Solving Semantic Analogy Problems Using ConceptNet

Mika Braginsky and Will Whitney

December 13, 2012

Summary

We implemented a system for solving SAT analogy problems, using ConceptNet. The system attempts to find the relationship between the target pair of words (such as *mason*:*stone*) and each option pair (such as *teacher*:*chalk*, *carpenter*:*wood*, *soldier*:*gun*, *photograph*:*camera*, *book*:*word*), scores the similarity of each option's relationship to the target's relationship, and selects the option with the highest score. On a dataset of 374 questions, it achieves an accuracy rate of 29.7%, with an accuracy rate of 55% on the 99 questions for which it can obtain at least minimal information.

1 Problem Overview

Reasoning by analogy has long been recognized in the cognitive sciences as a key component of cognition, in that it forms the basis for processes such as decision making, object recognition, face perception, problem solving, and creativity. Modeling this type of reasoning and enabling computer systems to make use of it should therefore be an important step in enabling artificial intelligence systems to emulate vital aspects of cognition.

As a way of exploring the more general phenomenon of reasoning by analogy, we implemented a system for solving analogy questions from the SAT (a standardized test used for college admissions). SAT analogy questions are specifically designed to test students' skill in solving problems using analogy-driven logic, so they are a reasonable test base for evaluating the ability of a computer system to reason by analogy.

An SAT analogy question consists of a target pair of words and five option pairs of words. The task is to select the option pair that "best expresses a relationship similar to that expressed in the original pair", as stated in the test's directions. For example:

```
    ostrich:bird
(a) lion:cat
(b) goose:flock
(c) ewe:sheep
(d) cub:bear
(e) primate:monkey
```

To solve the problem, one must determine the relationship between each word pair:

```
    An OSTRICH is a species of BIRD
(a) LION is a species of CAT
(b) FLOCK is a group of GOOSE
(c) EWE is a female SHEEP
(d) CUB is a young BEAR
(e) MONKEY is a sub-group of PRIMATE
```

Solving the problem requires a lot of semantic knowledge about how concepts are related to each other. In this question, the solver needs to not only know what ostriches and birds and lion and cats are, but also how ostriches are related to birds, how lions are realted to cats, etc. This knowledge must be also capture a rather specific level of detail – if the solver only knew that an ostrich is a type of bird, it would conclude that the same relation holds not just between lion and cat, but also between ewe and sheep and between cub and bear.

In general, the solver must make use of extensive information about the relationships between concepts and make inferences about the similarities between relationships. Achieving competence on this task, then, is an indicator of considerable mastery of reasoning abilities and common sense knowledge, both for a human and a computer system.

2 Previous Work

A number of algorithms have been devised to approach this problem (Table 1 shows their accuracy on a common test of 374 questions). Broadly speaking, they fall into two categories: lexicon-based and corpus-based (some are hybrid, a mix of both). The lexicon-based algorithms tend to use information from WordNet, a database of English words that encodes many lexical and some semantic relationships. The corpus-based algorithms use a corpus of questions to train their algorithm for the problem.

While some of the previous algorithms achieve almost human-level performance, they don't faithfully capture the nature of a human approach to solving the problem. Humans doesn't need to be "trained" on hundreds of questions before being able to answer further ones, and humans utilize much more complex and extensive semantic knowledge than can be provided by WordNet.

Reference for algorithm	**Type**	**Correct**
Random guessing	*Random*	*20.0%*
Jiang and Conrath (1997)	Hybrid	27.3%
Lin (1998)	Hybrid	27.3%
Leacock and Chodrow (1998)	Lexicon-based	31.3%
Hirst and St.-Onge (1998)	Lexicon-based	32.1%
Resnik (1995)	Hybrid	33.2%
Turney (2001)	Corpus-based	35.0%
Mangalath et al. (2004)	Corpus-based	42.0%
Veale (2004)	Lexicon-based	43.0%
Bicici and Yuret (2006)	Corpus-based	44.0%
Herdadelen and Baroni (2009)	Corpus-based	44.1%
Turney and Littman (2005)	Corpus-based	47.1%
Turney (2012)	Corpus-based	51.1%
Bollegala et al. (2009)	Corpus-based	51.1%
Turney (2008)	Corpus-based	52.1%
Turney (2006a)	Corpus-based	53.5%
Turney (2006b)	Corpus-based	56.1%
Average US college applicant	*Human*	*57.0%*

Table 1: Results of various algorithms on a common 374 question data set.[1]

[1] `http://aclweb.org/aclwiki/index.php?title=SAT_Analogy_Questions_%28State_of_the_art%29`

3 Approach

Attempting to solve analogy problems in a more plausibly human-like way requires access to a great deal of semantic information. Our approach was to obtain this information using ConceptNet, a semantic network that attempts to encode "common sense" knowledge and other information that computers should know about the world[2].

ConceptNet is a directed graph where the nodes are concepts and the edges are relationships between concepts. Its knowledge is assembled from a variety of sources, such as Wikipedia, Wiktionary, WordNet, and direct human contributions. It contains basic semantic relationships and everyday knowledge, such as:

`dog —IsA→ pet`

`learn —MotivatedByGoal→ know more information`

as well as more specialized cultural and scientific knowledge:

`saxophone —UsedFor→ jazz`

`plutinos —can cross→ Neptune's orbit`

If ConceptNet were identical to the full space of concepts and relationships known to a human, using its knowledge would allow a system to solve analogy questions as well as human. Such a system would determine the relationship between each word pair by finding the path from one word to the other, and then select an answer by picking the option with the same path (relationship) as the target pair.

However, ConceptNet is far from perfect – in trying to find the relationship between two concepts, ConceptNet has both not enough data and noisy data. On one hand, it has not enough data in that it just doesn't know enough about some concepts, such that finding a path between them is impossible. At the same time, its relationships are often very broad, so for some concepts it finds many paths between them. For example, for the concepts "note" and "music", it finds that:

[2]ConceptNet is developed by the Digital Intuition group, MIT Media Lab.

`music` —`have or involve`→ `note`

`note` —`RelatedTo`→ `music`

`music` —`HasProperty`→ `note`

`note` —`in`→ `music`

In light of noisy and insufficient data, our approach was to:

- Find the list of paths for the target pair
- Find the list of paths for each option pair
- For each option, compare the option's list of paths to the target's list of paths. Score their similarity using a scoring function.
- Pick the option with the highest similarity score

The closer that ConceptNet's knowledge is to a human's knowledge, the more similar this algorithm is to a human's problem solving approach.

4 Implementation

Our implementation of this system is written in Python, which was chosen for its large selection of libraries, acceptable speed, flexibility, and widespread use. Flexibility in particular was a key attribute, as graph traversal can quickly become unnecessarily complicated in many more structured languages. Our Python code finds data about the relevant terms on ConceptNet, then traverses the ConceptNet graph to find relationships between the terms. It then examines the relationships of the option pairs and compares them to the relationships of the target pair, as described above, and returns the correct answer.

4.1 ConceptNet Access

In implementing this system, we needed access to the data contained in ConceptNet. ConceptNet offers three options for accessing their data:

1. Download a flat `.json` file containing every concept and every edge
2. Run your own version of the ConceptNet server using Solr and Python
3. Access their online API with `HTTP GET` requests returning `JSON`

As with all graph-search and machine learning projects, performance is very important for this project. The ability to rapidly run tests at scale, analyze the results, and iterate is key. That made speed critical in our analysis of these methods.

We first considered the flat `JSON` method. Upon inspection, however, it quickly became obvious how unwieldy it was; once unzipped, the `JSON` files occupied approximately `8 GB` of hard disk space. While this on its own was not an issue, the fact that it is un-indexed means that in order to access the contained data, we would have to load all `8 GB` into memory before attempting to search. `8 GB`, already a large memory requirement, is made all the more unreasonable by the fact that standard Python in-memory structures occupy several times the space of `JSON`-encoded data.

We then began to consider the second option, running a Solr database and a Python web server. As this is the method used by ConceptNet themselves, this on its surface seemed reasonable, and having a proper database meant that queries would be faster and more flexible than those against the `JSON` files. Once we read further, though, we observed that ConceptNet was actually running their service with two servers, each a quad-core machine with 7.5 GB of RAM, and the database sharded between them. Furthermore, there would be significant implementation time involved even with getting the server up and running.

As such, we came to the third option: the ConceptNet web API. We had originally hesitated to consider this option due to the latencies inherent in an online API. At a minimum, every query requires the round-trip latency to the server location, in this case Virginia, which is typically on the order of a couple of hundred milliseconds. Since we needed to make approximately 3700 queries per run against our full dataset, that meant that every run would include at the very minimum twelve minutes of latency. In practice, this meant that each of those runs actually took about 25 minutes. However, since this was the only option that was technologically feasible, we went forward with it.

In the end we came up with two methods to improve our testing time, parallelization and caching. Our first experiment in these optimizations involved making very highly parallel requests (`O(1 per query)`, or 3700 simultaneous requests) to the ConceptNet API. While this solution was technically acceptable on our end, it rapidly became obvious that the ConceptNet servers were not able to keep up with the load and would still be a bottleneck for us, and furthermore would be unable to serve other requests while our tests were running. This being unacceptable, we began to investigate caching.

In order to reduce the strain on ConceptNet's servers and further decrease the time required for our testing cycle, we looked more into local solutions (like 1. and 2. from above), but from the point of view of caching. After exploring a few approaches, we decided to use Python's native `pickle` function to store the data from ConceptNet, then reload that data from local disk before every run.

`pickle` is a data serialization function implemented by Python, and it is intended to provide easy-to-use and compact serialization of all Python types while remaining moderately performant. In our case, using it meant building a dictionary containing all of the data that we received from ConceptNet during a normal run, a far smaller set of data than all of ConceptNet, then calling `pickle` on that dictionary and writing it to disk.

This resulted in a `.pickle` file of approximately `230 MB`, a very acceptable size. Once that data was stored locally, we no longer needed to make any queries to ConceptNet whatsoever. When we begin a test, Python loads the data into memory, taking at maximum about `1.5 GB`, then runs all queries against that in-memory structure. This reduced our runtime against our full data set of questions from 25 minutes to about 30 seconds.

4.2 Threading

To maximize the throughput of our system, we employed threading using the Python `threading.Thread` type. Since Python otherwise runs in only a single thread in a single process, the parallel computing power of modern multicore processors largely goes to waste, as all but one core will be sitting idle. Splitting the problem into threads up to the number of available cores increases the available power by a factor of the number of cores.

In our case, we split the set of questions to solve into four groups, one for each core of our largest machine, and then created a new thread to solve each group of questions. When done, these threads sent their results back to the main thread via a Python `Queue.Queue`, then the main thread collected the results and returned.

4.3 Pathfinding

Though all of the concepts from our SAT analogies have a direct relationship, many of those relationships are only represented indirectly (if at all) in ConceptNet, with both concepts having some relationship to an intermediate concept. In order to find those relationships as quickly as possible, we implemented a bidirectional graph search. This search looked from each of the 'root' concepts, mapping out every way to get from one to the other in two or fewer edges. ConceptNet would properly be represented as a directed acyclic graph (DAG), but as we are interested in relationships in either direction, we follow edges going to and from either 'root', recording the direction in which we traverse the edge. Wherever we find a path between the two 'roots', we record the edges (which in ConceptNet represent relationships) which were traversed along that path.

As we search the graph for exact matches, endpoint to endpoint, on ConceptNet, we also take into account some information hidden by the messiness of the ConceptNet data. ConceptNet has as its basic unit the "lemma", or concept, which is usually a single word; however, these lemmas are sometimes given as a word and its definition instead of the word alone. In order to use that definition data, we also check to see if both of the 'root' words are contained in the same lemma, and if so we give them the relationship ‘sameLemma’ in our similarity analysis.

4.4 Similarity

Finally, once we have a set of all relationships (paths) between the stem of the question and each of the pairs of words which are the choices of answer, we compare the relationships of the stem and each choice. For each relationship that they have the same, and each similar relationship, that choice’s score increases, and once all choices have had their similarity analyzed and scores computed, the system selects the choice with the greatest score.

We had several different categories of similarity which would add to an option’s score, each of them given different weights. Most types of relationship were weighted at 1.0 point, but some, such as the “IsA” relationship, proved less valuable than others, as they did not convey a specific enough meaning, and these were given a lower weighting.

For every relationship the choice had that was exactly shared by the stem, it was awarded the full weight of the relationship. For each relationship between the two words of the choice, that choice was awarded 0.01 points, to push it slightly above other choices which had no known relationships.

We experimented with several variations in our scoring, which did things like allow for matching on subsets of correct relationships, or normalized the score of a choice by the number of relationships it had. However, after testing a large grid of these possible variations, we discovered that they did more to hurt than to help. It remains possible that with a larger sample size or with more complete concept relationship data these methods would be helpful, but in our experiments this proved not to be the case.

5 Results

After optimizing our weighting, our system correctly answered 111 of 374 questions, giving us a success rate of 29.7%. This is slightly superior to the results of the hybrid systems of Jiang and Conrath in 1997 and Lin in 1998. In order to properly analyze this result, we should examine the causes of the system's incorrect answers.

5.1 Errors

When looking at the errors of this system, there are two main categories: errors where insufficient information was present, and cases where there was at least the minimum viable amount of information. We will define the minimum viable information as being the situation where there is at least one relationship for the stem and for each of the choices where some path between the two concepts contains at most one intermediate concept.

Errors in cases with insufficient data were by far the majority of our incorrect answers, representing 199 of our 263 incorrects, or 76%. In these situations, ConceptNet is simply lacking data about the words defining the question. For example,

```
 lull:trust
(a)balk:fortitude
(b)betray:loyalty
(c)cajole:compliance
(d)hinder:destination
(e)soothe:passion
```

In this case, ConceptNet has almost no information about at least one of these words in each pair. "Lull", for example, has connections to "lullaby", "calm", and "letup", but not to anything relating to trust or security or safety. Every one of the choices is in a similar position, with insufficient connections from each of the words to the other. This is frequently the case for the rarer words in the English language.

Even in situations where there is enough information to meet our minimum viability requirement, there is sometimes only a very vague connection found by ConceptNet. In the case `livid:anger`

`(a)querulous:reconciliation`

`(b)forlorn:hope`

`(c)radiant:happiness`

`(d)graceful:posture`

`(e)marvelous:wonder`

there is a relationship found between each pair, but in the stem (the most important relationship), there is only one, and it is a very weak one:

`livid` ←— `SimilarTo angry`

and

`anger` ←— `DerivedFrom angry`

These very general types of information, while better than nothing, are not sufficient to reliably distinguish the right answer from the wrong one.

However, even including questions in which the relationships are few and general, on questions with information above our threshold, our accuracy is a near-human 55%. This beats all of the prior projects on this subject except one.

6 Further Work

In the future, there is much which could be done to advance the state of projects like this. The most straightforward improvement would simply be adding data to ConceptNet or pulling in additional data sources with similar information. As shown above, the vast majority of the incorrectly answered questions had insufficient information about their words on ConceptNet; if the coverage of ConceptNet were up to our minimum viable data threshold for each question, perhaps a project like this one would achieve 55% accuracy overall, rivaling human students.

Additional data in ConceptNet would also solve problems like the `livid:anger` question shown above, where the graph density was simply too low to extract significant meaning.

Another approach, trying to make the best of the limited data currently available, might search the graph for paths containing more edges, finding more indirect relationships. This approach would, however, be faced with the exponential expansion of possibilities, with no simple pruning heuristic in sight. However, for a few more steps, and with very careful cache and memory management, this approach would be very feasible.

Even further out, there might be the possibility of mapping the whole of ConceptNet into a huge hierarchical graph. Then, when searching for at least a single first connection between two terms, a program could traverse hierarchically "up" the graph from each term until the two searches intersected. With proper interpretation of the ConceptNet relationships, this hierarchy could even represent a massive full conceptual classification of the world to power a system which knows not only that an ostrich is a type of bird, but that it's strange in that it can't fly, when most birds do, and that some birds migrate, and so do many other animals, but ostriches don't, and that they only live in Australia, which is an island continent in the Southern Hemisphere. It would be big and messy and all over the place.

But so are people.

References

Bicici, E., and Yuret, D. (2006). Clustering word pairs to answer analogy questions. Proceedings of the Fifteenth Turkish Symposium on Artificial Intelligence and Neural Networks (TAINN 2006).

Bollegala D., Matsuo Y., and Ishizuka M. (2009). Measuring the similarity between implicit semantic relations from the web. Proceedings of the 18th International Conference on World Wide Web, ACM, pages 651660.

Herdadelen A. and Baroni M. (2009) BagPack: A general framework to represent semantic relations. Proceedings of the EACL 2009 Geometrical Models for Natural Language Semantics (GEMS) Workshop, East Stroudsburg PA: ACL, 33-40.

Hirst, G., and St-Onge, D. (1998). Lexical chains as representation of context for the detection and correction of malapropisms. In C. Fellbaum (ed.), WordNet: An Electronic Lexical Database. Cambridge: MIT Press, 305-332.

Jiang, J.J., and Conrath, D.W. (1997). Semantic similarity based on corpus statistics and lexical taxonomy. Proceedings of the International Conference on Research in Computational Linguistics, Taiwan.

Leacock, C., and Chodorow, M. (1998). Combining local context and WordNet similarity for word sense identification. In C. Fellbaum (ed.), WordNet: An Electronic Lexical Database. Cambridge: MIT Press, pp. 265-283.

Lin, D. (1998). An information-theoretic definition of similarity. Proceedings of the 15th International Conference on Machine Learning (ICML-98), Madison, WI, pp. 296-304.

Mangalath, P., Quesada, J., and Kintsch, W. (2004). Analogy-making as predication using relational information and LSA vectors. In K.D. Forbus, D. Gentner & T. Regier (Eds.), Proceedings of the 26th Annual Meeting of the Cognitive Science Society. Chicago: Lawrence Erlbaum Associates.

Resnik, P. (1995). Using information content to evaluate semantic similarity. Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, pp. 448-453.

Turney, P.D., Littman, M.L., Bigham, J., and Shnayder, V. (2003). Combining independent modules to solve multiple-choice synonym and analogy problems. Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP-03), Borovets, Bulgaria, pp. 482-489.

Turney, P.D., and Littman, M.L. (2005). Corpus-based learning of analogies and semantic relations. Machine Learning, 60 (1-3), 251-278.

Turney, P.D. (2001). Mining the Web for synonyms: PMI-IR versus LSA on TOEFL. Proceedings of the Twelfth European Conference on Machine Learning (ECML-2001), Freiburg, Germany, pp. 491-502.

Turney, P.D. (2006a). Expressing implicit semantic relations without supervision. Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (Coling/ACL-06), Sydney, Australia, pp. 313-320.

Turney, P.D. (2006b). Similarity of semantic relations. Computational Linguistics, 32 (3), 379-416.

Turney, P.D. (2008). A uniform approach to analogies, synonyms, antonyms, and associations. Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008), Manchester, UK, pp. 905-912.

Turney, P.D. (2012). Domain and function: A dual-space model of semantic relations and compositions, Journal of Artificial Intelligence Research (JAIR), 44, 533-585.

Veale, T. (2004). WordNet sits the SAT: A knowledge-based approach to lexical analogy. Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004), pp. 606612, Valencia, Spain.