single instance, and there can be a great many training instances and a huge number of neighbors, this approach can be prohibitively slow. We present an alternative system $\Delta\!\!\Delta$ which employs a smarter evaluator that quickly computes the accuracy of a transformed theory $\theta_k(KB)$ by "looking inside" KB and reasoning about the effects of the θ_k transformation. We compare the performance of $\Delta\!\!\Delta$ with the naive wrapper system Δ on real-world theories obtained from a fielded expert system, and find that $\Delta\!\!\Delta$ runs over 35 times faster than Δ, while attaining the same accuracy. This paper also discusses $\Delta\!\!\Delta$'s source of power.

Keywords: theory revision, efficient algorithm, hill-climbing system

*We gratefully acknowledge the many helpful comments on this report from George Drastal, Chandra Mouleeswaran and Geoff Towell.

1 Introduction

Many currently deployed diagnostic expert systems use a knowledge base (a.k.a. "theory") to propose a repair for a device, based on a set of reported symptoms. Unfortunately, due to modifications of the basic devices, changes in the distribution of device faults as the device ages, installation of new devices, and errors in the original knowledge base, these proposed repairs may not always be appropriate. A "theory revision" system uses a set of labeled training examples to modify the incorrect theory, seeking a new theory that is more accurate. As no efficient algorithm is guaranteed to find the globally-optimal theory (assuming $P \neq NP$; see [Gre95]), most theory revision systems are implemented as hill-climbing processes; *cf.*, [Pol85, MB88, Coh90, OM94, CS90, WP93, LDRG94]. On each step, these revision systems compute the empirical accuracy, over the given set of examples, of the current theory KB and each of KB's "neighbors", where each neighbor is a slight modification of KB. These systems first determine which neighbor KB^* has the highest empirical accuracy, then if KB^*'s accuracy is greater than KB's, the system climbs to KB^* and the revision process iterates.

This paper presents an *efficient* way of evaluating the empirical accuracy of a theory and each of its neighbors, over a given set of labeled case reports. Section 2 provides the framework for this system, by describing the operation of the underlying expert system (*viz.*, determining the repair appropriate to a given set of symptoms), and summarizing the theory revision task (*viz.*, improving the "repair from symptom" function by modifying the underlying theory). It also describes a major complication of the revision task — *viz.*, , that the training data may not include some critical information. Section 3 then focuses on the "evaluator" component of the revision process, which computes the accuracy of each neighbor. It first presents the obvious implementation of such an evaluator which simply evaluates each individual neighboring theory on each instance; this approach resembles the "wrapper" model [JKP94], currently used when evaluating different sets of features in inductive inference tasks.[1]

Unfortunately, as it can be very expensive to evaluate a single theory on a single instance, and worse, there can be a great many examples and a huge number of neighbors, this approach is prohibitively slow. Section 3 then describes a more efficient "data-driven" theory revision algorithm, $\Delta\Delta$. Section 4 next presents empirical results which confirm that $\Delta\Delta$ works effectively, showing that it can run over 35 times faster than the naive wrapper-style system Δ, while returning essentially the same revised theories. This improvement is very important, as it allows users to use the revision system in a real-time fashion. Finally, Section 5 generalizes the specific ideas underlying $\Delta\Delta$.

Of course, a revision system, even a very efficient one, will only be used if it returns a more accurate theory. The earlier [LDRG94] demonstrates that the underlying Δ revision

[1] Of course, our theory revision task is significantly different from the feature-selection task for which the Wrapper model was designed. In particular, while it is fairly easy to analyze the revision process's evaluator, it is not as easy to determine which features will prove relevant to an induction system.

system (and hence the functionally equivalent $\Delta\!\!\!\Delta$) is *very* effective. In particular, it describes a battery of empirical tests, using real-world theories and data, which demonstrate that Δ (and hence $\Delta\!\!\!\Delta$) can quickly climb to a theory whose accuracy is over 98%, even when starting from theories whose accuracy is under 50%. Despite the complexities of these theories, it can work even when given as few as 20 labeled examples. This paper will not further discuss the accuracy-improvements that Δ, or $\Delta\!\!\!\Delta$, can obtain; instead, it focuses on ways of making these systems more efficient — fast enough that people will actually use these systems.

Related Research

There are, of course, other theory revision systems, which also start from an initial domain theory obtained from experts, and iteratively modify that theory to obtain a theory with improved accuracy on a set of training cases; *cf.*, [Pol85, MB88, Coh90, OM94, WP93, CS90]. Most of these systems focus on Horn clause knowledge bases or decision trees, which differ from the hypothesis space of fault hierarchies searched by $\Delta\!\!\!\Delta$ (although fault hierarchies can be translated into these representations). However, the transformations suggested by $\Delta\!\!\!\Delta$ typically do not correspond to one-step revisions suggested by existing systems. Further, single revisions suggested by the existing systems can radically change the structure of the fault hierarchy (see [LDRG94] for further discussion). This paper, in particular, focuses on the *efficiency* of our $\Delta\!\!\!\Delta$ system; the extended tech-report [GRM95] discusses how this compares with the efficiency measures used in other revision systems. It also provides a more comprehensive literature survey.

2 Framework

2.1 Theory (Fault Hierarchy) and Instances

This subsection first defines the structures of theories ("fault hierarchies") and "problem instances". It then describes how a fault-hierarchy–based expert system works; *i.e.*, how it evaluates a theory in the context of a problem instance to produce a repair.

Each "fault hierarchy" $KB = \langle\, N,\, E,\, \mathit{TS},\, \mathcal{R},\, t(\cdot),\, r(\cdot),\, \mathit{child}(\cdot,\cdot)\,\rangle$ is a directed-acyclic forest $\langle N, E\rangle$, whose nodes $n \in N$ represent faults, and whose edges $e \in E$ connect faults to subordinate faults. Each node $n \in N$ is labeled with a test $t(n) = t$ or $t(n) = \neg t$, where $t \in \mathit{TS}$; in addition, each leaf node n is also labeled with a "repair", $r(n) = r \in \mathcal{R}$. The arcs under each internal node are ordered; $\mathit{child}(n,\, i)$ refers to the "i^{th} child of n". For notation, let the $k\colon \mathcal{N} \mapsto \mathcal{Z}^+$ function map each node to its number of children.

For example, consider the KB_0 hierarchy shown in Figure 1, where the test associated with the node φ is $t(\varphi)$; hence, the test associated with the A node is $t(\mathtt{A})$, etc. The r_φ expression is the repair labeling the associated leaf node; hence, the repair associated with the node D (whose test is $t(\mathtt{D})$) is $r_{\mathtt{D}}$. A's children are, in order, C, D and E — hence, $\mathit{child}(A,\, 1) = C$, $\mathit{child}(A,\, 2) = D$ and $\mathit{child}(A,\, 3) = E$. Here, $k(A) = 3$.

$KB_0 \quad =$

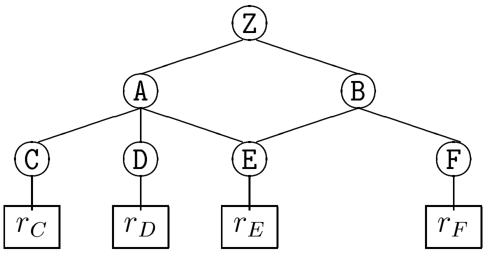

Figure 1: Structure of KB_0

Running an Expert System: Let "S_{KB}" denote the expert system that uses the KB hierarchy. When S_{KB_0} is "run" (using the KB_0 should in Figure 1) S_{KB_0} will ask the user a series of questions that correspond to a depth-first, left-to-right, no-backtrack traversal of (part of) the KB_0 structure.[2] This S_{KB_0} begins at the root, and asks the question associated with that node; here "Is $t(\texttt{Z})$ true?". If the user answers "Yes", S_{KB_0} descends to consider `Z`'s children, in left-to-right order — here next asking "Is $t(\texttt{A})$ true?". If the user responds "Yes", S_{KB_0} will descend to `A`'s children. If the user answers $t(\texttt{C})$ with "No", S_{KB_0} will continue to `C`'s right-sibling `D`, and ask about $t(\texttt{D})$. Assuming the user responds "Yes" here, S_{KB_0} will return the repair associated with that leaf node, here r_D. On the other hand, if the user had responded "No" to $t(\texttt{D})$, S_{KB_0} would have continued to ask about $t(\texttt{E})$. If this answer was "Yes", S_{KB_0} would return r_E. Otherwise, if this answer was also "No", S_{KB_0} would return the "no-proposed-repair" answer, $r_\perp$. *N.b.*, S_{KB_0} will not then continue to B — answering $t(\texttt{A})$ with "Yes" means the user will only consider tests and repairs under node A.

Ignoring the details of the actual user-interaction, each "(total) problem instance" is an assignment $\pi\colon TS \mapsto \{+,-\}$ that maps each test to one of $\{+,-\}$, where $+$ means the test was confirmed (passed), and $-$ means the test was disconfirmed (failed). Given a problem instance π, S_{KB} will return a repair $r \in \mathcal{R}$, written as $KB(\pi) = r$. This r is the value returned by EVALNODE(ROOT(KB), π), using the subroutine shown in Figure 2, where $n_{root} = \texttt{root(KB)}$ is KB's root. (We also assume that the associated test $t_{root} = t(n_{root})$ has already been confirmed, as we are viewing t_{root} as the "presenting symptom" or "triggering information".)

2.2 Accuracy of a Hierarchy

A labeled training instance includes the symptoms presented to the expert system plus any additional information supplied by a human expert, such as the "correct" repair ($r_{cor} \in \mathcal{R}$) and possibly additional test and their answers. The accuracy of the hierarchy KB for the

[2] This corresponds to using PROLOG, in which every clause includes a "`!`". For example, Figure 1 corresponds to a theory that includes "`top(R) :-` $\texttt{t}_Z$`, !, z(R).`", "`z(R) :-` $\texttt{t}_A$`, !, a(R).`", "`z(R) :-` $\texttt{t}_B$`, !, b(R).`", "`a(R) :-` $\texttt{t}_C$`, !, c(R).`", "`a(R) :-` $\texttt{t}_D$`, !, d(R).`", "`a(R) :-` $\texttt{t}_E$`, !, e(R).`", "`c(`$\texttt{r}_C$`).`", etc.

```
Procedure EvalNode( n: node, π: (total)_problem_instance )
  If n is a leaf node
  Then return( r(n) )
  Else For i := 1..k(n)
         n_i := child(n, i)
         If π( t(n_i) ) = +     /* ie, if result of test associated with n_i is + */
         Then return( EvalNode( n_i, π ) )
       End For
       return( r_⊥ )       /* Arriving here means NONE of n's children succeeded */
End EvalNode
```

Figure 2: EvalNode subroutine

"labeled (total) problem instance" $\langle \pi, r_{cor} \rangle$ is

$$acc(\, KB,\, \langle \pi, r_{cor} \rangle\,) \quad = \quad \begin{cases} 1 & \text{if } KB(\pi) = r_{cor} \\ 0 & \text{otherwise} \end{cases}$$

Over a set of labeled instances $\mathcal{C} = \{\langle \pi_i, r_i \rangle\}_i$, KB's (empirical) accuracy is

$$acc(\, KB,\, \mathcal{C}\,) \quad = \quad \sum_{\langle \pi, r \rangle \in \mathcal{C}} acc(\, KB,\, \langle \pi, r \rangle\,)$$

Partial Instances: In practice, as each training instance comes from a diagnosis session with the expert system, only a small subset of the tests in *TS* will be recorded. However, the above computations assume that S_{KB} is always able to obtain answers to all relevant tests. We must therefore use a "partial problem instance" $\pi : TS \mapsto \{+, -, ?\}$ where "$\pi(t) = ?$" means the value of the test t is not known.

Each such partial instance π really corresponds to some *total* instance π^*, where some of π's test values are not recorded. To state this more precisely: a *total* problem instance $\pi^* : TS \mapsto \{+, -\}$ is a *completion* of π iff π^* agrees with π whenever $\pi(t)$ is categorical (*i.e.*, is not "?"):

$$\pi^* \textit{ completes } \pi \qquad \textit{iff} \qquad [\pi(t) \neq ? \quad \Rightarrow \quad \pi^*(t) = \pi(t)]$$

Hence, the total instance

$$\pi_{T1} \quad = \quad \{\ t(\mathtt{Z})/+,\ t(\mathtt{A})/+,\ t(\mathtt{B})/-,\ t(\mathtt{C})/-,\ t(\mathtt{D})/+,\ t(\mathtt{E})/+,\ t(\mathtt{F})/-\ \}$$

is a completion of the partial instance

$$\pi_{P1} \quad = \quad \{\ t(\mathtt{Z})/+,\ t(\mathtt{A})/+,\ t(\mathtt{B})/?,\ t(\mathtt{C})/-,\ t(\mathtt{D})/+,\ t(\mathtt{E})/?,\ t(\mathtt{F})/?\ \}\ .$$

We let

$$\text{Complete}(\pi) \quad = \quad \{\ \pi^* : TS \mapsto \{+, -\} \mid \pi^* \text{ completes } \pi\ \}$$

refer to the set of total instances that complete a given partial instance.

In general, the probability $Pr[\pi^*|\pi]$ that the observed partial instance π corresponds to the total instance $\pi^* \in$ Complete(π) depends on the joint probability that each unobserved test t (t such that $\pi(t) =$ "?") has the specified categorical value $\pi^*(t)$. Here, the probability that the observed π_{P1} corresponds to the actual total π_{T1} depends on the probabilities that $t(\texttt{B}) = -$, $t(\texttt{E}) = +$ and $t(\texttt{F}) = -$. We assume that these tests are independent (of each other, and other context) which allows us to express this conditional probability in terms of the probability function $p\colon TS \mapsto [0,1]$, where $p(t)$ is the probability that the unobserved test t would be confirmed, if only it had been run and reported.

Notice also that each $\pi^* \in$ Complete(π) has an associated repair, r_{π^*} = EVALNODE(ROOT(KB), π^*); we can therefore use the $p(\cdot)$ values to compute the probability that S_{KB_0} will return each r_{π^*}, given the observed values π. In general, we will need to compute the probability that S_{KB_0} will return the correct repair r_{cor}, $Pr[\ S_{KB_0}$ returns $r_{cor} \mid \pi$ observed $]$. Using the observation that this quantity corresponds to $acc(\ KB,\ \langle \pi, r_{cor}\rangle\ )$ when π is a total instance (here, $Pr[\ S_{KB_0}$ returns $r_{cor} \mid \pi$ observed $] = acc(\ KB,\ \langle \pi, r_{cor}\rangle\ ) \in \{0,1\}$), we extend $acc(\ \cdot,\ \cdot\ )$ to be this probability value in general. (The $p(\cdot)$ function is implicit in the $acc(\ KB,\ \langle \pi, r_{cor}\rangle\ )$ description.)

Using the ACCNODE subroutine shown in Figure 3, ACCNODE(root(KB), π, r_{cor}) computes this probability. This subroutine uses:

$$p'(n) \quad = \quad \begin{cases} 0 & \text{if } \pi(t(n)) \ = - \\ 1 & \text{if } \pi(t(n)) \ = + \\ p(\,t(n)\,) & \text{if } \pi(t(n)) \ = \ ? \end{cases}$$

to refer to the probability of a node being confirmed, relative to the partial assignment π. Of course, if $t(n)$ is the negation of a test, *i.e.*, $t(n) = \neg t$, then $p(t(n)) = p(\neg t) = 1 - p(t)$ when $\pi(t(n)) =$ "?". This algorithm implicitly uses the fault hierarchy KB and the probability information $p(\cdot)$.

2.3 Theory Revision Task

As stated above, a theory revision system takes as input an initial theory KB_0 and a set of "labeled case reports" $\mathcal{C} = \{c_j\}$. Each such system also uses a set of transformations, $\Theta = \{\theta_k\}$, where each θ_k maps one hierarchy to another. We consider four classes of transformations:

- each $Delete_{par,n}$ transformation deletes the existing link between par and n. Hence, $Delete_{B,E}(KB_0)$ is a hierarchy KB_1 that includes all of the nodes of KB_0 and all of its arcs *except* the arc from B to E. Hence, in KB_1, $child(B,\ 1) = F$.
- each $Add_{par,n,i}$ transformation adds a new link between par and n, making n the i^{th} child under par. Notice $Delete_{A,F}(\ Add_{A,F,2}(KB_0)\ ) \equiv KB_0$.

```
Procedure AccNode( n: node, π: (partial)_problem_instance, r: repair )
  If n is a leaf node
  Then return( r(n) = r ?  1 :  0 )
  Else tot := 0;   reach_me = 1;
       For i :=  1..k(n)
         n_i  :=  child(n,  i)
         If π( t(n_i) ) = +
         Then tot := tot + (reach_me  ×  AccNode( n_i, π, r ) )
              return( tot )
         ElseIf π( t(n_i) )  = ?
              tot := tot + (reach_me × p'( n_i ) ×  AccNode( n_i, π, r ) )
              reach_me := reach_me * (1 − p'( n_i ))
         /* If π( t(n_i) ) = −, just go to next sibling */
       End For
       return( tot )
End AccNode
```

Figure 3: AccNode subroutine

- each $Move_{par1,par2,n,i}$ transformation deletes the existing link between $par1$ and n, and also adds a link from $par2$ to n, as the i^{th} arc under $par2$. Hence, $Move_{par1,par2,n,i}(KB) = Add_{par2,n,i}(\, Delete_{par1,n}(KB)\,)$.
- each $Switch_{par,n1,n2}$ transformation switches the positions under par of the links from par to $n1$, and from par to $n2$. Notice each $Switch_{n1,n2,par}$ transformation corresponds to at most two move transformations.

Notice that each operator θ has an associated "negation" operator, $\neg\theta$, such that $\neg\theta(\theta(KB)) = KB$. We let $\Theta = \{Delete_{par,n}\} \cup \{Add_{par,n,i}\} \cup \{Move_{par1,par2,n,i}\} \cup \{Switch_{par,n1,n2}\}$ include all meaningful transformations of each type (*e.g.*, we insist that par be an internal node that includes a link to n for each deletion, move and switch transformation, etc.); and let $\mathcal{N}(KB) = \{\theta(KB) \,|\, \theta \in \Theta\}$ be the set of KB's neighbors.

A theory revision system will use a set of labeled instances $\mathcal{C} = \{\langle \pi_i, r_i \rangle\}$ to compute $acc(\, KB, \mathcal{C}\,)$ and $acc(\, KB', \mathcal{C}\,)$ for each $KB' \in \mathcal{N}(KB)$, and then climb from KB to a $KB^* \in \mathcal{N}(KB)$ if $acc(\, KB^*, \mathcal{C}\,) > acc(\, KB, \mathcal{C}\,)$. Here, the revision system will recur, seeking a new $KB'' \in \mathcal{N}(KB')$ that is better than this KB', and so forth; see [LDRG94]. The paper focuses on a single step of this revision process: computing the best KB^* from the set $\mathcal{N}(KB)$.

3 Finding the Best Revision

The Δ system uses the obvious "wrapper-esque" algorithm to determine the best revised theory KB^*: It begins by loading in the KB theory and the first instance π_1, and uses S_{KB} to evaluate $KB(\pi_1)$ to compute $acc(\, KB, \langle \pi_1, r_1 \rangle\,)$. The Δ system then builds $KB_1 = \theta_1(KB)$ by applying the first transformation θ_1 to KB and uses S_{KB_1} to compute $acc(\, KB_1, \langle \pi_1, r_1 \rangle\,)$.

$$
\begin{array}{c|ccccc|c}
 & c_1 & \dots & c_j & \dots & c_{|\mathcal{C}|} & \mathcal{C} \\
\hline
KB & \cdot & \dots & \cdot & \dots & \cdot & acc(\,KB, \mathcal{C}\,) \\
\theta_1(KB) & \cdot & \dots & \cdot & \dots & \cdot & acc(\,\theta_1(KB), \mathcal{C}\,) \\
\vdots & \vdots & & \cdot & & \vdots & \vdots \\
\theta_i(KB) & \cdot & \dots & \boxed{acc(\,\theta_i(KB),\, c_j\,)} & \dots & \cdot & acc(\,\theta_i(KB), \mathcal{C}\,) \\
\vdots & \vdots & & \cdot & & \vdots & \vdots \\
\theta_m(KB) & \cdot & \dots & \cdot & \dots & \cdot & acc(\,\theta_m(KB), \mathcal{C}\,)
\end{array}
$$

Figure 4: Operations required by the Naive Wrapper-esque Algorithm

It then builds $KB_2 = \theta_2(KB)$ from KB_1, by applying $\neg\theta_1$ (to restore KB) followed by θ_2. In this fashion, Δ computes $acc(\,KB_i,\, \langle\pi_1, r_1\rangle\,)$, for all $|\Theta|$ transformations in $\mathcal{N}(KB)$. At the end of the first iteration, Δ unloads the first instance, π_1, and loads the second instances, π_2, and proceeds as above for each $\langle\pi_j, r_j\rangle \in \mathcal{C}$. In essence, this involves sequentially computing each column of the matrix shown in Figure 4, which involves $(1 + |\Theta|) \times |\mathcal{C}|$ non-trivial computations, $\{acc(\,\theta_i(KB),\, c_j\,)\}_{ij}$. Finally, Δ computes $acc(\,KB_i,\, \mathcal{C}\,) = \sum_j acc(\,KB_i,\, \langle\pi_j, r_j\rangle\,)$ (summing every row of the matrix in Figure 4),[3] and finds which of these transformations produced the largest of the values $KB^* = \text{argmax}_i\{acc(\,KB_i,\, \mathcal{C}\,)\}$, and climbs to this theory if $acc(\,KB^*,\, \mathcal{C}\,) > acc(\,KB,\, \mathcal{C}\,)$. This process can be very expensive as both $|\Theta|$ and $|\mathcal{C}|$ can be huge — *e.g.*, in the study discussed below, we found values of $|\Theta| \approx 1,000$.

3.1 Beyond the Wrapper Model

There are two basic ideas for improving the efficiency: The first is to evaluate a smaller number of the $\{acc(\,\theta_i(KB),\, c_j\,)\}_{i,j}$ entries; we show below three techniques for this. The second is to provide a more efficient way to compute this quantity; for reasons explained below, we will actually use the equally-useful values

$$
\begin{aligned}
\text{diff}'(\,\theta,\, c\,) &= acc(\,\theta(KB),\, c\,) \;-\; acc(\,KB,\, c\,) && (1) \\
\text{Diff}(\,\theta,\, k\,) &= \sum_{j=1..k} \text{diff}'(\,\theta,\, c_j\,) && (2)
\end{aligned}
$$

where $\text{diff}'(\,\theta,\, c\,)$ is the difference in accuracy between $\theta(KB)$ and KB, on instance c; and $\text{Diff}(\,\theta,\, k\,)$ is the relative score for the transformed $\theta(KB)$ knowledge base, after k instances. The resulting $\Delta\!\!\Delta$ system is sketched in Figure 5.

RT$_I$: Restrict the Transformations, based on Instance. The first idea exploits the particular set of transformations we are considering, using the observation that, for almost every π_j, the value of $acc(\,\theta_i(KB),\, c_j\,)$ is equal to $acc(\,KB,\, c_j\,)$ for a great many θ_is. This

[3] The more "natural" method of sequentially computing each row of the matrix shown in Figure 4 directly would require Δ to load and unload transformations, θ, $|\mathcal{C}|$ times, and to load and unload instances, π, $(1 + |\Theta|) \times |\mathcal{C}|$ times. Because loading and unloading π is 1–2 orders of magnitude more expensive than loading and unloading θ, we sequentially computed the columns of the matrix, as described above.

```
Procedure ΔΔ( KB: fault_hierarchy, C: set_of_problem_instances ):  transformation;
  Q := {}          /* Q will store set of active ⟨θ, Diff(θ, k)⟩ pairs.
  rem_insts := |C|;                    /* = M − 0, the number of instances remaining */
  Err := Σ_{c_i ∈ C}(1 − acc(KB, c))   /* remaining "error" of all instances; Err_M */
  s* := 0;                             /* current highest value Diff(θ, 0) score */
  For  ⟨π,r⟩  in C
    QE_ComputeRhoTau( KB, π, r );
       /* Modifies KB by assigning ρ(n), τ(n) values to each node in KB, based on π, r */
    GoodThetas := RTi_Eligible( KB, ⟨π,r⟩ );
       /* GoodThetas = transformations θ where acc( KB, ⟨π,r⟩ ) may be different from acc( θ(KB), ⟨π,r⟩ ) */
    For  θ  in GoodThetas
      s = CurrentScore( θ, Q );          /* if ∃⟨θ,s⟩ ∈ Q, then use s; otherwise, set s := 0 */
      if ( RTp( s, Err ) && RTm( s, rem_insts, s* ) )
         s' := s + QE_Diff( KB, θ );
              /* QE_Diff uses KB, which includes values of ρ(n), τ(n), to compute diff'(θ, c). */
         Q := Replace( Q, θ, s' );      /* Q now includes ⟨θ,s'⟩ entry */
    End For
    rem_insts := rem_insts - 1;          /* this is M − k */
    Err := Err - (1- acc(KB, ⟨π,r⟩)); /* this is Err_k */
    ⟨θ*, s*⟩ := Best( Q );               /* ⟨θ*, s*⟩ is in Q, and s* is largest score in Q. */
  End For
  If (s* > 0)  return( θ* )
  Else         return( NoOp )            /* here, nothing was better */
End ΔΔ

Procedure RTp( s: Real, Err: Real):  boolean;
  return( s > - Err );
End RTp

Procedure RTm( s: Real, rem_insts: Int, s*: Real):  boolean;
  return( s > s* - rem_insts );
End RTm
```

Figure 5: Overview of ΔΔ Algorithm

means that most transformations θ_i do not influence whether a hierarchy will be correct on an instance. As an illustration, consider the KB_0 hierarchy (Figure 1) and the partial instance $\pi_1 = \{t(\mathtt{Z})/+,\ t(\mathtt{A})/+,\ t(\mathtt{C})/-,\ t(\mathtt{D})/+\}$ that confirms $t(\mathtt{Z})$, $t(\mathtt{A})$ and $t(\mathtt{D})$ and disconfirms $t(\mathtt{C})$; here, S_{KB_0} returns $r_D = KB_0(\pi_1)$.[4] Now consider the $Delete_{B,E}$ transformation that deletes the $\langle B, E\rangle$ arc, and observe that, as the original KB_0 arrives at its answer (r_D) before ever reaching B, this new $KB_1 = Delete_{B,E}(KB_0)$ will also reach the same decision; here r_D. As $KB_0(\pi_1) = KB_1(\pi_1)$, clearly $acc(\, KB_0,\, \langle \pi_1, r\rangle\,) = acc(\, Delete_{B,E}(KB_0),\, \langle\pi_1, r\rangle\,)$. (Notice this does *not* depend or whether r_D was the correct answer: if KB_0 was correct on π_1, then so is KB_1; likewise, if KB_0 was incorrect on π_1, then so is KB_1.) The extended paper presents other classes of deletions which will not change the accuracy score, as well as showing when additions, switches and moves are similarly no-ops.

Stated more precisely, $\Delta\!\!\Delta$ will only consider the subset of the transformations, returned by the RTI_ELIGIBLE routine, call the set $\mathrm{RT}_I(KB, \mathcal{N}, c_j)$, which excludes many transformations θ for which $\mathrm{diff}'(\,\theta',\, c_j\,) = 0$. (The extended [GRM95] presents the actual implementation, which uses the $\rho(\cdot)$ and $\tau(\cdot)$ values, defined below.)

Of course, many other theory revision systems use some variant of this basic idea — *viz.*, they too focus on the subset of transformations that appear "relevant" for at least one instance. The novelty in our approach is the way we determine which transformations to consider, which is complicated by both the fault-hierarchy representation used, and by our use of partial instances.

RT_P : Restrict the Transforms, based on Positive score. The second insight comes from two observations: First, $\Delta\!\!\Delta$ will only climb from KB to the transformed theory $KB' = \theta(KB)$ if KB$'$'s empirical score, after all M instances, is strictly greater than KB's; *i.e.*, if $\mathrm{Diff}(\,\theta,\, M\,) > 0$. Second, as $acc(\,\theta(KB),\, c_k\,) \le 1$, the difference between $\mathrm{Diff}(\,\theta_i,\, k\,)$ and $\mathrm{Diff}(\,\theta_i,\, k-1\,)$ can be at most

$$\begin{aligned}\mathrm{Diff}(\,\theta,\, k\,) - \mathrm{Diff}(\,\theta,\, k-1\,) &= acc(\,\theta(KB),\, c_k\,) - acc(\,KB,\, c_k\,)\\ &\le 1 - acc(\,KB,\, c_k\,)\end{aligned}$$

This trivially means that $\mathrm{Diff}(\,\theta,\, M\,) \le \mathrm{Diff}(\,\theta,\, k\,) + Err_k$, where $Err_k = \sum_{j=k+1}^{M}(1 - acc(\,KB,\, \langle\pi_j, r_j\rangle\,))$ is the total error of KB on the remaining $M-k$ instances. If $\mathrm{Diff}(\,\theta,\, k\,) \le -Err_k$, then clearly $\mathrm{Diff}(\,\theta,\, M\,) \le 0$, which means $\Delta\!\!\Delta$ will *not* climb to $\theta(KB)$. Hence,[5]

$$\mathrm{RT}_P(KB, \mathcal{N}, Err_k, \mathrm{Diff}(\,\cdot,\, k\,)) \quad = \quad \{\theta \in \mathcal{N} \mid \mathrm{Diff}(\,\theta,\, k\,) > -Err_k\} \tag{3}$$

RT_M : Restrict the Transforms, based on Maximal score. The third insight is related to the RT_P filter presented above, and is also based on simple dynamic programming.

[4] Notice S_{KB_0} would only ask about these 4 tests, and only record their values. The other tests would not be asked, as their values do not matter here; hence, π_1 maps each other test to "?".

[5] To produce slightly more efficient code, we first sorted the cases in order of increasing accuracy. This way, the $\mathrm{Diff}(\,\theta,\, k\,)$ scores go to 0 fastest, which increases the number of transformations θ that will fail the RT_P test.

It uses two observations: First, $\Delta\!\!\Delta$ will only climb from KB to the transformed version $KB' = \theta'(KB)$ if KB$'$'s empirical score is the largest over all members of $\mathcal{N}(KB)$; *i.e.*, if $\forall\theta_i \in \mathcal{N}, \mathrm{Diff}(\,\theta',\, M\,) \geq \mathrm{Diff}(\,\theta_i,\, M\,)$. Second, for any $\theta_i, \theta_j \in \mathcal{N}$, the difference between $\mathrm{Diff}(\,\theta_i,\, k+1\,) - \mathrm{Diff}(\,\theta_j,\, k+1\,)$ and $\mathrm{Diff}(\,\theta_i,\, k\,) - \mathrm{Diff}(\,\theta_j,\, k\,)$ can be at most 1. This trivially means that

$$\mathrm{Diff}(\,\theta_i,\, M\,) - \mathrm{Diff}(\,\theta_j,\, M\,) \quad \leq \quad (\mathrm{Diff}(\,\theta_i,\, k\,) - \mathrm{Diff}(\,\theta_j,\, k\,)) + (M-k)$$

and so

$$\text{if } \mathrm{Diff}(\,\theta_i,\, k\,) < \mathrm{Diff}(\,\theta_j,\, k\,) - (M-k), \quad \text{then} \quad \mathrm{Diff}(\,\theta_i,\, M\,) < \mathrm{Diff}(\,\theta_j,\, M\,),$$

which means $\Delta\!\!\Delta$ will not climb to $\theta_i(KB)$. Letting $\mathrm{Best}_k = \max_{\theta\in\mathcal{N}}(\mathrm{Diff}(\,\theta,\, k\,))$ be the largest empirical (relative) score, over the transformations, after k samples, we define

$$\mathrm{RT}_M(KB, \mathcal{N}, k, M, \mathrm{Diff}(\,\cdot,\, k\,)) = \{\ \theta_i \in \mathcal{N} \mid \mathrm{Diff}(\,\theta_i,\, k\,) > \mathrm{Best}_k - (M-k)\ \}.$$

QE**: Quick Evaluation.** Given the above analysis, we need only compute the values of $acc(\,\theta_i,\, c_k\,)$ for only a subset of the transformations; *i.e.*, only for θ_i in the set

$$\mathrm{RT}_I(KB, \mathcal{N}, c_k) \bigcap \mathrm{RT}_P(KB, \mathcal{N}, Err_k, \mathrm{Diff}(\,\cdot,\, k\,)) \bigcap \mathrm{RT}_M(KB, \mathcal{N}, k, M, \mathrm{Diff}(\,\cdot,\, k\,)) \ .$$

We could compute these $acc(\,\theta_i(KB),\, c\,)$ values by first synthesizing each $KB_j = \theta_j(KB)$, and then "running" each KB_j on each $c_i = \langle \pi_i, r_i\rangle$ instance to get their respective repairs and associated accuracy scores. Our $\Delta\!\!\Delta$ algorithm, however, uses a more effective way of computing these accuracy scores:

Notice first that it is sufficient to compute $\mathrm{diff}'(\,\theta_m,\, \pi\,)$, rather than $acc(\,\theta_m(KB),\, \pi\,)$; this quantity turns out to be relatively easy to compute (especially as $\mathrm{diff}'(\,\theta_m,\, \pi\,) = 0$ holds for many $\langle\theta_m, \pi\rangle$ pairs; see above discussion of RT_I). Given a (partial) instance π and the correct repair r_{cor}, the QE_ComputeRhoTau subroutine identifies, with each node n in the initial KB, the probability $\rho(n)$ that S_{KB} would reach r_{cor}, given that it has reached n (this is essentially AccNode(n, π, r_{cor})). QE_ComputeRhoTau also identifies each arc a in KB (resp., node n in KB) with the probability that S_{KB} will reach and traverse this arc $\tau(a)$ (resp., the probability that S_{KB} will reach n, $\tau(n)$). (This information is computed in time linear in the size of the hierarchy.) Then, for each transformation $\theta \in \Theta$, QE_Diff uses these $\rho(\cdot)$ and $\tau(\cdot)$ values to compute $\mathrm{diff}'(\,\theta,\, \langle\pi, r_{cor}\rangle\,)$, in constant time.[6]

N.b., our $\Delta\!\!\Delta$ revision algorithm does *not* produce an explicit $S_{\theta_i(KB)}$ performance system, nor does it explicitly compute the value $acc(\,\theta_i(KB),\, \pi\,)$ on the instance π. Instead, it uses the $\rho(\cdot)$ and $\tau(\cdot)$ values to produce the information it needs, directly.

The extended [GRM95] supplies the many (tedious) details required to specify exactly how to use this information effectively, which basically requires a non-trivial case statement for each of the four types of transformation. To give a flavor for the basic idea, the appendix below sketches how this process works for one type of transformation.

[6]The time is really bounded by the largest number of arcs descending from any node, which we are assuming is a small constant.

4 Empirical Data

We have implemented this $\Delta\!\!\Delta$ system, and confirmed that its generalization accuracy was equivalent to that of the wrapper-esque Δ system, in over 1700 climbs, when revising a theory from a fielded expert system. However, $\Delta\!\!\Delta$'s average run time is over 35 times faster than Δ's. To state this more precisely: We consider 127 different climbs, taken from 52 different contexts, each involving one of 13 different initial theories T_i (each a corruption of the particular theory T_{ewsd}[7]), with unique training sets of various sizes. While $\Delta\!\!\Delta$'s times range from 5.6 to 54.4 CPU seconds and Δ, from 175.0 to 2252.1, the ratios of Δ to $\Delta\!\!\Delta$'s run times in corresponding contexts range from 29.3 to 49.5, with a mean ratio of 36.3 ± 3.6.[8]

To understand how $\Delta\!\!\Delta$ achieves its improvement over Δ, we collected some additional statistics during each climb. In particular, for this set of climbs, the number of neighboring theories $|\mathcal{N}(T_i)|$ ranges from 686 to 798, with mean 743.6 ± 39.8. The wrapper-esque Δ considers them all. By contrast, $\Delta\!\!\Delta$ uses its RT_I, RT_P and RT_M filters to ignore a great many of these revisions, when processing each training sample. We therefore record ℓ_i, the number of theories considered when processing the i^{th} sample, and use them to obtain $L = \frac{1}{N}\sum_{i=1..N} \ell_i$, the average number of revised theories considered for each sample. (Each ℓ_i is the number of "meaningful" entries in the i^{th} column of the Figure 4 matrix.) These values are different in different situations, as they depend on details of the structure of the initial T_i and the distribution of instances. Here, L varies from 2.4 to 71.1 (mean 22.6 ± 15.4). By comparing the two means, we see this reduction alone accounts for a speed-up by a factor of $743/22.6 = 32.9$.

The other influence is the time required to evaluate each revised theory on each sample. Δ actually produces the new revised $\theta_i(KB)$, and then evaluates it in the context of the labeled query π_j; by contrast, $\Delta\!\!\Delta$ uses the quick evaluation scheme (QE) to quickly compute the value of $\mathrm{diff}'(\,\theta_i,\ \langle \pi_j, r\rangle\,)$. As both systems are implemented within the same framework, and share essentially the same overhead code, we attribute the remaining $\approx 36.3/32.9 = 1.10$ (*i.e.*, 10%) improvement to $\Delta\!\!\Delta$'s use of its Quick Evaluation process.[9]

Finally, the extended [GRM95] presents this data in more detail, and also describes how much the various ideas (RT_I ,RT_P ,RT_M and QE) contribute to the overall speedup. It also presents the similar speed-ups we obtained when we compared Δ and $\Delta\!\!\Delta$ with respect to other fault hierarchies, from different, fielded expect systems.

[7] This T_{ewsd}, viewed as a hierarchy, had 19 "nodes", and leads to 13 distinct repairs.

[8] The "3.6" here is one standard deviation.

[9] In fact, the best least-square fit of the timing data when $\Delta\!\!\Delta$ is processing N instances, and considering an average of L revised theories / instance, is $\mathrm{time}_{\Delta\!\!\Delta}(L, N) \;=\; 0.76 + 0.27N + 0.00087LN$. where $\mathrm{time}_{\Delta\!\!\Delta}(L, N)$ is the number of CPU seconds required by a Sparc10/42.

5 Conclusion

$\Delta\Delta$ is based on some simple ideas that can certainly be exploited in other contexts as well. The first idea is to understand the task's objectives. Here, our goal is NOT to evaluate the accuracy of each theory, but rather to identify the best neighboring theory, providing it is better than the current theory. This allows us to ignore any neighbor that is guaranteed to be either strictly worse than the current theory, or strictly worse than another neighbor. A second idea is to analyze the underlying performance system, and then seek efficient data-structures that encode information common to many subtasks, which can be used to produce the needed information. Here, for example, the $\rho(\cdot)$ and $\tau(\cdot)$ information can be used for computing the difference in the accuracy scores of the initial theory and each modified one.

We used these ideas to improve the wrapper-esque Δ, producing the $\Delta\Delta$ theory revision system. We have also demonstrated that $\Delta\Delta$ requires considerably less time to run than Δ — by a factor of over 35 — while producing effectively the same results, when handling real-world knowledge bases. This improvement is significant, as it means the revision system can be used interactively, rather than in batch-mode.

References

[Coh90] William W. Cohen. Learning from textbook knowledge: A case study. In *Proceeding of AAAI-90*, 1990.

[CS90] Susan Craw and Derek Sleeman. Automating the refinement of knowledge-based systems. In L.C. Aiello, editor, *Proceedings of ECAI 90*. Pitman, 1990.

[Gre95] Russell Greiner. The complexity of theory revision. In *Proceedings of IJCAI-95*, 1995.

[GRM95] Russell Greiner, R. Bharat Rao, and Glenn Meredith. An optimized theory revision module. Technical Report SCR-95-TR-538, Siemens Corporate Research, 1995.

[JKP94] George H. John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. In *Proceedings of the Eleventh International Machine Learning Conference*, pages 121–129, N.J., 1994. Morgan Kaufmann.

[LDRG94] Pat Langley, George Drastal, R. Bharat Rao, and Russell Greiner. Theory revision in fault hierarchies. In *Proceedings of The Fifth International Workshop on Principles of Diagnosis (DX-94)*, New Paltz, NY, 1994.

[MB88] S. Muggleton and W. Buntine. Machine invention of first order predicates by inverting resolution. In *Proceedings of IML-88*, pages 339–51. Morgan Kaufmann, 1988.

[OM94] Dirk Ourston and Raymond J. Mooney. Theory refinement combining analytical and empirical methods. *Artificial Intelligence*, 66(2):273–310, 1994.

[Pol85] P.G. Politakis. *Empirical Analysis for Expert Systems*. Pitman Research Notes in Artificial Intelligence, 1985.

[WP93] James Wogulis and Michael J. Pazzani. A methodology for evaluating theory revision systems: Results with Audrey II. In *Proceedings of IJCAI-93*, pages 1128–1134, 1993.

A Computing $acc(\, Delete_\chi(KB),\, c\,) - acc(\, KB,\, c\,)$

We[10] first describe the internal data structure we use: Given a labeled (partial) instance $\langle \pi, r_{cor} \rangle$ and a theory KB, for each node n in KB, define $\rho(n)$ to be the probability of reaching the correct repair r_{cor}, given that S_{KB} has reached n. Notice that $acc(\, KB,\, \langle \pi, r_{cor} \rangle\,) = \rho(\, n_{root}\,)$, where n_{root} is the root of KB; and also that $\rho(n) \;=\; p'(n) \times \text{AccNode}^*(n,\, \pi,\, r_{cor}\,)$, using the AccNode* routine shown in Figure 6.

For each arc $e = \langle n_1, n_2 \rangle$ in KB, let $\tau(e)$ represent the probability that S_{KB} will reach e. To define this quantity, we first define $\tau(n)$ as the probability of reaching the *node* n. Clearly $\tau(n_{root}) = 1$. Given an internal node n with children $n_i = child(n,\, i)$, let $e_i = \langle n, n_i \rangle$ be the arc connecting n to n_i. Then recursively define

$$\begin{aligned} \tau(e_1) &= \tau(n) \times p'(n) \\ \tau(e_{j+1}) &= \tau(e_j) \times (1 - p'(n_j)) \end{aligned}$$

Finally, to compute each $\tau(n_j)$, just add up the τ-values of the arcs reaching n_j; *i.e.*, $\tau(m) = \sum_{\langle n,m \rangle \in E} \tau(\langle n, m \rangle)$. Notice this can be computed in time linear in the size of the theory.

We can now use those $\rho(\cdot)$ and $\tau(\cdot)$ values to compute $\text{diff}'(\, Delete_{,}\, c\,)$: Let $KB' = Delete_{a,b}(KB)$ the theory produced by deleting from KB the link connecting node a to node b. Clearly $acc(\, KB',\, \langle \pi, r \rangle\,) = acc(\, KB,\, \langle \pi, r \rangle\,)$ whenever either $\pi(t(a)) = -$ or $\pi(t(b)) = -$. We will therefore deal only with situations when $\pi(t(a)) \in \{+, ?\}$ and $\pi(t(b)) \in \{+, ?\}$. W.l.o.g., we can write

$$\begin{aligned} acc(\, KB,\, \langle \pi, r_{cor} \rangle\,) \;&=\; \text{P[success before } \langle a, b \rangle] \;+\; \tau[\langle a, b \rangle] \times \rho(b) \\ &+\; \tau[\langle a, b \rangle](1 - p'(b)) \times \text{P[success after } \langle a, b \rangle \,|\, \text{reached } \langle a, b \rangle] \\ &+\; \tau[a](1 - p'(a))\text{P[success after } a] \end{aligned}$$

where P[success before $\langle a, b \rangle$] is the probability of reaching r_{cor} following the arcs that occur strictly *before* reaching the $\langle a, b \rangle$ arc (in the obvious traversal of KB); P[success after $\langle a, b \rangle \,|\, \text{reached } \langle a, b \rangle$] is the probability of reaching r_{cor} following the arcs that occur strictly *after* the $\langle a, b \rangle$ arc, given that S_{KB} has reached the $\langle a, b \rangle$ arc; and P[success after a] is the probability of reaching r_{cor} following the arcs that occur strictly *after* the a node.

For example, using KB_0 from Figure 1, identify the a with the fault node "A" and b with "E". Then P[success before $\langle a, b \rangle$] is the probability that either (1) $t(\texttt{Z})$, $t(\texttt{A})$ and $t(\texttt{C})$ all succeed and the correct repair is $r_{\texttt{C}}$, or (2) $t(\texttt{Z})$, $t(\texttt{A})$ and $t(\texttt{D})$ all succeed, $t(\texttt{C})$ fails, and the correct repair is $r_{\texttt{D}}$. Here, P[success after $\langle a, b \rangle \,|\, \text{reached } \langle a, b \rangle$] is 0, as this is the probability of reaching a success node under A, strictly after reaching and traversing $\langle a, b \rangle \equiv \langle A, E \rangle$. (Notice E has no "right-siblings" under A.)[11] Finally, P[success after a] is the probability that $t(\texttt{A})$ fails, $t(\texttt{B})$ succeeds and either $t(\texttt{E})$ succeeds and the correct repair is $r_{\texttt{E}}$, or $t(\texttt{E})$ fails $t(\texttt{F})$ succeeds and the correct repair is $r_{\texttt{F}}$.

[10] The reviewers should view the material in this appendix as strictly *optional* reading.

[11] If we had, instead, identified a with "A" and b with "D", then P[success after $\langle a, b \rangle \,|\, \text{reached } \langle a, b \rangle$] = P[success after $\langle A, D \rangle \,|\, \text{reached } \langle A, D \rangle$] is the probability that $t(\texttt{E})$ succeeds and the correct repair is $r_{\texttt{E}}$.

```
Procedure AccNode*( n: node, π: (partial)_problem_instance, r: repair )
 If n is a leaf node
 Then return( r(n) = r ?  1 :  0 )
 Else tot = 0;  reach_me = 1;   unknowns = {}
      For i = 1..k(n)
        n_i = child(n, i)
        If π( t(n_i) ) = +
        Then  tot += reach_me × AccNode*( n_i, π, r )
              return( tot )
        ElseIf π( t(n_i) ) = ?
        Then  If ¬t(n_i) ∈ unknowns
              Then tot += reach_me × AccNode*( n_i, π, r )
                   return( tot )
              ElseIf t(n_i) ∉ unknowns
              Then unknowns = unknowns ∪ t(n_i)
                   tot += reach_me ×p(n_i)× AccNode*( n_i, π, r )
                   reach_me *= 1 − p(n_i)
              /* if "t(n_i) ∈ unknowns", then either */
              /*    if π(t(n_i)) = −, this node will be ignored;    or
                    if π(t(n_i)) = +, S_KB has already followed earlier branch. */
      End For
      return( tot )
End AccNode*
```

Figure 6: AccNode* subroutine

Similarly,

$$\begin{aligned} acc(\, KB',\, \langle \pi, r_{cor} \rangle\,) \quad &= \quad \text{P[success before } \langle a,b \rangle] \\ &+ \quad \tau[\langle a,b \rangle] \times \text{P[success after } \langle a,b \rangle \,|\, \text{reached } \langle a,b \rangle] \\ &+ \quad \tau[a](1-p'(a))\text{P[success after } a] \end{aligned}$$

Subtracting these quantities, we find

$$\text{diff}'(\, Delete_{a,b},\, \langle \pi, r_{cor} \rangle\,) \quad = \quad \tau[\langle a,b \rangle]\ \{\ [p'(b) \times P'] - \rho(b)\ \}$$

where $P' = \text{P[success after } \langle a,b \rangle \,|\, \text{reached } \langle a,b \rangle] = \sum_{\ell=m+1}^{k} \tau[\langle a, a_\ell \rangle] \times \rho(a_\ell)/\tau[\langle a,b \rangle]$, where b is the m^{th} child of a, and a has $k \geq m$ children, $\{a_1, \ldots, a_{m-1}, a_m = b, a_{m+1}, \ldots, a_k\}$.