Constraints on Analogical Mapping:

A Comparison of Three Models

By

Mark T. Keane

Department of Computer Science,
University of Dublin, Trinity College,
Dublin, IRELAND

Tim Ledgeway & Stuart Duff

School of Psychology
University of Wales College of Cardiff,
Cardiff, WALES

Correspondence and proofs to be sent to:

Dr. Mark Keane,
Cognitive Science Group,
Department of Computer Science,
Trinity College,
Dublin 2,
IRELAND

Running head: ANALOGICAL MAPPING

ABSTRACT

Three theories of analogy have been proposed which are supported by computational models and data from experiments on human analogical abilities. In this paper, we show how these theories can be unified within a common meta-theoretical framework which distinguishes between levels of informational, behavioural and hardware constraints. This framework makes clear the distinctions between three computational models in the literature (the Analogical Constraint Mapping Engine, the Structure-Mapping Engine and the Incremental Analogy Machine) . The paper then goes on to develop a methodology for the comparative testing of these models. In two different manipulations of an analogical-mapping task we compare the results of computational experiments with these models against the results of psychological experiments. In the first experiment, we show that increasing the number of similar elements in two analogical domains, decreases the response time taken to reach the correct mapping for an analogy problem. In the second psychological experiment we find that the order in which the elements of the two domains are presented has significant facilitative effects on the ease of analogical mapping. Only one of the three models, that model which embodies behavioural constraints, the Incremental Analogy Machine, predicts both of these results. Finally, the immediate implications of these results for analogy research are discussed, along with the wider implications the research has for cognitive science methodology.

INTRODUCTION

Analogical thinking is central to problem solving, learning and creativity (see e.g., Gentner, 1983; Gick & Holyoak, 1980; Keane, 1988a; Koestler, 1964; Mayer & Bromage, 1980). Since 1980, considerable psychological and computational research has been carried out to elucidate this form of thought. Empirical evidence has been gathered on subjects' analogical thinking in a variety of task situations and different computational models of this behaviour have been specified. In these respects, work on analogy is in an advanced state and has the potential to be an exemplar for cognitive science theory and methodology.

In this paper, we make theoretical, methodological, and empirical contributions to analogy research. First, we show that analogy theory can benefit from being organized within a meta-theoretical framework. Frameworks, such as Marr's□(1982), help us separate theoretical proposals about the basic competence underlying a behaviour, from proposals about the performance of that behaviour and the neurological substrate in which it is grounded. We show that apparently diverse proposals on analogy can be unified productively within just such a framework. In particular, we show that the framework helps us separate the "competence" aspects of analogising from the "performance" aspects of the behaviour.

Second, we develop a methodology for comparing various computational models. In cognitive science, computational models have tended to be used in a minimalist fashion. They are used to show that theoretical proposals are well-specified enough to be programmed. Models which compete on computational, as well as theoretical and empirical grounds, are seldom developed and even when such models exist the competitive testing of them is a rare occurrence. Yet, clearly, this sort of testing is what we should be doing with these models. In this paper, we

develop a basis for comparing three distinct models of analogical mapping and perform computational experiments on these three models in specific task situations.

Third, we report psychological experiments that parallel the tests carried out with the computational models. The goal here is to compare the results of these experiments against the outputs produced by the various models. The task situations we examine are concerned with the effects of similarity and order of presentation on analogical mapping. The results of these experiments contribute new evidence to the corpus of empirical findings in the literature. But before going into each of these areas in detail, let us first outline the current state of analogy theory.

The Stages of Analogical Thinking

It has become clear in recent years that several sub-processes constitute analogical thinking; representation, retrieval, mapping, adaptation and induction. In order to solve a problem by analogy one must first *represent* the problem in some form. It has been shown, using tests on experts and novices in a domain, that exactly how an individual represents a problem affects the success of subsequent analogising (see Novick, 1988). Then one must *retrieve* a useful analogous case to the problem. For instance, faced with the problem of understanding the structure of the atom one might retrieve a putative analogous case like the solar system (see e.g., Gentner & Landers, 1985; Gentner & Rattermann, in press; Holyoak & Koh, 1987; Keane, 1987; Thagard, Holyoak, Nelson & Gochfeld, 1990; Wharton, Holyoak, Downing, Lange & Wickens, 1991, 1992). Having done this retrieval, one can then draw the analogy between the two domains -- the solar system and the atom -- or perform an *analogical mapping* between them (see e.g., Burstein 1986; Carbonell 1983; Gentner 1983, 1989; Gick & Holyoak 1980, 1983; Holyoak 1985; Keane 1985, 1988a). An analogous solution may not always solve the problem

immediately; it may have to be *validated* or tested and *adapted* to the reality of the problem situation (see Anderson & Thompson, 1989; Falkenhainer, 1987; 1990; Keane, 1990; Novick & Holyoak, 1991). Finally, as a result of this mapping a higher-order schema might be *induced* based on the two domains of information; for example, a schema encoding the higher-order concept of central force systems might be formed (see e.g., Gick & Holyoak, 1983).

The core sub-process in analogical thinking, the process that is unique to it, is analogical mapping. It is also the process that has received the most attention in the literature. Typically, when an analogical mapping is made two distinct computations occur; first, the corresponding concepts in both domains are *matched* and, second, a portion of the conceptual structure of one domain is *transferred* (or carried over) into the other domain to form the basis of analogical inferences. For example, when mapping the solar system domain to the atom domain you might *match* the corresponding REVOLVES relations in both domains and *transfer* relations of ATTRACTION from the solar system domain to apply in the atom domain.

In general, when people analogically map two domains they have to solve several tricky computational problems. Even though two domains may have a one-to-one correspondence between their parts, there may be a large number of ambiguous matches -- many-to-one and one-to-many matches -- between their parts. Psychologically, many of these problems only surface when we make the mapping task difficult, as in the attribute-mapping problem from Holyoak & Thagard (1989; see Table 1).

---------- Insert Table 1 About Here ----------

In this task, subjects are asked to say which things in list A correspond to which things in list B (ignoring the meaning of the words). Essentially, subjects have to discover a one-to-one mapping between all the individuals and attributes in list A

and list B. This is quite a difficult task as a lot of ambiguous matches have to be resolved. For example, *smart* may match *hungry* or *friendly* or *frisky* and the correct match can only be determined by eliminating the inconsistent matches which follow from all but one of these matches. Furthermore, even for quite small domains such as these, there are over 700 possible matches. The unique one-to-one mapping which solves the problem is to match Steve and Fido, Bill and Rover, Tom and Blackie, smart and hungry, tall and friendly, and timid and frisky.

In this paper, we will concentrate on current computational models of the analogical mapping stage of analogical thinking. The remainder of this paper is divided up into six main parts. First, we show that current theory can be unified within a common meta-theoretical framework. Second, we outline how the various models of analogical mapping relate to this statement of analogy theory. Third, we describe one of these models in some detail. Fourth, we identify a common measure for comparing the predictions of these models. Fifth, using an attribute mapping task similar the above example, we carry out two studies examining the factors affecting analogical mapping. Each study has two parts to it; a computational experiment and a corresponding psychological experiment. The computational experiment determines the outputs the different models make for the target manipulation. The psychological experiment tests these outputs against people's performance on the task. The two studies examine the effects of similarity and of order on analogical mapping, respectively. Finally, we will consider the immediate implications of these findings for analogy research and the wider implications they have for cognitive science.

META-THEORIES, THEORIES AND MODELS OF ANALOGICAL MAPPING

In recent years, cognitive scientists have come to appreciate the need to stratify their theoretical proposals within a meta-theoretical framework. Marr's (1982)

framework distinguishes between three levels of theory (i.e., the computational, algorithmic, and hardware levels). In several areas of cognitive science, most notably vision research, it has proved possible to elaborate theories at each of these levels. However, many have been pessimistic about the feasibility of computational-level theories of higher-thought processes (e.g., Fodor, 1983; Marr, 1982). Fortunately, this view has been countered, in general, by Anderson (1991) and, specifically, in the areas of deduction (Johnson-Laird & Byrne, 1991) and analogical thinking (Palmer, 1989).

Palmer (1989) has pointed out that any adequate theory of analogical mapping will have to operate at several levels of description. At the highest level, one needs to characterise the *informational constraints* implied by the task situation; this level is concerned with describing what an analogy is; that is, what needs to be computed to produce appropriate outputs given certain inputs (akin to Marr's computational level). Below this level is one of *behavioural constraints* which have to capture the empirical facts of people's observable analogical behaviour (Marr's algorithmic level). Hence, this level should include constraints that predict when one analogy is harder than another, the relative differences in processing times for different analogies and the sorts of errors that people produce. Finally, there is the level of *hardware constraints* which aim to capture the neurological primitives of analogical thought (Marr's hardware level).

We make use of Palmer's meta-theoretical framework in this paper, but it should be said that terminologically both frameworks have certain advantages and disadvantages. Palmer's use of the term "informational" is more apt for our purposes because it captures the notion that this level is concerned with determining the *information* that needs to be computed to perform analogies. At the next level, the labels "algorithmic" and "behavioural" are informative in different ways. *Algorithmic* captures the idea that this level is concerned with *how* a computation is

carried out; it also conveys the idea that many different algorithms may instantiate a given computational level description. By calling this level *behavioural*, we stress the idea that this level of description is specifically concerned with observed behaviour. At this level there are further constraints which reduce the space of possible algorithms, instantiating the computational level, to those which produce outputs that parallel the observed behaviour of people on a given task.

Accepting these caveats, we adopt Palmer's framework, although we organize analogy theories quite differently within his framework (see also Keane, Ledgeway & Duff, 1991). Having done this, the crux of our argument is that current theories of analogy have discovered many informational constraints but say little about behavioural and hardware constraints. Then, we attempt to improve this situation by considering two possible behavioural constraints. Finally, we show how current computational models of analogy instantiate these constraints.

Informational Constraints on Analogical Mapping

In the attribute-mapping problem there are several difficulties associated with achieving optimal analogical mappings. As we saw earlier, there are many ambiguous matches that need to be resolved and a large number of alternative matches to choose between. Several informational constraints have been proposed to solve these sorts of mapping difficulties (see e.g., Gentner, 1983; Holyoak & Thagard, 1989).

The most important set of constraints are *structural constraints*. These constraints are used to enforce a one-to-one mapping between the two domains (Falkenhainer, Forbus & Gentner, 1986, 1989; Holyoak & Thagard, 1989). Structural constraints rely on several techniques:

- *make matches only between entities of the same type*; only attributes are matched with attributes, objects with objects and two-place predicates with

two-place predicates. For example, in matching REVOLVES(A B) and REVOLVES(C□D), the REVOLVES predicate would never be matched with the object C. This reduces the total number of matches that need to be considered (see Gentner, 1983; Holyoak & Thagard, 1989).

- *exploit structural consistency*; that is, if the propositions REVOLVES(A B) and REVOLVES(C□D) match, then the arguments of both should also be matched appropriately, A with C and B with□D. This is especially useful in eliminating many-to-one and one-to-many matches (see Falkenhainer et al., 1986, 1989).

- *favour systematic sets of matches* (Gentner's systematicity principle); that is, if one has two alternative sets of matches then the mapping with the most higher-order connectivity should be chosen. This constraint aids the choice of an optimal mapping from among many alternative mappings.

These techniques have been shown to be very powerful. In many cases, structural constraints alone can find the optimal mapping between two domains (as in the above attribute-mapping example).

A *similarity constraint* can also be used to disambiguate between alternative matches. When this constraint is applied only identical concepts are matched between the two domains (Gentner, 1983) or, more loosely, semantically-similar concepts are matched (Gick & Holyoak, 1980; Holyoak & Thagard, 1989). Semantic similarity can be used to disambiguate matches; if one match in a set of one-to-many matches is more similar than the others, it can be preferred.

Finally, there are *pragmatic constraints* (e.g.,□Holyoak, 1985; Holyoak & Thagard, 1989; Keane, 1985). Again, these constraints may disambiguate a set of matches. For example, in a certain analogical mapping situation, one match may be pragmatically more important (or goal-relevant) than other alternatives and so it will

be preferred over these alternatives. We propose that many task demands provide pragmatic constraints on analogical mapping. These task demands would include the specific instructions given in a task (e.g., in the attribute mapping problem subjects are asked to map all of the elements) or the way in which materials are presented (e.g., the information may be presented in a certain sequence). These task demands, which constitute the "pragmatics of the situation", have received little research attention.

Informational constraints constitute a high-level specification of what makes a particular comparison between two domains an *analogical* comparison. They constitute a *competence* theory of analogical mapping (see also Gentner, 1989). Computationally, they can be and have been implemented in a variety of different models (as we will see later). As such, they capture the significant informational aspects of analogical comparisons. However, this level of description on its own is not sufficient to constitute a cognitive model (cf. Newell, 1990, for more general arguments on this point). For an adequate cognitive model of analogical mapping we need to elaborate the behavioural constraints on analogising. These behavioural constraints reduce the set of possible algorithms which instantiate the informational level theory. Indeed, the addition of behavioural level constraints should result in algorithms that predict the detailed *performance* of subjects in analogical mapping tasks.

Behavioural Constraints on Analogical Mapping

Informational constraints help to characterise significant aspects of analogical competence. An adequate cognitive model of the analogical performance of human thinkers also requires the elaboration of behavioural constraints. We have elaborated two such constraints: working memory limitations and the effects of background knowledge (see Keane, 1988b, 1990, 1991).

Working Memory Constraints

It is well known that working memory limits both problem solving and reasoning performance (see e.g., Atwood & Polson, 1976; Johnson-Laird & Byrne, 1991). There are at least two ways in which working memory limitations might influence the nature and course of analogical thinking. First, working memory limitations may result in information loss, and thus produce errors in analogising. When working memory is overloaded, some critical part of the representation of a domain may be lost or forgotten. This degradation in the inputs to the mapping process could produce a corresponding degradation in outputs from it. In short, errors would be found even when it was clear that subjects had the prerequisite mapping competence. In support of this claim, Keane (1990, 1991) has found that domains which contain more conceptual information result in more mapping errors.

Second, the existence of working memory limitations is likely to influence the *way* in which analogies are processed. Since working memory can be overloaded, subjects should tend to use mapping techniques that reduce the processing load. For instance, rather than considering all the possible matches and sets of matches between two domains, they are more likely to select some small subset of these possibilities (see Forbus & Oblinger, 1990; Keane, 1988b). These methods may only be successful some of the time and, therefore, should result in systematic errors or mis-analogies. These methods should also result in differential response times for different analogies, depending on the ease with which the method can resolve a particular mapping problem. Later, we present one model which incorporates such techniques.

The Influence of Background Knowledge

Background knowledge has also been shown to have a significant influence in problem solving and reasoning (e.g., Eysenck & Keane, 1990; Kotovsky, Hayes

& Simon, 1985; Johnson-Laird & Byrne, 1991). For example, in deductive reasoning, background knowledge can facilitate or inhibit "correct" reasoning depending on its relationship to the inferences to be made (Byrne, 1989; Cheng & Holyoak, 1985). Similarly, in analogical thinking, Keane (1991) has shown that a mapping task can be performed faster if the set of mappings required are consistent with background knowledge, than if they are inconsistent or neutral with respect to background knowledge. Apart from affecting the time-course of performance, background knowledge may also be a source of errors in analogising when the products of mapping conflict with background knowledge of a domain.

Summary

An analogy theory which just consists of informational constraints, and no behavioural constraints, will not be an adequate cognitive model. Some facets of analogical performance will not emerge from a purely informational analysis of task situations. First, informational constraints will tell us little about the sources and pattern of errors in subjects' performance. In contrast, when behavioural constraints are added to an informational account, the way the model processes analogies should produce various performance effects (such as errors). Second, even when the outputs for a given analogy are predicted by informational constraints, response times for the task may be accurately predicted only by a model which includes behavioural constraints. Informational constraints deal with the basic competence that produces certain normative outputs given certain inputs; they are less adequate at predicting performance. Later, we report experimental comparisons of a model which incorporates behavioural constraints with models that omit them. But before we do this we need to consider briefly how the different models instantiate the above constraints.

HOW MODELS INSTANTIATE CONSTRAINTS

We have specified analogy theory in terms of constraints which unify many diverse statements by different theorists within a single framework (see Falkenhainer et al., 1989; Gentner, 1983, 1989; Holyoak & Thagard, 1989; Keane, 1988a, 1990). The diversity between the different views emerges from the way theorists have instantiated these constraints in various computational models. We will concern ourselves with three models that have addressed human analogising: the Structure Mapping Engine (SME; Falkenhainer et al., 1986, 1989), the Analogical Constraint Mapping Engine (ACME; Holyoak & Thagard, 1989), and the Incremental Analogy Machine (IAM; Keane, 1988b, 1990; Keane & Brayshaw, 1988). As we shall see, all three of these models implement most of the informational constraints underlying analogical mapping. Indeed, many aspects of these constraints are now well-understood and supported by empirical research. However, it is now clearly time to attempt a more fine-grained analysis of analogical performance, to begin to include various behavioural constraints and to explore these constraints empirically. This is one of the main aims of the current paper.

Structure-Mapping Engine

Falkenhainer et al.'s (1986, 1989) *Structure Mapping Engine* (SME) implements both structural and similarity constraints in a serial way. SME finds all the legal local matches between two domains and then combines these into alternative interpretations of the comparison. SME is explicitly designed to construct all possible maximal interpretations for a given comparison between two domains. When SME has an appropriate set of match rules -- the analogy match rules -- it instantiates Gentner's structure-mapping theory. However, it can be also be used as a tool, when different match rule sets are used. When SME is run on the attribute-mapping problem, with an appropriate set of match rules, it generates 32 alternative interpretations (see Appendix A). These are all the possible, maximal

interpretations that can be generated for the problem (made up from all the possible matches). Among these alternative mappings is the optimal or "correct" one which is indicated by the "goodness score" it receives. Of course, other less optimal interpretations will also be included (e.g., one interpretation just includes a match between *bill-rover*, *tom-blackie* and the attributes *smart-friendly* and *tall-hungry*). As such, SME uses a "generate and select" technique, where all the possible interpretations are generated and the best is selected from these using structural constraints like systematicity; that is, the interpretation with the most higher-order connectivity between its elements is selected. Gentner (1989) has proposed a general architecture within which the Structure Mapping Engine can work. Forbus & Oblinger (1990) have extended SME to implement some pragmatic constraints and their "greedy merge algorithm" reduces the number of interpretations produced to one (or a few) "best" interpretations.

Analogical Constraint Mapping Engine

Holyoak & Thagard's (1989) *Analogical Constraint Mapping Engine* (ACME) uses parallel constraint satisfaction in an interactive network to find the optimal mapping between two domains. It implements the structural, similarity and most pragmatic constraints. ACME establishes a network of units or nodes. Each node represents a legal match between two predicates. The excitatory and inhibitory connections between these nodes implement the various constraints. So, for example, the match between SMART(steve) and HUNGRY(fido) involves nodes representing the matches between SMART=HUNGRY and steve=fido. To implement structural consistency, there are excitatory links between the SMART=HUNGRY node and the steve=fido node. To enforce a one-to-one mapping there are inhibitory links between the SMART=HUNGRY and SMART=FRIENDLY nodes, and the SMART=HUNGRY and SMART=FRISKY nodes and also between the steve=fido node and the steve=blackie and steve=rover nodes.

When the network has been constructed, it is run until the activations of the nodes settle into a stable state. The nodes whose activation exceed a threshold correspond to the optimal set of matches between the two domains. In one sense, ACME produces just one mapping, a single optimal interpretation. Holyoak & Thagard's (1989) measure of mapping difficulty is the number of cycles the network goes through before it reaches the correct mapping. When ACME is run on the attribute mapping problem it creates a network of 43 units with 440 links between these units. With Holyoak & Thagard's parameter settings, we found that the network goes through 72 cycles before it stabilises. It reaches the correct mapping on its eight cycle (see Appendix A).

The Incremental Analogy Machine

Keane & Brayshaw's (1988; Keane, 1988b, 1990, 1991b) *Incremental Analogy Machine* (IAM) implements all the informational and behavioural constraints mentioned above using serial constraint-satisfaction. It generates a single, optimal interpretation based on a small subset of the possible mappings between the two domains. IAM builds up this mapping incrementally by selecting a small portion of a base domain for mapping, mapping it and then moving on to map another portion. Typically, it will construct a single mapping which will tend to be the optimal interpretation. However, if it has to, IAM can consider several alternative interpretations. Again, it deals with these alternatives incrementally, one after the other. So, if the first mapping that is built is less than optimal, IAM will undo the matches found and try an alternative mapping. For instance, depending on the version of the attribute-mapping problem it is given, IAM will generate between 1 and 8 alternative mappings. In order to appreciate how IAM instantiates both informational and behavioural constraints and how it differs from the other models we will consider some of its main features in more detail.

THE INCREMENTAL ANALOGY MACHINE

IAM implements the informational constraints that have been used so successfully in SME and ACME. However, its algorithm is quite different because it attempts to take into account certain behavioural constraints. For the purposes of the present paper, the working memory constraint is the most important (see later section for a discussion of the background knowledge constraint). There are a number of different ways that working memory limitations might be taken into account. One option would be to have a storage structure -- a working memory -- with a limited capacity. This would help to model some types of error production. However, at present, IAM does not include a limited-capacity working memory. In contrast, IAM uses an algorithm which is *designed to run in a limited-capacity system*; namely, an algorithm which minimises the amount of processing required.

The IAM algorithm reduces the processing involved in analogical mapping by incrementally mapping portions of a base domain, rather than mapping every element in the domain. Thus, it builds up a single interpretation based on the selected portion of the domain rather than many alternative interpretations. The algorithm is designed to generate the optimal mapping between the two domains but if the mapping it produces is not optimal then this mapping will be abandoned and another constructed. We expect this algorithm to be a better approximation to people's analogical behaviour; first, in its ability to generate complex analogical interpretations quickly and accurately and, second, in having the facility to re-consider a mapping made between two domains and to generate new alternatives. Of course, the time the model takes to re-consider alternative mappings should parallel the time people take to do this. In the following sub-sections, we outline the IAM algorithm in more detail and show how it deals with some classic examples from the literature.

---------- Insert Table 2 About Here ----------

The IAM Algorithm

Table 2 has a short, informal description of the IAM algorithm (see Appendix B for pseudo code and also Keane, 1993). In essence, IAM forms a mapping from a sub-set of the predicates/elements in a base domain rather than all the elements in the base. In particular, it selects that group of predicates which have the most higher-order connectivity between its elements. Having selected this, so-called *seed group*, it chooses an element from this group and finds the best match between this element and all the elements in the target domain (called the *seed match*). The seed match is used to *grow* the mapping of the other predicates in the seed group. All the legal matches between the other elements of the seed-group and the target domain are found and a unique one-to-one set of matches is formed. These matches are found by using serial constraint satisfaction - applying pragmatic, structural and similarity constraints. The seed match plays an important role in elminiating ambiguities arising from matches made from other seed-group elements. Depending on task demands and the success of this attempt to map the seed group, IAM may backtrack to try an alternative seed match or go on to map other groups of predicates in the base domain. Normally, however, a single interpretation based on the seed group will suffice as the output for the analogical comparison.

More detail on each of these stages is provided below with reference the solar-system / atom analogy introduced by Gentner (1983; see Figure 1). Table 3 indicates some of the slot contents of IAM's working memory after this analogy has been processed.

---------- Insert Figure 1 About Here ----------

Select Seed Group

The key to IAM's reduction of processing in analogical mapping is its selection of one part of the base domain, the seed group, for mapping to the target. A *group* is any inter-connected or systematic set of predicates in a domain. Most domains will have several groups. For example, in the solar system / atom analogy the base domain (i.e., the solar system) might contain the following predicates:

cause((and (weight-difference(sun, planet), attracts(sun, planet)),
revolve-around(planet, sun)) &
yellow(sun) &
enable (oxygen-atmosphere(planet), habitable(planet))

which make up three distinct groups:

Group 1: *cause((and (weight-difference(sun, planet),*
attracts(sun, planet)),
revolve-around(planet, sun))
Group 2: *enable (oxygen-atmosphere(planet), habitable(planet))*
Group 3: *yellow(sun)*

The first step in the IAM algorithm is to rank-order these groups so that the group with, for example, the most higher-order, systematic structure receives the highest ranking. The highest ranked group is then adopted as that portion of the base domain used for mapping (i.e., the seed group). So, in the solar-system domain Group 1 would be chosen as the seed group over Groups 2 and 3. While the structural criterion is the main one used in choosing seed-groups other criteria are also used; for instance, the number of predicates in the group, whether the group contains pragmatically-important elements or whether it just happens to be the first group encountered (see Appendix B for details).

---------- Insert Table 3 About Here ----------

Find Seed Match

Before the seed-group is mapped one special match, called the *seed match*, is found. A seed match is found so that there is one match that is adopted unambiguously before the rest of the seed group is matched. This seed match then plays an important part in resolving the ambiguous mappings found for the remaining elements in the seed group.

In order to find a seed-match, one must first select a *seed element* from among the elements of the seed group. At present, IAM favours relational elements that take multiple objects as seed elements (see Appendices A and B). All the legal matches between this seed element and the elements in target domain, according to IAM's *match rules*, are then found. IAM's *match constraints* are then applied to these matches to disambiguate them (should this be required). So, pragmatic, similarity and structural constraints are used to select one match as the seed match. At this time, other alternative seed matches and candidate seed elements in the group are noted for use in later backtracking (again, should they be required).

In the solar-system domain, Group 1 has been chosen as the seed group and *weight-difference(sun, planet)* is chosen as the seed element, mainly because it a relational element taking multiple objects (see Table 3). This element unambiguously matches *weight-difference(nucleus, electron)* in the atom domain; so, the seed match pairs *weight-difference=weight-difference*, *sun=planet* and *planet=electron*. The object mappings established by this match can be used to exclude relational mappings that violate these matches. Other alternative seed-elements are noted, namely, *revolve-around(planet, sun)* and *attracts(sun, planet)* which may be used later if this seed match fails to deliver an optimal mapping (see Table 3).

Find Isomorphic Matches for Group

After selecting a seed match, all the legal matches between the remaining elements in the seed-group and the target domain's elements are found, using IAM's *match rules*. The *match-rules* allow relations to match if they have the same functor and/or if they are of the same type (as in functions). The legal matches that result will tend to be ambiguous, with a number of one-to-many mappings between base and target elements. IAM then applies its *match-constraints* using serial constraint satisfaction to resolve these ambiguities (see Appendix B for details). Hence, structurally-consistent matches in an ambiguous match-set will be preferred over ones that lack such consistency, pragmatically-important matches will be preferred over those lacking such importance, and matches that are more similar will take precedence over those that are less similar in an ambiguous set. Amongst the pragmatic constraints IAM has a *favour-first constraint,* which is applied when no structural, pragmatic-importance or similarity constraints disambiguate a match-set; it is this constraint which favours matches encountered before other matches (see Study 2 later). This constraint satisfaction finds an isomorphic mapping between the elements in the base, seed group and those of the target domain.

For example, this step will find the following matches between the seed-group in the solar-system domain and the atom domain (see Table 3):

weight-difference(sun, planet) = weight-difference(nucleus, electron)
revolve-around(planet, sun) = revolve-around(electron, nucleus)
attracts(sun, planet) = attracts(nucleus, electron)

These relational matches support the following object mappings: *planet=electron* and *sun=nucleus*.

Find Transfers for Group

Those elements in the base, seed-group which are not included in the one-to-one matches found in the previous stage can now be transferred into the target domain to constitute *analogical transfers* or *analogical, candidate inferences*. These are inferences suggested by the mapping which hold *by analogy* in the target domain. These transfers are an important source of new knowledge in the target.

So, for example, in the solar-system analogy the *cause* and the *and* relations in the seed group have not been matched and can be transferred. These are transferred resulting in the following mapping between the two domains (see also Table 3):

weight-difference(sun, planet) = weight-difference(nucleus, electron)
revolve-around(planet, sun) = revolve-around(electron, nucleus)
attracts(sun, planet) = attracts(nucleus, electron)
*and(weight-difference, attracts) = *t*and(weight-difference, attracts)*
cause(and-weight-diff-attracts, revolve-around) =
**t*cause(and-weight-diff-attracts, revolve-around)*

where the "*t*" indicates that this is a transferred element. In essence, this means that we can now infer by analogy that the weight-difference *and* attraction between the electron and the nucleus should *cause* the electron to revolve around the nucleus.

Evaluate Group Mapping

After establishing a mapping for the seed group, IAM performs a minimal evaluation on it. The evaluation criteria at this point could be quite complicated but at present in the program they are very simple. IAM's evaluation function simply looks at how many matches have been made relative to the number of predicates in the seed group. If more than half of the predicates have been matched successfully

then the mapping is accepted as optimal, otherwise it is not optimal. This type of evaluation has sufficed for all the domains used in our tests.

If the mapping of the seed group is not optimal, then IAM will begin to backtrack, to re-consider the mapping. This type of "backtracking" is designed to approximate subjects' phenomenal experience of trying one set of mappings and then another. At first, IAM will consider alternative seed-matches if any exist. If there are alternatives and they fail to deliver an optimal mapping then it will consider alternative seed elements. If all of these mappings are not optimal, IAM may consider another group as the seed group. So, IAM has considerable backtracking potential. In practice this potential tends not to be used. However, the attribute mapping problem is an exception. We shall see later, in some versions of this problem a number of alternative seed matches have to be tried. We should also say that IAM avoids "awkward loops" like *oscillations*; where one mapping is produced and rejected (call it mapping-1) and then another mapping is produced and rejected (mapping-2), before considering mapping-1 again. This sort of looping cannot happen given the way in which IAM backtracks.

In the solar-system / atom example, the mapping found is evaluated as being optimal. So, no backtracking occurs. If the mapping had not been optimal then another seed element (e.g., *attracts(sun, planet)*) would have been tried, because there are no alternative seed matches for the *weight-difference* element.

Find Other Group Mappings

Typically, IAM will halt after it has mapped the seed group. We believe that usually the point of an analogy will be to develop a swift interpretation of the analogy, which delivers some analogical inferences about the target. This is what happens in the solar system / atom analogy. However, there are occasions when task demands are different. If people are asked to map all the elements in the base

domain, then IAM will go on to map *incrementally* the other groups (this is required in the attribute mapping problem, see also Appendix A). Similarly, when people are asked to match up two domains in a relatively complete fashion they should map as many of the groups in the base as they can. The mapping of other groups iterates through the above steps (see Table 2). However, all these subsequent mappings must conform to the mappings that have been established by the first group mapping; in particular, they must be structurally consistent with it.

Summary

The IAM algorithm goes through six main stages in analogical mapping: it selects a seed group and finds a seed match for that group, then it finds isomorphic matches and transfers for that group, it will then evaluate that group mapping and finding other group mappings if they are required. The IAM algorithm is specifically designed to reduce the processing load involved in analogical mapping, in order to unburden a limited-capacity working memory. It does this by (i) mapping just one of the many possible groups in a base domain and (ii) incrementally undoing and re-considering mappings if they are unsuccessful in the quest for the best mapping, rather than computing all possible mappings and (iii) in mapping other groups incrementally (if this is required). As such, the algorithm is shaped by one of the important behavioural constraints in analogical mapping. □

IAM : Some Examples

IAM has been applied to the standard examples reported in the literature (see Falkenhainer et al, 1989; Holyoak & Thagard, 1989; Keane & Brayshaw, 1988). In general, its capabilities match those of SME and ACME (see Keane, 1993, for full details on these tests). While there is insufficient space here to describe all of these tests in detail, some representative examples from the literature are reported .

The representations for these analogies, as they were given to the program are provided in Appendix C.

---------- Insert Figure 2 About Here ----------

The Water Flow / Heat Flow Analogy

Falkenhainer et al (1989) use the water flow / heat flow analogy as a key example in presenting SME (from Buckley, 1979). This is a situation where water flow from a beaker along a pipe to a vial is seen as being analogous to the flow of heat from a cup of coffee along a bar to a ice-cube (see Figure 2 and Appendix C). Table 4 shows the seed-group, seed-element and seed-match that are formed for this analogy (see Table 4). The seed group chosen, because it is has the most higher-order structure, is that group which encodes the causal relationship between the pressure difference and the flow of the water along the pipe from one container to the other. The *flow(beaker, vial, water, pipe)* predicate is chosen as the seed element and the unambiguous seed match formed from it is between *flow(beaker, vial, water, pipe)* and *flow(coffee, ice-cube, heat, bar)*. The overall mapping found by IAM for this analogy corresponds to that found by SME. The *causal* relation holding between *pressure-difference* and *flow* in the water-flow domain is transferred to hold between the *temperature-difference* and *flow* relations in the heat-flow domain (see Table 4).

---------- Insert Table 4 About Here ----------

As in SME, IAM's match rules allow a function to match any other function. So, potentially, there are many legal but ambiguous matches for all the functions considered in both domains: for instance, *pressure(beaker) = temperature(water)*, *pressure(beaker) = temperature(ice-cube)*. However, some of these matches are never considered because they are not members of the chosen seed-group: for instance, matches from *diameter(beaker)* and *diameter(vial)* are never considered.

The remaining ambiguity is resolved by structural constraints which remove mappings that conflict with known mappings and which favour matches supported by previous mappings.

---------- Insert Figure 3 About Here ----------

The SME Atom Solar-System Analogy

Falkenhainer et al (1989) have used a more complicated version of the solar-system / atom analogy, than the one presented above (see Figure 3 and Appendix C). In this representation of the analogy there are more groups (with a greater degree of higher-order structure) in both domains. This version of the analogy admits a much greater potential for ambiguity as a result of the presence of many functions and higher-order, causal structures in both domains. In terms of IAM's criteria the best group is that which involves the predicates about the mass-difference, attraction and their causal connection to revolution. Again, IAM makes the mapping interpretation typically generated by people and SME, based on this structure (see Table 5).

---------- Insert Table 5 About Here ----------

The Attribute Mapping Problem

The attribute mapping problem differs in several respects from the above mapping examples. First, in all of the above examples the analogy is developed from the first mapping considered. So, IAM never has to backtrack to re-consider the mapping it has produced. However, typically, in the attribute-mapping problem (shown in Table 1) several alternative mappings have to be considered. Second, this analogy has specific task demands which do not arise in the previous analogies; specifically, all the items in list A have to be mapped to all the items in list B. In IAM, this means that all the groups in the base domain, not just the seed-group,

have to be mapped. Furthermore, the potential for mapping ambiguity in this problem is much greater than in previous problems and this ambiguity is not easily resolved.

IAM divides up the base domain of the attribute mapping problem (i.e., list A) into five, single-attribute groups. There are no criteria for preferring one of these groups over another, so the first encountered is chosen as the seed group. This group consists of a single element (i.e., *tall(Bill)*). Given the demands of the problem, any element can match any other element so there are five alternative, candidate matches for the seed-match: *tall(Bill) = hungry(Fido), tall(Bill) = friendly(Blackie), tall(Bill) = frisky(Blackie), tall(Bill) = hungry(Rover), and tall(Bill) = friendly(Rover).* However, none of the constraints can disambiguate these matches, so the *favour-first* constraint simply takes the first encountered *tall(Bill) = hungry(Fido)* and uses it as the seed-match. However, starting with this seed match will not allow us to reach an isomorphic mapping between the two lists, so when the mapping from this seed match fails, IAM will backtrack, re-consider the mapping and use the next alternative seed match.

---------- Insert Table 6 About Here ----------

Table 6 shows the contents of some of the working memory slots after the mapping has been achieved. Note that the alternative-seed-matches list, which would have originally held all the matches from *tall(Bill)* listed above, has been reduced to nil, as each of these alternatives has been tried as the seed match. For this version of the attribute mapping problem, IAM has to re-consider the mapping made four times, so it generates five alternative mappings for the problem before the correct one is found. This behaviour, as we shall see in the experiments reported here, has important implications for predicting the time course of analogical mapping on this problem.

Other Features of IAM

The above examples give some indication of how IAM works on particular examples. The pseudo code description of the algorithm in Appendix B gives more detail on how the program works. In this section, we review briefly some of the other features of IAM; specifically, in comparison to its close relative, SME.

IAM Can be Used as a Tool

IAM, like SME, can be used as a tool to implement different types of comparison. Specifically, there are four different sets of functions that can be re-configured in IAM (see Appendix A, section 4, for the configuration used in this study):

- *seed-group choice criteria* - the functions which rank-order the groups in the base domain, to choose the seed group
- *seed-element choice criteria* - the functions used to choose a seed-element
- *match rules* - the rules that determine what counts as a legal match between items (i.e., elements or objects) in the base and target domains
- *match constraints* - the constraints that are used to disambiguate the matches between the base and target domain

In all the examples reported above and in the current computational experiments the seed-group choice criteria, seed-element choice criteria and the match constraints were kept constant. For example, the match constraints were always the pragmatic, similarity and structural constraints and they were always applied in the same order (see Appendix A).

However, the match rules had to be varied for the attribute-mapping problem. In this problem, we want to match attributes irrespective of their meaning, therefore any relational element can be matched legally with any other relational element.

Normally, two relational elements would be required to have the same functor, unless they were functions (see Appendix A).

Structural Constraints and Systematicity

IAM, like ACME, implements the structural constraints which were first realised in SME. However, the way IAM computes such constraints differs considerably from SME because of the nature of the IAM algorithm.

First, in the SME algorithm, the computation of isomorphic mappings is a hard constraint. That is, this constraint is inviolate, mappings always *have* to preserve isomorphism. In ACME, however, this constraint can be applied as a soft constraint, merely as a pressure towards a one-to-one mapping, that can be relaxed relative to other constraints. This is an important property for an analogy program because people seem to show some tolerance for violating isomorphism (see Reed, Ernst & Banerji, 1975; Holyoak & Thagard, 1989). In IAM, the computation of isomorphic mappings by the structural constraint is also a hard constraint. However, it might be feasible to develop a different structural constraint to use in IAM that does not apply isomorphism in such a procrustean fashion.

Second, in SME, the computation of systematicity is one of the central achievements. Each match is given an evidence score and the scores for the matches in a given interpretation are combined in order to derive an overall goodness score for a given mapping. The scheme which combines these match scores implements systematicity and leads to the most systematic mapping being given the highest goodness scores. IAM has no such weighting scheme and does not assign evidence to local matches. In SME the computation of such evidence is computationally very expensive, so IAM is less expensive and much simpler in this respect. If IAM can be said to compute systematicity at all it does so in its

preference for groups with the higher-order connectivity. It might be more precise to say that IAM computes one of the important conditions for systematicity.

Background Knowledge Constraint

In this paper, we concentrate on one of the two proposed behavioural constraints; namely, the working memory constraint. The second constraint concerns the influence of background knowledge on analogical mapping. Our proposal is that this influence enters into the formation and evaluation of analogical transfers. Specifically, we believe that analogical transfers or candidate inferences can be *validated* locally with respect to background knowledge. Keane (1985) has pointed out that analogical transfers may suggest new objects in the target domain (see also Falkenhainer et al., 1989). For example, in the general story / radiation problem analogy (from Gick & Holyoak, 1980), people are given a story about a general's attempt to attack a fortress by sending small groups of men along different roads so that they converge on the fortress. This is designed to suggest the analogical solution to Duncker's (1945) radiation problem of sending multiple, low-intensity rays along a number of paths so that they converge on a tumour to destroy it. Among the many analogical transfers here that people have to make, is the transfer of the base object "roads" to become the target object "paths". But, the concept *path* is not mentioned in the statement of the radiation problem; so this concept has to be added to the target representation.

In SME, this sort of object transfer is handled by the use of skolemized objects in the target domain. Falkenhainer (1987, 1990) has proposed a later verification stage where an analogical solution is treated as a qualitative model of the physical world which can be validated and refined. This verification stage should find appropriate objects for such skolemized entities. In IAM, we assume that relational and object transfers are checked when transfers are being formed for a mapping. So, there is a validation sub-routine in step 5 of the IAM algorithm

which checks candidate inferences against background knowledge of the target domain (assuming that such knowledge is available). This proposal leads to the prediction that if these transfers suggest something which is familiar with respect to background knowledge then the mapping should be easier than if they suggest an unfamiliar inference. Keane (1991) has tested this prediction and found evidence to support it. For present purposes, it is sufficient to note that IAM has a method for implementing the other behavioural constraint mentioned here.

COGNITIVE MODELS AND MEASURES FOR COMPARATIVE COMPUTATIONAL TESTS

The core argument in this paper is that in order to develop cognitive models of analogical mapping we need to take *both* informational and behavioural constraints into account. As we have seen, SME and ACME essentially implement algorithms based on informational constraints alone. However, this does *not* mean that SME and ACME cannot make predictions about subjects' behaviour. Models based on informational constraints *can* make behavioural predictions because these constraints capture significant aspects of the task situation. For example, Skorstadt, Falkenhainer & Gentner (1987) have shown that goodness scores generated by SME for different comparisons exhibit relative differences that parallel subjects' soundness ratings for the same comparisons. Similarly, Holyoak & Thagard (1989) have shown that the number of cycles ACME goes through to reach the correct mapping reflects the relative difficulty experienced by subjects on the same analogies. However, we will show that finer-grained predictions can be made from models which include behavioural constraints, as well.

In the remainder of this paper, we illustrate this point by comparing SME, ACME and IAM in a number of task situations. However, in order to do this, we require a common measure across the three models [1]. The most obvious candidate is *the number of alternative mappings that a model computes*. That is, the number of

alternative mapping interpretations of a comparison; what are called G-maps in SME. In essence, Holyoak & Thagard (1989) use this measure in ACME when they count the number of cycles to the correct global mapping between the elements of two domains, because each cycle of ACME is really a state in which a putative mapping interpretation has been formed. So, the number of cycles to the correct mapping indicates the number of alternative mappings considered [2]. Similarly, in IAM, alternative mappings are easy to count by looking at the number of times the program backtracks to consider either a new seed-match, a new seed-element or another seed-group. It is also easy to use this measure in SME, because its outputs consist of alternative mappings (i.e., G-maps) scored in terms of their structural goodness.

However, the use of this measure in SME needs to come with a caveat. SME is designed to generate all possible, maximal mappings in order to explore the space of alternative mappings between two domains. Hence, for SME, this measure is not intended to be a predictor of human behaviour. It is, however, useful to employ SME in comparative tests for this very reason, because it indicates relative to the other models, how many mappings are possible for a given analogy. We have, therefore, retained this measure here for SME (later we discuss other possible measures).

Assuming this common measure for assessing the outputs of the different models, we can now look at the specific predictions that are made for two psychological experiments. The first of these experiments examines the facilitative effects of similarity on analogising, and the second examines order effects on analogical processing. In both cases, we will first outline the general form of the experimental manipulation and the rationale for it, then describe the computational experiment carried out to generate predictions, and, finally, the psychological experiment used to test these predictions.

STUDY 1: THE EFFECTS OF SIMILARITY

Many experiments have shown that varying the similarity of two analogues affects the ease of analogical mapping (e.g., Gentner & Landers, 1985; Gentner & Toupin, 1986; Keane, 1985; Ross, 1987). Holyoak & Koh (1987) have distinguished between surface and structural similarity and have shown that the latter has a significant effect on the production of correct mappings using various story analogues to Duncker's radiation problem. Typically, this facilitation has been measured by the frequency of correct mappings for different groups of subjects. Differences in response times taken to perform a complex mapping, as a function of similarity, have not been examined. Traditionally, response time is one of the finest measures used in psychological research, yet to-date it has not been used in analogy research.

In these experiments, we systematically manipulate the similarity constraint by varying the number of predicates that are similar, while controlling the other constraints. For instance, a modified version of the problem in which all the attributes are similar should be easier:

A	B
Bill is intelligent.	Fido is clever.
Bill is tall.	Blackie is big.
Tom is timid.	Blackie is shy.
Tom is tall.	Rover is clever.
Steve is intelligent.	Rover is big.

To test the prediction that semantic similarity can facilitate response time on an analogical mapping task, three different versions of the attribute-mapping problem were used which had either no similar attributes, one set of similar attributes or three sets of similar attributes (see Table 7).

---------- Insert Table 7 About Here ----------

Experiment 1A: Computational Tests of Similarity

The inputs to the computational models were predicate calculus representations of the different versions of the problems. In the models which instantiate similarity constraints, we included information about which attributes were similar to one another (see Appendices A and C).

Method

Materials & Design. The materials used in the computational experiment were predicate calculus representations of the problems shown in Table 7. The order of attributes in each domain was randomised, with the constraint that attributes about the same individual were kept together. Ten such randomly-ordered problems were generated (the specific attributes and names in the problems were held constant). Each of these ten problems were then modified to produce versions in which either none of the attributes were similar, one attribute was similar or all the attributes were similar in each list. The materials thus consisted of three sets of ten problems used in three experimental conditions; the None-Similar, One-Similar and All-Similar conditions.

Procedure. Each of the problems were run on the three different models; SME, ACME and IAM (see Appendix A for implementation details). After running a problem on a particular program the number of alternative mappings taken to reach the correct solution were recorded.

---------- Insert Figure 4 About Here ----------

Results & Discussion

Figure 4 shows the mean number of alternative mappings generated by the different models for the different conditions of the experiment. □ The figure shows that both ACME and IAM manifest a trend indicating that increasing similarity results in a decreasing number of alternative mappings. The frequency of mappings gradually decreases from the None-Similar condition, to the One-Similar and the All-Similar conditions. SME shows us that there are many possible mappings for the different versions of the problem and they do not differ across conditions.

For ACME, there is a reliable trend in the mean number of mappings, decreasing from the None-Similar (M = 8, SD = 0) to the One-Similar (M = 4, SD□= 0) and All-Similar (M = 3, SD = 0) conditions. For IAM there is also a reliable trend from the None-Similar (M = 3.2, SD = 2.25), One-Similar (M = 2.4, SD = 1.43), and All-Similar (M = 1.3, SD = .48) conditions [$F(2, 18) = 6.88$, $p < .01$]. SME shows us that the complete set of alternative mappings for a problem, irrespective of similarity, is 32. It also identifies the correct mapping for the problem as that interpretation with the best score.

This computational experiment shows us that both ACME and IAM predict that as the similarity between the domains in a problem increases the problem should be easier to solve; with the All-Similar problems being easier than the One-Similar problems, which in turn should be easier than the None-Similar problems. In ACME these outputs are produced because similar matches receive more positive activation and hence the network stabilises faster on the correct mapping. In IAM the similarity between predicates helps to reduce the list of seed-matches and hence the number of times that IAM backtracks to consider alternative seed-matches. For the reasons outlined earlier, SME shows no differences between the conditions (see conclusions section for alternative measures for SME). Let us now look at what people *actually* do.

Experiment 1B: Psychological Tests of Similarity

We have seen the sorts of outputs produced by the computational tests in Experiment 1A. Here we test these predictions against the evidence of a psychological experiment which parallels the computational tests.

Method

Materials. We used three versions of the attribute-mapping problem (see Table 7). Each version had two lists of attributes. In each list, there were three individuals and three attributes; two individuals had two attributes and one individual had a single attribute. The three versions differed in terms of the number of attributes that were similar in both lists. In the None-Similar version none of the attributes were semantically similar; in the One-Similar version one set of the attributes was semantically similar ("intelligent" in list A and "clever" in list B); in the All-Similar version, all the attributes in one list had a semantically-similar, parallel attribute in the second list.

Procedure. Subjects were instructed in writing that their "task is to figure out what in the left set corresponds to what in the right set of sentences". A single column below list A listed the names of the individuals and attributes in that list. Next to each was a space for subjects to write the corresponding name or attribute from list B. The order of sentences in each list was randomised with the proviso that sentences with attributes about the same individual were kept together.

Subjects were first shown the instructions and were asked to read them carefully. They were then shown the problem and asked to solve it. A stop-watch was used to time them from this point to when they solved the problem. If subjects produced an incorrect answer they were told so and asked to continue solving the problem. Only when the correct answer was produced was the clock stopped and the elapsed time recorded.

*Subjects & Design***.** Twenty-four undergraduates at the University of Wales College of Cardiff took part voluntarily in the experiment. The experiment used a between-subjects design and subjects were assigned randomly to one of the three conditions; the None-Similar, One-Similar and All-Similar conditions. Three subjects were dropped from the experiment before any data analysis because they misunderstood the experimental instructions. Data analysis was carried out on the remaining 21 subjects, who were equally distributed across the three conditions.

---------- Insert Figure 5 About Here ----------

Results & Discussion

The results corroborate the predictions of the ACME and IAM models. The presence of semantic similarities between the elements of the two domains has an important facilitating effect on the ease of analogical mapping (see Figure 5). The elapsed time taken to solve the problem gradually decreased across the three conditions, with subjects in the None-Similar Condition being the slowest (M = 210.9 secs.), those in the One-Similar Condition were faster (M = 164.9 secs.) and those in the All-Similar Condition the fastest [M = 69.7 secs.; $F(2,12) = 12.022$, $p < .01$].

Previous tests of similarity effects have hinged largely on differences in the frequency of correct solutions in analogical problem solving (cf. Gick & Holyoak, 1980; Keane, 1987). However, this study is the first demonstration that domains which systematically differ in terms of their similarity give rise to systematically differing response times.

Both ACME and IAM predict the similarity effects found, but an important difference between these two models emerged in this study. In both Experiments 1A and 1B the order of attributes in the lists were randomised. In Experiment 1A, the results from ACME in each of the conditions had no variance in the scores; that

is, in the None-Similar condition ACME *always* reached success in 8 cycles, in the One-Similar condition it *always* succeeded in 4 cycles and it *always* took 3 cycles in the All-Similar condition. This lack of variance within conditions indicates that the random ordering of attributes had no effect on ACME's performance. In contrast, IAM exhibited variance in all of the conditions indicating that the ordering of attributes matters to IAM. We examined these order effects in Study 2.

STUDY 2: THE EFFECTS OF ORDER

A wide variety of task-demands may influence analogical processing (i.e., as other manifestations of pragmatic constraints). These demands may include specific task instructions and features of the task itself. In Study 2, we examine one such task demand, the order in which domain information is presented to subjects. We expect that it will affect the ease of analogical mapping. In Study 1, we randomised the order of presentation of the attributes in both lists thus controlling for any order effects. But, consider how order might make some versions of the attribute-mapping problem easier to solve.

In the original version of the attribute-mapping problem (i.e., the None-Similar version), each list has two individuals (e.g., Bill and Tom) with two attributes and a remaining individual (i.e., Steve) who has just one attribute (See Table 1). Matching up the single individuals in both lists (i.e., Steve and Fido) is the key to achieving the isomorphic mapping. The presence of these single individuals with one attribute (which we will call *singletons*) disambiguates the set of matches between the two lists (this argument should also apply to the single attribute in both lists). So, if the singletons are matched *before* the other attributes then the correct mapping should be found more easily. We should, therefore, be able to test for the effects of order by placing both of the singletons at the beginning of the lists rather than at some other position in the lists, where they are less likely to be encountered first. This prediction should follow from a model that is sensitive to such task

demands. In the next section, we consider the outputs of the various models on this matter.

---------- Insert Table 8 About Here ----------

Experiment 2A: Computational Tests of Order

As before, we carried out computational experiments using the three models to determine their outputs for different versions of the attribute-mapping problem.

Method

Materials, Design & Procedure. In the computational tests, two sets of ten problems were generated; (i) the Singleton-First set contained problems in which both of the singletons were at the beginning of the lists (and the remaining attributes were randomly ordered as in Study 1) and (ii) the Singleton-Last set was made up of problems in which the singleton in List A was always in the last position, while the singleton in list B was in the first position (see Table 8). The experiment had two conditions, the Singleton-First and Singleton-Last conditions. The ten problems in the Singleton-First set and the ten problems in the Singleton-Last set were run on each of the models and the number of alternative mappings produced for each problem were noted.

---------- Insert Figure 6 About Here ----------

Results & Discussion

Figure 6 shows the mean number of alternative mappings generated by the different computational models for the two sets of problems; the Singleton-First and Singleton-Last problems. For SME and ACME the products of the mapping process were the same in both the Singleton-First and Singleton-Last conditions, with M = 32 (for SME) and M = 8 (for ACME; SD = 0 for both). SME's outputs

show us, as we would expect, that the number of possible mappings does not change for these different versions of the problem. ACME predicts that order has no effect on the ease of analogical mapping.

In contrast, IAM predicts a marked difference between the two versions of the problem. The Singleton-First problems were solved after just one mapping was generated (M = 1, SD = 0), whereas the Singleton-Last problems on average required more mappings to be produced (M = 2.9, SD = 1.29) and this difference was statistically reliable on an independent t-test [$t(18) = -4.67$, $p < 0.001$].

This pattern of results is found in IAM because its algorithm is constrained by the behavioural constraint of working-memory limitations. Specifically, it arises from the way in which the algorithm deals with this task situation. First, as we saw earlier, in the attribute mapping problem all the groups in list A are structurally equal. This means that the choice of a seed group is based on the first group encountered in the task. Second, the single element in the seed-group can be matched legally with all five elements in the target domain; so it gives rise to five alternative possible seed-matches. Again, the only criterion for choosing one of these is to select the first encountered (the other constraints do not resolve this ambiguity).

Both of these factors mean that, in the Singleton-First problems, the singleton in list A is chosen as the seed group and the match between it and the first element of list B (i.e., its singleton) is chosen as the seed match. Since this match is the most unambiguous match to start with in mapping the problem, the correct mapping is found at once. In contrast, in the Singleton-Last problems a non-singleton attribute is chosen as the seed group and the seed-match is between it and the singleton in list B. This seed match will not deliver a single consistent mapping for all the elements in the two domains, so at least one alternative seed-match (and hence an alternative mapping) has to be considered. Normally, several alternative

seed matches will have to be considered for these Singleton-Last problems and hence this class of problem takes longer than Singleton-First problems. In short, the differences in these two conditions is predicted by IAM because of its incremental mapping heuristic and the sensitivity of this heuristic to the pragmatic constraints of task demands.

Experiment 2B: Psychological Tests of Order

In Experiment 2A we saw that ACME and IAM lead to different predictions about the effects of order on analogical mapping. ACME predicts that order should have no effect whereas IAM predicts an advantage for Singleton-First problems over Singleton-Last problems. In this psychological experiment, we examined which of these models best approximates the performance of people facing the same manipulation.

Method

Subjects & Design. Twenty-three undergraduates at the University of Wales College of Cardiff took part voluntarily in the experiment. The experiment employed a between-subjects design and subjects were assigned at random to one of the two conditions; Singleton-First or Singleton-Last conditions. Three subjects were excluded from the experiment prior to data analysis because they misunderstood the experimental instructions. Data analysis was carried out on the remaining 20 subjects, who were equally distributed across the two conditions.

Materials & Procedure. The materials consisted of two None-Similar versions of the attribute-matching problem (see Table 8). In the Singleton-First version the singletons were at the top of both lists, while in the Singleton-Last version the singleton in list A was in the last position, while the singleton in list B was in the first position. The order of the remaining sentences was randomised as in Experiment 1B.

The instructions were as in Experiment 1B, except for the introduction of the following sentence: "The meaning of the words in the sentences is irrelevant". Holyoak and Thagard (1989) used this instruction when they gave subjects the original version of the problem. Subjects were shown the sheet containing the instructions and problem, and were timed, as before.

---------- Insert Figure 7 About Here ----------

Results & Discussion

The results corroborate the predictions of the IAM model and run counter to the those of the ACME model. The slight change in the ordering of the singletons has a marked effect on the ease of analogical mapping (see Figure 7). Subjects in the Singleton-First condition were almost twice as fast at solving the problem (M=178.0 secs) compared to the Singleton-Last condition (M=363.1 secs) and this difference was reliable [Mann-Whitney U= 7, $p < .005$, 1-tailed].

This result shows that the order in which the attributes are presented affects subjects' latency to solve the problem. This result is predicted by IAM because of its incremental mapping and sensitivity to pragmatic constraints.

CONCLUSIONS

Theoretical proposals on analogical mapping can be organized within a meta-theoretical framework, which makes a distinction between informational and behavioural constraints. The main informational constraints of importance are the structural, similarity and pragmatic constraints. The primary behavioural constraints are working memory limitations and background knowledge. The three current models of analogical mapping instantiate these constraints to different degrees. SME mainly instantiates the informational, structural and pragmatic constraints. ACME and IAM model all three informational constraints. But IAM

alone models the behavioural constraints. In the present studies, we have seen that a model which includes the behavioural constraints is better able to predict people's performance.

In our first study, we saw the effects of similarity on the ease of analogical mapping. A gradual improvement in performance as a function of the number of similar elements between the two domains was reflected in the predictions of ACME and IAM. In particular, we found that increasing the number of similar attributes between two domains resulted in a parallel decrease in the time taken to perform a difficult analogical mapping. Previous research has shown that similarity affects the likelihood of success in analogical mapping, but latencies have never been used as a measure (see e.g., Holyoak & Koh, 1987). The research evidence is thus unanimous on the centrality of similarity constraints. In our second study, we saw that the ACME model, which excludes behavioural constraints, fails to predict the effects of order on analogical mapping. In contrast, IAM captured the marked difference we found for versions of the problem which differ in the order in which information is presented.

This research has implications for our understanding of analogical mapping, for the conduct of analogy research and for cognitive science, in general. In this concluding section we discuss briefly these implications.

Computational and Empirical Issues

The present work raises an number of issues for computational models of analogy and for the course of future empirical work. From a computational perspective, the natural question to ask is how existing models might be modified to include behavioural constraints. Empirically, these studies suggest several lines of future research and the possibility that a more fine-grained analysis of subject's analogical behaviour is possible.

Can ACME and SME Capture these Phenomena ?

Can ACME and SME be naturally extended to encompass the phenomena found in these studies ? We have seen that ACME makes appropriate predictions for similarity manipulations in these experiments, but that it falls down on order predictions. We find it hard to see how minor modifications to ACME would improve this state of affairs. One option would be to assign extra activation to match-units that are encountered earlier in the task. But, it is not clear whether this would deliver the required performance. It would also make ACME much more complex and unwieldy. The basic problem lies in the fact that ACME is inherently *parallel* and the behaviour we see in these problems is inherently *serial*. In our view, the best option would be to develop a hybrid model which imported aspects of IAM into ACME. So, one could have a serial, IAM-like component in ACME that would handle the choice of seed groups, seed elements and seed matches and the backtracking between mappings. But, the mapping of a given group would be handled as before by ACME's network.

It is somewhat harder to bring SME into contact with these findings. The first problem is that, as we saw earlier, SME was designed to produce all possible, maximal interpretations of an analogical comparison. So, SME was never intended to make psychological predictions based on the number-of-alternative-mappings measure. There are versions of SME which have been designed to produce fewer alternative mappings. Forbus & Oblinger's (1990) greedy-merge algorithm produces a single best interpretation and can be set to produce different numbers of alternatives. However, greedy-merge is still not constrained in a psychological fashion, as the number of alternative mappings produced is just set as one of the parameters of the system. The algorithm should generate different frequencies of mappings as a function of the nature of the mapping task. Such a solution would

appear to require a more radical modification of SME's control scheme (Forbus, 1993, considers this problem).

Another alternative is to use a different measure as a psychological predictor. One possibility is to consider the number of *root matches* that SME generates for a given mapping (suggested by Forbus, 1993). These root mappings are more basic than the interpretations SME generates and might provide a better indicator of performance differences. However, on the basis of our analysis of SME, it looks like SME should always generate 25 such mappings for all the versions of the attribute mapping problem used in these present studies. So, again, one faces the problem of how SME might be modified so that the structure of the problem would result in different numbers of root mappings being generated. As far as we can see, the most feasible option would be to modify the control structure of SME, so that some sub-set of root mappings are generated before another sub-set of mappings (a step which would amount to making SME more like IAM).

Future Empirical Directions

The present experiments have important implications for the methodology and substance of future empirical research. First, they bring us closer to subjects' analogical behaviour. One of the major problems in analogy research has been finding suitable measures of analogical performance. Traditionally, researchers have used relatively coarse measures; frequency counts of success/failure in problem solving or ratings of the soundness of analogies. The experiments show that response latencies can also be informative, that similarity affects the real-time course of analogical performance. This is a more fine-grained measure of analogical performance and combined with the increasing sophistication of our computational models shows great promise.

Apart from adding to the measures used in analogy research, the work also suggests a number of substantive research issues. We have shown that the ordering of elements in a domain has marked effects on the ease of analogical mapping (see Experiment 2B). This phenomenon is one of the effects of task demands on analogising. In general, the role of task demands has been neglected in analogy research. We have argued that they are best thought of as being a form of pragmatic constraint. Clearly, much future work needs to be carried out on these factors. For instance, one important question for future research concerns the extent to which specific task demands recur across different situations. We have shown that these effects hold in an attribute-mapping problem. This problem involves an "odd" sort of analogy, involving only attributes and no relations, although it allows us to uncover significant aspects of the mechanism which draws analogies. One future question is whether similar results are to be found for relational mappings. IAM predicts that such results should be found for problems of the same form involving relations:

A	B
Mark is beside Ronan.	Lisa hugs Jenny.
Mark motivates Ronan.	Laura employs Ruth.
Conor is beside Paul.	Laura hugs Ruth.
Conor fears Paul.	Mary sees Ali.
Joe motivates Steven.	Mary employs Ali.

So, moving the singletons around in the above problem (i.e., Joe motivates Steven and Lisa hugs Jenny) should have the same order effects we observed for the attribute-mapping problem. Similarity effects should also be observed by using sentences like "Joe cuddles Steven". Furthermore, relational-mapping problems could be extended to examine other factors. For example, if we establish strong

causal relations between the relations in the non-singleton elements (e.g., Laura employs Ruth and Laura pays Ruth), then this should change the predicate-grouping in the domain, thus influencing the mappings people make and the relative difficulty of these mappings. We are currently considering such issues.

Wider Implications for Analogy Research

Theories of analogical mapping have been converging for some time now. We have tried to speed this convergence by showing how the different theoretical constraints proposed can be related together in a meta-theoretical framework. From this endeavour it becomes clear that the differences in computational models can be traced directly to the various high-level constraints taken into account by the models. Given this more unified state of affairs, we are presented with the opportunity to modify our research strategy.

Rather than using the traditional falsification strategy for conducting research, a collaborative, approximation strategy may be appropriate at this stage (see also Newell, 1990). That is, given the state of theory in this area is makes more sense to attempt to refine our models so that they become closer and closer approximations to subjects' behaviour. As we have argued here, there are many aspects of subjects' analogical performance, rather than their analogical competence, which have not been given much attention (e.g., the sources of errors and response times).

Implications for Cognitive Science Practice

The present research supports several propositions about the practice of cognitive science. First, it shows that computational theories of higher-level cognitive processes are possible. Second, it indicates that the statement of such theories as sets of constraints (at an informational and behavioural level) is quite natural and successful. Finally, it provides grounds for a cognitive science methodology

which promotes the more comparative use of computational models. In one sense, to date, the use of computational models has been a bit half-hearted. Researchers have tended to produce their models to demonstrate that their theories are well-specified. Models are seldom taken "seriously" by being used to make comparative predictions in specific situations. The present work shows that the use of models in this way is feasible, even between models with widely differing architectures, and gives rise to informative empirical tests of various theories. Therefore, we hope that the present work will act as a paradigmatic case for the role of computational models in cognitive science.

AUTHOR NOTES

IAM was originally developed in 1987, in Prolog, by Mark Keane and Mike Brayshaw at the Open University (see Keane & Brayshaw, 1988). The present version retains the spirit of the original program but the algorithm has been completely revised and re-written in CommonLisp by Mark Keane. Tim Ledgeway and Stuart Duff did Experiments 1B and 2B, respectively with Mark Keane. The authors would like to thank Ruth Byrne and anonomous referees for helpful comments on earlier drafts of this paper. Special thanks are due to Barry Smyth (Hitachi Dublin Laboratory, TCD) who helped to write the pseudo code for the IAM algorithm. We would also like to thank Dedre Gentner, Ken Forbus (now at Northwestern University) and the Qualitative Reasoning Group at the University of Illinois for helpful criticisms and a copy of SME and Paul Thagard (University of Waterloo) for providing us with a copy of ACME and a lot of free advice about how to run it.

FOOTNOTES

1 This is a strong comparative test, where we make all the models conform to the same measure. A weaker alternative, is to allow each model to use its own measure and then to look for a difference across different tasks/materials for this model.

2 There is one further qualification that needs to be made about measuring ACME's performance in this way. As Holyoak & Thagard (1989) point out, the number of cycles the network goes through before reaching the optimal mapping can change depending on the initial parameters adopted for it. We have, therefore, tried to use parameters that maximise the speed with which the network reaches the correct mapping (see Appendix A, section 1).

REFERENCES

Anderson, J.R. (1991). Is human cognition adaptive□? *Behavioural & Brain Sciences, 14*, 471-485.

Anderson, J.R. & Thompson, R. (1989). Use of analogy in a production system architecture. In S. Vosniadou & A. Ortony (Eds.), *Similarity and Analogical Reasoning*. (pp. 267-297). Cambridge: Cambridge University Press.

Atwood, M.E., & Polson, P.G. (1976). A process model for water jug problems. *Cognitive Psychology, 8*, 191-216.

Buckley, S. (1979). *Sun Up to Sun Down*. New York: McGraw-Hill.

Burstein, M.H. (1986). Concept formation by incremental analogical reasoning and debugging. In R.S. Michalski, J.G. Carbonell & J.M. Mitchell (Eds.), *Machine Learning II: An Artificial Intelligence Approach* . Los Altos, Calif.: Kaufmann.

Byrne, R.M.J. (1989). Suppressing valid inferences with conditionals. *Cognition, 31*, 61-83.

Carbonell, J.G. (1983). Learning by analogy: Formalizing and generalising plans from past experience. In M.S. Michalski, J.G. Carbonell & T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*. Amsterdam: Springer-Verlag.

Cheng, P., & Holyoak, K.J. (1985). Pragmatic reasoning schemas. *Cognitive Psychology, 17*, 391-416.

Duncker, K. (1945). On problem solving. *Psychological Monographs*, *58* (Whole No. 270).

Eysenck, M.W., & Keane, M.T. (1990). *Cognitive Psychology: A Student's Handbook*. London: Lawrence Erlbaum.

Falkenhainer, B. (1987). An examination of the third stage in the analogy process: Verification-based analogical learning. In *Proceedings of the Tenth*

International Joint Conference on Artificial Intelligence. Los Altos: Morgan Kaufmann.

Falkenhainer, B. (1990). A unified approach to explanation and theory formation. In Shrager, J. & Langley, P. (Eds.), *Computational Models of Scientific Discovery and Theory Formation*. San Mateo, CA: Morgan Kaufmann.

Falkenhainer, B., Forbus, K.D., & Gentner, D. (1986). Structure-mapping engine. *Proceedings of the Annual Conference of the American Association for Artificial Intelligence*.

Falkenhainer, B., Forbus, K.D., & Gentner, D. (1989). Structure-mapping engine. *Artificial Intelligence, 41*, 1-63.

Forbus, K.D. (1983). Personal communication.

Forbus, K.D. & Oblinger, D. (1990). Making SME greedy and pragmatic. *Twelfth Annual Conference of the Cognitive Science Society*. Hillsdale: Erlbaum.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, *7* , 155-170.

Gentner, D. (1989). Mechanisms of analogical learning. In S. Vosniadou & A. Ortony (Eds.), *Similarity and Analogical Reasoning*. (pp. 267-297). Cambridge: Cambridge University Press.

Gentner, D., & Landers, R. (1985). Analogical reminding: A good match is hard to find. *Proceedings of the International Conference on Systems, Man and Cybernetics* , Tucson, Arizona, November.

Gentner, D. & Rattermann, M.J. (in press). The role of similarity in transfer. *Cognitive Psychology*.

Gentner, D. & Toupin, C. (1986). Systematicity and surface similarity in the development of analogy. *Cognitive Science*, *10*, 227-300.

Gick, M.L., & Holyoak, K.J. (1980). Analogical problem solving. *Cognitive Psychology*, *12*, 306-355.

Gick, M.L., & Holyoak, K.J. (1983). Schema induction in analogical transfer. *Cognitive Psychology*, *15*, 1-38.

Holyoak, K.J. (1985). The pragmatics of analogical transfer. *The Psychology of Learning and Motivation*, *19* , 59-87.

Holyoak, K.J. & Koh, K. (1987). Surface and structural similarity in analogical transfer. *Memory & Cognition* , *15*, 332-340.

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

Johnson-Laird, P.N., & Byrne, R.M.J. (1991). *Deduction*. London: Lawrence Erlbaum.

Keane, M.T. (1985). On drawing analogies when solving problems: A theory and test of solution generation in an analogical problem solving task. *British Journal of Psychology*, *76* , 449-458.

Keane, M.T. (1987). On retrieving analogues when solving problems. *Quarterly Journal of Experimental Psychology*, *39A* , 29-41.

Keane, M.T. (1988a). *Analogical Problem Solving*. Chichester: Ellis Horwood (Simon & Schuster in N.America).

Keane, M. (1988b). Analogical Mechanisms. *Artificial Intelligence Review*, *2(4)*, 229-251.

Keane, M.T. (1990). Incremental analogising: Theory and model. In K.J. Gilhooly, M.T. Keane, R. Logie & G. Erdos (Eds), *Lines of Thinking: Reflections on the Psychology of Thought*. Vol. 1. Chichester: John Wiley.

Keane, M.T. (1991). The Role of Background Knowledge in Analogical Mapping. *CS-TCD-TR*, May.

Keane, M.T. (1993). IAM àlà Lisp: Description and Examples. *CS-TCD-TR*, July.

Keane, M.T., & Brayshaw, M. (1988). The Incremental Analogy Machine: A computational model of analogy. In D. Sleeman (Ed.), *Third European*

Working Session on Machine Learning. London: Pitman/San Mateo, Calif.: Morgan Kaufmann.

Keane, M.T, Ledgeway, T. & Duff, S. (1991). Constraints on analogical mapping. In K.J. Hammond & D. Gentner (Eds.), *Thirteenth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum.

Koestler, A. (1964). *The Act of Creation* . London: Picador.

Kotovsky, K., Hayes, J.R., & Simon, H.A. (1985). □Why are some problems□?: Evidence form the Tower of Hanoi. *Cognitive Psychology, 17*, 248-294.

Marr, D. (1982). *Vision*. San Francisco: Freeman.

Mayer, R.E. & Bromage, B.K. (1980). Different recall protocols for technical texts due to advance organizers. *Journal of Educational Psychology* , *72* , 209-225.

Newell, A. (1990). *Unified Theories of Cognition*. Harvard: Harvard University Press.

Novick, L.R. (1988). Analogical transfer, problem similarity, and expertise. *Journal of Experimental Psychology: Learning, Memory & Cognition, 14*, 510-520.

Novick, L.R. & Holyoak, K.J. (1991). Mathematical problem solving by analogy. *Journal of Experimental Psychology: Learning, Memory and Cognition*, *17*, 398-415.

Palmer, S.E. (1989). Levels of description in information processing theories of analogy. In S. Vosniadou & A. Ortony (Eds.), *Similarity and Analogical Reasoning*. (pp. 267-297). Cambridge: Cambridge University Press.

Ross, B.H. (1987). This is like that: The use of earlier problems and the separation of similarity effects. *Journal of Experimental Psychology: Learning, Memory & Cognition, 13*, 629-639.

Skorstadt, J., Falkenhainer, B., and Gentner, D. (1987) Analogical processing; A simulation and empirical corroboration. *Proceedings of AAAI'87*. American Association of Artificial Intelligence.

Thagard, P, Holyoak, K.J., Nelson, G. & Gochfeld, D. (1990). Analogue retrieval by constraint satisfaction. *Artificial Intelligence, 46*, 259-310.

Wharton, C.M., Holyoak, K.J., Downing, P.E., Lange, T.E. & Wickens, T.D. (1991). Retrieval competition in memory for analogies. In K.J. Hammond & D. Gentner (Eds.),*Thirteenth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum.

Wharton, C.M., Holyoak, K.J., Downing, P.E., Lange, T.E. & Wickens, T.D. (1992). The story with reminding: Memory retrieval is influenced by analogical similarity. In J.K. Kruschke (Ed.), *Fourteenth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum.

APPENDIX A

Implementational Details of Models

In order to run the different models in a comparative experiment certain implementational issues must be resolved about them. This appendix lays out the different modifications and versions to ACME and SME used in Experiments 1A and 2A.

1. ACME's Parameters in Tests

The critical measure we used for ACME was the number of cycles the network went through before reaching the correct mapping. This varies depending on the parameters adopted for the network. In Holyoak and Thagard's (1989) tests they used a version of the Grossberg activation rule and the following parameter values:

Decay: 0.1

Excitation: 0.1

Inhibition: -0.2

Threshold for output from units: 0.0

Minimum activation: -0.99

Maximum activation: 0.99

Asymptote criterion: 0.001

Concept inhibition: -0.2

Object inhibition: -0.2

Proposition inhibition: -0.2

Similarity of identical predicates: 0.1

Holyoak & Thagard also point out that decay rates from .001 to .2 work well with all of the examples they reported in their paper, with higher values producing a faster rate of settling. However, the rates of settling and of reaching the correct

mapping do not necessarily covary with changes in the decay value. At different decay values, a network can settle at the same cycle, while the number of cycles to success continue to drop. In line with Holyoak & Thagard's suggestions we adopted the highest decay rate they used (i.e., .2) in order to be consistent with their tests of other examples.

Similarly, Holyoak & Thagard report that higher values of excitation lead to faster settling (values from .01 to .12 work on all examples). But, excitation values higher than .12 tend to be disruptive. In our tests, we found that increasing the excitation values, when the decay rate was .2, did not have a significant impact on reducing the number of cycles to the correct solution. So, throughout all the tests reported in this paper the only change to Holyoak & Thagard's parameters was to change the decay parameter to .2.

2. ACME's Similarity Tests Used Synonyms

In ACME's similarity component predicates can be treated using different degrees of similarity; for example, they can be treated as identical (in which case they are given a similarity score of 1) or as synonyms (in which case they get a score of .8). In the tests we performed on similarity we assumed that all of our similar attributes were synonyms. We examined a few variants on similarity scores, treating some attributes as being identical rather than similar, but found that it did not change the character of the results.

3. Using SME for Tests

In all our tests we used version 2E of SME. SME is run with its "free-for-all" match rules which allows any predicate to be matched to any other predicate. When SME is run with the "analogy" match rules, it is a strict instantiation of Gentner's structure-mapping theory.

4. IAM used Configuration

IAM has a Rule Setup file which lays out all the criteria, match rule and match constraints used during a particular comparison. So, different configurations of criteria and rules can be loaded into IAM. The configuration for the attribute mapping problem were as follows:

```
seed-gp-choice-criteria:
        choose-pragmatic-groups
        choose-highest-order-gp
        choose-biggest-gp
        choose-first-gp

seed-ele-choice-criteria:
        remove-tainted-ele
        remove-higher-order-ele
        choose-pragmatic-ele
        choose-most-args-ele
        choose-first-ele)

match-rules:
        both-objects?
        both-elements?          ; for other examples this would be
                                ; both-objects? both-functions? same-type-&-functor?
match-constraints:
        favour-pragmatic
        favour-similar
        favour-structural
        favour-first

preferences:    map all         ; normally some
seed-only:      nil             ; normally      t
```

APPENDIX B THE IAM ALGORITHM (PSEUDOCODE)*

---

initialise

Begin-Definition initialise

base-domain	<--	base predicates and objects
target-domain	<--	target predicates and objects
pragmatically-important-list	<--	matches that are pragmatically-important
used-seed-elements	<--	()
known-mappings	<--	()
map-other-groups	<--	() or True

End-Definition initialise

select-seed-group

Begin-Definition select-seed-group

base-group-list-0	<--	groups of inter-connected predicates in base-domain
base-group-list	<--	sort base-group-list-0 in terms of pragmatic-importance, higher-order connectivity and then by number of predicates
seed-group	<--	pop (base-group-list)

End-Definition select-seed-group

find-seed-match

Begin-Definition find-seed-match

seed-element-list-1	<--	seed-group
seed-element-list-2	<--	remove elements of seed-element-list-1 that appear in used-seed-elements
seed-element-list-3	<--	remove all higher-order elements (i.e., elements that take elements as arguments) from seed-element-list-2
seed-element-list-4	<--	sort seed-element-list-3 in ascending order of pragmatic importance
seed-element-list	<--	sort seed-element-list-4 in terms of number of object arguments
seed-element	<--	pop (seed-element-list)
poss-seed-matches	<--	apply-match-rules(seed-element, target-domain)
seed-match	<--	apply-constraints(poss-seed-matches)

push(seed-match, known-mappings)

End-Definition find-seed-match

find-isomorphic-group-matches

Begin-Definition find-isomorphic-group-matches

```
apply-match-rules(seed-group , target-domain)
Begin-Definition  apply-match-rules
    legal-matches          <--       ()
    For bi  ε  seed-group
        For  tj  ε  target-domain
            If  ∃M a match rule such that M(bi , tj ) = True
                then push ( (bi  tj) , legal-matches)
            End-If
        End-For
    End-For
End-Definition  apply-match-rules

find-ambiguous-matches(legal-matches)
Begin-Definition  find-ambiguous-matches
    ambiguous-matches-list <--    ()
    For bi  ε  seed-group
        ambiguous-matches-sublist  <--   ()
        For  lmi  ε  legal-matches
            If (bi = base-element(lmi ) AND
              lmi  ∉  ambiguous-matches-sublist)
                then push ( lmi, ambiguous-matches-sublist)
            End-if
        End-For
        push(ambiguous-matches-sublist, ambiguous-matches-list)
    End-For
End-Definition  find-ambiguous-matches
```

```
apply-constraints(ambiguous-matches-list)
Begin-Definition  apply-constraints

pragmatic-constraint(ambiguous-matches-list)
Begin-Definition  pragmatic-constraint
        ambiguous-matches-list-1    <--        ambiguous-matches-list
        If  ambiguous-matches-list = ()
             then  ambiguous-matches-list-1
        Else-if  ambiguous-matches-list ≠ ()
            then
            For ambig-sublist_i ε  ambiguous-matches-list
                    For match_i  ε  ambig-sublist_i
                            if match_i  ε  pragmatically-important-list
                                    then     push(match_i , known-mappings)
                                             remove(ambig-sublist_i,ambiguous-matches-list-1)
                            End-if
                    End-for
            End-for
        End-if
End-Definition  pragmatic-constraint
```

```
similarity-constraint(ambiguous-matches-list-1)
Begin-Definition  similarity-constraint
    ambiguous-matches-list-2 = ambiguous-matches-list-1
    If  ambiguous-matches-list-1 = ()
        then   ambiguous-matches-list-2
    Else-if ambiguous-matches-list-1 ≠ ()
        For ambig-sublist_i ε  ambiguous-matches-list-1
            ambig-sublist-1 <--   sort ambig-sublist_i  in terms of highest
                                  match similarity score
            best-match      <--   pop(ambig-sublist-1)
            best-score      <--   similarity-score(best-match)
            ambig-sublist-2 <--   all matches of ambig-sublist-1 with
                                  similarity-score = best-score
            if length(ambig-sublist-2) = 1
                then      push(best-match, known-mappings)
                          remove(ambig-sublist_i, ambiguous-matches-list-2)
            Else-if   length(ambig-sublist-2) > 1
                then      remove(ambig-sublist, ambiguous-matches-list-2)
                          push(ambig-sublist-2, ambiguous-matches-list-2)
            End-If
        End-For
    End-If
End-Definition  similarity-constraint
```

```
structural-constraint(ambiguous-matches-list-2)
Begin-Definition  structural-constraint
    known-mappings-1          <--     known-mappings
    If  (ambiguous-matches-list-2 = ()  OR  known-mappings=() )
        then ambiguous-matches-list-3   <--     ambiguous-matches-list-2
    Else-if  ambiguous-matches-list-2 ≠ ()
        Until  known-mappings = ()
            Mapi     <--     pop(known-mappings)
            ambiguous-matches-list-3        <--     ()
            ; remove-conflicting-matches
            For ambig-sublisti  ε  ambiguous-matches-list-2
                For  new-ambig-sublist  <--     ()
                    For  Mapi'  ε  ambig-sublisti
                        If not(conflicting(Mapi, Mapi')
                            then push(Mapi', new-ambig-sublist)
                        End-If
                    End-For
                    If length(new-ambig-sublist) = 1
                        then    push (first (new-ambig-sublist),
                                    known-mappings)
                                push (first (new-ambig-sublist),
                                    known-mappings-1)
                    Else-if length(new-ambig-sublist) > 1
                        then push(new-ambig-sublist,
                                ambiguous-matches-list-3)
                    End-If
                End-For
            End-For
```

```
; check-support
        ambiguous-matches-list-4    <--    ambiguous-matches-list-3
        known-mappings-1            <--    known-mappings
        supported-matches           <--    find-supported-matches(Map_i)
        For  Match_i  ε  supported-matches
            For ambig-sublist_i  ε  ambiguous-matches-list-3
                If Match_i  ε  ambig-sublist_i
                    then   push(Match_i , known-mappings)
                           push(Match_i , known-mappings-1)
                           remove(ambig-sublist_i, ambiguous-matches-list-4)
                End-If
            End-For
        End-Until
        unsupported-consistent-matches    <--    ambiguous-matches-list-4
    End-if
End-Definition  structural-constraint

favour-first-constraint(ambiguous-matches-list-4)
Begin-Definition  favour-first-constraint
    For ambig-sublist_i  ε  ambiguous-matches-list-4
        push(first(ambig-sublist_i,  known-mappings-1)
    End-For
End-Definition  favour-first-constraint

End-Definition  apply-constraints

End-Definition  find-isomorphic-group-matches
```

find-group-transfers

```
Begin-Definition  find-group-transfers
        known-mappings-2          <--       known-mappings-1
        For  bi  ε  seed-group
            For Mapi  ε  known-mappings-1
                If  bi  ≠  base-element(Mapi )
                        then     New-Map          <--       form-transfer(bi)
                                 push(New-Map, known-mappings-2)
                End-if
            End-For
        End-For
End-Definition   find-group-transfers
```

evaluate-group-mapping

```
Begin-Definition  evaluate-group-mapping
        half-no-of-base-elements            <--     length(seed-group)/2
        evaluation-of-mapping       <--     ()
        If  length(known-mappings-2) >=  half-no-of-base-elements
            then  evaluation-of-mapping     <--     good
        Else-if  length(known-mappings-2) <  half-no-of-base-elements
            then   evaluation-of-mapping    <--     bad
        End-if
        If  evaluation-of-mapping = good
            then   known-mappings-2
        Else-if  evaluation-of-mapping = bad
            then   if poss-seed-matches = ()
                        then    if  seed-element-list = ()
                                    then    if  base-group-list = ()
                                                then  mapping-of-analogy-has-failed
                                            Else-if  base-group-list ≠ ()
                                                ; try another group as the seed-group
                                                then    known-mappings   <--    ()
                                                        seed-group  <--  pop(base-group-list)
                                                        find-seed-match
                                                        find-isomorphic-group-matches
                                                        find-group-transfers
                                                        evaluate-group-mapping
                                            End-if
                                Else-if seed-element-list ≠ ()
                                        ; try  another base-group element as the seed-element
                                        then    known-mappings   <--    ()
```

```
                                        push(seed-element, used-seed-elements)

                                        seed-element  <--
                                                pop (seed-element-list)

                                        poss-seed--matches <--
                                                apply-match-rules(seed-element, target-domain)

                                        seed-match     <--
                                                apply-constraints(poss-seed--matches)

                                        find-isomorphic-group-matches

                                        find-group-transfers

                                        evaluate-group-mapping
                                End-if

                        Else-if  poss-seed-matches ≠ ()

                                ; try an alternative seed-match as the seed-match

                                then    known-mappings  <--  ()

                                        seed-match     <--     pop(poss-seed--matches)

                                        find-isomorphic-group-matches

                                        find-group-transfers

                                        evaluate-group-mapping

                        End-if

                End-if

        End-if

End-Definition  evaluate-group-mapping
```

find-other-group-mappings

```
Begin-Definition  find-other-group-mapping
```

```
    If  map-other-groups = ()
        then  mappings-for-analogy   <--      known-mappings-2
    Else-if  map-other-groups = True
        then  For groupi  ε  base-group-list
                    known-mappings        <--      known-mappings-2
                    find-seed-match
                    find-isomorphic-group-matches
                    find-group-transfers
                    evaluate-group-mapping
              End-For
    End-If
End-Definition   find-other-group-mapping
```

–

*In the above description of the IAM algorithm, the functions *push, pop, first, length, not* are as they are defined in Common Lisp. The *remove* function is slightly different, it takes an item and a list (which has been assigned to a variable) and will update the variable to be the list with the item removed from it. "<--" represents the assignment of a value to a variable; so, "x <-- y" will assign the value y to the variable x.

Other Undefined Functions Mentioned in Algorithm Description

–

conflicting matches - this function takes two matches, match-1 and match-2 and returns T is the two matches violate isomorphism (or if the object matches they support violate isomorphism), otherwise it returns NIL

form-transfer - this function takes a set of mappings between two domains and forms the candidate inferences or transfers suggested by the base representation. Once these transfers are formed in the target they are linked to the appropriate arguments they should have, which may be further transfers or existing elements in the target.

–

APPENDIX C

Domain Representations for IAM Examples

The following are the representations of the different analogy examples reported in this paper.

Example 1: Simple Atom-Solar System Analogy

```
(domain solar-system
        (objects sun planet)
        (predicates
            (weight-difference (sun planet) weight-difference-ss)
            (attracts (sun planet) attracts-sun-planet)
            (revolve-around (planet sun) revolve-planet-sun)
            (and (weight-difference-ss attracts-sun-planet) weight-diff-&-attracts)
            (cause (weight-diff-&-attracts revolve-planet-sun)
                   cause-~weight-diff-&-attracts~-revolve)
            (yellow (sun) yellow-sun)
            (habitable (planet) habitable-planet)
            (oxygen-atmosphere (planet) oxygen-planet)
            (enable (oxygen-planet habitable-planet)  enable-oxygen-habitation)      ))

(domain atom
        (objects nucleus electron)
        (predicates
           (attracts (nucleus electron) attracts-nucleus-electron)
           (revolve-around (electron nucleus) revolve-electron-nucleus)
           (weight-difference (nucleus electron) weight-diff-atom)     ))
```

Example 2: Heat-flow Water-flow Analogy

```
(domain simple-water-flow
        (objects water beaker vial pipe)
       (predicates
            (flow (beaker vial water pipe) water-flow)
            (pressure (beaker) pressure-beaker)
           (pressure (vial) pressure-vial)
            (greater (pressure-beaker pressure-vial) pressure-difference)
           (diameter (beaker) diameter-beaker)
          (diameter (vial) diameter-vial)
            (greater (diameter-beaker diameter-vial) diameter-difference)
            (cause (pressure-difference water-flow) cause-pressure-diff-flow)
           (flat-top (water) flat-top-water)
           (liquid (water) liquid-water)          ))

(domain simple-heat-flow
        (objects coffee ice-cube bar heat)
       (predicates
          (flow (coffee ice-cube heat bar) heat-flow)
          (temperature (coffee) temp-coffee)
          (temperature (ice-cube) temp-ice-cube)
           (greater (temp-coffee temp-ice-cube) temp-difference)
          (flat-top (coffee) flat-top-coffee)
          (liquid (coffee) liquid-coffee)          ))
```

Example 3: SME Solar System / Atom Analogy

```
(domain solar-system
        (objects sun planet)
        (predicates
            (mass (sun) mass-sun)
            (mass (planet) mass-planet)
            (greater (mass-sun mass-planet) mass-difference-ss)
            (attracts (sun planet) attracts-sun-planet)
            (revolve-around (planet sun) revolve-planet-sun)
            (and (mass-difference-ss attracts-sun-planet) mass-diff-&-attracts)
            (cause (mass-diff-&-attracts revolve-planet-sun)
                    cause-~mass-diff-&-attraction~-revolve)
            (temperature (sun) temp-sun)
            (temperature (planet) temp-planet)
            (greater (temp-sun temp-planet) temp-difference)
            (gravity (mass-sun mass-planet) force-gravity)
            (cause (force-gravity attracts-sun-planet) cause-gravity-attracts)  ))

(domain atom
        (objects nucleus electron)
        (predicates
            (mass (nucleus) mass-nucleus)
            (mass (electron) mass-electron)
            (greater (mass-nucleus mass-electron) mass-difference-a)
            (attracts (nucleus electron) attracts-nucleus-electron)
            (revolve-around (electron nucleus) revolve-electron-nucleus)
            (charge (electron) q-electron)
            (charge (nucleus) q-nucleus)
```

(opposite-sign (q-electron q-nucleus) charge-difference)

(cause (charge-difference attracts) cause-charge-diff-attracts)))

Example 4: Attribute Mapping Problem (none-similar)

(map-all-groups)(map-abstract)

(domain men (objects bill steve tom)

(predicates

(tall (bill) tall-bill)

(smart (bill) smart-bill)

(tall (tom) tall-tom)

(timid (tom) timid-tom)

(smart (steve) smart-steve)))

(domain dogs (objects rover fido blackie)

(predicates

(hungry (fido) hungry-fido)

(friendly (blackie) friendly-blackie)

(frisky (blackie) frisky-blackie)

(hungry (rover) hungry-rover)

(friendly (rover) friendly-rover)))

Example 5: Attribute Mapping Problem (all-similar)

(map-all-groups)(map-abstract)

(domain men (objects bill steve tom)

(predicates

(tall (tom) f4)

(timid (tom) f5)

(intelligent (bill) f3)

(tall (bill) f2)

```
        (intelligent (steve) f1)    ))

(domain dogs (objects rover fido blackie)
      (predicates
        (clever (rover) s4)
        (big (rover) s5)
        (clever (fido) s1)
        (shy (blackie) s3)
        (big (blackie) s2)          ))
(similar ((f2 s2 .8)  (f2 s5 .8)  (f3 s1 .8)  (f3 s4 .8)  (f4 s2 .8)  (f4 s5 .8)  (f5 s3 .8)  (f1 s1 .8)  (f1 s4 .8)))
```

Table 1 The Attribute Mapping Problem (Holyoak & Thagard, 1989)

A	B
Bill is tall.	Fido is hungry.
Bill is smart.	Blackie is friendly.
Tom is tall.	Blackie is frisky.
Tom is timid.	Rover is hungry.
Steve is smart.	Rover is friendly.

Table 2 The IAM Algorithm

---

1. *Select Seed Group* - Rank-order groups of connected predicates in the base domain and select the first in the list as the *seed group*

2. *Find Seed Match* - Find a seed match from a selected element in the seed group and note alternative seed matches which may be possible from this element

3. *Find Isomorphic Matches for Group* - Find all the legal matches between the elements of the selected group and the target domain and use serial constraint satisfaction to find a one-to-one set of matches that disambiguates these matches, using pragmatic, similarity and structural constraints

4. *Find Transfers for Group* - Add the transfers or candidate inferences supported by the matches found

5. *Evaluate Group Mapping* - If the resultant mapping is evaluated as being good then continue (step 6), otherwise try an alternative seed match (step 2), or failing that try another group as the seed group (step 1)

6. *Find Other Group Mappings* - If task demands require many groups to be mapped then incrementally map each of the other groups (performing steps 1-5 on each one), such that the mappings formed do not violate the mappings found already (as dictated by the constraints); otherwise just return the mappings found for the seed group

---

Table 3 Summary of IAM's Working Memory After Mapping the Solar System / Atom Analogy

---

base: <DOMAIN-solar-system>

target: <DOMAIN-atom>

seed-group:

(<weight-difference-ss> <attracts-sun-planet> <revolve-planet-sun>
<weight-diff-&-attracts> <cause-~weight-diff-&-attracts~-revolve>)

seed-group-alternatives:

((<habitable-planet> <oxygen-planet> <enable-oxygen-habitation>)
(<yellow-sun>))

seed-element: <weight-difference-ss>

seed-element-alternatives: (<attracts-sun-planet> <revolve-planet-sun>)

seed-match: (<weight-difference-ss> <weight-difference-atom>)

seed-match-alternatives: nil

mappings-found:

((<weight-diff-&-attracts> <*t*weight-diff-&-attracts>)
(<cause-~weight-diff-&-attracts~-revolve> <*t*cause-~weight-diff-&-attracts~-revolve>)
(<revolve-planet-sun> <revolve-electron-nucleus>)
(<attracts-sun-planet> <attracts-nucleus-electron>)
(<weight-difference-ss> <weight-difference-atom>))

obj-mappings-found:

((<planet24> <electron26>) (<sun23> <nucleus25>))

Table 4 Summary of IAM's Working Memory After Mapping the Water Flow / Heat Flow Analogy

---

base: <DOMAIN-simple-water-flow>

target: <DOMAIN-simple-heat-flow>

seed-group:

(<water-flow> <pressure-beaker> <pressure-vial> <pressure-difference> <cause-pressure-diff-flow>)

seed-group-alternatives:

((<diameter-beaker> <diameter-vial> <diameter-difference>)
(<flat-top-water>)
(<liquid-water>))

seed-element: <water-flow>

seed-element-alternatives: (<pressure-beaker> <pressure-vial>)

seed-match: (<water-flow> <heat-flow>)

seed-match-alternatives: nil

mappings-found:

((<cause-pressure-diff-flow> <*t*cause-pressure-diff-flow>)
(<pressure-difference> <temp-difference>)
(<pressure-vial><temp-ice-cube>)
(<pressure-beaker> <temp-coffee>)
(<water-flow> <heat-flow>))

obj-mappings-found:

((<pipe52> <bar55>)
(<water49> <heat56>)
(<vial51> <ice-cube54>)
(<beaker50> <coffee53>))

Table 5 Summary of IAM's Working Memory After Mapping the SME Solar System / Atom Analogy

---

base: <DOMAIN-solar-system>

target: <DOMAIN-atom>

seed-group:

(<mass-sun> <mass-planet> <mass-difference-ss>
<attracts-sun-planet> <revolve-planet-sun> <mass-diff-&-attracts>
<cause-~mass-diff-&-attraction~-revolve>)

seed-group-alternatives:

((<mass-sun> <mass-planet> <attracts-sun-planet> <force-gravity>
<cause-gravity-attracts>)
(<temp-sun> <temp-planet> <temp-difference>))

seed-element: <attracts-sun-planet>

seed-element-alternatives:

(<revolve-planet-sun><mass-sun> <mass-planet>)

seed-match: (<attracts-sun-planet> <attracts-nucleus-electron>)

seed-match-alternatives: nil

mappings-found:

((<mass-diff-&-attracts> <*t*mass-diff-&-attracts>)
(<cause-~mass-diff-&-attraction~-revolve>
<*t*cause-~mass-diff-&-attraction~-revolve>)
(<revolve-planet-sun> <revolve-electron-nucleus>)
(<mass-difference-ss> <mass-difference-a>)
(<mass-sun> <mass-nucleus>)
(<mass-planet> <mass-electron>)
(<attracts-sun-planet> <attracts-nucleus-electron>))

obj-mappings-found:

((<planet6> <electron8>) (<sun5> <nucleus7>))

Table 6 Summary of IAM's Working Memory After Mapping the Attribute-Mapping Problem

---

base: <DOMAIN-men>

target: <DOMAIN-dogs>

seed-group: (<tall-bill>)

seed-group-alternatives:

((<smart-bill>) (<tall-tom>) (<timid-tom>)
(<smart-steve>))

seed-element: <tall-bill>

seed-element-alternatives: nil

seed-match: (<tall-bill> <friendly-rover>)

seed-match-alternatives: nil [1]

mappings-found:

((<smart-steve> <hungry-fido>)
(<timid-tom> <frisky-blackie>)
(<tall-tom> <friendly-blackie>)
(<smart-bill> <hungry-rover>)
(<tall-bill> <friendly-rover>))

obj-mappings-found:

((<steve64> <fido67>) (<tom65> <blackie68>) (<bill63> <rover66>))

---

note 1: Initially this slot has the following alternative seed-matches [(<tall-bill> <hungry-fido>), (<tall-bill> <friendly-blackie>), (<tall-bill> <frisky-blackie>), (<tall-bill> <hungry-rover>), (<tall-bill> <friendly-rover>)] but all of them fail to generate a successful mapping, hence each is tried and rejected until (<tall-bill> <friendly-rover>) is tried.

Table 7 The Versions of the Attribute-Mapping Problem Used in Study 1 (Only B differed in each)

A	*B (None-Similar)*	*B (One-Similar)*	*B(All-Similar)*
Bill is intelligent.	Fido is hungry.	Fido is clever.	Fido is clever.
Bill is tall.	Blackie is friendly.	Blackie is friendly.	Blackie is big.
Tom is timid.	Blackie is frisky.	Blackie is frisky.	Blackie is shy.
Tom is tall.	Rover is hungry.	Rover is clever.	Rover is clever.
Steve is intelligent.	Rover is friendly.	Rover is friendly.	Rover is shy.

Table 8 The Two Versions of the Attribute-Mapping Problem Used in Study 2 (*a singleton)

Singleton-First		*Singleton-Last*	
A	B	A	B
Steve is smart.*	Fido is hungry.*	Bill is smart.	Fido is hungry.*
Bill is tall.	Blackie is friendly.	Bill is tall.	Blackie is friendly.
Bill is smart.	Blackie is frisky.	Tom is timid.	Blackie is frisky.
Tom is tall.	Rover is hungry.	Tom is tall.	Rover is hungry.
Tom is timid.	Rover is friendly.	Steve is smart.*	Rover is friendly.

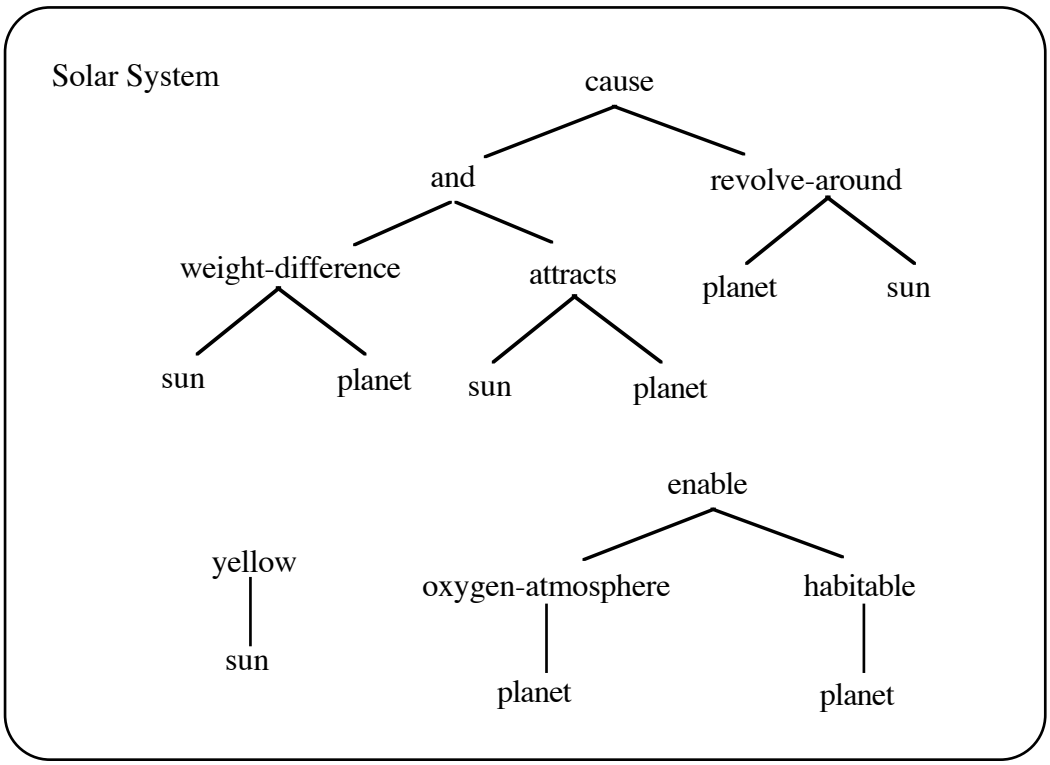
FIGURE CAPTIONS

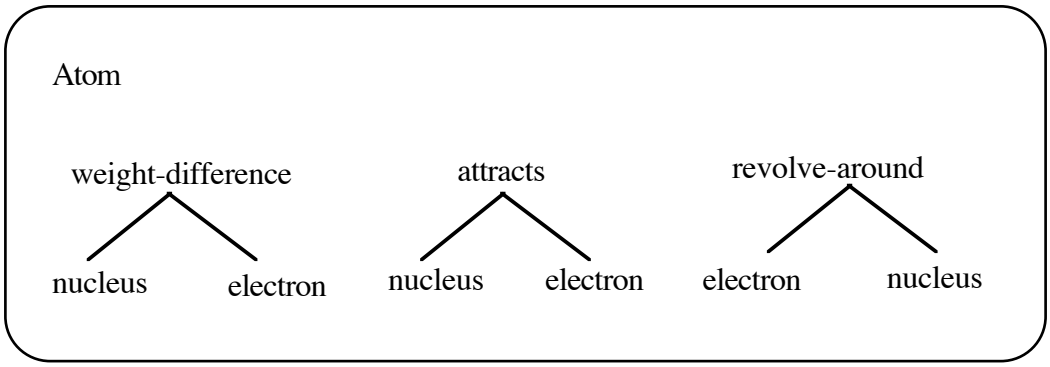
Figure 1 Simple Version of the Solar System and Atom Domains

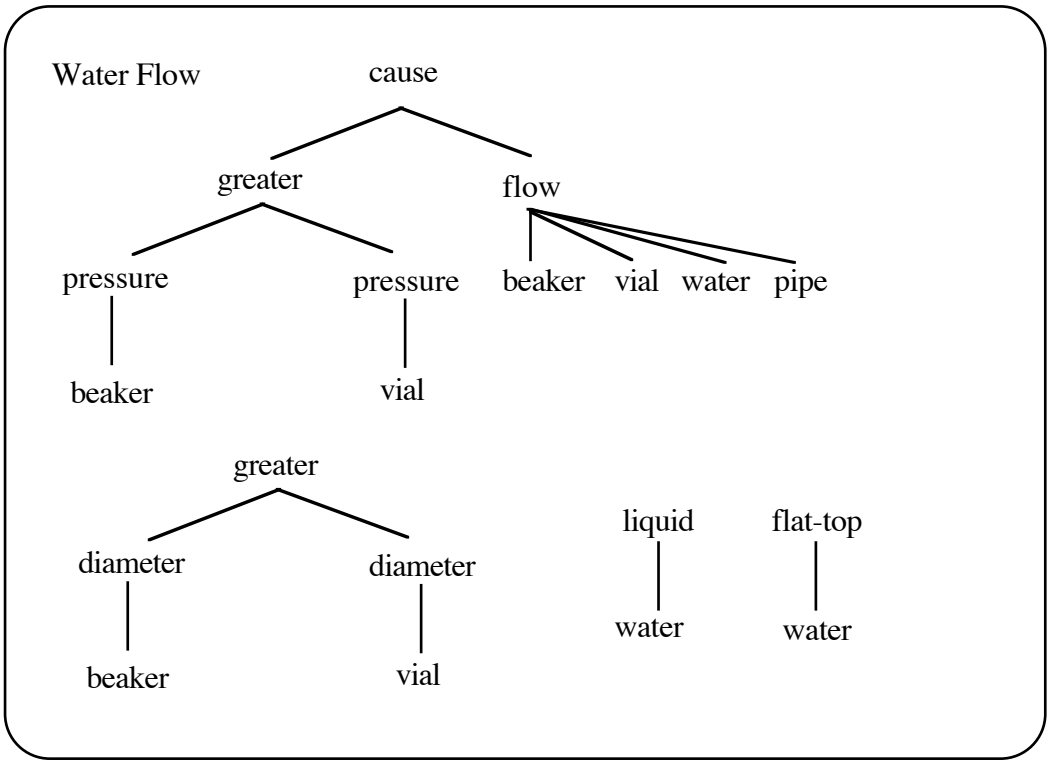
Figure 2 The Water Flow and Heat Flow Domains

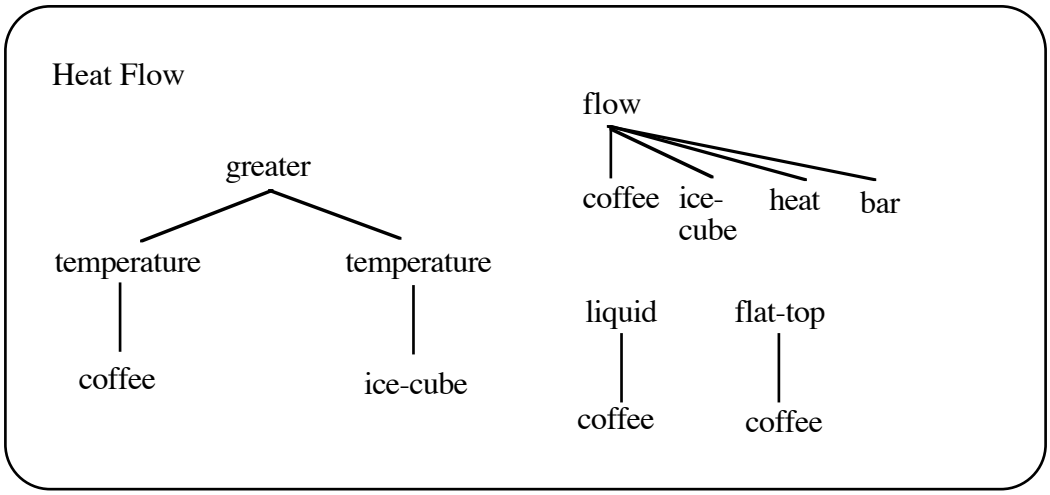
Figure 3 SME Version of the Solar System and Atom Domains

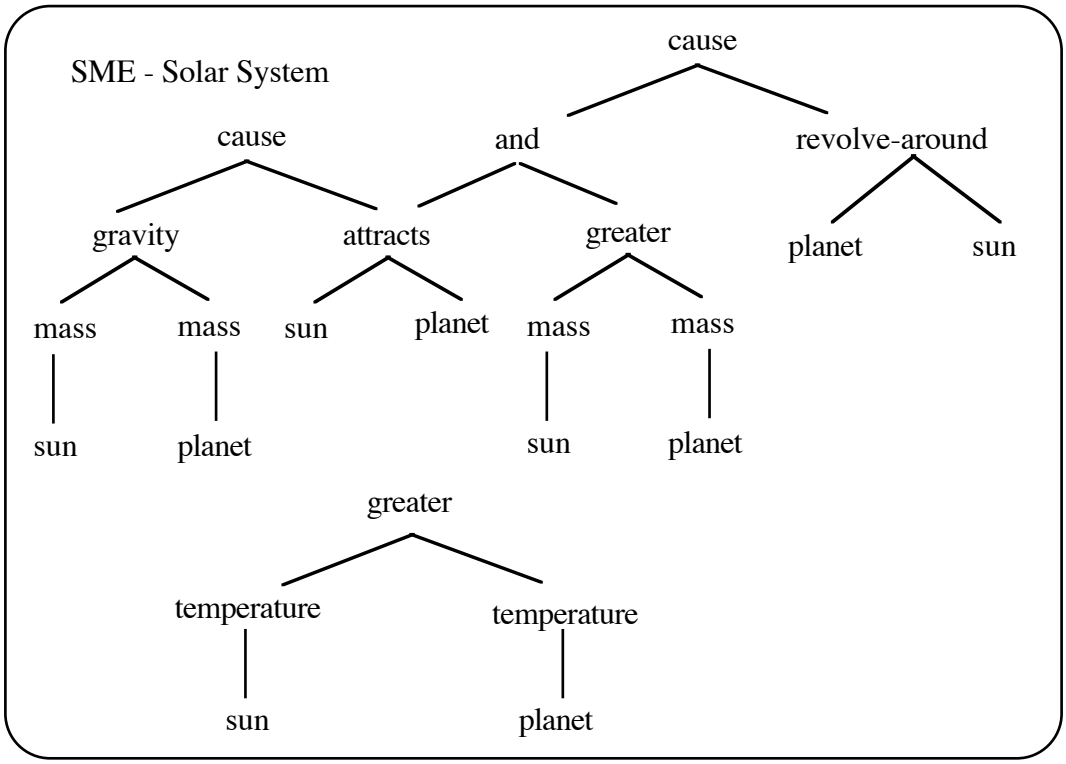
Figure 4 The Mean Number of Mappings Produced by SME, ACME and IAM for the Different Versions of the Attribute-Mapping Problem in Experiment□1A

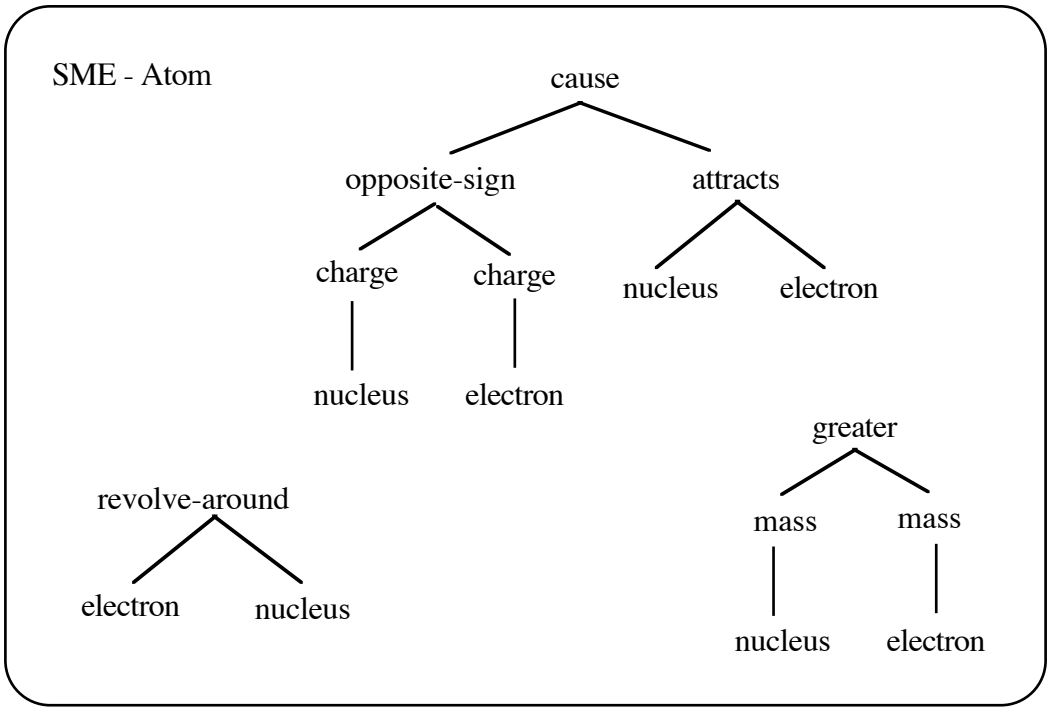

Solar System
cause
and
revolve-around
weight-difference
attracts
planet
sun
sun
planet
sun
planet
enable
yellow
oxygen-atmosphere
habitable
sun
planet
planet

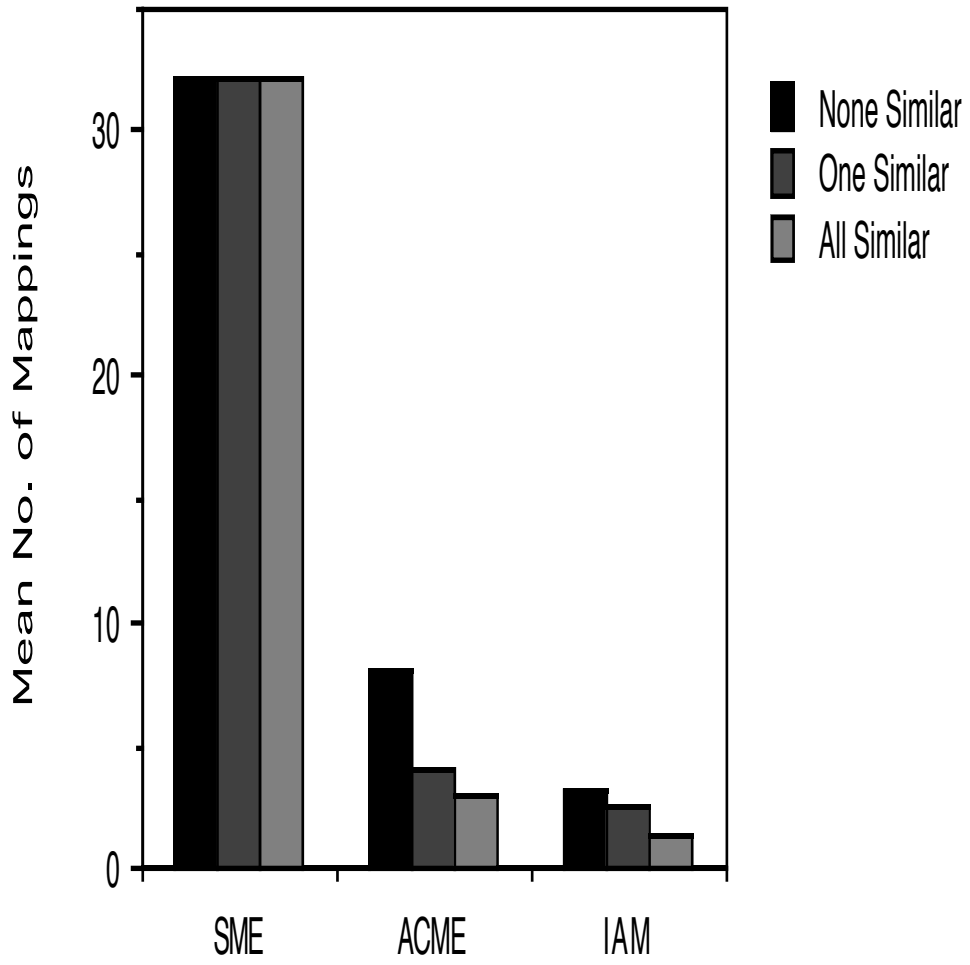

Atom
weight-difference
attracts
revolve-around
nucleus
electron
nucleus
electron
electron
nucleus

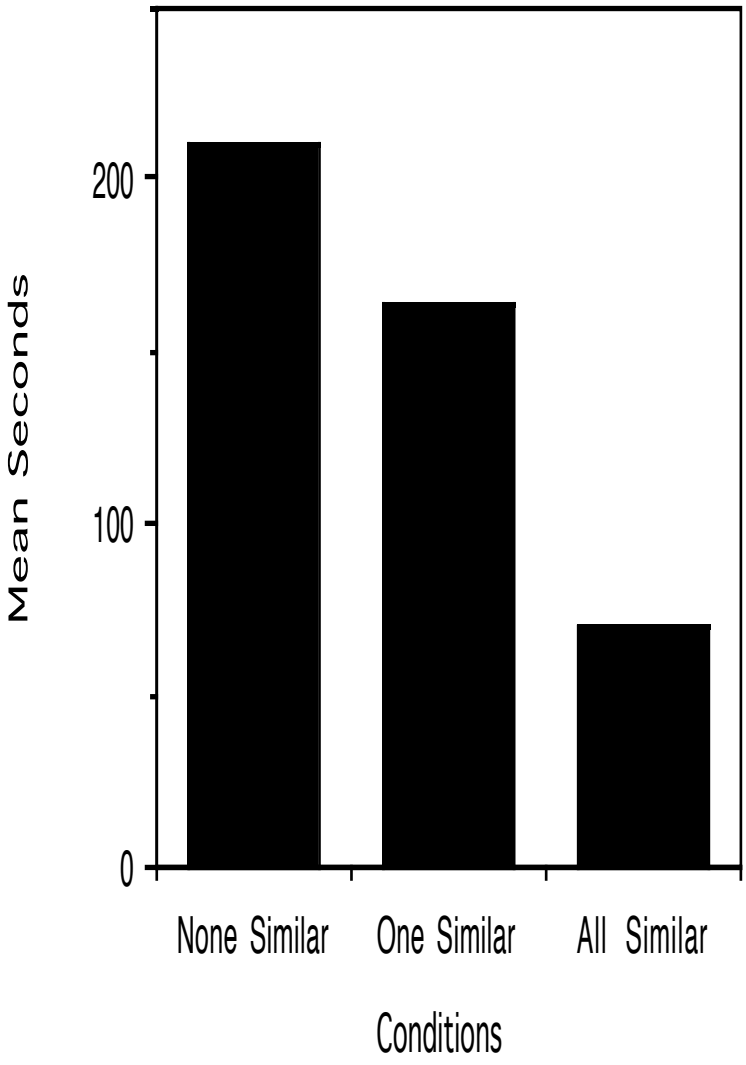

Water Flow
cause
greater
flow
pressure
pressure
beaker
vial
water
pipe
beaker
vial
greater
diameter
diameter
beaker
vial
liquid
flat-top
water
water

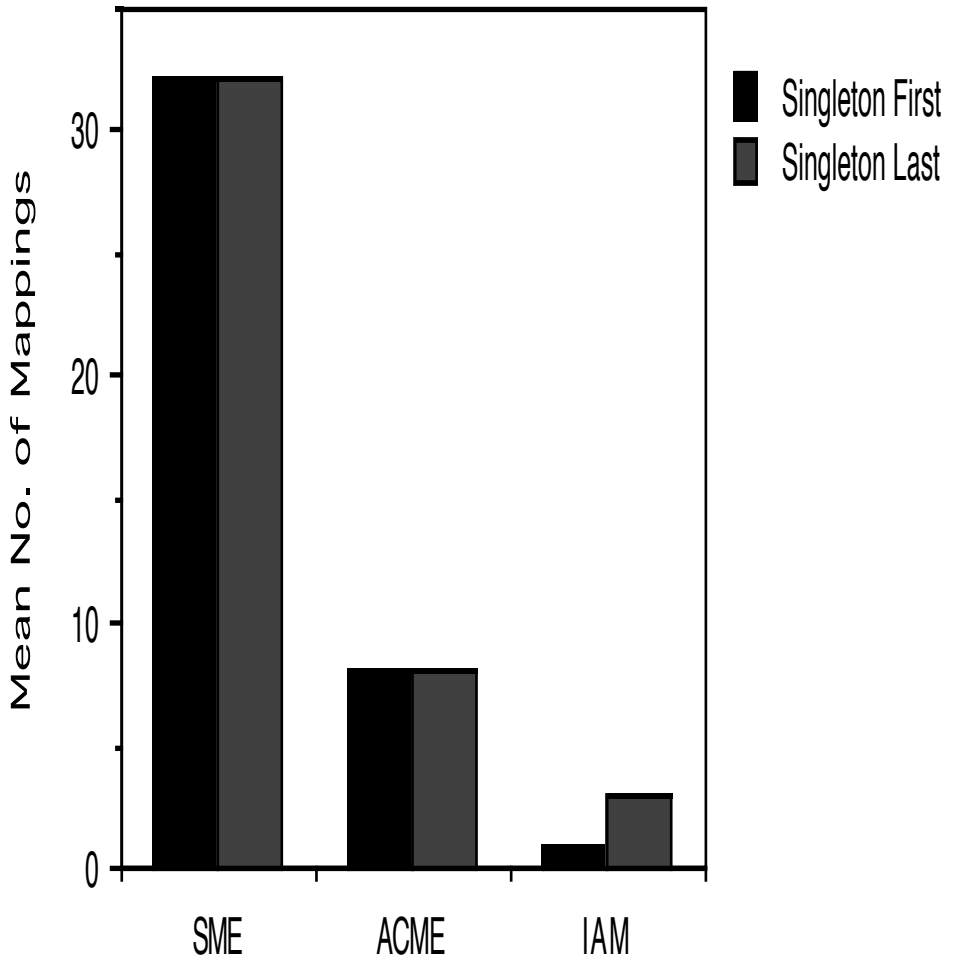

Heat Flow
flow
greater
coffee
ice-cube
heat
bar
temperature
temperature
coffee
ice-cube
liquid
flat-top
coffee
coffee

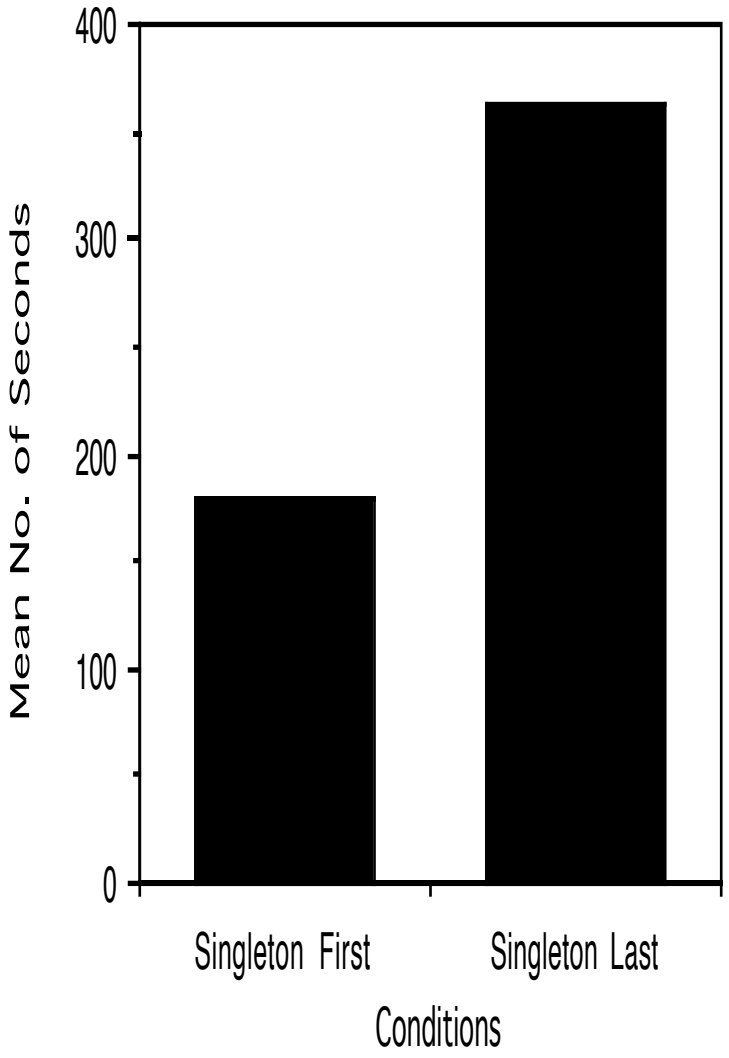

SME - Solar System
cause
cause
and
revolve-around
gravity
attracts
greater
planet
sun
mass
mass
sun
planet
mass
mass
sun
planet
sun
planet
greater
temperature
temperature
sun
planet


SME - Atom
cause
opposite-sign
attracts
charge
charge
nucleus
electron
nucleus
electron
greater
revolve-around
mass
mass
electron
nucleus
nucleus
electron


Mean No. of Mappings
30
20
10
0
SME
ACME
IAM
None Similar
One Similar
All Similar


Mean Seconds
200
100
0
None Similar
One Similar
All Similar
Conditions


Mean No. of Mappings
30
20
10
0
SME
ACME
IAM
Singleton First
Singleton Last


400
300
200
100
0
Mean No. of Seconds
Singleton First
Singleton Last
Conditions