Competing Models of Analogy: ACME Versus Copycat

Bruce D. Burns
Department of Psychology
University of California, LA
Los Angeles, CA 90024-1563
burns@psych.ucla.edu

Keith J. Holyoak
Department of Psychology
University of California, LA
Los Angeles, CA 90024-1563
holyoak@psych.ucla.edu

Abstract

ACME and Copycat have been viewed as competing models of analogy making. Mitchell (1993) makes three major criticisms of ACME in arguing for Copycat's superiority: that because ACME considers all syntactically possible mappings it is psychologically implausible and computationally infeasible; that its representations are rigid and hand-tailored for each problem; and that ACME's representations are semantically empty. To evaluate these criticisms we applied ACME to simulating problems in the only domain addressed by Copycat, letter-string analogies such as, "If **abc** is changed into **abd**, how would you change **kji** in the same way?" Using representations that include only knowledge available to Copycat, ACME generated the most common solutions that people and Copycat produce. In addition, ACME was able to generate some solutions produced by people but that are impossible for Copycat, demonstrating that in some respects ACME is a more flexible analogical reasoner than is Copycat. These simulations answer each of Mitchell's criticisms of ACME. ACME can incorporate domain-relevant knowledge to allow a principled reduction in the number of mappings considered; it can generate novel representations based on its domain-general constraints; and it can incorporate semantic content into its representations. In addition, ACME has the advantage of being applicable to many different domains.

Introduction

Copycat attempts to computationally model the processes underlying the creation of analogies (Hofstadter, 1984; Mitchell, 1993). Analogies are produced by the interaction of processes for building structured representations of the source and target analogs, including processes that lead to "slippage" of concepts to allow mappings between non-identical concepts. ACME (Holyoak & Thagard, 1989), another computational model of analogy making, finds a systematic mapping between a source domain and a target domain by building a network based on multiple soft constraints (in particular, semantic, structural and pragmatic constraints), and then allowing it to settle using parallel constraint satisfaction. ACME and Copycat have been viewed as competing models of analogy, and Mitchell (1993) has criticized ACME's approach to making analogies. It has been difficult to evaluate these alternative models, however, because they were not designed to be applied to the same analogy problems. Copycat only solves problems in the micro-domain of letter-string analogies (e.g., If **abc** is changed into **abd**, how would you change **kji** in the same way?). Copycat has had some success in its restricted domain, both computationally and when compared to human data (Burns & Schreiner, 1992), but it is unclear what factors provide the basis for its successes. ACME is designed to be applicable to analogies in any domain, but in practice it has usually been applied to cases in which a complex source story is mapped to a complex target analog, often in the context of solving the target analog as a problem. Misunderstandings related to the comparative advantages and disadvantages of the two models may be partly due to them being tested on very different problems. In this paper we try to assess the models by applying ACME to letter-string analogies

The approaches of ACME and Copycat are similar in important ways. In particular, the models share the idea that analogies develop from competition between multiple soft constraints. Indeed, Mitchell (1993, p. 210) points out that Copycat has counterparts for each of the three basic classes of constraints that ACME uses. However, Mitchell directs three major criticisms at ACME. First, ACME creates all possible syntactic mappings, which appears to be computationally infeasible and psychologically implausible. Second, Mitchell claims that ACME uses knowledge representations that are unduly rigid and are hand-tailored for each new analogy. If true, this would imply that ACME lacks the ability to do the kind of re-representation that is central to Copycat. Third, Mitchell claims that ACME's representations are devoid of semantic content. Given that these criticisms are also applied to another major analogy model, SME (Falkenhainer, Forbus, & Gentner, 1989), Mitchell presents Copycat as an approach radically different from previous analogy models. The status of these criticisms will be evaluated after describing our attempt to use ACME to simulate the solution of analogies drawn from Copycat's chosen domain.

Simulating Solutions to Letter-String Analogies

ACME Principles

ACME computes a mapping between a source and a target domain, represented using a predicate-calculus style representation, by forming a network containing units representing potential mappings between elements of the source and target. The network is constructed to conform to structural, semantic and pragmatic constraints, and the final mapping is derived by allowing the network to settle under parallel constraint satisfaction. The structural constraints play the dominant role in determining mappings, as is consistent with findings from research on human analogical mapping (see Falkenhainer et al., 1989).

Solving letter-string analogy problems requires more than mapping, however, as new elements must be generated in order to form the solution. An extension to ACME proposed by Holyoak, Novick, and Melz (1994) uses a *copying with substitution and generation* algorithm (CWSG) to allow ACME to perform analogical pattern completion by generating new elements where they are necessary. This mechanism assumes that if some proposition exists in one analog, but has no corresponding proposition in the other, then a new "image" proposition in the other analog may be generated by substituting mapped predicates and objects and generating counterparts for unmapped components.

Representations

Given that many of the criticisms Mitchell makes of ACME are related to its representations, it is necessary to carefully explain how our representations were constructed. We will focus on the problem, "If **abc** is changed into **abd**, then how would you change **kji** in the same way?" Because the **abc:abd** part of the problem was always the same in problems used here, problems will be referred to by the string to be changed, for example, **kji**.

The **abc:abd** part of the analogy is treated as the source domain while the **kji** and answer strings form the target. The representation of the source was divided into a number of logically distinct fields. In ACME, elements of a source field are only mapped to elements in the same type of field in the target domain. Thus by using fields we limit the number of mapping units formed and increase the likelihood of appropriate mappings being formed.

A *semantic* field was declared that contained information about what is the predecessor and successor of each letter of the alphabet (where the "letters" represent types, rather than tokens, of letters that are part of a string). Hence the source domain semantic field consisted of predicate-calculus statements of the form:

(PRED_OF (A* B*) A_predecessor_of_B)

(SUCC_OF (B* A*) B_successor_of_A)

These two statements respectively indicate that the letter type *a* (types are denoted by *) is the predecessor of letter type *b*, and that *b* is the successor of *a*. While this field should in principle contain the successor and predecessor relations between all letters in the alphabet, for the present simulations only relevant letters were included in this field in order to avoid the formation of a large number of irrelevant mapping units. The predecessor and successor relations for letters *a* through *e* were included. Copycat never produces answers to this problem that require knowledge about letters outside of this range.

A *first_string* field contained propositions about the first string (**abc**), such as which letters are left of each other, which are right of each other, and what the start, middle and end letters were. For example:

(RIGHT (B1 A1) B1_right_of_A1)
(START (A1) A1_starts string)

The first statement indicates that a token of the letter *b* in **abc** (string 1) is to the right of a token of the letter *a*. Left and right relations for all three letters in **abc** were represented. The second statement indicates that the letter *a* is at the start of the string.

A *second_string* field contained information about the second string (**abd**), including the same type of information given for the first string, that is, which letters are to the right and left of each other and which are at the start, middle and end of the string. For example,

(RIGHT (D2 B2) D2_right_of_B2)

In addition, statements were included that indicated which letter types had been retained from the first string, which had been deleted, and which added in order to form the second string. In total these leave, add and delete statements were:

(LEAVE (A2) A2_retained)
(LEAVE (B2) B2_retained)
(DELETE (C2) C2_deleted)
(ADD (D2) D2_added)

First_string_instantiated and *second_string_instantiated* fields served to link the letter type information in the semantic field to specific letter tokens. The following are examples for the first and second strings respectively,

(INSTANTIATE (A1 A*) A1_isa_A)
(INSTANTIATE (A2 A*) A2_isa_A)

All letter tokens used in the string were instantiated in this way.

A *relations* field specified which letters stay as the same type and which change between the two strings. This field consisted of the following statements:

(SAME (A1 A2) A1_stays_as_A2)
(SAME (B1 B2) B1_stays_as_B2)
(CHANGE (C1 D2) C1_changed_into_D2)

The representation of the target domain was structured in the same way as the source domain. The *semantics* field had the same type of successor and predecessor statements, but for the letters *f* through *m*. The *first_string* field contained the same type of statements as the source version did, but for the string **kji** instead of **abc**. The *second_string* field was left highly impoverished, in that it was limited to listing the potential additions and deletions from that string. The second-string field contained LEAVE statements for K2, J2 and I2, and DELETE statements for the same letters. In addition there were ADD statements for all letters between *d* and *m* (including new tokens of the *k*, *j* and *i* letter types). The *relations* field was left blank. The details for the later two fields, which provide ACME's answer to the problem, had to be filled in by the CWSG algorithm. The *instantiation* fields instantiated all letters that were used in any other field.

Evaluating ACME's Performance

Our use of ACME to simulate solution of letter-string analogies was guided by several constraints. First, we based our representations entirely on concepts that have been incorporated into Copycat's "slipnet": *rightmost*, *leftmost*, *successor*, *predecessor*, *first*, and *last*. The slipnet also contains all the letters of the alphabet and links from each letter to its successor and predecessor, and Copycat makes a type-token distinction for letters. By equating its representational elements with those assumed by Copycat, we ensured that ACME's representations would be as semantically rich as those used by Copycat, as intuitively obvious, and as general in their applicability to multiple letter-string analogies.

People produce a wide range of answers to these problems. Copycat also produces a variety of answers because on different runs it builds different structures and mappings that underlie alternative answers, by probabilistically invoking different small pieces of structure-building code. In contrast, ACME operates by over-generating all syntactically possible mappings, and selecting the most coherent subset by a deterministic algorithm. Accordingly, the representation of the second string in the target included an overly-general list of possibilities, different subsets of which would constitute alternative answers. ACME's pragmatic constraint provides features that allow it to simulate variations in active representations and mappings. In different runs, alternative critical mappings can be promoted by being *presumed*, and different parts of structure can differ in *importance*. A presumed mapping is given additional external excitation, and less important components are inhibited (Spellman & Holyoak, 1993). Critical mappings would be those for *predecessor*, *successor*, *right*, *left*, *start* and *end*, for which the alternative mappings (to the same or to the opposite relation) appear to underlie many of the answers Copycat produces for letter-string analogy problems. By varying which of these mappings were presumed on different runs we attempted to generate alternative answers using ACME's CWSG algorithm. On some runs we deleted some structural elements (a crude implementation of differential importance) to simulate generation of answers that appear to reflect insensitivity to certain aspects of structure.

It is important to note that using "presumed" mappings does not in itself provide ACME with a solution to the problem. The presumed mappings do not themselves constitute an answer; rather, they bias the concept mappings from which an answer emerges after constraint satisfaction is performed. Copycat arrives at these mappings though the probabilistic running of codelets. In its current implementation ACME is deterministic; however, it would be possible to have the strength of the critical mappings altered probabilistically. But whether a model is implemented in a probabilistic or deterministic fashion does not appear to be a major issue at stake in evaluating models of analogy.

Our aim in this effort was not to exhaustively generate all possible answers, nor to match the details of the probability distribution of answers produced by people, which would require a thorough search of parameter space. Rather, we focused on generating the most common solutions to problems that have been extensively tested with people. If ACME can in fact generate the answers most frequently provided by people using plausible assumptions about differential presumed mappings and importance of structural elements, this would demonstrate that its approach to analogy is at least viable as a psychological model of solving letter-string analogies. In addition, we will report examples of solutions generated by ACME (and some people) that Copycat is unable to generate. Such cases undermine claims that Copycat is inherently less rigid.

Simulation Results

In all simulations the following parameters were kept constant: excitation, .005; inhibition (structural), -.16; decay, .005; similarity of identical predicates, .005; starting activation of all units .001. The Grossberg updating rule, with maximum activation of 1.0 and minimum activation

of .3, was used to settle the network. These parameters were chosen because they had been used to explore other domains with ACME. The mappings of LEAVE=LEAVE, ADD=ADD, and DELETE=DELETE were always presumed, as these predicates have no other sensible mappings.

ACME was presented with two problems: the **kji** problem, and the **xyz** problem: "If **abc** was changed into **abd**, then how could **xyz** be changed in the same way?" (with the *xya* answer prohibited. Note that solutions to letter-string problems will be given in *italics*, problems themselves will be in **bold**.) These problems were chosen because they have been solved by Copycat, and Burns (1994) has administered them to large groups of people. (Another problem, **mrrjjj**, meets this criterion as well, but requires addressing the issue of how groupings of elements can be represented, which we have not yet attempted.)

The kji problem

Copycat has been applied to the letter-string analogy **kji**, and Mitchell (1993, p. 80) reports that over a thousand runs its most common solutions were *kjh*, *kjj* and *lji*. Burns and Schreiner (1992) gave the **kji** problem to college students and found that *kjh*, *kjj* and *lji* were the most common solutions, although people also generated a wide variety of additional answers.

Each ACME run used the representation described above, from which a network of 421 units and 4463 links was formed. The first run of the program was made with no presumed mappings. The following relations field was generated by the CWSG algorithm:

(SAME (K1 K2) T12)
(SAME (J1 J2) T13)
(CHANGE (I1 H2) T14)

This field represents the answer *kjh*, as K1 and J1 stay as tokens of the same letters (*k* and *j*, respectively), while I1 is changed into an *h*. Thus the answer that is most easily produced by ACME is also commonly produced by both Copycat and by people. The predominance of the answer is not surprising, as it is a highly structurally coherent solution (based on mapping *successor* to *predecessor* and vice versa), and ACME is strongly driven by structural constraints.

In order to try to produce the *lji* answer the mappings of START=END and END=START were presumed. This mapping underlies Copycat's generation of this answer (see Mitchell, 1993, p. 113). Running the program with such a presumption generated the propositions:

(SAME (I1 I2) T12)
(SAME (J1 J2) T13)
(CHANGE (K1 L2) T14)

ACME thus succeeded in producing the *lji* solution. A run of the program that instead presumed the mappings of LEFT=RIGHT and vice versa also generated the *lji* answer.

The other common answer produced by Copycat and by people was *kjj*. The key to producing this solution appears to be ignoring the fact that *k* is the successor to *j*, whereas *a* is the predecessor of *b*, while nonetheless mapping *successor* to *successor* and applying this mapping to the change in the last element. Loosening of the structural constraints by ignoring the relationship between *k* and *j* was simulated by removing the propositions concerning K2 and J2 in the target domain's *second_string_instantiated* field. Running this representation produced the *kjj* answer. In effect, this run simulated Copycat failing to build or maintain the links between *k* and *j*, as it must fail to do in order to generate the *kjj* answer.

In another run we presumed the mappings SUCC_OF=SUCC_OF and PRED_OF=PRED_OF, as well as START=START, END=END, obtaining the answer *kjl*. This solution requires tolerating the inconsistency between the two pairs of presumed mappings. This answer is occasionally produced by people (Burns, 1994), but has never been reported to have been produced by Copycat.

The xyz problem

The **xyz** problem is of interest because the instructions block the most natural answer, *xya*, which arises from people's tendency to view the alphabet as circular. Copycat's most common answer to the **xyz** problem was *xyd*, and its next most common was *wyz* (Mitchell, 1993, p. 82). Burns (1994) found that for **xyz** people generated *wyz* most often, as well as a large range of other answers. ACME's representation of this problem was identical to that for **kji** in the source domain, but in the target domain every *k* was changed into an *x*, every *j* to *y* and every *i* to *z*. In addition, the *semantics* and *instantiate* fields, as well as the ADD propositions in the *second_string* field, were modified to reflect the use of a range of letters between *u* and *z*, as well as *d*.

The first run using this representation used no presumed mappings and generated the answer *zyw*. This answer is never reported to have been produced by Copycat, but a number of people do generate it (Burns, 1994). By setting as presumed the mappings of START=END and END=START, the answer *wyz* was generated. The *xyd* answer appears to be a simplistic solution derived by ignoring all predecessor and successor information. We successfully simulated generation of *xyd* by replacing the semantic fields with more primitive letter-type definitions, such as (A_type (A*)).

Discussion

Our simulations show that the ACME model, which has been applied to a wide range of analogical mapping and transfer problems in many different domains, can also produce reasonable solutions to letter-string analogies. For the two problems we have investigated ACME can find the most common solutions generated by both people and Copycat, and in addition can find some solutions that people generate but Copycat cannot. In light of this success we can evaluate the status of Mitchell's (1993) criticisms of ACME.

The first criticism was that ACME is unrealistic in creating mapping units for all possible syntactic matches. Our simulations reveal that principled use of subfields in the representation can reduce the number of mapping units formed, providing an illustration of the general point that domain knowledge, when available, can be used to reduce the space of possible mappings. Mappings only occur between members of the same subfields, each of which can be small. Subfields reduce the computational explosion that can occur when units for all possible syntactic mappings are formed. The psychologically plausibility of the mappings formed is harder to determine, as we do not know what mappings people may initially form at an implicit level. Copycat restricts the possible mappings even further than ACME because the former model is designed to deal only with a single domain of problems. More specialized network-construction rules based on domain-specific knowledge could also be built into ACME, but this would not seem to constitute a theoretical advance. In fact, the construction of a diverse set of potential mappings contributes to ACME's flexibility in constructing a wide range of answers, some of which turn out to be meaningful. These include some human solutions that Copycat's restrictions render it unable to generate. It should also be noted that ACME's basic constraints can be realized in architectures that eliminate the need for explicit generation of mapping units (Hummel, Burns & Holyoak, 1994).

The second criticism of ACME was that its representations are inflexible, because they supposedly are hand-tailored for each problem and do not change during the running of the program. This criticism is misleading, in that it hinges on what is meant by "re-representation". In a sense, ACME dynamically re-represents the problem as it runs, because its representation is a product of the current states of activation of the mapping units. Furthermore, ACME finds "slippages" in the form of mappings between non-identical concepts, such as *successor* and *predecessor*, and it builds new structure using its CWSG procedure. It is certainly true that the mappings that can be made are constrained by the mapping units that are initially formed; however, Copycat is similarly limited by the range of slippages permitted in its slipnet. ACME's ability to generate solutions that people produce but Copycat cannot demonstrates that ACME can actually be more flexible than Copycat, even in the specific domain to which Copycat has been tailored. The *kjl* answer to the **kji** problem is fundamentally impossible for Copycat to generate, as it involves defining a change relative to one letter (*k*) but then applying it to a different letter (*i*). In Copycat the required double slippage of *end* to *start* and back again is restricted to occur only once. For the same reason Copycat is also unable to produce the *zyw* answer to the **xyz** problem. ACME is less brittle than Copycat in this regard because it treats structural consistency as a soft constraint, which does not require global consistency of the entire set of favored mappings.

The current ACME simulations are representationally limited in that the model lacks the ability to organize the elements of the problem into meaningful groups, and thus is unable to deal with problems such as **mrrjjj**. Copycat includes specialized procedures for building structures representing certain groupings, and hence can produce reasonable answers for such problems. Nonetheless, it is important to recognize that Copycat is also limited by the structure-building routines that have been programmed into it. For example, one of the more common answers offered to the **mrrjjj** problem by people is *mrsjjk*, in which every third letter is changed into its successor (Burns, 1994). Because Copycat does not include a procedure to link every third letter, it cannot generate this solution. The apparent successes of Copycat in dealing with groupings, as well as its limitations in this area, arise from it being programmed with specific knowledge. Such knowledge could also improve ACME's performance if the latter model were modified to deal with a specific domain.

The third criticism, that ACME is semantically empty, dissipates once it is recognized that ACME can readily incorporate the same semantic knowledge as is included in Copycat. Mitchell (1993) claims that Copycat's concepts are semantically meaningful because they are embedded in a network of related concepts; but the same claim can be made for any model that accepts hand-coded representations of domain concepts and their interconnections.

Our ACME simulations illustrate that ACME and Copycat share an emphasis on producing structured solutions. The fundamental commonalties of the two approaches are highlighted by Mitchell's observation that Copycat includes counterparts to the constraints on human analogy making that ACME takes as fundamental. Nonetheless, the implementations of the programs differ vastly, and it is possible that the Copycat approach will eventually prove more successful than that of ACME. It is yet to be demonstrated, however, that Copycat is superior on either psychological or computational grounds, or that its computational realization embodies any distinctive theoretical constraints on analogy making. In addition,

until it is shown that Copycat can be generalized, the model will remain vulnerable to the criticism that its successes depend more on its specialized domain knowledge than on general principles that underlie human analogy making.

Acknowledgments

This research was supported by NSF grants SBR-9310614 and DIR-9024251.

References

Burns, B. D. (1994). *Representations in analogical problem solving*. Doctoral dissertation. Los Angeles: University of California, Los Angels, Department of Psychology.

Burns, B. D., & Schreiner, M. E. (1992). Analogy and representation: Support for the Copycat model. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society* (pp. 737-742). Hillsdale, NJ: Erlbaum.

Falkenhainer, B., Forbus, K. D., & Gentner, D. (1989). The Structure-Mapping Engine: Algorithm and examples. *Artificial Intelligence, 41*, 1-63.

Hofstadter, D. R. (1984). *The Copycat project: An experiment in nondeterministic and creative analogies.* Cambridge, MA: MIT A.I. Laboratory Memo 755.

Holyoak, K.J., Novick, L. R., & Melz, E. R. (1994). Component processes in analogical transfer: Mapping, pattern completion, and adaptation. In K. J. Holyoak & J. A. Barnden (Eds.), *Advances in connectionist and neural computation theory, Vol 2: Analogical connections* (pp. 113-180). Norwood, NJ: Ablex.

Holyoak, K. J., & Thagard, P. (1989). Analogical mapping by constraint satisfaction. *Cognitive Science, 13*, 295-355.

Hummel, J. E., Burns, B. D., & Holyoak, K. J. (1994). Analogical mapping by neural synchrony: Preliminary investigations. In K. J. Holyoak & J. A. Barnden (Eds.), *Advances in connectionist and neural computation theory, Vol 2: Analogical connections* (pp. 416-445). Norwood, NJ: Ablex.

Mitchell, M. (1993). *Analogy-making as perception.* Cambridge, MA: MIT Press.

Spellman, B. A., & Holyoak, K. J. (1993). An inhibitory mechanism for goal-directed analogical mapping. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society* (pp. 947-952). Hillsdale, NJ: Erlbaum.