Source Selection for Analogical Reasoning An Empirical Approach

William A. Stubblefield

Sandia National Laboratories
P. O. Box 5800
Albuquerque, New Mexico 87185
wastubb@sandia.gov

George F. Luger

Department of Computer Science
University of New Mexico
Albuquerque, New Mexico 87131
luger@cs.unm.edu

Abstract

The effectiveness of an analogical reasoner depends upon its ability to select a relevant analogical source. In many problem domains, however, too little is known about target problems to support effective source selection. This paper describes the design and evaluation of SCAVENGER, an analogical reasoner that applies two techniques to this problem: (1) An assumption-based approach to matching that allows properties of candidate sources to match unknown target properties in the absence of evidence to the contrary. (2) The use of empirical learning to improve memory organization based on problem solving experience.

Introduction

There are infinite things on earth; any one of them may be likened to any other. Likening stars to leaves is no less arbitrary than likening them to fish or birds.

Jorge Luis Borges
Labyrinths

Analogical reasoning (Gentner 1983; Hall 1989) makes inferences about novel *target* problems by transferring knowledge to them from a better understood *source* domain. Analogical reasoners generally choose a source that is known to have properties in common with the target under the assumption that this implies further commonalities and the potential for valid inferences. While this is a reasonable heuristic, many current implementations of it suffer from a number of limitations, including:

1. The requirement that enough be known about the target to select a relevant source. If the target domain is poorly understood, this assumption may not be justified.
2. Reliance on a restricted *retrieval vocabulary* (Kolodner 1993) to specify properties the reasoner may consider in choosing useful sources. Although efficiency requires such restrictions, many retrieval methods ignore the problem of their selection by assuming an a priori definition of a retrieval vocabulary.
3. The use of context independent measures of similarity. For example, spreading activation techniques (Mitchell 1993; Thagard 1988) measure the similarity of two concepts as a function of their closeness in a semantic network. By fixing similarity measures in the structure of the network, such methods have difficulty in responding to changing problem situations.
4. Failure to consider problem solving context when organizing source memory. Many systems use clustering algorithms like UNIMEM (Lebowitz 1980) and COBWEB (Fisher 1987) to construct a hierarchical source organization. These and similar methods ignore the structure of target problems when defining hierarchies. If such approaches are to take the problem solving context into account at all, they must do so implicitly, through biases in the retrieval vocabulary.

Hoffman (1995) has demonstrated that reliance upon a single "form, format or ontology" for background knowledge will obscure many analogies that would seem reasonable and useful to humans. These limitations restrict both the theoretical depth and practical usefulness of many current models of analogy. This paper discusses the design

and evaluation of SCAVENGER, an analogical reasoner that attempts to correct these problems by (1) integrating source selection and inference in an assumption-based retrieval mechanism, and (2) using empirical learning to improve memory organization through problem solving experience.

Like most analogical reasoners, SCAVENGER organizes its source base under a hierarchical index. Each index node describes properties that are common to a set of similar sources. Each node adds information to its parent, and indexes a subset of its parent's sources. SCAVENGER searches its hierarchy for nodes whose descriptions match components of the target. However, SCAVENGER takes an *assumption-based* approach to matching target problems with the patterns stored at index nodes: It requires properties known for the target to match exactly, but allows properties unknown in the target to assume any values required for a match. In a sense, rather than using the target problem description to select a source, it uses it to eliminate sources known to be irrelevant. Unlike approaches that treat retrieval and inference as separate steps, assumption-based retrieval integrates analogical inference with source selection.

On constructing a successful analogy, SCAVENGER updates its hierarchy by specializing the node that led to retrieval of the relevant analogical sources. *Empirical memory management* selects properties that best distinguish successful analogies from their competitors by using a variation of the ID3 learning algorithm (Quinlan 1986). It then uses these properties to refine the index hierarchy. By limiting the properties it considers to those that were effective in solving target problems, SCAVENGER both eliminates the need for a priori restrictions on its retrieval vocabulary, and considers problem solving experience in organizing source memory.

Although promising, this approach raises a number of questions and potential problems:

1. If little is known about a target problem, assumption-based retrieval will allow many potentially contradictory matches. It is possible that the overhead of maintaining these alternative hypotheses may cancel any advantage it provides over exhaustive search of all sources.

2. Because analogy is an unsound inference method, there is no guarantee that properties that characterized a useful source in one situation will be useful for other target problems. In using the ID3 learning algorithm with such potentially unsound data, we are violating many of the assumptions underlying its design. This may undermine its effectiveness.

The rest of this paper examines SCAVENGER's design and its ability to overcome these problems.

The SCAVENGER Architecture

Although we have tested SCAVENGER on several domains (Stubblefield 1995), this paper discusses its application to the problem of diagnosing children's errors in simple subtraction problems (Brown and VanLehn 1980; Burton 1982; VanLehn 1990). Where their DEBUGGY system used an analytic approach that considered all likely bugs in forming a diagnosis, we have used their diagnostic models and problem data to ask a different question: Can SCAVENGER learn enough to make effective first guesses about the cause of an error without performing an extensive analysis? There are several reasons this is an interesting test for SCAVENGER's retrieval algorithm:

1. Although human teachers are generally good at inferring the cause of a student's error, it seems unlikely that they use DEBUGGY's analytic methods. It is more likely that they recognize similarities between current problems and those they have seen in the past. This suggests the use of some form of analogical reasoning.

2. Subtraction problems offer few surface cues to the underlying causes of mistakes. This makes them a difficult challenge for an analogical reasoner.

3. A single error may result from multiple interacting bugs, further complicating diagnosis.

SCAVENGER represents analogical sources as class and method definitions using the Common LISP Object System. The source base used in these tests is a set of LISP functions that reproduce the bugs described in (Brown and VanLehn, 1980). SCAVENGER decomposes each subtraction problem into a series of unknown operations on pairs of digits. This takes the form of a LISP program containing unknown functions. For example, in diagnosing the cause of the erroneous subtraction:

634 - 468 = 276

SCAVENGER reformulates the problem as a sequence of LISP function evaluations:

```
(#:G873 4 8 w) -> ?
(#:G874 3 6 w) -> ?
(#:G875 6 4 w) -> ?
(show-result w) -> 276
```

In this target, w is an instance of the class working-

memory. w contains two slots: a borrow slot that records the value to be decremented from the next column, and a result slot that accumulates the column results as the program proceeds. Show-result returns the result accumulated in working memory. The methods, #:G873, #:G874 and #:G875, indicate unknown target operations. SCAVENGER diagnoses the error by finding an analogical mapping of target methods onto source operators that reproduces the erroneous behavior.

The source library contains methods for subtracting digits. Each of these takes two digits and an instance of working-memory, and returns an integer. Sources include the normal-subtract method and such error methods as borrow-no-decrement, borrow-from-zero, always-borrow, etc. For example, borrow-no-decrement borrows under the appropriate circumstances, but fails to decrement the column borrowed from.

Trying alternative mappings onto the target operators, SCAVENGER eventually reproduces the behavior seen in the target problem. In the example, it diagnoses the error's cause as a borrow-no-decrement bug:

(normal-subtract 4 8 w) -> 6
(borrow-no-decrement 3 6 w) -> 7
(borrow-no-decrement 6 4 w) -> 2
(show-result w) -> 276

Assumption-based Retrieval

Each operator in SCAVENGER's source base is stored along with:

1. Its *signature*, specifying the types of its arguments. Types used in this problem include a digit type, and sub-types for each individual digit (0, 1, . . .). For example, the signature of the borrow-from-zero operator is:

   digit-0 x digit x working-memory -> digit

   The numbers and types of arguments are the only information SCAVENGER uses to restrict matches between targets and sources.

2. A description of the bug. The original DEBUGGY research derived bugs from a procedural model of subtraction skills (Brown and VanLehn 1980; Burton 1982; VanLehn 1990). Each bug represents a different failure of one step in this model. In order to avoid representational biases that might distort our evaluation of SCAVENGER, we described each bug according to the stage of this model it effects. This yielded 6 bug descriptors:

   1. normal-op. This describes normal subtraction.
   2. transpose-error. An error in which the student subtracts the top digit from the bottom digit.
   3. borrow-error. Any failure in borrowing, such as failing to borrow, or always borrowing.
   4. decrement-error. Any failure in decrementing the digit borrowed from.
   5. subtract-error. A failure in subtracting digits, such as assuming that n-0 = 0.
   6. add-error. Adding instead of subtracting.

For example, borrow-no-decrement is represented by:

Operator: borrow-no-decrement
Signature: digit x digit x working-memory -> digit
Description: (decrement-error)
Definition: (defmethod . . .) ; the LISP definition

SCAVENGER stores sources under a hierachical index, where each node contains the signature and description shared by a class of similar operators. In the example hierarchy of Figure 1, the root contains no description pattern, and the child node describes borrow-no-decrement and 10 similar operators.

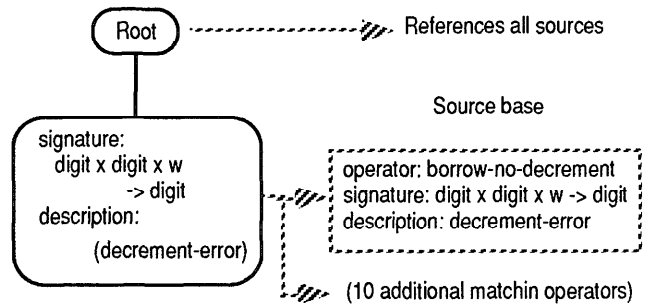

Figure 1

In searching this hierarchy, SCAVENGER forms and evaluates alternative hypotheses about the target. Each hypothesis results from a different sequence of index matches, and reflects different assumptions about the target. SCAVENGER constructs and evaluates hypotheses using the following algorithm:

1. Create an initial hypothesis based on the root node, in which no targets are matched. Move down the hierarchy, creating a new hypothesis for every matching index by:

   1.1. For each hypothesis, examine the children of the most specific index node that it has already matched.

   1.2. For each match between the description stored at a child index and an unmatched target operator in the hypothesis, create a new hypothesis. In it, record the matching index, the target that is matched, and transfer the index

signature and description to the target operator.

1.3. Repeat 1.1 & 1.2 until no more hypothesis are produced.

2. Sort all hypotheses according to heuristic merit.

3. Trying each hypothesis in order, retrieve all sources stored under its matching indices, and construct a partial analogy for each consistent combination of retrieved sources.

4. Complete each of these partial analogies using all sources that match the remaining target methods. Note that a single partial analogy may have multiple completions.

5. Test each candidate analogy by attempting to reproduce the target behavior. Repeat steps 3 - 5 until finding an analogy that duplicates the target behavior.

Matching (step 1) uses type information stored at an index to prevent invalid matches. For example, the digit 5 cannot match the type digit-0. On matching, the index signature and source description transfer to the target. This can restrict further extensions to the hypothesis as described in (Stubblefield 1995).

The heuristics of step 2 use the information transferred to the target under step 1 to rank hypotheses. The heuristics used in this problem were (1) a specificity criterion that preferred matches deeper in the hierarchy, and (2) a simplicity criterion that favored matches that transferred the fewest different source descriptions to the target. (Stubblefield 1995) discussses the use of assumptions made in retrieval to evaluate competing hypotheses.

Because SCAVENGER uses assumed information to match index nodes, it must evaluate all hypotheses, whether produced by internal or leaf nodes. If all other indices fail to produce a viable analogy, the algorithm will eventually "fail back" to the root, where it effects an exhaustive search of the source base. Since this process may generate the same analogies several times, SCAVENGER keeps a list of previously tried analogies, and checks it to avoid testing the same analogy twice. Although this adds to the algorithm's overhead, the results of section 3 show that it does not outweigh its benefits.

In step 4, SCAVENGER completes a hypothesized analogy by matching each of its unmatched targets with all matching sources. Consequently, a single partial analogy can have many completions.

Continuing with the example problem, assumption-based retrieval produces four hypotheses, based on the initial match with the root, and a match between the child node and each target operator (#:G873, #G874 and #:G875). SCAVENGER evaluates each hypothesis in turn by retrieving all source operators referenced under the matching index. For example, evaluating the match between target #:G873 and the child node of Figure 1, produced 11 partial analogies, including one that eventually led to a correct diagnosis:

#:G873 -----> borrow-no-decrement
#:G874 -----> ?
#:G875 -----> ?

SCAVENGER generates and tests all possible extensions to each partial analogy.
Although a given error may have different possible diagnoses, SCAVENGER stops after finding the first. This is not as thorough as the approach taken by DEBUGGY, but it fits our stated goal of testing SCAVENGER's ability to efficiently find relevant analogies.

successful analogy

#:G873	operator: borrow-no-decrement signature: digit x digit x w -> digit description: decrement-error
#:G874	operator: normal-subtract signature: digit x digit x w -> digit description: normal-op
#:G875	operator: borrow-no-decrement signature: digit x digit x w -> digit description: decrement-error

failed analogy

#:G873	operator: borrow-no-decrement signature: digit x digit x w -> digit description: decrement-error
#:G874	operator: normal-subtract signature: digit x digit x w -> digit description: normal-op
#:G875	operator: normal-subtract signature: digit x digit x w -> digit description: normal-op

Figure 2

Empirical Memory Management

When an analogy correctly reproduces the target behavior, SCAVENGER specializes the index that led to its construction according to the algorithm:

1. Consider the index whose match led to the successful analogy. This may be either a leaf or an internal index node.

2. For each target function in the successful analogy that was not matched to an index pattern but matched a source when the partial analogy was extended:

2.1. Use the signature and description transferred to that function in the successful analogy to partition all analogies produced at the index into matching and non-matching sets.

2.2. Using ID3's information theoretic evaluation function (Quinlan 1986), rank each partition according to its ability to distinguish successful and failed analogies.

2. Create a new child node using the function description that best partitioned the candidate analogies.

For example, Figure 2 shows a successful and a failed analogy that resulted from the match of target #:G873 with the index node of Figure 1.

Figure 3 shows two potential extensions to the index of Figure 1, resulting from the respective mappings of #:G874 onto normal-op, and #:G875 onto borrow-no-decrement in the successful analogy. Candidate child node #1 partitions the analogies considered into those that mapped #:G875 onto a source operator from the set of decrement-errors and those that gave this target function a different interpretation. Candidate child node #2 partitions them into those that mapped #:G874 onto normal-op and those that gave it a different interpretation.

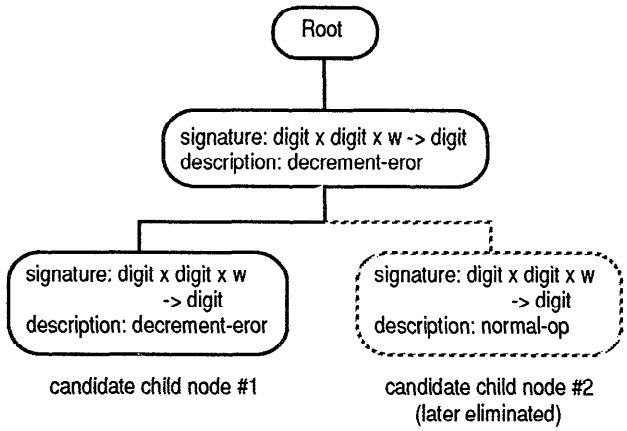

Figure 3

SCAVENGER rates each candidate child node according to its ability to distinguish the successful and failed analogies using the information theoretic evaluation function from the ID3 learning algorithm. For details, see Quinlan (1986). In our example, it determined that mapping target operator #:G875 onto the borrow-no-decrement source made the greatest contribution to the successful analogy, and selected candidate specialization #1 from Figure 3. This new node indexes all matching sources.

It is important to note that SCAVENGER is using ID3's evaluation function differently than was originally intended. Each specialization of the index hierarchy results from a different target problem. Since an analogy that proved useful for solving one problem will not necessarily be correct for another, the examples used to construct the indices lack the global consistency usually assumed by ID3. In addition, since SCAVENGER stops evaluating analogies on finding one that solves the target problem, many of the analogies evaluated in step 2.2 may not have been tested. SCAVENGER assumes these to be failures. One of the questions we consider in the next section concerns the algorithm's ability to produce a useful hierarchy under these circumstances.

Evaluating SCAVENGER

We evaluated SCAVENGER on a Power Macintosh 6100 computer. The source base consisted of 67 operators. Although this is less than the 97 bugs that were considered in the original DEBUGGY work, many of those were either combinations of simpler bugs or restrictions of bugs to a specific context (e.g., the leftmost column). Consequently, SCAVENGER was able to duplicate all the original bugs when constructing analogies. We randomly chose 500 buggy subtractions from VanLehn's (1990) study of children's subtraction; to these, we added the correct solutions to the problems, producing a set of 575 potential test problems. Our test procedure randomly chose 161 problems from this set, dividing them into a training set containing 75 problems, and a test set containing 86 problems.

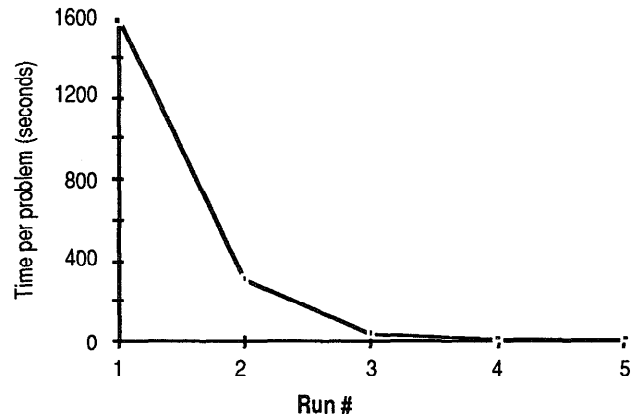

Figure 4

Figure 4 shows SCAVENGER's improvement across 5 trials of the training data. Due to the many possible combinations of bugs it had to test, the untrained algorithm took approximately 1600 seconds per problem. The trained version took an average of 11 seconds per problem. The longest solution time for the trained algorithm was 30.5 seconds, the shortest was just over 2 seconds. The untrained version of the algorithm generated an av-

erage of 3096 candidate analogies per problem. After training, it generated an average of 78 per problem.

In applying SCAVENGER to the test set (problems that the learning algorithm had not seen), the untrained version took about 1500 seconds per problem. This improved to 76 seconds after training. Although this is a strong result, it is worth noting that it was skewed by the presence of a small number of completely novel problems that, consequently, took a very long time (2220, 1556, 593, 569, 568 and 116 seconds) to solve. Excluding these, the average time on the test problems drops to under 11 seconds.

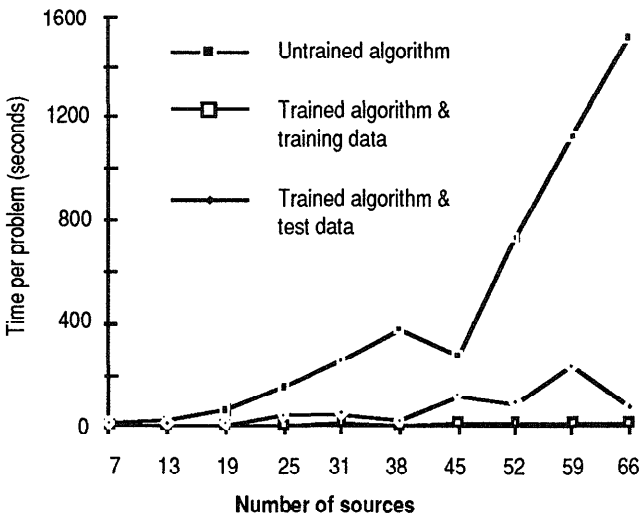

Figure 5

The algorithm scales well as the source base grows. To examine its scaling behavior, we divided the source base into 10 sets of 6 or 7 buggy operators. Each of these "test units" included approximately 16 problems that could be solved with those operators. Repeated trials "grew" the source base by adding test units and problems. At each stage, we randomly divided the problems into test and training sets and repeated the test described above. As the source base grows, the untrained algorithm shows the exponential rise in complexity we would expect of exhaustive search (Figure 5). However, times for the trained algorithm remain nearly flat when applied to problems it has trained on. The algorithm also remains efficient on the test problems, although the results show some fluctuation due to the existence of novel problems that caused it to perform badly.

Conclusion

The effectiveness of SCAVENGER's retrieval algorithm rests on two factors: the first is the repetition of useful patterns of analogy in the problem domain. These tests confirmed the existence of such common patterns in our test domain. The second is the ability of the learning algorithm to construct effective hierarchies. Although SCAVENGER uses a variation of the well tested ID3 learning algorithm, it did so in a very different way than was originally intended. SCAVENGER used a different analogy for each extension of its index hierarchy. In addition, it stopped evaluating analogies on success, and assumed unevaluated analogies to be failures. This contrasts with ID3's usual global analysis of entire sets of training data, and it was not clear that it would support construction of a useful index hierarchy. The results of our evaluation strongly suggest that it does, providing another example of the robustness of the ID3 algorithm.

The positive results on scaling tests indicate that in spite of the algorithm's complexity and the large numbers of hypotheses it typically evaluates, the algorithm behaves well as its source base grows.

SCAVENGER provides an analogical retrieval mechanism that overcomes the four limitations of traditional approaches that were listed in the introduction:

1. By projecting commonly useful patterns of analogy onto poorly defined target problems, assumption-based retrieval enables reasonable analogies in poorly defined problem domains.
2. The use of empirical learning to select properties that are effective predictors of source utility eliminates the need for a priori restrictions on the retrieval vocabulary.
3. SCAVENGER determines the similarity of sources and targets by heuristically ranking hypothesized matches between the target and different index nodes. This is more flexible and expressive than context independent approaches.
4. Empirical memory management takes the structure of target problems into account in organizing the retrieval system.

It is interesting to contrast SCAVENGER with the more analytic approach taken by DEBUGGY and most expert systems. SCAVENGER does not reason about problems: it simply remembers useful patterns of analogy. This "analogize-test-remember" approach could be useful in poorly understood problem domains. For example, if we are diagnosing

problems in systems where failure of one component could cause or interact with failures of another in poorly understood ways, SCAVENGER could be a useful tool for discovering and recording patterns of interacting failures.

The SCAVENGER experiments corroborate the viability of assumption-based retrieval and empirical memory management for analogical source retrieval. In particular, they show that the experience gained in constructing analogies, although limited and contingent in nature, will support construction of effective hierarchical indices for future source selection. Finally, SCAVENGER provides an alternative to the separation of retrieval and analogical inference common to many models of analogy. In doing so, it offers what we believe to be an elegant, flexible and theoretically interesting model of analogical reasoning.

Acknowledgements

We wish to thank Kurt VanLehn for graciously providing the original data from his Southbay study of children's performance on subtraction problems. We also thank the Computer Science Departments at Dartmouth College and The University of New Mexico for providing the intellectual communities that made this research possible.

References

Brown, J. S. and K. VanLehn. 1980. Repair theory: a generative theory of bugs in procedural skills. *Cognitive Science* 4:379-426.

Burton, R. R. 1982. *Diagnosing bugs in a simple procedural skill.* in Sleeman and Brown (1982).

Gentner, Dedre. 1983. Structure-Mapping: A Theoretical Framework for Analogy. *Cognitive Science* 7: 155-170.

Hall, R.P. 1989. Computational Approaches to Analogical Reasoning: A Comparative Analysis. *Artificial Intelligence* 39(1): 39-120.

Hoffman, R. P. 1995. Monster Analogies. *AI Magazine* 16(3):11-35.

Kolodner, J. L. 1993. *Case-Based Reasoning*. San Mateo, Cal.: Morgan Kaufmann.

Mitchell, M. 1993. *Analogy-making as Perception.* Cambridge, Mass.: MIT Press.

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

Sleeman, D. and Brown, J. S. 1982. *Intelligent Tutoring Systems*. New York: Academic Press.

Stubblefield, W. A., 1995. *Source Selection for Analogical Reasoning: An Interactionist Approach*. PhD Dissertation, Department of Computer Science, University of New Mexico.

Thagard, P. 1988. *Computational Philosophy of Science.* Cambridge, Mass.: MIT Press.

VanLehn, K. 1990. *Mind Bugs: The origins of procedural misconceptions*. Cambridge, Mass: MIT Press.