Analogical Retrieval

Dervla O'Keeffe and Fintan Costello
University College Dublin

Technical Report UCD-CSI-2007-11

November 29, 2007

Abstract

We observe that thus far all computational models of analogy have modelled memory as a set of disjoint, encapsulated, domains. As there does not appear to be any psychological evidence for modelling memory in this way, we suggest that a more realistic model of analogy could be constructed if memory was modelled as one large data structure. We argue that the retrieval sub-process of analogy may not be independent of the mapping sub-process, and that both processes may well be governed by structural similarity. We describe a computational model of analogy which incorporates these three ideas; it models mapping and retrieval together, uses structural similarity to govern matching, and models memory as one large data structure. Retrieval in this system corresponds to the searching of the data structure for analogical matches to a supplied probe. We suggest a practical and efficient algorithm for such retrieval.

1 Analogy

Analogy is the process of understanding something new in terms of something familiar [GH97]. The new domain is called the *target analog*, and the familiar domain is called the *source analog* or the *base analog*. An analogy *maps* elements from the source domain to elements of the target domain, although elements of both domains may remain unmapped.

A classic example of an analogy is the analogy between the atom and the solar system. There are many reasons for thinking that the atom "is like" the solar system: the electrons orbit the nucleus, and the planets orbit the sun; the mass of the nucleus is greater than the mass of the electrons, and the mass of the sun is greater than the mass of the planets; the positive charge of the nucleus attracts the negatively charged electrons, and gravity attracts the planets to the sun. Certain elements may be put into correspondence between the two *domains*: electrons and planets; the sun and the nucleus; the sun's gravity and the opposing charges of the electrons and the nucleus.

Analogy may be used as both an aid to memory, and to understanding. Conjectures about unfamiliar domains may also be triggered by analogy. These conjectures are often referred to as *candidate inferences*.

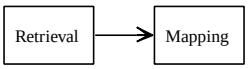

Figure 1: Serial: Retrieval *then* Mapping

1.1 The Sub-Processes of Analogy

Various authors have split analogy into stages or sub-processes, with many authors adopting their own naming conventions for these sub-processes. Hall [Hal89] identifies four sub-processes; *recognition*, *elaboration*, *evaluation*, and *consolidation*. Falkenhainer et al. [FFG89] identify three sub-processes; *access*, *mapping and inference*, *evaluation and use*. In spite of differing naming conventions, most authors have agreed at least that a source must be retrieved from memory, a mapping performed between the retrieved source and the target, candidate inferences suggested, and candidate inferences evaluated. This paper uses the nomenclature of Kokinov and French [KF03], which identifies six sub-processes; *representation building*, *retrieval*, *mapping*, *transfer*, *evauation*, and *learning*. The retrieval and mapping sub-processes are the most relevant to this paper.

1.1.1 Ordering of the Sub-Processes

It was originally thought that these six sub-processes may occur sequentially, one after the other. This is called the *modular view*. Evidence, however, has been presented in favour of what is referred to as the *interactionist view*. This view is that all the sub-processes of analogy run in parallel and can interact with each other. Most of the discussion regarding the order of the sub-processes in analogy has concerned the retrieval and mapping stages.

Kokinov and Petrov [KP01] refer to an unpublished experiment carried out by Ross and Sofk in 1986 where they found that reminding is "a slow and gradual rather than an instantaneous process and that it runs in parallel and interacts with mapping". Petrov and Kokinov [PK98] describe a simulation they carried out using a computational model of analogy called AMBR[1]. The AMBR system can be set up to run retrieval and mapping sequentially as separate sub-processes, or to run the sub-processes in parallel (its usual mode of operation). This ability provided an opportunity for comparing the consequences of the modular and interactionist views with regard to the mapping and retrieval sub-processes. A simulation comparing the two modes of operation resulted in the finding that it *is* possible for the mapping sub-process to influence the retrieval sub-process in order to find an appropriate source. An appropriate source was retrieved when the sub-processes were run in parallel, as illustrated in Figure 2. When the two sub-processes were run sequentially, as illustrated in Figure 1, an *inappropriate, superficially similiar*, source was retrieved. This result does as least show how it may be possible for the mapping sub-process to influence the retrieval sub-process.

1.1.2 Retrieval

This sub-process is the equivalent of Hall's *recognition* sub-process and the *access* sub-process of Falkenhainer et al. It involves the retrieval of a source domain from long-

[1] Associative Memory-Based Reasoning

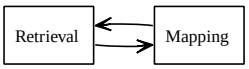

Figure 2: Parallel: Retrieval and Mapping Interacting

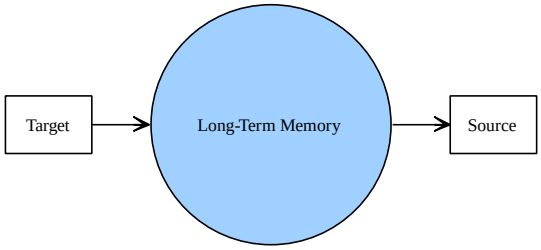

Figure 3: The Retrieval Stage of Analogy

term memory (LTM) in response to a supplied target, as illustrated in Figure 3. The retrieved source domain is analogous to the target domain. The target is sometimes referred to as the *probe* in the context of retrieval. In terms of the solar system and atom analogy retrieval is the identification of the model of the solar system in LTM as analogous to the target representation of the atom.

1.2 Similarity and Analogy

Two domains are deemed to be *structurally similar* if the *relations* that hold between the elements of one domain also hold between the elements of the other domain. The similarity of the objects in one domain to objects in the other domain is not important. Structural similarity *between domains* may be easily examined using graphs of the domains. Two domains are said to be *superficially similar* if the properties *of the elements* of one domain are similar to the properties *of the elements* of the other domain [Kea94] and/or if the domains share similar relations and themes [PK98].

It was initally thought that analogical retrieval (access) was governed by superficial similarity [GH83, GH80, Kea87], while mapping was governed by structural similatrity [GRF93]. The opinion was that when people were provided with some target, the sources they would retrieve from memory in response to this target would be superficially similar to the target. People would however, according to the Structure-Mapping Theory [Gen83], regard those sources retrieved which were most structurally similar to the target as "better" than those which were merely superficially similar [Gen83]. This lead to the hypothesis that the retrieval sub-process returned a collection of domains from memory based on superficial similarity and then somehow chose the one of these that was most structurally similar to the target for use in the mapping sub-process.

As the retrieval sub-process was thought to be most influenced by superficial similarity, and the mapping sub-process most influenced by structural similarity, this was seen as evidence that the two processes were either seperate [Gen93] or seperable. This, in turn, lead to various attempts at the independant computational modelling of retrieval and mapping [PK98]. Examples of mapping models include SME [FFG89] and ACME [HT89], while examples of retrieval models include MAC/FAC [GF91]

and ARCS [THNG90]. The models of retrieval concentrated on evaluating superficial similarity between *disjoint domains*, while the models of mapping concentrated on assessing the degree of structural similarity *between these disjoint domains*.

Further investigation, however, has revealed that the divide between the use of superficial and structural similarity in the sub-processes of retrieval and mapping is not as clear-cut as previously thought. A study described by Catrambone [Cat02] found that *both* superficial and structural may be important for retrieval, and may even be *equally* important under some conditions. Blanchette and Dunbar [BD00] found that structural similarity may even be more important than superficial similarity for retrieval.

1.2.1 Reception Paradigm Vs. Production Paradigm

Blanchette and Dunbar [BD00] have cast doubts over the methodology used in previous experiments [GH80, GRF93, Cat97, Kea87] which aimed to discover the type of similarity most important to the retrieval process. Blanchette and Dunbar label the methodology used in these experiments as using a "reception paradigm" and describe their own series of experiments which consists of two experiments based on their "production paradigm" and one using the reception paradigm. The contents of Blanchette & Dunbar's paper are summarised and discussed here as the paper was instrumental in the formation of the ideas which lead to the model of memory and associated search algorithm described later in this paper.

In the "reception paradigm" experiments subjects are presented with various source stories. Subsequently, they are provided with a cue story, and asked which of the source stories (if any) they are reminded of by this cue story. Gick and Holyoak [GH80] use a similar paradigm for analogical problem-solving. Their subjects were presented with a source story which described a problem and solution. The subjects were then provided with a problem which could be solved by drawing an analogy between the current problem and the problem in the source story. The defining feature of all these "reception paradigm" experiments is that *the subjects were given the sources to be retrived*, and this is what makes the paradigm one of reception.

In Blanchette & Dunbar's experiments, a "production paradigm" is used by allowing subjects to retrieve any source from their LTM in response to a cue story. No sources are provided to the subjects by the experimenters. The sources retrieved by the subjects are documented and then compared with the cue story. The results of the experiment show that the retrieved sources exhibit little superficial similarity to the cue. The retrieved sources and the cue were found to share deep *structural* similarities. These results imply that the apparent importance of superficial similarity in retrieval as documented in the papers using the reception paradigm may be an artifact of the (possibly flawed) methodology used, and that in *real life* structural similarities are the most important type of similarity involved in retrieval.

Blanchette and Dunbar considered the possibility that the difference between the results of the experiments using the differing paradigms may have been due to a difference in the level of expertise of subjects in the domains involved in the experiments. As experts have been shown to be able to use the structure of domains [EC94], it is possible that the domain used in the production paradigm experiments was simply more familiar and well-understood than the domains of the reception paradigm stories. The subjects may have"fallen back" on superficial similarity when dealing with the largerly unfamiliar, unsurprising, and (probably, to some) uninteresting stories used in the reception paradigm experiments as they lacked the expertise in the domains to use structural similarity. The third experiment described by Blanchette & Dunbar aims to investigate this

Figure 4: The Model of Memory assumed in MAC/FAC

possibility by carrying out a reception paradigm experiment using sources similiar to those retrieved using the production paradigm in their first two experiments. In this experiment the sources retrieved were those superficially similiar to the cue, implying that it is the paradigm used which governs the results of the experiments rather than the subjects' expertise, or lack thereof, in the domains.

2 Memory in Models of Analogy

Memory in computational models of analogy has been modelled as a collection of disjoint domains, as shown in Figure 4, rather than as one large structure. We here examine in detail the operation of one computational model of analogical retrieval (MAC/FAC), which we consider representative, with the aim of explaining the assumptions of the model, and discussing some of the possible issues with the assumptions.

Memory is modelled in MAC/FAC [GF91, FGL94] simply as a collection of distinct domains as shown in Figure 4. MAC/FAC stands for Many Are Called/Few Are Chosen. The MAC stage calculates the number of occurrences of each relation in each domain in memory and forms the results into a *content vector*. This vector is then used in much the same way as a similarity metric in case-based reasoning. A content vector is considered to provide some measure of similarity, but does not provide any indictation of *structural* similarity between domains. Thus an estimation of the superficial similarity between the target and each domain in memory may be easily and cheaply calculated using the content vectors. MAC/FAC does just this, by calculating the dot product of the content vector of the target and the content vector of each domain in memory. The MAC stage returns the domain that provides the best dot product match to the target, and any other domains that are within 10% of the best match.

The FAC stage acts on the domains returned by the MAC stage. It passes each of those domains to SME to be structurally matched against the target. The identicality constraint of SME is relaxed by allowing *functions* (in the SME sense of the word) to match if they are embedded in a structure that would otherwise match under the identicality constraint. A numerical threshold is also part of the system. The system always returns the best match from memory. The best match is found using the SES, and the numerical threshold is set to be 10% lower than the SES of this best match. Any comparison between a domain from memory and the target which achieves a score in excess of the threshold is added to the output from the retrieval model. In most cases this 10% lower threshold policy has been observed to return only one result, although it will of course return multiple results when the SES scores for multiple domains are very close.

At the core of the MAC/FAC model is the aforementioned simple view of memory as a collection of distinct domains. This is what allows each "domain" to be treated in the same way as a case in case-based reasoning. Contemporary theories of memory however do not posit that memory is composed of distinct domains as assumed in

MAC/FAC. The MAC/FAC model of memory therefore appears to be a simplification without any phychological basis, but with obvious computational benefits. MAC/FAC is not the only model to model memory in this way, in fact all models of analogy that include a model of LTM use a similar model. The use of this model of memory defines the nature of the retrieval algorithm used by MAC/FAC. Kokinov and Petrov [KP01] have identified this "encapsulated centralized and frozen" model of memory as "at least questionable", but due to the consequences of the model of memory on the entire retrieval algorithm, and the possibly closely-coupled nature of retrieval and mapping, the consequences of using this model may be more far-reaching than previously thought.

Gentner and Forbus [GF91] have as their stated goal the production a model capable of reproducing the retrieval patterns found in psychological data [GL85] [GRF93] [RG87] to which they refer. To this end, they endeavour to construct a system based on the assumption that superficial similarities are used for retrieval, while structural similarities are used for mapping. As discussed in Section 1.2 this may not be a valid assumption.

That MAC/FAC actually uses SME, a *mapping* engine, as part of what is a computational model of retrieval provides further evidence for the argument that it may not be possible to model retrieval in isolation (i.e. without also modelling mapping).

3 An Alternative Model of Memory

People generally retrieve information from memory very quickly. This suggests that they may not be performing brute-force exhaustive searches of memory. One way that exhaustive searches of memory could be avoided would be if memory was *structured* in such a way as to support rapid searching. The amount of searching through memory *at query time* when retrieving information could be drastically reduced by the introduction and maintenance of certain links or connections between different parts of memory. In human memory it is possible that such connections are introduced and maintained as information is added to memory. The time and effort sacrificed in introducing and maintaining the links would be both worthwhile and justifiable if it allowed memory to be searched in a non-exhaustive manner, resulting in fast retrieval from memory. This situation is reminiscent of that in computer language compilers, where much effort is expended optimising the target language code generated by the compiler as the generated code may be run many many times. Any small optimisation that can be made to the generated code during compilation is very worthwhile as the resulting saving will be made every time the generated code in run.

Two types of computational model of the suggested memory and retrieval mechanism are possible; a static model, and a dynamic model. A dynamic computational model would start with an empty memory, and add data piecemeal, introducing and maintaining the necessary connections as required during the population of memory with data. A static model would start with all the data already in memory, and all the relevant connections for that data already in place.

In a dynamic model, as data would be added to memory gradually and over relatively long periods of time, there is very likely to be plenty of time which could be used for updating these connections. The cost of updating one connection is very low. It is envisioned that these very low cost connection maintenance operations when spread out over time in a dynamic model could be achieved with little, if any, performance cost to other memory operations.

A static computational model could be constructed as a proof-of-concept system;

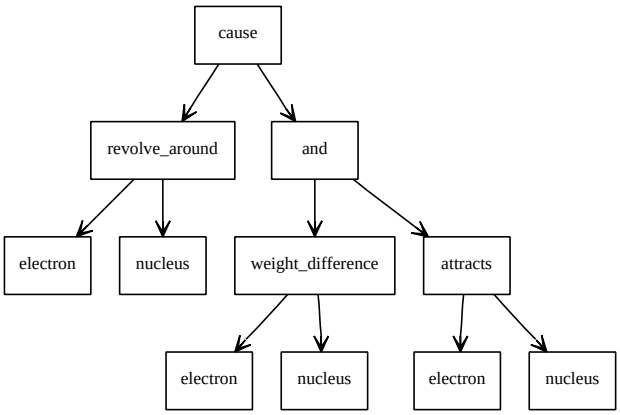

Figure 5: Example Probe *p* in Propositional Form (adapted from [Kea94])

it would be simpler to implement than a dynamic model and would seek just to capture the state of a dynamic model at one moment in time. In this model we assume that we are "pausing" the dynamic model at some moment in time and therefore have the data and connections present before using our search algorithm. As explained above, we are not simply ignoring the costs associated with adding this data to memory and maintaining the appropriate connections to the static model, we are suggesting that although those costs may be considerable if the addition and maintenance of the links for all the data was to be calculated *at the same time*, as in the static model, that these many low cost operations would be spead out over time, avoiding such a performance cost, in a more realistic dynamic model.

Analogous domains share relations. Because relations are important in memory it follows that if we were to add connections to memory, we would like these connections to connect all instances of each relation in memory together. This would allow the modelling of analogical retrieval based on structural similarity, as suggested by Blanchette and Dunbar [BD00]. If such connections were present in memory, it would dramatically reduce the search space for analogical retrieval from memory. We model memory here as one structure, which is not split up into disjoint domains.

We therefore set out to design and construct a static model of memory with these connections, to show that the use of such connections allows for fast and efficient searching of memory. Whether this is what actually occurs in human memory we do not know; here we simply seek to use a static computational model of memory to show that the introduction of such connections provides *one way of* obtaining rapid analogical retrieval from memory.

3.1 The Search Algorithm: Explanation

3.1.1 Relations

In the following example we consider a *connected* probe; there are no unconnected components that we seek to join via "missing" information that we aim to discover via analogical transfer. Any information to be added via analogical transfer would simply enlarge the connected probe. We also assume that all the relations in the probe must occur in memory in the exact same formation as they do in the probe; we call

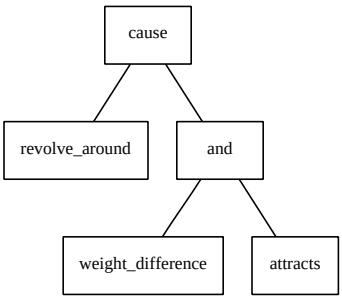

Figure 6: Example Probe *p* in Higher-Order Relational Form (HORF)

this *complete matching*. We concentrate on this simplified situation (connected probe, complete matching) first.

We consider the atom and solar system analogy, assuming the atom is the probe. The propositional representation of the atom domain, adapted from Keane [Kea94], is shown in Figure 5. The corresponding representation in Higher-Order Relational Form (HORF) is illustrated in Figure 6. This form of representation is used because we will focus on the relations in the probe. The object information is still accessible; however, it is not shown when using the HORF. The example probe, *p*, is shown in Figure 6.

We here endeavour to explain the search mechanism of the model in response to such a probe. Initially we ignore the objects and the attributes of the objects in the probe, and concentrate on the relations. This focus on relations is highlighted via the use of the HORF. We use the word "match" as shorthand for "analogous match".

The probe contains five relations: *cause*, *and*, *revolve_around*, *weight_difference*, *attracts*. Assuming connections between identical relations in memory, it is now trivial to locate all instances of each of these relations in memory. To find a match to this probe, we simply need to start by choosing some arbitrary leaf relation, say *weight_difference*, from the probe and find all the instances of that relation in memory. The list *instances(weight_differences)* is the list of instances of *weight_difference* in memory. As we can not have a *complete match* without the relation *weight_difference*, if the relation does not occur in memory (i.e. *instances(weight_difference)* is empty) we know that there can be no match to the probe in memory. If *instances(weight_difference)* is not empty, one or more of the instances on the list may be an instance in a domain analogous to the probe *p*. We therefore would like to investigate further around these instances of *weight_difference* in memory. If memory has 3,000,000 instances of relations, and there are approximately 30,000 distinct relations, then we have just narrowed down the search considerably. How much we have narrowed the search down depends on the frequency of the relation *weight_difference* in memory. There may be 100 (assuming equal frequency of all 30,000 relations), or many more or less (if the frequency of relations in memory is unequal), instances of *weight difference* in the list *instances(weight_difference)*.

We then look to see which relations in the probe are adjacent to the instance of the relation *weight_difference* there. The adjacencies for the probe *p* are shown in Table 7. From Table 7 it can be seen that there is only one relation adjacent to the first relation *weight_difference*; this adjacent relation is *and*. In any complete match analogous to the probe in memory, the relation *and* must be adjacent to the relation *weight_difference*. We therefore look at each element of *instances(weight_difference)* in memory to see

Number	Relation Name	Adjacent Relations
0	revolve_around	cause (1)
1	cause	and, revolve_around (2,0)
2	and	cause, weight_difference, attracts (1,3,4)
3	weight_difference	and (2)
4	attracts	and (2)

Figure 7: The Adjacent Relations for each Relation in *p*

if an instance of the relation *and* is adjacent to any of them. If *and* is not adjacent to *weight_difference* in any of these locations, we know that the two relations do not occur adjacent in memory, and so we can safely conclude that there is no match to the probe *p* in memory.

If we do find one of more locations where *and* occurs adjacent to *weight_diffference*, the list of these instances of is called *instances(weight_difference,and)*. We then want to check and see if any of these instances of *and* adjacent to *weight_difference* is in turn adjacent to the other relations (excluding *weight_difference*) adjacent to *and* in *p*. The only other relations adjacent to *and* are *cause* and *attracts*, so we now look at all the elements of *instances(weight_difference,and)* in memory for any instances of *and* that are adjacent to an instance of both *cause* and *attracts*. This process is repeated until either we have found a match in memory which contains all the relations from *p* in the correct structure, or we fail to find such a match. When this relational matching process has completed, it is trivial to establish the relation-to-relation correspondances between the relations in the probe and the relations in each match.

3.1.2 Objects and Inferences

Thus far we have been solely concerned with relations, both in memory and in the probe. Now that the relational correspondances have been established, the correspondences between objects in the probe and objects in the match(es) in memory must be considered. The arguments of a relation may be either relations or objects, or a mixture of both relations and objects. If the instance of a particular relation in a match in memory has the same number of object arguments as that relation has in the probe and these object arguments take the same position in the arguments of the relation, we hypothesise that the object argument in position *n* of the relation in memory may be put in correspondence with the object argument in position *n* of the relation in the probe. When we have collected our object-matching hypotheses, we must then examine them for consistency. In systems of relations we only want to allow correspondences between objects in a consistent manner; if the first object argument of relation A in the probe is the same as the first object argument of relation B in the probe, we would like the first object argument of the instances of the relations A and B in memory to also share the an object. For example, in the situation where the atom is the probe; because the first object argument of *revolve_around*, *weight_difference*, and *attracts* is the same (electron) we only want to match with any complete relational match found in memory if the instances of those relations in the match also share their first object argument.

The situation is more complicated however when the instance of the relation in memory has a different arity or different number of object arguments to the relation in the probe. There are many ways in which the object arguments from the match and the probe could be put into correspondance. It is likely that the preference for consistent

object mappings could be used in this case to help narrow down the possible object mappings.

With the corresponences between relations and objects decided, a naive inference process could simply transfer one extra layer of information from around the match in memory to around the probe. All of these potentially transferred nodes would be considered as candidate inferences, or conjectures, and could then be evaluated. As we hope to have saved time over a traditional mapping and retrieval analogy model, we hope to be able to spend some time evaluating these inferences. This could possibly be done using data from a knowledge base or by making use of category information from a source such as Wordnet.

3.2 The Search Algorithm: Pseudocode

A series of initialisations and calculations are performed by the algorithm, before the recursive searching via traversal *of the probe* begins. Pseudocode for these initial calculations are shown in Figure 8, along with a post-traversal check for a complete match. The variables `current`, `match`, `previous`, and `saved_current` are all vectors. A complete match has been found if there is an entry in `match` which contains the same number of relations (*not the same number of entries, as one entry may contain one or more relations*) as there are in the probe. The pseudocode algorithm for the recursive function *traverse* is detailed in Figure 9. Note that, for the purposes of the *traverse* function, an active entry is one that matched all the elements of adjacent for the last relation examined from `current`.

4 Implementation

The data in memory is assumed to be a (probably unconnected) sparse graph, and therefore will be implemented using an adjacency list. Figure 10 gives an idea of how the information about the solar system domain would be represented using a type of adjacency list. This figure shows only a small subsection of the graph for the purposes of illustration, and we do not mean to imply via this figure that the information about each domain is stored *contiguously* in memory. The connections between instances of the same relation are stored seperate from the adjacency list (which defines the structure of the graph representing memory).

5 Conclusions

We have discussed how memory has been modelled as a collection of *disjoint domains* in previous computational models of retrieval, and suggested an alternative way of modelling memory as one large structure with connections between instances of the same relation. This alternative way of modelling memory seems as if it might be very powerful for retrieval if we assume that analogical retrieval is governed by structural similarity, as suggested by Blanchette and Dunbar. We also model retrieval and mapping together, instead of trying to separate the sub-processes. We have developed an efficient algorithm for searching such a model of memory for a complete match in response to a connected probe and have presented the pseudocode for this algorithm.

```
search(probe,memory)
{
  result = null;
  current[0] = any_leaf_relation_in_probe()

  //Find and save the instances of current
  //in memory:
  instances = get_instances_in_memory(current);

  //If instances is empty then end in failure.
  if (empty(instances))
  {
    end(fail);
  }

  //Save the partial match:
  match = instances;

  //Search for current adjacent to previous
  //in memory:
  traverse(current,previous)

  //check match to see if a complete match
  //was found during the traversal:
  if (max_count_relations(match)) ==
     (count_relations(probe)) )
  {
    result = match[index_of_max_count_relations(match)];
  }

  return result;
}
```

Figure 8: Pseudocode for the Search Algorithm

```
traverse(current,previous)
{
  for (int c=0; c<length(current); c++)
  {
    adjacent = get_adjacent_in_probe(current[c])

    //if previous[c] is the only element of adjacent
    //then end without failing.
    if (length(adjacent)==1) &&
       (adjacent[0]==previous[c])
    {
      end(not_fail);
    }

    remove(previous,adjacent);

    //update active match entries
    for (int m=0; m<length(match); m++)
    {
      //if that match entry is still active
      if (contains(last(match[m]),current[c]))
      {
        to_add = are_adjacent(match[m],adjacent);

        if (to_add != null)
        {
          match[m].push_back(to_add);
        }
      }
    }

    saved_current[c] = current[c]

    for (int a=0; a<length(adjacent); a++)
    {
      if (adjacent[a] != previous)
      {
        current[a] = adjacent[a];
      }
    }

    previous[c] = saved_current[c]

    traverse(current,previous)
  }
}
```

Figure 9: Pseudocode for the Travserse Function (Part of the Search Algorithm)

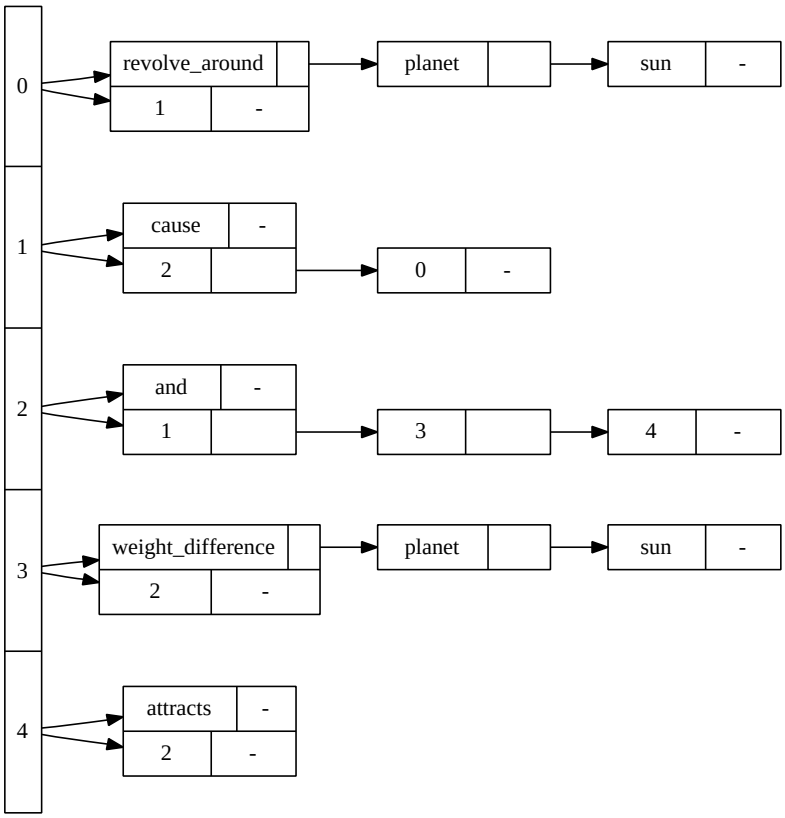

Figure 10: A Part of Memory as Implemented via an Adjacency List

6 Future Work

Future work includes the implementation and testing of our described computational model of memory; first searching for complete matches in response to connected probes. We will work out the complexity of our suggested algorithm with a view to improving it further, and for the purposes of comparision with other models. We must also consider how the performance of our model is to be compared with other models such as: MAC/FAC and SME, ARCS and ACME. We would like to devise some task which we could use to compare the performance and efficiency of the algorithms. This will be challenging as the underlying assumptions of the models greatly differ.

A subsequent version of the model will allow for unconnected probes to be used. In this case, a complete match search for each unconnected component of the probe will be preformed and the results then collated; if matches to two unconnected components are located in memory close to each other (decided by checking if the components are separated in memory by less than some number of relations which could be checked using the adjacency list) then it may be possible that the relations connecting them in memory should be transferred to the probe as candidate inferences. It is possible that this process could be performed incrementally.

Another version of the model may allow for partial matching of the probe in mem-

ory. Although our current model retains the identicality constraint of SME, future versions will aim to allow matching of synonyomous relations. We are considering using WordNet to implement this functionality.

For each relation in memory our current model stores the list of every location that an instance of that relation occurs in memory. We have not assumed any specific ordering of the locations of these instances, although many are possible. We will carry out experiments using a variety of schemes including Most Recently Used (MRU) and Most Frequently Used (MFU) with the aim of further increasing the performance of our computational model.

Eventually we would like to construct a dynamic model of memory and analogy, which would allow complete or partial matching and connected or unconnected probes.

Our current model considers relations in analogous *systems* may be matched using connections in memory which connect instances of the same relation throughout memory. It is possible that the addition of similar connections which connect all instances of the same object attribute throughout memory would be useful for locating *objects* (e.g. a pen is similar to a pencil) which are analogous in memory. We aim to investigate this idea in our future work.

7 Acknowledgments

This research was supported by the FP6 NEST Programme of the European Commission (ANALOGY: Humans the Analogy-Making Species: STREP Contr. No. 029088).

References

[BD00] I. Blanchette and K. Dunbar. How analogies are generated: the roles of structural and superficial similarity. *Memory and Cognition*, 28(1):108–24, 2000.

[Cat97] R. Catrambone. Reinvestigating the effects of surface and structural features on analogical access. In *Proceedings of the Nineteenth Annual Conference of the Cognitive Science Society*, pages 90–95. Erlbaum, 1997.

[Cat02] R. Catrambone. The effects of surface and structural feature matches on the access of story analogs. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 28:318–334, 2002.

[EC94] K. A. Ericsson and N. Charness. Expert performance: Its structure and acquisition. *American Psychologist*, 8:725–747, 1994.

[FFG89] Brian Falkenhainer, Kenneth D. Forbus, and Dedre Gentner. The Structure-Mapping Engine: Algorithm and Examples. *Artificial Intelligence*, 41:1–63, 1989.

[FGL94] K. D. Forbus, D. Gentner, and K. Law. *Cognitive Science*, 19:141–205, 1994.

[Gen83] Dedre Gentner. Structure-mapping: A Theoretical Framework for Analogy. *Cognitive Science*, 7, 1983.

[Gen93] D. Gentner. The mechanisms of analogical learning. In B. G. Buchanan and D. C. Wilkins, editors, *Readings in Knowledge Acquisition and Learning: Automating the Construction and Improvement of Expert Systems*, pages 673–694. Kaufmann, San Mateo, CA, 1993.

[GF91] Dedre Gentner and Kenneth D. Forbus. MAC/FAC: a model of similarity-based retrieval. In *Proceedings of the 13th Cognitive Science Conference*, pages 504–509, Chicago, 1991. Erlbaum, Hillsdale.

[GH80] Mary L. Gick and Keith J. Holyoak. Analogical problem solving. *Cognitive Psychology*, 12:306–355, 1980.

[GH83] Mary L. Gick and Keith J. Holyoak. Schema induction and analogical transfer. *Cognitive Psychology*, 15:1–38, 1983.

[GH97] Dedre Gentner and Keith J. Holyoak. Reasoning and Learning by Analogy: Introduction. *American Psychologist*, 52:32–34, 1997.

[GL85] D. Gentner and R. Landers. Analogical reminding: A good match is hard to find. In *Proceedings of the International Conference on Cybernetics and Society*, pages 607–613, 1985.

[GRF93] D. Gentner, M. J. Rattermann, and K. D. Forbus. The roles of similarity in transfer: separating retrievability from inferential soundness. *Cognitive Psychology*, 25(4):524–575, October 1993.

[Hal89] Rogers P. Hall. Computational approaches to analogical reasoning: A comparative analysis. *Artificial Intelligence*, 39(1):39–120, 1989.

[HT89] Keith J. Holyoak and Paul Thagard. Analogical mapping by constraint satisfaction. *Cognitive Science*, 13:295–355, 1989.

[Kea87] M. Keane. On retrieving analogues when solving problems. *Quarterly Journal of Experimental Psychology*, 39A:29–41, 1987.

[Kea94] Mark T. Keane. Constraints on analogical mapping: A comparison of three models. *Cognitive Science*, 18(3):387–438, 1994.

[KF03] B. Kokinov and R. M. French. *Encyclopedia of Cognitive Science*, chapter Computational Models of Analogy-Making, pages 113–118. Number 1. Nature Publishing Group, 2003.

[KP01] B. Kokinov and A. Petrov. *The Analogical Mind: Perspectives from Cognitive Science*, chapter Integration of memory and reasoning in analogy-making: the AMBR model. The MIT Press, 2001.

[PK98] A. Petrov and B. Kokinov. *Advances in analogy research: Integration of theory and data from the cognitive, computational, and neural sciences*, chapter Mapping and access in analogy-making: Independent or interactive? A Simulation Experiment with AMBR, pages 124–134. NBU Press, 1998.

[RG87] M.J. Ratterman and D. Gentner. Analogy and similarity: Determinants of accessibility and inferential soundness. In *Proceedings of the ninth Annual Meeting of the Cognitive Science Society*, pages 23–34, Seattle, WA, 1987.

[THNG90] Paul Thagard, Keith J. Holyoak, Greg Nelson, and David Gochfeld. Analog retrieval by constraint satisfaction. *Artificial Intelligence*, 46(3):259–310, 1990.