Two Heads are Better than One: Causality and Similarity in Misconception Discovery

Raymund Sison (sison@cs.titech.ac.jp)
Masayuki Numao (numao@cs.titech.ac.jp)
Department of Computer Science
Tokyo Institute of Technology, Japan

Masamichi Shimura (shimura@ia.noda.sut.ac.jp)
Department of Industrial Administration
Science University of Tokyo, Japan

Abstract

MMD is an algorithm that learns without supervision intensional definitions of classes of knowledge errors using data (similarity) and theory (causality). Causality can be especially useful when similarity fails to discover certain errors due to their entanglement in complex behaviors, while similarity can be especially useful when no causal relationships for robust co-occurring discrepancies are present in the background knowledge. This paper examines the individual and combined effectiveness of MMD's similarity and causality components in discovering error classes and classifying behaviors in which these errors occur. Experimental results show how similarity and causality can serve to complement each other in the discovery of novice PROLOG programmer errors.

Introduction

Although most concept formation systems in artificial intelligence (Gennari, Langley & Fisher, 1989) as well as concept learning models in cognitive psychology (Komatsu, 1992) have tended to rely almost exclusively on similarities in the data, evidence is mounting that theories and goals are at least as important as data in the formation of concepts (Murphy & Medin, 1985; Barsalou, 1991; Rips & Collins, 1993; Wisniewski & Medin, 1994).

In (Sison, Numao & Shimura, 1997) an algorithm called MMD[1], was presented that utilizes similarity and causality for unsupervised concept formation in a more tightly coupled way than previous systems have. Moreover, MMD deals with the formation of categories that are intended to represent classes of *knowledge errors*.[2] Thus, the usual problem of concept formation is complicated by the additional requirement that the conceptual descriptions that are formed also need to be explainable, at least in terms of causal relationships, since a conceptual description that is an ordinary conjunction of seemingly correlated discrepancies in novice behavior can hardly be considered as representing a knowledge error unless causal relationships among the discrepancies can be found.

This paper further examines the utility of coupling data and theory in the discovery of knowledge errors of novice PROLOG programmers. Specifically, it examines (1) the usefulness of the causality component of MMD especially when the similarity component fails to discover certain errors due to these errors' entanglement in complex behaviors (particularly in novice PROLOG programs with multiple bugs), and (2) the usefulness of the similarity component when no causal relationships for robust co-occurring discrepancies are present in the background knowledge. In what follows, we first review the basic representation and algorithm of MMD. We then present and discuss results of experiments that reveal how similarity and causality can serve to complement each other in the discovery of novice PROLOG programmer errors.

MMD: A Review and Closer Look

Representation

The objects that MMD deals with — discrepancies in behavior — are represented as sets of relational descriptions (specifically, as atomic formulas in the function-free first order logic). Consider the following ideal behavior in the form of a correct PROLOG clause for the reversal of the elements of a list:[3]

```
% correct clause %
reverse([H|T],R) :-        % head
     reverse(T,T1),        % subgoal1
     append(T1,[H],R).     % subgoal2
```

The correct clause has a head, `reverse([H|T],R)`, which states that the reverse of a list that is made up of a first element, `H`, called the list's head, and a sublist, `T`, called the list's tail, is `R`. `R` is determined in the clause's body, which has two subgoals. The first subgoal, `reverse(T,T1)`, states that the reverse of the list `T` is `T1`. The second subgoal, `append(T1,[H],R)`, states that `R` is just the concatenation of the list `T1` and the element `H`. In short, the clause as a whole recursively states that the reverse of a list is the concatenation of the reverse of its tail and its head.

[1] Multistrategy Misconception Discovery

[2] *Misconceptions* are incorrect or inconsistent knowledge — facts, procedures, concepts, principles, schemata or strategies — about a domain that result in errors in behavior. Behavioral errors can also be due to insufficient knowledge, however, and we use the term *knowledge error* to include insufficient knowledge as well as incorrect or inconsistent knowledge.

[3] Words or phrases that come after a % symbol in a PROLOG program are considered comments; they are not part of clause definitions.

Now consider the following actual student behavior, which is incorrect.

```
% buggy clause %
reverse([H|T],[T1|H]) :-
    reverse(T,T1).
```

The above clause differs from the correct clause in two ways. First, in place of the variable `R` in the head of the correct clause, the buggy clause has the list `[T1|H]` in its head. Expressed in relational form, this discrepancy is:

`replace(head,R,[T1|H]).`

Second, the second subgoal in the correct clause is omitted in the buggy clause. Expressed in relational form, this discrepancy is:

`remove(subgoal2).`

The two discrepancies just described form a *discrepancy set*, and constitute one input object to MMD. MMD takes one such discrepancy set at a time and clusters it into an error hierachy (Figure 1A; we shall explain the figure shortly).

For each input discrepancy set, MMD outputs two things: (1) a revised error hierarchy (Figure 1B) and set of intensionally defined error classes,[4] and (2) the error class or classes to which a discrepancy set (as well as its corresponding program) belongs.

Algorithm

MMD uses a similarity measure that adapts Tversky's (1977) contrast model, and a set of causality heuristics to classify a discrepancy set into an error hierarchy. The similarity measure is:

$$Sim(C,O) = \theta f(C \cap O) - \alpha f(C - O) - \beta f(O - C)$$

where C and O are sets of behavioral discrepancies; θ, α, and β are user-redefinable parameters; $f(X)$ returns the cardinality of X; and, the set of commonalities $(C \cap O)$ is:

$$(C \cap O) = Com(C,O) = \bigcup_{i=1}^{m} \bigcup_{j=1}^{n} lgg(C_i, O_j)$$

where *lgg(x,y)* is the *least general generalization* (Plotkin, 1970; Muggleton & Feng, 1990) of discrepancies x and y in the function-free first order logic, and m and n are the number of discrepancies in C and O, respectively.

The causality heuristics are:

1. *Component-level causality:* Causal relationships among the components of the ideal behavior that are present in a set of discrepancies suggest causal relationships among these discrepancies.
2. *Concept-level causality:* The co-occurrence of two discrepancies *d1* and *d2* in a generalization node of an error hierarchy, where *d1* is an intersection generalization and *d2* is a variabilization generalization, suggests that *d1* causes *d2*.

[4]Each subtree (minus its variable-instantiating leaves) under the root node of an MMD-induced hierarchy forms an intensional definition of a class of errors which, in turn, represents a misconception or other knowledge error.

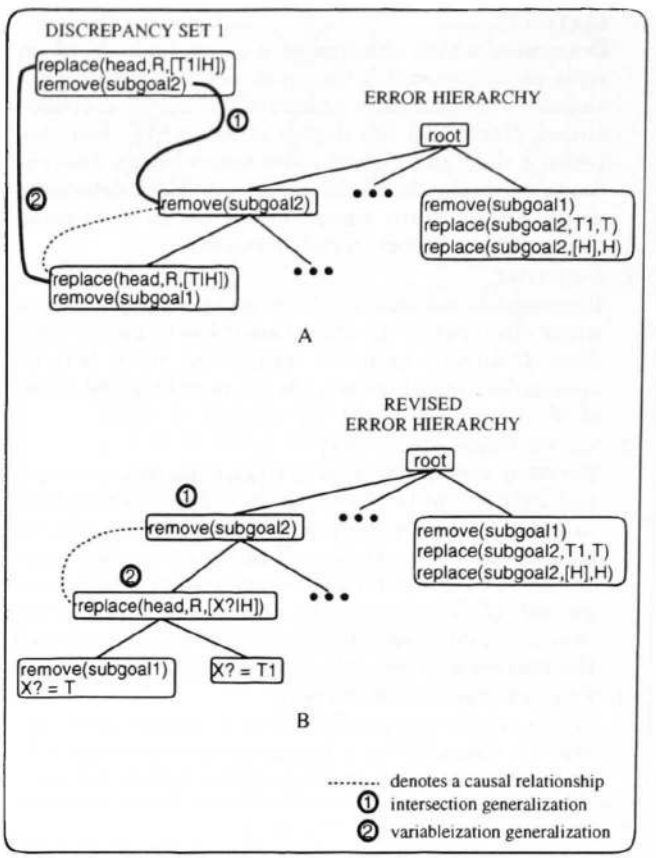

Figure 1: Clustering set of behavioral discrepancies into an error hierarchy I (See explanation in text.)

3. *Subconcept-level causality:* Causal relationships between a parent node *d1* of an error hierarchy and its child *d2* suggests that, all other things being equal, *d2* causes *d1*.

We shall illustrate the use of the first heuristic later.[5]

MMD, whose algorithm is summarized in Figure 2, takes one discrepancy set at a time and classifies this recursively into the nodes of the error hierarchy that match it to a certain degree (Figure 2, steps 1 and 2). Referring back to Figure 1A, for example, note that the node `remove(subgoal2)` below the `root` node of the error hierarchy is actually one of the discrepancies in the input discrepancy set (`remove(subgoal2)` is called an 'intersection' generalization of the two). Assuming that the γ parameter of step 1 indicates that this is a match, the input discrepancy set will therefore be classified into this subtree, among possible others.

Next, the remaining discrepancy `replace(head,R,[T1|H])` of the input discrepancy set is compared against the child node [`replace(head,R,[T|H])`, `remove(subgoal1)`]. This time, a 'variabilization' gen-

[5]The second and third heuristics do not directly affect the results we discuss in this paper, and are provided only for completeness. Further details regarding the similarity measures, causality heuristics, and algorithm of MMD can be found in (Sison, Numao & Shimura, 1998).

1. MATCH.
   Determine which children of a given node N of an error hierarchy *match* the input set of input discrepancies D. Specifically, compute the set of commonalities, Com, and the degree of similarity, Sim, between a child and D, and determine whether Sim exceeds a system threshold, γ. In addition, determine causal relationships among discrepancies in D using the component-level causality heuristic.
2. POSITION.
   If no match is found, or if the input discrepancy is a single discrepancy (in which case it has no causal ties), place D directly under N. Otherwise, place D in its appropriate position vis-a-vis the matching child(ren) of N.
3. SEVER (unnecessary ties).
   For every new node created in (2), determine concept- and subconcept-level causalities. If no concept-level causality exists among discrepancies in this node, retain the node nevertheless. If this node is a leaf node, and no subconcept-level causality exists between components of this node and its parent, sever the link between this child and its parent, and reclassify it (and the current subtree) into the hierarchy.
4. FORGET (the occasional slip).
   Nodes whose weights fall below a system parameter may be discarded on a regular or demand basis. (No nodes are discarded in the current implementation.)

Figure 2: Basic procedure of MMD

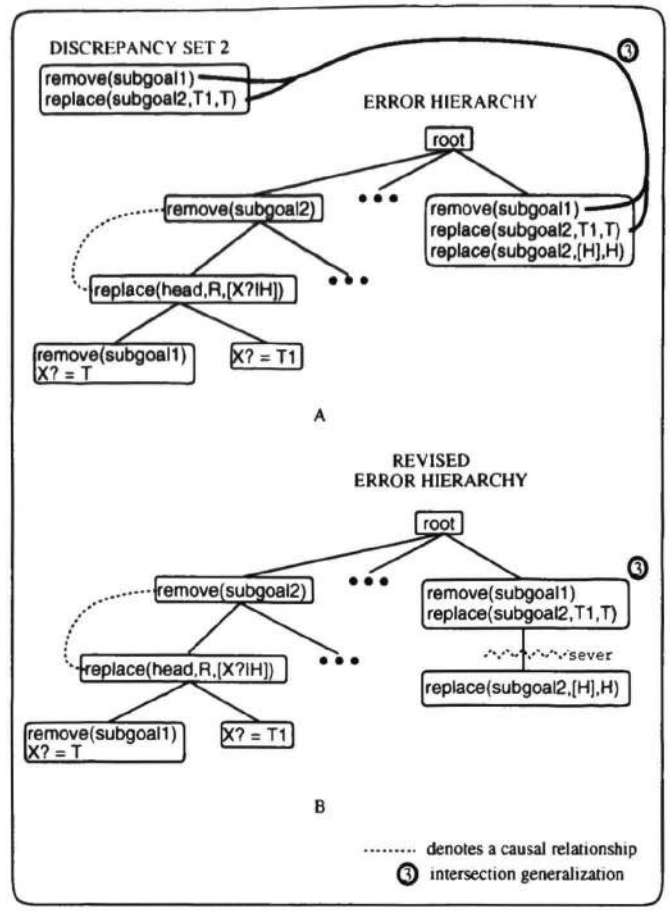

Figure 3: Clustering a set of behavioral discrepancies into an error hierarchy II (See explanation in text.)

eralization (`replace(head,R,[X?|H])`) occurs,[6] and so a node for this variabilization is created, and the instantiations of the pattern variable `X?` (i.e., `X?=T` and `X?=T1`), among others, are pushed down to the next level. Figure 1B shows the revised hierarchy. The left subtree in Figure 1B (though not complete in the figure) characterizes a misconception in which the `append/3` relation (in `subgoal2`) and the `[|]` operator are thought to be functionally the same, at least as far as concatenating two lists is concerned. This is in fact the misconception underlying the student's buggy clause presented earlier.

The SEVER Operator In addition to the hierarchical reorganization that step 2 entails, the absence of causal relationships between a parent and a child in a hierarchy can cause further reorganization: said child can be severed from its parent and then reclassified (step 3 (SEVER)). This allows the algorithm to disentangle the multiple misconceptions or knowledge errors that may have produced the discrepancies in a student's behavior. Consider the following buggy clause:

```
% buggy clause 2 %
reverse([H|T],R) :-
     append(T,H,R).
```

The discrepancies between the above clause and the correct clause presented earlier are:

```
remove(subgoal1),
replace(head,T1,T).
```

Clustering the discrepancies of the above buggy clause into the error hierarchy in Figure 3A produces the revised error hierarchy in Figure 3B. Now note that the node `replace(subgoal2,[H],H)` in the right subtree of Figure 3B is not at all (causally) related to its parent ([`remove(subgoal1)`, `replace(subgoal2,T1,T)`]).

Since the child `replace(subgoal2,[H],H)` is not related to its parent, MMD severs (denoted in the figure by a jagged line) the child from the parent and then reclassifies the child into the hierarchy. This particular severing operation is consistent with the fact that omitting or forgetting to put the list brackets `[]` around a variable (denoted by the discrepancy `replace(subgoal2,[H],H)`) has nothing to do with omitting the first subgoal of the correct clause (i.e., the discrepancy `remove(subgoal1)`), and the natural consequence of the latter, which is using some other variable in place of `T1` in the second subgoal of the correct clause (i.e., the discrepancy `replace(subgoal2,T1,T)`).

It will be noted that the SEVER operator only applies between a child and an unrelated parent; i,e., a subset of discrepancies will not be severed from its original set, S, even if no causal relationship can be found between it and S, unless it has already been 'gently pushed out of'

[6] Variabilization generalizations correspond to Markman and Gentner's (1993) 'alignable differences.'

S, only connected to S by a parent-child link. In other words, the SEVER operator will not split a node even though its contents are unrelated with respect to the background knowledge. This conservative approach of retaining nodes despite the absence of component-level causalities thus takes into account the possibility that the background knowledge may be incomplete.

Experiments and Discussion

Experiments

To empirically examine the individual and combined effectiveness of MMD's similarity and causality components in discovering error classes and classifying behaviors accordingly, we compare the performance of MMD against its similarity and causality components working 'alone' on the discrepancy sets of 64 buggy `reverse/2` and 56 buggy `sumlist/2` PROLOG programs[7] obtained from third year undergraduate students learning the language. We shall call the similarity and causality components SMD and CMD, respectively. Specifically, SMD is MMD without the checks for causality in steps 1 and 3 (of Figure 2), while CMD is MMD in which the last part of step 1 would be changed to read "... *threshold*, γ and causal relationships exist among the discrepancies in *Com*." The latter modification implies that for two sets of discrepancies to match, their similarity value must not only be above the system threshold, but their commonalities must also have (the same) inter-discrepancy causal relationships.

Performance is viewed from two perspectives, namely, (1) *program classification* (i.e., the percentage of buggy programs, or more specifically, their corresponding discrepancy sets, that are correctly classified) and (2) *error discovery* (i.e., the percentage of misconceptions and knowledge errors, or more specifically, their corresponding error classes, that are correctly discovered). Performance accuracy is determined by comparing the classification and discovery results of SMD/CMD/MMD against the error categories and groupings that were produced by a team of PROLOG teachers who have examined (1) the buggy programs and (2) MMD's output. Specifically, performance accuracy is the percentage of buggy programs (discrepancy sets) correctly classified or knowledge errors (error classes) correctly discovered by SMD/CMD/MMD with respect to those of the experts.

Earlier we have noted that a buggy program can exhibit more than one misconception or knowledge error. Thus, in our experiments, a buggy program is considered *fully* classified only if all the misconceptions and knowledge errors underlying it are detected; otherwise, it is only *partially* classified, receiving only a partial point ($\frac{1}{n}$), where n is the total number of knowledge errors underlying the program. Similarly, a knowledge error is considered *fully* discovered (with respect to past data) only if the knowledge error's intensional definition contains all the manifestations that the error can assume in a buggy program; otherwise, it is only *partially* discovered, receiving only a partial point ($\frac{1}{m}$), where m is the total number of possible ways in which the error has been observed to appear in a program. Partially classified buggy programs or partially discovered error classes can, of course, be awarded full points. This is appropriate when, for example, all that is required in a certain application is for one student error to be determined.

[7] For the naive reversal of elements of a list and for summing the elements of a list of numbers, respectively.

Table 1: Results for the `reverse` dataset

	BC		ED	
Algorithms	P-P	P-F	P-P	P-F
SMD	61	79	34	38
CMD	68	81	44	50
MMD	84	94	70	88

BC: Buggy Programs Correctly Classified (%)
ED: Error Classes Correctly Discovered (%)
P-P: Partial classification/discovery awarded Partial point
P-F: Partial classification/discovery awarded Full point

Table 2: Results for the `sumlist` dataset

	BC		ED	
Algorithms	P-P	P-F	P-P	P-F
SMD	76	81	55	58
CMD	44	45	63	63
MMD	95	97	75	75

BC: Buggy Programs Correctly Classified (%)
ED: Error Classes Correctly Discovered (%)
P-P: Partial classification/discovery awarded Partial point
P-F: Partial classification/discovery awarded Full point

The mean accuracies of SMD, CMD, and MMD for 5 random orderings of the input datasets are shown in Tables 1 and 2. These results were obtained using the following parameter settings: $\theta = 1$, $\alpha = 0.5$, $\beta = 0.5$, and $\gamma \geq 0$. These values of θ, α and β are intuitive and give good results for $\gamma \geq 0$. Other values of γ are possible, but are somewhat more arbitrary and contrived. Using different values for α and β (e.g., to model some asymmetry) do not yield better results, at least not in the two datasets. (Sison *et al.*, 1997) shows some results for different values of these parameters. The first causality heuristic is implemented using input-output relationships between components in the ideal behavior (see appendix).

Results and Discussion

The overall result is that MMD can identify the bugs in most of the programs, and can do this more effectively than either of its similarity or causality components working alone. Tables 1 and 2 show that MMD's classification performance (P-P) was 38% better than SMD's and 24% better than CMD's on the `reverse` dataset, and 25% better than SMD's and more than a hundred percent (!) better than CMD's on the `sumlist` dataset. As far as discovery performance is concerned, MMD (P-P) was almost a hundred percent (!) better than SMD and almost fifty percent better than CMD on the `reverse` dataset, and 36% better than SMD and 19% better than CMD on the `sumlist` dataset. The dif-

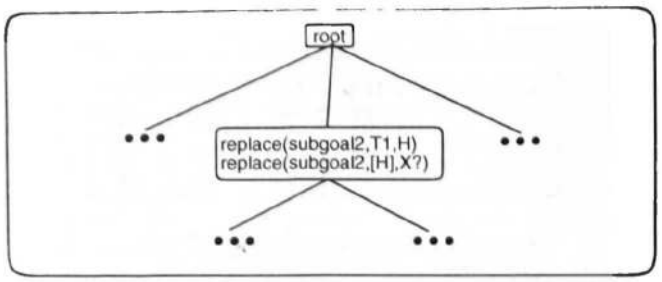

Figure 4: An error unlearnable by CMD

ferences were even greater when partially classified programs or partially discovered error classes were given full points (P-F).

Moreover, MMD's P-P-to-P-F ratios for program classification performance (84/94 or .89 for `reverse`, 95/97 or .98 for `sumlist`) were higher than those of SMD's (61/79 or .77 for `reverse`, 76/81 or .94 for `sumlist`), indicating that MMD was better able to disentangle multiple errors as a result of its causality component. This is the main reason for what was reported in previous papers as MMD's superior performance compared to SMD.

Although CMD's P-P-to-P-F ratio for program classification performance was quite high, quite as high in fact as MMD's for the `sumlist` dataset (44/45 or .98), its rather surprisingly dismal performance (both P-P and P-F values for classification accuracy were only half as good as MMD's on this dataset) quickly negates this slight P-P-to-P-F advantage. This poor performance can be traced to the fact that CMD requires that two similar discrepancy sets have (the same) inter-discrepancy causal relationships. Thus, for example, CMD was not able to learn the subtree in Figure 4 because the discrepancies in the node [`replace(subgoal1,T1,H)`, `replace(subgoal2,[H],X?)`] are not causally related according to the first causality heuristic and the background knowledge (in the appendix).

Unfortunately, not all co-occurring discrepancies in the above experiments had causal relationships that could be found in or explained by the background knowledge. This is not to say that there are are no causal relationships between these co-occurring discrepancies; this only says that the background knowledge of causal relationships that was used in the experiments was not complete. While a perfectly complete background knowledge is desirable, it might not always be feasible. By retaining such robust (i.e., frequently co-occurring but unseparatable by a parent-child link) groups of discrepancies in spite of the absence of causal relationships in the background knowledge, MMD can in fact correctly discover more errors and classify more programs than CMD. In addition, such as-yet-not-fully-understandable co-occurrences[8] can be used to trigger a new kind of discovery process involving the search for new causal relationships.

The results also indicate some important differences between the two datasets. For example, the `sumlist` dataset seems to be simpler than the `reverse` dataset in the sense that buggy programs in the former tend to each have only one misconception or knowledge error. In contrast, many programs in the `reverse` dataset are more complex, having multiple knowledge errors. Thus, the ratios of the P-P values to the P-F values are smaller in the `reverse` dataset than in the `sumlist` dataset for SMD, CMD, and MMD.

Finally, although MMD was not able to discover all error classes, MMD succeeded in discovering the more common ones. This explains why the classification rates were higher relative to the discovery rates.

Related Work

The similarity component of MMD is similar to UNIMEM (Lebowitz, 1987) and COBWEB (Fisher, 1987), which are also incremental concept formation systems. Like UNIMEM, it adopts a set theoretic concept representation. UNIMEM's similarity measure, however, considers only the differences between two sets of features. Moreover, UNIMEM only deals with attribute-value descriptions. COBWEB uses a probabilistic concept representation and a corresponding probabilistic similarity measure (category utility (Gluck & Corter, 1985)), and can thus only produce disjoint clusters of whole objects. In terms of explaining errors in novice behavior, this means that a buggy behavior can only be classified under one misconception, though it may well be symptomatic of several. Like UNIMEM, COBWEB deals only with attribute-value descriptions (but see the COBWEB variant in (Thompson & Langley, 1991)).

Causal relationships among features that have been previously clustered can be induced or deduced in a variety of ways. In the UNIMEM extension proposed in (Lebowitz, 1986), for example, frequently occurring features in other concepts are considered causative, from which one can forward-chain to the other features using heuristic, low-level, causal domain rules. In MMD, causality between discrepancies is determined by the existence of causal relationships among their components, and whether a generalized discrepancy is an intersection or a variabilization, or if it is a parent or a child. There is no need in MMD for a given set of heuristic directional causal rules linking discrepancies, though the causal relationships among their components can be viewed as a lower-level and less ad hoc form of such. In (Pazzani, 1993), which likewise uses a UNIMEM-like clusterer, there are two kinds of features, namely, actions and state changes, and actions are always the causative features. Explanation is achieved by instantiating general causal domain rules or templates. In our approach, any discrepancy is potentially causative, and no general causal domain rule templates are necessary.

[8] We say 'not fully' understandable or explainable because, although MMD's second causality heuristic can be used to inductively determine which discrepancy causes which (e.g., the second causality heuristics suggests that the first discrepancy in Figure 4 causes the second), this still does not specify the exact nature of the relationship.

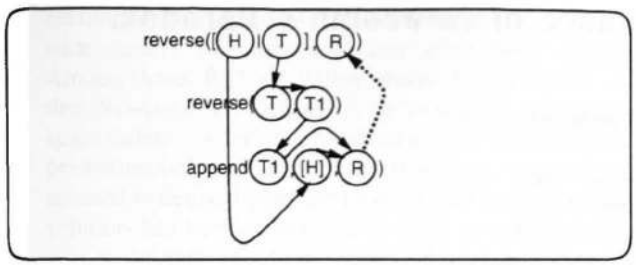

Figure 5: Input-output (causal) relationships between ideal behavior components

Conclusion

In this paper, we have reviewed the basic representation and algorithm of MMD, and presented and discussed results of experiments in which the individual and combined use of MMD's similarity and causality components were examined to evaluate their usefulness in discovering novice programmer errors. Results have shown that MMD can effectively discover such errors and classify discrepant behaviors according to these errors. Moreover, results have revealed the usefulness of the causality component of MMD especially when the similarity component fails to discover certain errors due to these errors' entanglement in complex behaviors (particularly in PROLOG programs with multiple bugs), and the usefulness of the similarity component when no causal relationships for robust co-occurring discrepancies are present in the background knowledge. Future work involves the incorporation of a failure-driven mechanism for extending the set of causal relationships in the background knowledge. MMD is part of MULSMS (Sison & Shimura, 1996), a framework for a multistrategic learning student modeling system.

Appendix

Figure 5 shows the input-output relationships between components of the correct `reverse` clause presented earlier. To illustrate, recall the discrepancy set of Figure 1A, which is made up of the following two discrepancies:

```
replace(head,R,[T1|H]),
remove(subgoal2).
```

These two discrepancies are causally related according to the first causality heuristic because both involve the user variable `R`, and the `R` involved in the second discrepancy can be viewed as causing (emphasized in Figure 5 using a broken line) the `R` involved in the first. Of course, the `R` in the head and the `R` in the second subgoal of the correct clause refer to one and the same thing. To be more precise, therefore, what we mean by the `R` in the subgoal "causing" the `R` in the head is that: given that both `R`'s are output variables, the subgoal (rather than the head) is responsible for binding `R` to a value.

Acknowledgment

We thank Ethel Chua Joy and Philip Chan for their help in collecting and analyzing the buggy programs.

References

Barsalou, L. (1991). Deriving categories to achieve goals. *The Psychology of Learning and Motivation, 27*, 1–64.

Fisher, D. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning, 2*, 139–172.

Gennari, J., Langley, P., & Fisher, D. (1989). Models of incremental concept formation. *Artificial Intelligence, 40*, 11–61.

Gluck, M. & Corter, J. (1985). Information, uncertainty, and the utility of categories. *Proceedings of the Seventh Annual Conference of the Cognitive Science Society.*

Komatsu, L. (1992). Recent views of conceptual structure. *Psychological Bulletin, 112*(3), 500–526.

Lebowitz, M. (1986). Integrated learning: Controlling explanation. *Cognitive Science, 10*, 219–240.

Lebowitz, M. (1987). Experiments with incremental concept formation: Unimem. *Machine Learning, 2*, 103–138.

Markman, A. & Gentner, D. (1993). Splitting the differences: A structural alignment view of similarity. *Journal of Memory and Language, 32*, 517–535.

Muggleton, S. & Feng, C. (1990). Efficient induction of logic programs. *Proceedings of the Conference on Algorithmic Learning Theory.*

Murphy, G. & Medin, D. (1985). The role of theories in conceptual coherence. *Psychological Review, 92*(3), 289–316.

Pazzani, M. (1993). Learning causal patterns: Making a transition from data-driven to theory-driven learning. *Machine Learning, 11*(2/3), 173–194.

Plotkin, G. (1970). A note on inductive generalization. *Machine Intelligence, 5*, 153–163.

Rips, L. & Collins, A. (1993). Categories and resemblance. *Journal of Experimental Psychology: General, 122*(4), 468–486.

Sison, R. & Shimura, M. (1996). The application of machine learning to student modeling: Toward a multistrategic learning student modeling system. *Proceedings of the European Conference on Artificial Intelligence in Education.*

Sison, R., Numao, M., & Shimura, M. (1997). On using theory and data in misconception discovery. *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society.*

Sison, R., Numao, M., & Shimura, M. (1998). Discovering error classes from discrepancies in novice behaviors via multistrategy conceptual clustering. *User Modeling and User-Adapted Interaction (Special Issue on Machine Learning for User Modeling), 8.* To appear.

Thompson, K. & Langley, P. (1991). Concept formation in structured domains. In D. Fisher, M. Pazzani, & P. Langley (Eds.), *Concept Formation: Knowledge and Experience in Unsupervised Learning.* San Mateo, CA: Morgan Kaufmann.

Tversky, A. (1977). Features of similarity. *Psychological Review, 84*(4), 327–352.

Wisniewski, E. & Medin, D. (1994). On the interaction of theory and data in concept learning. *Cognitive Science, 18*, 221–281.