A Data Driven Approach for Making Analogies

Mei Si (sim@rpi.edu)
Department of Cognitive Science, Rensselaer Polytechnic Institute
Troy, NY 12180 USA

Craig Carlson (carlsc2@rpi.edu)
Department of Computer Science, Rensselaer Polytechnic Institute
Troy, NY 12180 USA

Abstract

Making analogies is an important way for people to explain and understand new concepts. Though making analogies is natural for human beings, it is not a trivial task for a dialogue agent. Making analogies requires the agent to establish a correspondence between concepts in two different domains. In this work, we explore a data-driven approach for making analogies automatically. Our proposed approach works with data represented as a flat graphical structure, which can either be designed manually or extracted from Internet data. For a given concept from the base domain, our analogy agent can automatically suggest a corresponding concept from the target domain, and a set of mappings between the relationships each concept has as supporting evidence. We demonstrate the working of this algorithm by both reproducing a classical example of analogy inference and making analogies in new domains generated from DBPedia data.

Keywords: creativity; analogy; intelligent agents

Introduction

This work proposes a data-driven approach for dialogue agents to make analogies between concepts. Analogies describe the comparative relationships between two sets of concepts, i.e. concepts A and B are related in a similar way to how concepts C and D are related. Analogies are widely used in writings and dialogues for explaining new concepts or for making the narration more vivid and more interesting. Typically, one set of concepts is more familiar to the audience than the other. Analogies can, therefore, help the audience understand concepts in unfamiliar domains.

Though making analogies is natural for human beings, it is not a trivial task for dialogue agents. There are at least two challenges associated with this task. One is how to find out and represent what people know about a domain. The other is the computational complexity of establishing mappings between two domains. Both challenges become more significant when the domains the agent tries to make analogies with are not defined explicitly. For example, it is much harder to represent what people know about music genres than linear algebra. There is both more uncertainty and more information in the first case. In addition, there may be multiple good mappings between the concepts in the two domains. For example, one's life can both be mapped to a tree or a road depending on the purpose of making the analogy.

Many cognitive theories have been proposed for explaining how people form analogies (Keane, 2012; Kubose, Holyoak, & Hummel, 2002; Larkey & Love, 2003). Structure-Mapping Theory (SMT) is one of most influential theories for analogies and has been supported by a number of empirical studies using human subjects (Falkenhainer, Forbus, & Gentner, 1989; Gentner, 1983; Gentner & Smith, 2012). According to SMT, an analogical mapping is created by establishing a structural alignment of relationships between two sets of concepts (in two different domains). The closer the structural match is, the more optimal the inferred analogy will be.

One of the main challenges of implementing SMT is its computational complexity. Many researchers have pointed out that the computational time of establishing the mapping is intractable. Heuristics and alternative theories have been developed to form analogies and cut down the computational time. Holyoak and Thagard's Multiconstraint Theory reduces analogy inference to a constraint satisfaction problem (1989). (Forbus & Oblinger, 1990; Grootswagers, 2013; van Rooij, 2008; Wareham, Evans, & van Rooij, 2011) have all worked on creating heuristics for speeding up the structural mapping process.

Another challenge comes from applying SMT or other similar theories to dialogue agents. They typically require a hierarchical relationship structure in the data. For example, the analogy between the solar system and the Rutherford model is a classic example used in computational models of analogy. Figure 1 is taken from (Falkenhainer, Forbus, and Gentner, 1989) for illustrating the solar system domain. For representing this domain, SMT prefers to know not only the relationships between the concepts, e.g. the planet revolves around the sun, the sun's mass is greater than the planet's mass, and the sun attracts the planet but also the relationships among relationships, i.e. the latter two relationships are the cause for the first one. When designing virtual characters with automatically generated or crowd-sourced dialogue content, we often do not have such hierarchical information. The alternative is to design content solely by hand, which creates a huge authoring burden. This challenge is particularly significant when we study

analogy inference not only for understanding human cognition but also for procedurally generating dialogues for virtual characters.

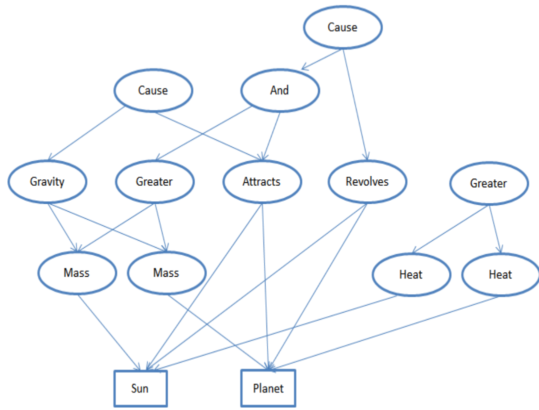

Figure 1: Solar System

In this work, we experiment with loosening up the constraints on input data and using a flat graphical structure for representing the agent's knowledge, i.e. our proposed algorithm only needs to know the relationships between each pair of concepts. Instead of mapping the structures of the relationships, we seek to map the type of relationships from one domain to another. This algorithm is completely data driven; there is no manually designed mapping rule. Our algorithm generates comparable results with SMT when being applied to a classical analogy inference example. We also demonstrate applying the algorithm to larger domains that were automatically generated by crawling data from DBpedia (Bizer, Lehmann, Kobilarov, Auer, Becker, Cyganiak, & Hellmann, 2009).

The results from the analogy-making module will be integrated into an automated narrative agent we developed for making presentations using data gathered through crowdsourcing or from the Internet (Si, Battad, & Carlson, 2016). The success of this project will contribute greatly to creating interesting dialogues and computational creativity. The analogy-making module is self-contained, and the details of the presentation agent are skipped in this paper. In the next sections, we will first describe our input data's format and example domains. Then, we will present our analogy-making algorithm, and results generated by this algorithm, followed by discussions and future work.

Example Domains and Knowledge Representation

We want to use a knowledge representation that is both compatible with structured data, such as the results from querying DBpedia, and is intuitive enough for non-technical authors to manually design and edit the knowledge base. We use a XML format that encodes knowledge as a directed graph. Each concept is represented as a node with a unique ID. The nodes are linked to each other by their relationships, and thus form a directed graph.

We will demonstrate the application of our algorithm using two examples. The first one makes analogies between the solar System and the Rutherford model. This example has been discussed extensively by Gentner et al. (see (Falkenhainer, Forbus, & Gentner, 1989; Gentner, 1983) for more detailed descriptions of the example.) Figure 2 shows the solar system represented as a knowledge graph in our system. Because we don't use hierarchical relationships in our data representation, the higher-level relationships, such as "And" and "Cause" in Figure 1 are lost. However, the relationships between each pair of concepts, such as "Attracts" and "Revolve" are kept. We created a new relationship "More massive" for representing the sun's mass is greater than the planet. Our representation does not use attributes and functions. For attributes, we converted them into a relationship the concept has with another concept, e.g. the sun has a relationship with a concept called Yellow. Currently, we don't have a corresponding encoding for SMT's concept of function in our system.

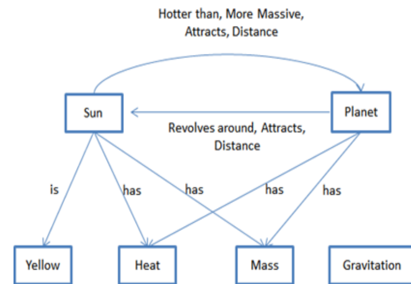

Figure 2: Solar System without Hierarchical Relationship Structure

The first example only contains about a dozen concepts. For examining how well our algorithm scales up, we created a second set of example domains which are much larger. One domain is about music genres, and the other is about programming languages. In this work, we used Wikipedia data as the base of knowledge. The two domains are generated by crawling for information from DBpedia using a tool we developed in the lab. The tool uses one or more DBpedia entries as the starting points and iteratively expanding the graph by including neighbors of the entries that are already in the graph.

Each entry in DBpedia is converted to a node in our knowledge graph and represents a unique concept. The type of link between them in DBpedia becomes the relationship link in our data. These domains are significantly larger than the ones in the first example. The music genres domain contains 999 nodes and 6418 relationships. The programming language domain contains 2589 nodes and 9952 relationships. Figure 3 shows part of the data from the music domain.

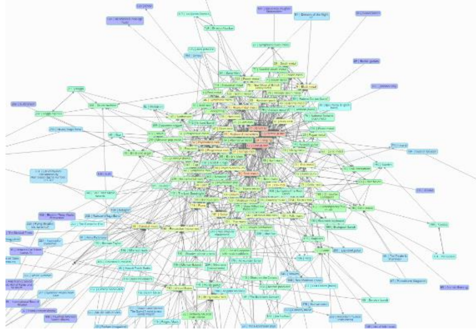

Figure 3: Music Genre Data

Proposed Approach

In this work, our main objective is to provide a dialogue agent or a game character a tool for conducting richer and more interesting dialogues, or for making explanations for a new concept to the user. We hope to help create dialogues that are creative and innovative. Therefore, different from most existing work on analogy inference, we do not necessarily need to find the best analogy we can make given the two domains. Instead, we want to be able to make analogies that are interesting and explainable. Furthermore, the computation needs to complete in a reasonable amount of time.

Our proposed algorithm follows the same philosophy as SMT in that we want to find mappings between concepts and relationships that are supported by mappings between other concepts and relationships. In other words, we want all of the mappings to be consistent with each other. Our algorithm seeks to achieve these goals while working with large and uniformly structured data.

More specifically, instead of trying to map a relationship structure, we seek to map relationship types from one domain to another. These mappings are supported by the similarities in the concepts being linked to, and the relationships related to those concepts. Because our data is large and not manually designed, there may not be a single mapping that is better than all the alternatives. Instead, there may be multiple good candidates. Therefore, instead of looking for the best mapping for all possible hypotheses between the concepts and the relationships in the base and destination domains, we seek to find the best analogy we can make just about a single concept.

Algorithms 1-4 contain the pseudo code for our proposed algorithm. On a high level, it works in two steps: 1) computes a unique index for each concept and each relationship type. This index can be used for comparing the similarities between two concepts or two relationship types; 2) generates and evaluates the hypotheses of mapping a concept in the base domain to a concept in the target domain.

Algorithm 1 Index_Relationship_Type (*domain*):

```
loss, gain, same, diff, index = {} # empty dictionar-
ies
# n: concept; r: relationship; d: destination concept
of r
for each n in domain do
  for each r, d of n do
    # compare n's relationships with d's relation-
  ships
    loss[r] += n.relationship - d.relationship
    gain[r] += d.relationship - n.relationship
    same[r] += Common(n.relationship, d.relationship)
    diff[r] += Difference(n.relationship,
                          d.relationship)
  end for
end for
for each r in domain do
  index[r] = (Jaccard_index(loss[r], gain[r]),
              Jaccard_index(loss[r], same[r]),
              Jaccard_index(loss[r], diff[r]),
              Jaccard_index(gain[r], same[r]),
              Jaccard_index(gain[r], diff[r]),
              Jaccard_index(same[r], diff[r]))
end for
return index
```

Algorithm 1 creates a vector of size 6 for describing each relationship type in a domain. Inspired by the structural mapping process in SMT, here we argue two relationship types are similar if they are always used in similar contexts. Because we don't have the relational structure for providing a context, we operationally defined the context as the origin and the destination concepts linked by the relationship, and we judge the similarity of these two concepts by looking at the differences between the relationships they have and what they share in common. For example, for the relationship "Hotter than" in Figure 2, n is Sun and d is Planet. The *loss* set, in this case, equals to ["More massive", "Is"]. It contains all the relationships the Sun has, but the Planet does not have. If "Hotter than" also links other concepts in the knowledge base, the loss set will be appended every time this relationship is used. The *gain* set contains all the relationships the destination concept has, but the origin concept doesn't. The *same* set contains all the relationships the origin and the destination concepts have in common, and *diff* contains all the relationships that are either in loss or gain. Currently, we are only using the measurements that represent the results of basic set operations, i.e. complement, intersection, symmetric and difference. As part of our future work, we will be looking for other measurements that can help with differentiating the relationship types.

For each relationship type, Algorithm 1 aggregates the results from every time it is used in the domain. The second for-loop converts the information in the four sets, i.e. *loss, gain, same and diff* into a one-dimensional vector by calculating the Jaccard indices between them. We used Jaccard index because it can provide a numerical measurement of the similarities between two sets.

```
Algorithm 2 Get_Node_Index (n,rtype_index):
# rtype_index: the relationship indexes computed by
Algorithm 1
# n: concept; r: relationship
tmpv = (0,0,0,0,0,0) # a zero vector
for each r of n do
  tmpv += rtype_index[r]
end for
return Normalize(tmpv)
```

Based on the relationship indices computed by Algorithm 1, Algorithm 2 returns an index for a concept. Similarly, this index will be used for computing the difference between two concepts in the knowledge network. We used a simple heuristic here: a concept's index is decided by the sum of the index values of all the relationships it has. This value is then normalized to a unit vector.

Finally, Algorithm 3 generates and tests the matching hypotheses, and Algorithm 4 creates a one-to-one mapping between all the relationships a concept *n* has in the base domain to the relationships in the target domain. With this mapping, it is straightforward to find the concept in the target domain that has the most mapping relationships with *n*.

For establishing the mapping, Algorithm 3 generates all the possible hypotheses of mapping the relationships and destinations (*r1, d1*) associated with *n* to another pair of relationship and destination (*r2, d2*) in the destination domain. For evaluating the quality of this mapping, Algorithm 3 looks at both how different the two relationships (*r1* and *r2*) are -- *rdiff*, and how different the two destinations (*d1* and *d2*) are -- *ndiff*. The difference here is given by the cosine similarity between the two vectors. These two difference values are combined. The smaller the overall difference is, the stronger the mapping is. The variable "*hypotheses*" contains the list of all the hypotheses and their strengths.

```
Algorithm 3 Generate_Hypotheses(n, B, T):
# B: base domain
# T: destination domain
hypotheses = [] # hypotheses for mapping
rtype_index = Index_Relationship_Type(B, T)
# index for source node
svec = Get_Node_Index(n, rtype_index)
for each node t in T do
  # index for candidate node
  cvec = Get_Node_Index(t, rtype_index)
  for each r2, d2 of t do
    for each r1, d1 of n do
      rdiff = Cosine_Similarity(rtype_index[r1],
                                rtype_index[r2])
      d1vec = Get_Node_Index(d1, rtype_index)
      diff1 = svec - d1vec
      d2vec = Get_Node_Index(d2, rtype_index)
      diff2 = cvec - d2vec
      ndiff = Cosine_Similarity(diff1, diff2)
      normalized_score = (rdiff + ndiff)/2
      hypotheses.Append(normalized_score,r1,d1,r2,d2)
    end for
  end for
end for
return hypotheses
```

Similar to SMT, we want the mappings to be unambiguous. We used a greedy algorithm to resolve the conflicts in the hypotheses. In case there are hypotheses for both mapping (*r1, d1*) to (*r2, d2*), and to (*r3, d3*) in the destination domain, we simply accept the best -- the mapping that has the highest score -- hypotheses first, and reject any subsequent mappings that intend to revise an existing one (Algorithm 4).

```
Algorithm 4 Map_Relationships(hypotheses):
map = {}
# sort the hypotheses based on normalized_score
hypotheses.Sort_Descending()
for each h in hypotheses do
  # ensure a one-to-one mapping
  if both h.r1 and h.r2 are not mapped then
    # map r1 in base to r2 in destination
    map[r1] = r2
  end if
end for
return map
```

Example Results and Discussion

The proposed algorithm has been applied to making analogies in the two example scenarios described in the Example Domains and Knowledge Representation section.

The Solar System and the Rutherford Model

For making analogies between the solar system and the Rutherford model, we obtained perfect results. Our algorithm correctly generated the mapping between the Nucleus and the Sun, and between the Electron and the Planet. Our algorithm does not produce mapping relationship structure for supporting the analogy. Instead, it produces matching pairs of relationships and destination concepts. Tables 1 and 2 list the evidence for these two mappings.

Table 1: Mappings between Nucleus and Sun

Nucleus	**Sun**
(Attracts, Electron)	(Attracts, Planet)
(Distance, Electron)	(Distance, Planet)
(Has, Electric charge)	(Has, Mass)
(More massive than, Electron)	(Hotter than, Planet)

In Table 1, all the mappings except the last one are straightforward. We checked the intermediate results. The last mapping was an artifact. (Hotter than, Planet) and (More massive than, Planet) received the same score, and the system did not know how to break the tie. All the mappings in Table 2 are consistent with the original example. We are quite encouraged to get this result without the need of using data with hierarchical relationships. We believe the flat concept-relationship structure we designed in Fig-

ure 2 is friendlier to both human designers and automated programs that convert data from other sources.

Table 2: Mappings between Electron and Planet

Electron	Planet
(Attracts, Nucleus)	(Attracts, Sun)
(Distance, Nucleus)	(Distance, Sun)
(Has, Electric charge)	(Has, Mass)
(Revolves around, Nucleus)	(Revolves around, Sun)

Music Genres and Programming Languages

Making analogies between these two domains generated some interesting results, and inspired us with directions for future work. These domains are much larger than the solar system and the Rutherford model. In our evaluation, the typical running time is less than a second on a Lenovo T430 laptop. We will discuss two pieces of example results below.

Table 3: Mapping Relationships between Punk rock and LPC

Punk Rock	LPC
Music fusion genre	Influenced
Stylistic origin	Influenced by
Instrument	Paradigm

In the first example, the system mapped the music genre Punk Rock to the programming language LPC. Because of space limitation, in Table 3 we only list the matching relationship types the system provided for this analogy. Two of these mappings are quite reasonable. By mapping “Stylistic origin” to “Influenced by”, the system provided us supporting evidence such as “the stylistic origin of Punk Rock is Garage Rock, Glam Rock, and Surf Music, just like LPC is influenced by Lisp, Perl, and C.” By mapping “Music fusion genre” to “Influenced”, the system provided corresponding supporting evidence “Celtic Punk is a music fusion genre of Punk Rock, just like LPC influenced Pike.” Intuitively, these two examples make sense. Music genres that are influenced by other music genres have their styles originating from those genres. The inverse works as well; if genre A is a music fusion genre for genre B, then A influenced B. The system was able to equate these relation types without any explicit help.

The mapping from “Instrument” to “Paradigm” isn’t as a clear cut as the other mappings. The evidence provided is “the relationship between Punk rock and Bass guitar or Electric guitar is Instrument, just like the relationship between LPC and Procedural programming and Functional programming is Paradigm.” This assertion isn’t inherently wrong. However, to a human observer this mapping may not seem intuitive enough.

Interestingly, we also asked the system to make an analogy about the programming language Python, and the system responded with Hardcore Punk. Table 4 provides the matching relationships for this analogy.

Most of the relationship mappings in Table 4 are reasonable. For example, we can say “Python influenced F Sharp, Ruby, and Swift just like Black Metal, Thrash Metal, and Industrial Metal are derivatives of Hardcore Punk.”

Table 4: Mapping Relationships between Python and Hardcore Punk

Python	Hardcore Punk
Influenced	Derivative
Influenced by	Stylistic origin
Operating system	Instrument
Paradigm	Format

The most interesting part of this example is the assertion that Python is influenced by Perl in the same way as the stylistic origin of Hardcore Punk is Punk Rock (and hence Hardcore Punk is influenced by Punk Rock). This goes against the previous analogy of Punk Rock being comparable to LPC, since Python is not influenced by LPC. We think this shows the weakness of our approach. Without the hierarchical relationship information which in fact provides a global structure of the data, our algorithm does not do a good job in creating analogies that are globally consistent. However, the analogies are still locally consistent for a given topic because of Algorithm 4.

Another thing to note is the difference in mapping between Instrument and Paradigm. In Table 3, “Instrument” is mapped to “Paradigm”, but in Table 4, “Instrument” is mapped to “Operating system.” LPC does not have a relationship of the type “Operating system”, so no mapping could have been made. Table 4 indicates that “Instrument” is more analogous to “Operating system” than to “Paradigm.” As mentioned before, our system cannot enforce global consistency yet. Realistically, however, it’s hard to say which is truly correct in this case. A similar phenomenon can be observed with “Influenced” and “Music fusion genre” in Table 3. This time in Table 4, “Influenced” is mapped to “Derivative” because the match is better, not because Hardcore Punk lacks that relation type. In Table 4, mappings from “Influenced” to “Music fusion genre” are ignored because a one-to-one mapping of relation types is enforced by the algorithm. Currently, one-to-one mappings must be enforced in order for coherent analogies to be made. However, there are cases when using many-to-one mappings is more suitable. This is especially true when using crowd-sourced data or data from the Internet where sometimes the only real difference in relationship type is semantics (e.g. “Instrument” / “Instruments”).

Future Work

We have planned future work both in the direction of improving our algorithms for finding better mappings and discovering more creative uses of the algorithms.

First of all, we want to address the issue of the agent sometimes creating conflicted mappings between two pairs of concepts. When working with a large data set, exclusively checking all the possible conflictions would be very time-consuming. Instead, we plan to develop a greedy solution. When the agent needs to make a new analogy, it will assume all the relationship mappings it used to support its previous analogies are already true. This way, instead of asking two separate questions of "what is LPC like" and "what is Python like", we are asking the system "If LPC is like Punk Rock, what Python would be like?"

Secondly, we are looking for better ways for indexing the relationships and the concepts. Right now, the semantic information of the relationship types is rarely used. Algorithm 1 only looks at whether they are different or not. We are considering using other semantic tools for helping us to get a direct measure of how close two relationship types are, and even how close two concept descriptions are. This would solve the aforementioned problem caused by the one-to-one mapping restriction. Another consideration in the indexing process is the fact that when dealing with human authored content, there is no guarantee that different contributors will use the same relation type in the same way. Such inconsistencies could throw off the results of Algorithm 1, leading to bad analogies.

Thirdly, many benchmarks have been created for analogy inference, such as (Holyoak & Thagard, 1989). Most of the benchmarks' formats are compatible with SMT and the algorithms derived from it. We will be looking into ways of evaluating our algorithm using a standard benchmark.

Finally, we believe this work has great potential of contributing to creating rich and vivid virtual characters, interesting and interactive stories, and computational creativity. We are interested in finding new and innovative applications of our proposed algorithms in addition to making analogies for a single concept. In particular, we are interested in exploring how these algorithms can be used in creating digital stories. As one of our next steps, we plan to experiment with using this algorithm to learn how a person tells a story or how a good story is constructed and then apply the learning results for telling new stories using data from a different domain.

Acknowledgments

This work was supported in part by the Cognitive and Immersive Systems Laboratory -- a collaboration between the IBM Research and RPI in the IBM Cognitive Horizons Network.

References

Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., & Hellmann, S. (2009). DBpedia-A crystallization point for the Web of Data. *Web Semantics: science, services and agents on the world wide web*, *7*(3), 154-165.

Falkenhainer, B., Forbus, K., & Gentner, D. (1989). The structure-mapping engine: Algorithm and examples. *Artificial Intelligence, 41,* 1–63.

Forbus, K., & Oblinger, D. (1990). Making SME greedy and pragmatic. *Proceedings of the 12th Annual Conference of the Cognitive Science Society* (pp. 61–68).

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science, 7 (2),* 155–170.

Gentner, D., & Smith, L. (2012). Analogical reasoning. In V. Ramachandran (Ed.), *Encyclopedia of human behavior* (2nd ed.) (pp. 130–136). Elsevier; Oxford, UK.

Grootswagers, T. (2013). Having your cake and eating it too: Towards a fast and optimal method for analogy derivation. Master dissertation, Radboud University, The Netherlands.

Holyoak, K., & Thagard, P. (1989). Analogical mapping by constraint satisfaction. *Cognitive Science, 13 (3),* 295–355.

Kline, P. J. (1983). Computing the similarity of structured objects by means of a heuristic search for correspondences. Doctoral dissertation, University of Michigan.

Kubose, T. T., Holyoak, K. J., & Hummel, J. E. (2002). The role of textual coherence in incremental analogical mapping. *Journal of memory and language*, *47*(3), 407-435.

Larkey, L. B., & Love, B. C. (2003). CAB: Connectionist analogy builder. *Cognitive Science, 27(5),* 781-794.

Si, M., Battad, Z. & Carlson, C. (2016) Intertwined storylines with anchor points. *Proceedings of the 9th International Conference on Interactive Digital Storytelling (ICIDS)* (pp 247-257), Los Angeles, CA.

van Rooij, I. (2008). The tractable cognition thesis. *Cognitive science*, *32*(6), 939-984.

Wareham, T., Evans, P., & van Rooij, I. (2011). What does (and doesn't) make analogical problem solving easy? A complexity-theoretic perspective. *The Journal of Problem Solving, 3 (2)*, 30–71.