The Report Committee for Viet Dinh Tran
Certifies that this is the approved version of the following report:

Application of the Semantic Network ConceptNet

APPROVED BY

SUPERVISING COMMITTEE:

Supervisor:

Constantine Caramanis

Omar Chavez

Application of the Semantic Network ConceptNet

by

Viet Dinh Tran, B.S.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

August 2016

Abstract

Application of the Semantic Network ConceptNet

Viet Dinh Tran, M.S.E.

The University of Texas at Austin, 2016

Supervisor: Constantine Caramanis

This report introduces a way to use the semantic network ConceptNet to create a tool to aid in a writer's creative process. ConceptNet is a project that maps concepts and their relationship to each other. The goal is to mimic our own creative search process of semantic knowledge. To achieve this we describe and implement two core components which are the search and the ranker. We show that these two components by themselves can provide useful semantic insight by identifying relationships between concepts and their relevancy even on seemingly unrelated concepts. We also present some more complex and practical application of these two components for future work.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This report presents an application idea of the semantic network ConceptNet. ConceptNet is a free and crowd-sourced project that maps concepts and their relationship to each other [1]. Concepts are ideas represented by words or short phrases of natural language. The data contain in ConceptNet is sourced from many other crowd-sourced projects. Our goal is to present a way to utilize ConceptNet to create a creative writing aid.

Chapter 2

Background and Motivation

2.1 BACKGROUND

We will begin by providing background information on ConceptNet and what kind of information we can extract from it. ConceptNet is described from its homepage as a multilingual knowledge base [1]. In essence, it is a network of nodes and edges. A node is some concept represented by a word or a short phrase. The edge contains several useful pieces of information. It includes the source where this knowledge was collected, the relation of the nodes to each other, and the weight of the edge. For our purposes we are only interested in the relation and the weight of the edge. The weight relates to the frequency of its appearance in the same knowledge source or multiple different sources.

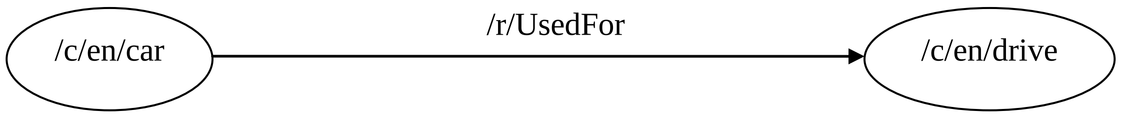

Figure 2.1: Representation of ConceptNet's node and edge

ConceptNet defines its own URI (uniform resource identifier) for uniquely identifying its different types of resources. In Figure 2.1 above, the "/c/" identifies the resource as a concept resource and the "/en/" identifies the language which in this case is English. The "/r/" on the edge identifies the resource as a language-independent relation. The URI pattern is defined on ConceptNet's homepage. This is important since we access this database through their application programming interface (API) that returns

JSON (JavaScript Object Notation) that we must parse to be utilized in our application. Also available on their homepage is information on how to set up your own copy of ConceptNet and the accompanying API documentation to access its data.

2.2 MOTIVATION

The motivation behind this report is from the observation that in works of creative writing there is a bridging of seemingly unrelated concepts that makes the work interesting. It is the presentation of new connections between concepts. The more ways in which we can connect one concept to another the more the relationship between these concepts makes sense. Also the farther out the connections between the concepts the more creative it appears. This is only true to some extent since if it's too far out then the relevancy between concepts diminishes. Another way of thinking about this observation is that we are joining the common characteristics of two or more concepts, resulting in connecting pathways between the concepts. This report presents a way to computationally mimic our natural discovery process of these connections with the use of the semantic network ConceptNet.

2.3 POTENTIAL APPLICATION

There are a few ways we can use this tool in practice. An example is narrative theme. A narrative theme is some idea or concept that is central to the story, often repeated in different forms. The characteristic of this theme concept is applied across the many different elements of the story. That is, the different elements in the story share some common characteristics with the narrative theme. Another example is simile or metaphors. These two figurative language types take some characteristic from one concept and applying it to another much like a theme but is only applied to one other concept and not the entire narrative.

2.4 GOAL

To achieve this, we need to design a way to traverse ConceptNet to find concepts and rank the relevancy of those concepts. The first part will be referred to as the "Search" component and the second part the "Ranker" component. The search component is responsible for traversing the network and storing the concepts discovered and their relationships to each other. This is needed for the ranker and also to visually graph the network. The ranker component's purpose is to find the most relevant concepts in the search result. This is achieved by using a form of Google's PageRank algorithm [3]. Our goal in this report is to design and implement the search and ranker components. We also limit the scope for this report to two concepts but we can generalize it for any number of concepts in future work.

Chapter 3

Methodology

3.1 THE SEARCH COMPONENT

The search component will be responsible for traversing the semantic network to collect all the concepts and their corresponding relations starting from some root concept since each concept is related to other concepts through a relation. This will essentially build an *n*-ary tree data structure where concepts are nodes in the tree and edges are the relations between concepts. This tree represents the characteristics of the root concept which are effectively its child concepts. The farther away a child concept is from the root concept the less related and less of a defined characteristic of the root concept it is expected to be. We now describe the implementation for this component.

3.1.1 The Search Tree Data Structure

Before we start we need to create the *n*-ary tree data structure to store our search result. The root node of a search tree is some root concept that contains any *n* number of pointers to its child concepts and each child concepts may contain any *n* number of pointers to its own child concepts. We perform a breadth-first search, that is, we start at the root concept and find its child concepts (first layer) and then we find the child concepts of these concepts (second layer) and so on. We also need a map of all the concepts contained in the tree. This is because we need a way to quickly check if certain concepts already exist in the tree before adding it. Duplicate concepts will create cycles in the search process that will yield no new useful information as shown in Figure 3.1. The red node represents the root concept while the blue represent the child concepts. We see that if concept *C3* is related to *C4* and if we try to add that relation into the tree it will create a cycle since *C4* was already added with a direct relation to the root concept *C1*. If we are performing a search for a high number of layers, the search algorithm will assume *C4* is a child of *C3* and perform a search on *C4* in which *C1* will be a result and the cycle continues until we reached our defined number of layers. No new information result from this.

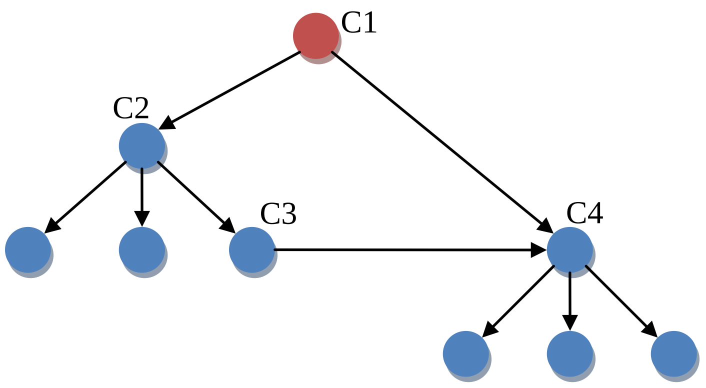

Figure 3.1: *N*-ary tree structure with a cycle

We cannot ignore that there is a relation from *C3* to *C4*, however. This is important information for the ranking process since the rank depends on the number of edges connected to a given node. So in addition to a map of unique concepts we also need to store a list of all relations we encounter in the tree as the search progress. The relation from *C3* to *C4* is stored in this list but *C3* will not have a pointer from to *C4* in the tree itself for the purpose of avoiding cycles. There are advantages to avoiding cycles this way when coupled with a breadth-first search. The first is that we can depend on each node to have one and only one parent. This is useful for traversing from a child concept back to the root concept to reveal one unique path of relations. The second advantage is that we get the most direct (shortest) path from the root concept to any given child concept. For example, in Figure 3.1 we see *C4* has a direct relation to *C1*. This information is more useful to us as compared to going from *C1* to *C2* to *C3* to *C4* because the relation *C1* to *C4* reveals a more relevant connection in one jump of the tree versus three jumps. As mentioned before, the farther a concept is from a root concept the less related it is expected to be to the root.

We now discuss what kind of information is stored in the nodes. Nodes will contain the concept, a list of pointers to its children, a pointer to its parent, the relation to its parent, the direction of this relation, and the weight of the relation. Since relations can be directed, so do the edges in the tree as exemplified in Figure 3.2. This is useful information for when we want generate a visual graph of the search tree.

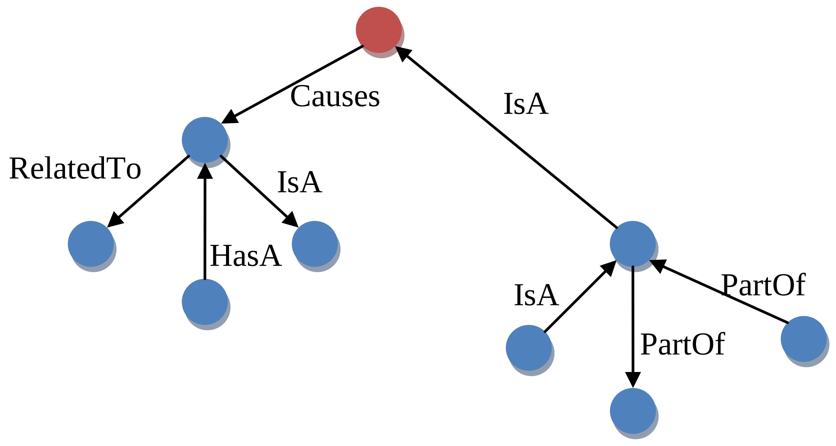

Figure 3.2: Example of a search tree with directed relation

3.1.2 The Search Process

Now that we have the data structure we can begin describing the search process. In general, the search process is a breadth-first traversal. That is, we start from the root concept and search for concepts related to it. We then repeat for each child of the root concept as seen in Figure 3.3.

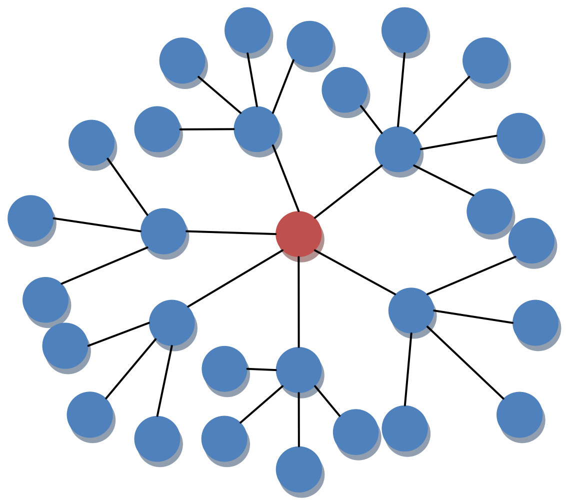

Figure 3.3: Example 2-layer search result tree

First, the input for the search component will be a string of natural language words and also the number of layers to traverse. For this report we limit the number of words to two but as a future improvement this will be extended to any *n* number of words. Layers are how many levels we want the search to perform starting from the root node. We perform the search for each word in the input string resulting in an *n*-ary tree for each of the words. Before we perform the search for each word we first need to get the stem of the word. The stem of a word is the base word where affixes can be attached. For example, the stem of "mice" is "mouse" and "fried" is "fry". This is necessary because the concepts stored in ConceptNet are in this base form. ConceptNet provides an API called the "URI" API that allows us to do this stemming. In addition to stemming we also need a process to prune stop words. Stop words are common words such as "of", "or", and "as" that provides no useful characteristics to the root concept.

ConceptNet provides us with a “Search” API that allows us to search using relations. That is, we can pass some concept C and a relation R and ConceptNet will return a list of concepts that relates to C by relation R. Knowing this we can do a search of all possible types of relations in the semantic network to build something we call a "concept space". A concept space is the first layer of concepts that connects to some concept. This concept space gives us a general sense of what this concept is and its characteristics; it describes its parent concept through the relation between them. For example, if we apply this search to the concept "cat" then the resulting search will be as shown in Figure 3.4.

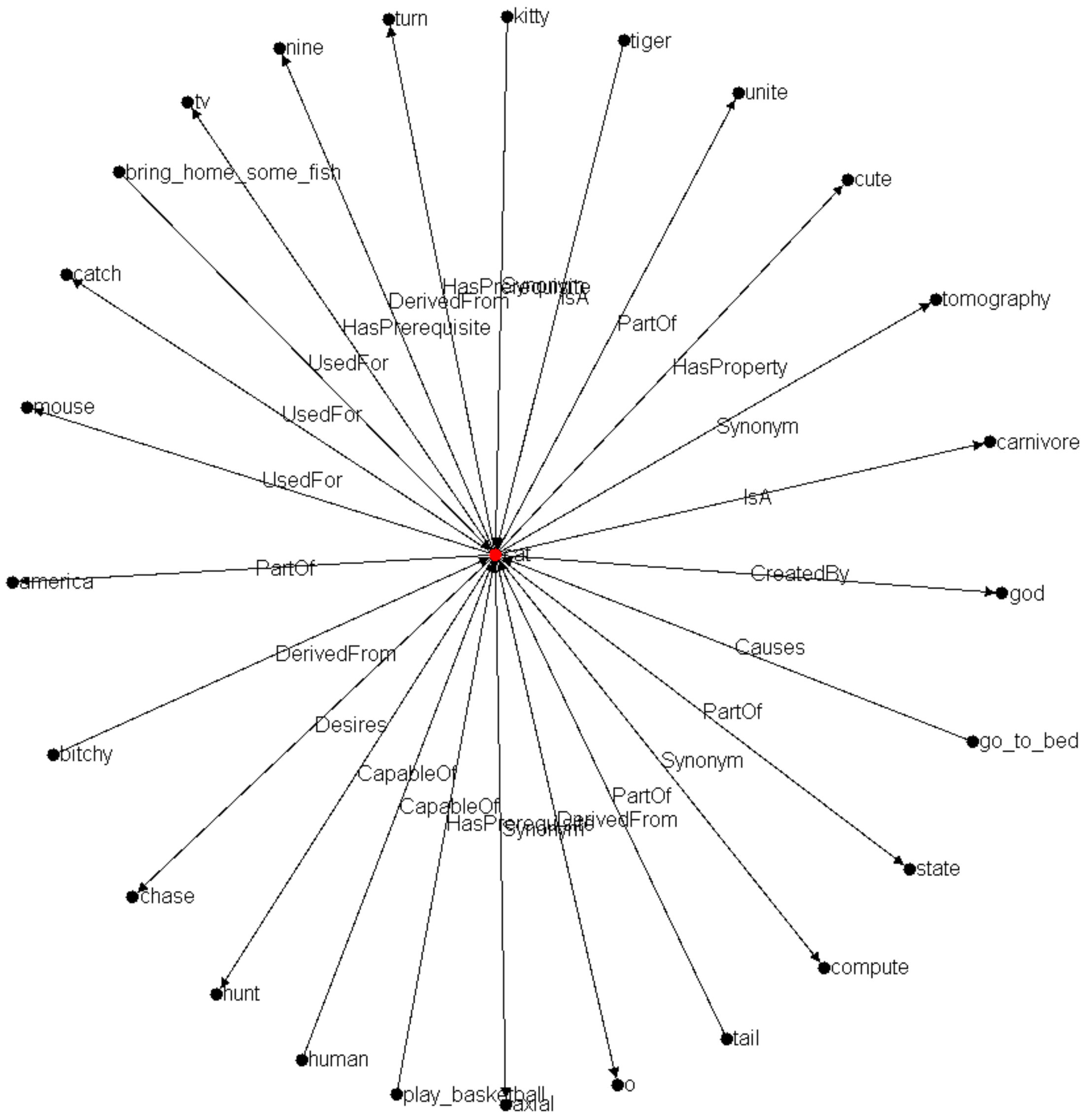

Figure 3.4: Search output for the concept "cat"

In summary, the search process takes a string of words (concepts) and builds a search tree for each word. It will take each root word and search all possible relations available in ConceptNet to get the concepts that relates to the root word by that relation if it exists. There may be more than one concept that relates to the root word through the same relation so we can make this a parameter for the user to limit how many concepts are added to the tree per unique relation. This process can be repeated for subsequent child in the tree. The number of search layer can be set and should be kept low since the search cost is exponential per layer. If we do an n number of relation search for each concept and have k number of layers then we will have n^k searches.

3.1.3 Application of the Search

Now that we have a search tree for each of the words from the input string there is something interesting we can do with it. Since all the concepts in each tree are unique within its own tree we can do a set intersection between the trees. If there is sufficient layers in each tree then this intersection would result in concepts that are common to both trees. Knowing this we can effectively join the two trees with the common concepts being the bridge between the two.

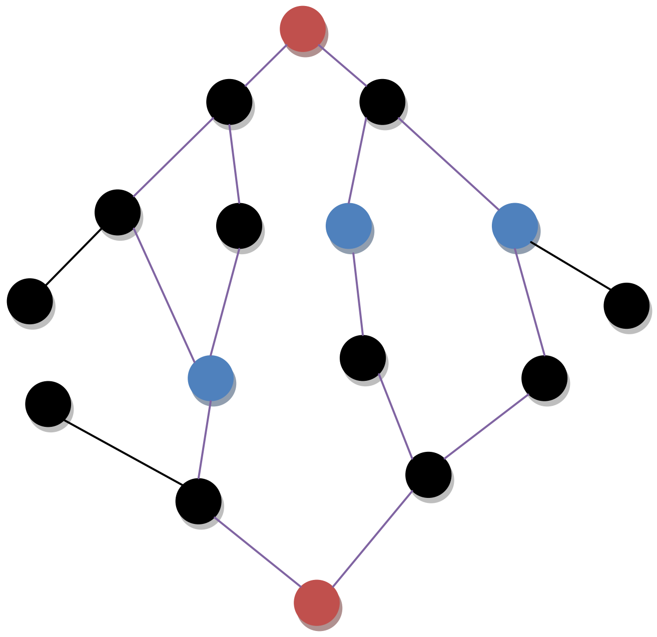

Figure 3.5: Joining of two concept trees

Figure 3.5 is an example of what it would look like if we were to join two trees by their intersecting concepts if they exist. The red nodes are the two root concepts and the blue nodes are the intersecting concepts between the two root concepts. In actuality any nodes along the paths that join the two root concepts are intersecting concepts. The reason is if we increase the number of search layers for one node and lower the number of search layers for the other we will effectively shift the point of intersection either up or down. If we remove the nodes from Figure 3.5 that are not in the connecting paths we have something like Figure 3.6.

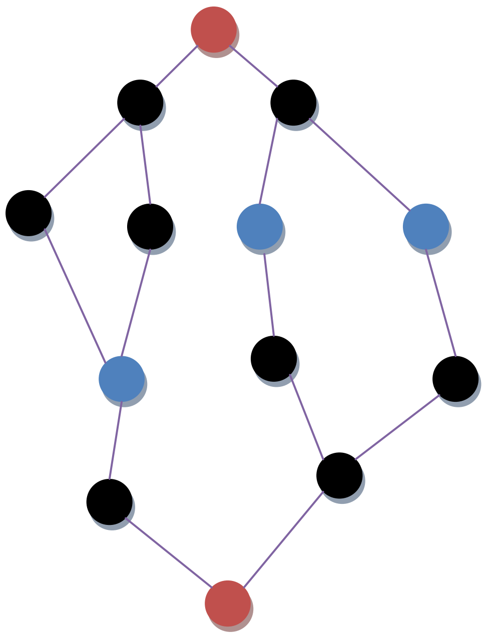

Figure 3.6: Graph of two joined root concepts and their connecting paths

There is useful insight we can gain from this graph. We see the path of relations between the two root concepts, that is, the way in which the two concepts are connected. We have found how the two root concepts relate to each other. Any of the non-root concepts in the graph are potential concepts that can be used as a bridging concept. All of this information will be stored in a search result graph object which is basically a collection of nodes and edges. This graph object will be used later by the ranker. As an example, we see a graph generated from our search component showing how the concepts "water" and "electricity" connect in Figure 3.7.

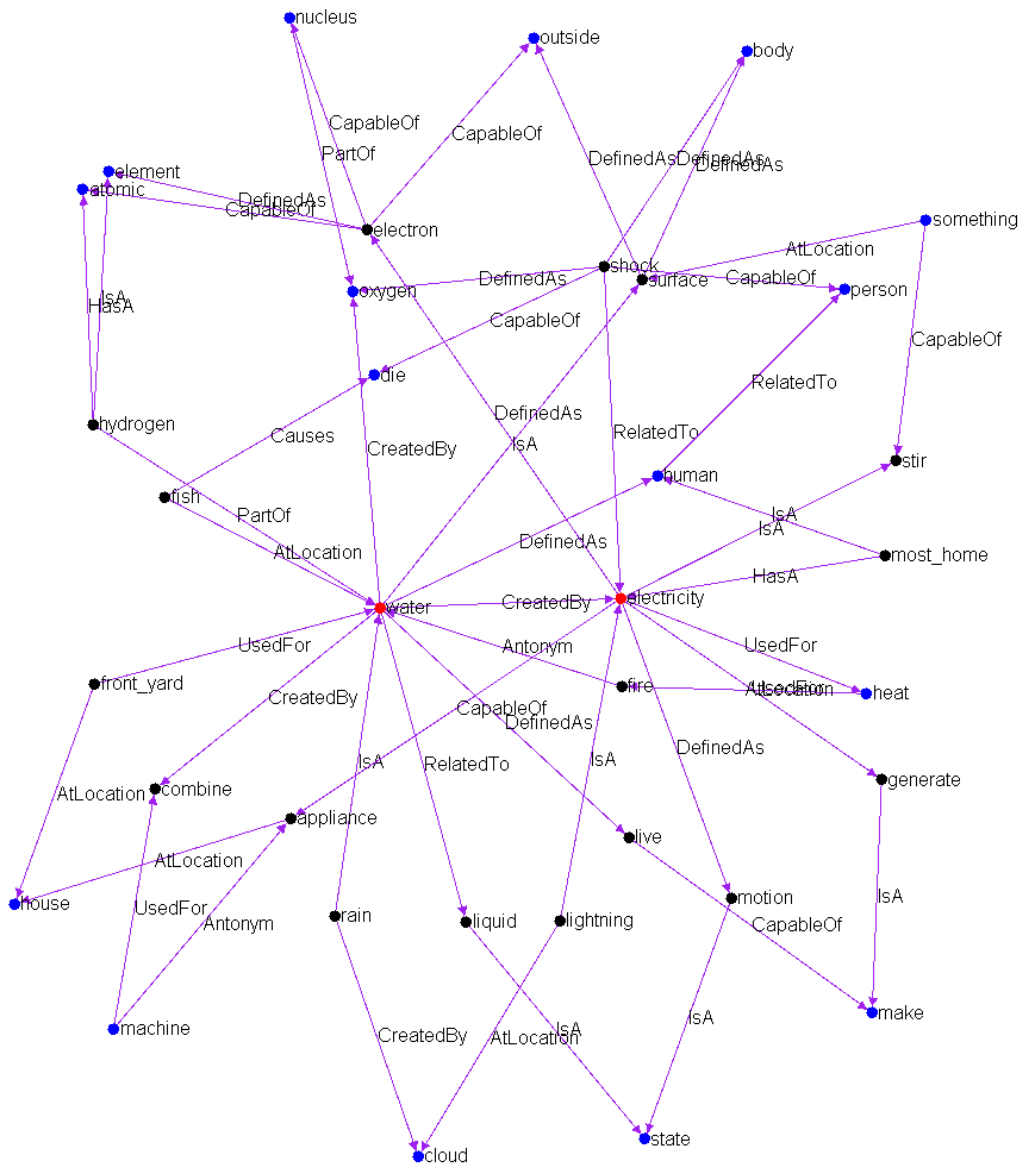

Figure 3.7: Graph of the joined concepts "water" and "electricity"

3.2 THE RANKER COMPONENT

The ranker is responsible for ranking the concepts found in the paths connecting the two root concepts. A concept's rank determines how relevant it is in the search graph. To resolve a concept's rank we draw attention to the fact that the search result is a network of nodes and edges. This is analogous to the World Wide Web. There exist algorithms to measure the relative importance of nodes in such a network, one of which is the famous PageRank algorithm used by Google [3]. For this report we will use a modified version of this algorithm to determine the rank for each concept. To actually apply the algorithm and compute the rank we will use the Markov chain [4].

3.2.1 PageRank and Markov Chain

We now give a brief overview of the PageRank algorithm and the Markov chain. A detailed explanation is outside the scope of this report but there are many resources available on the web on these topics. We will start with the PageRank algorithm. The PageRank value is given by the equation:

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)} \tag{3.1}$$

The equation states that the PageRank value for some node *u* is dependent on the PageRank value of node *v* where *v* is in the set B_u. B_u is the set of all nodes that link to node *u* and *L(v)* is the number of links from node *v* to node *u*. We do not know each node's PageRank value initially because each node's value is dependent on the other nodes' PageRank value. Due to this, we initialize all the nodes' value to 1/*n* where *n* is the number of nodes in the network. If we apply the algorithm for several iterations we will begin to notice that the PageRank values start to converge to a value that's closer to the theoretical value.

To actually calculate the PageRank values we will employ the Markov chain. A Markov chain is a mathematical system that transition from one state to another with a certain probability. Recall that ConceptNet provides us a weight for each edge in our search graph. If we take all the edges connected to a particular node and divide it by the total weight value of its edges we effectively have the probability of going down one edge versus the other. For example, in Figure 3.8 the edge *C1* to *C2* has a probability of 5/(5+7) or 5/12 and the edge from *C1* to *C3* has 7/12.

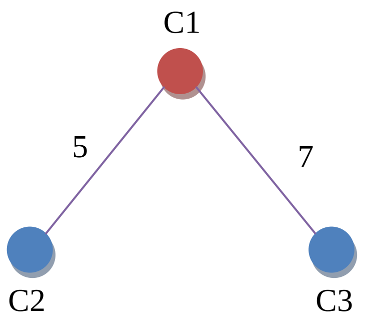

Figure 3.8: An example graph with weighted edges

If we obtain these values for the entire search graph we can build a transition matrix. Figure 3.10 shows an example transition matrix *A* for Figure 3.9. In the matrix *A* the first column is the connection from *C1* to *C1...C6* then the second column is the connection from *C2* to *C1...C6* and so on. For example, the first value in the first column is the value for the connection *C1* to *C1* which is 0, the second value is for the connection *C1* to *C2* which is 5/12, the third value is for the connection *C1* to *C3* which is 7/12, etc.

The vector V is the initial value of each node. We do $A^n V$ some n number of times to get the convergence of the values of each concept node. These values are the rank of each node. Typically the rank of the root nodes will end up being the highest ranked but since we are not interested in them we will only look at the nodes in the connecting paths.

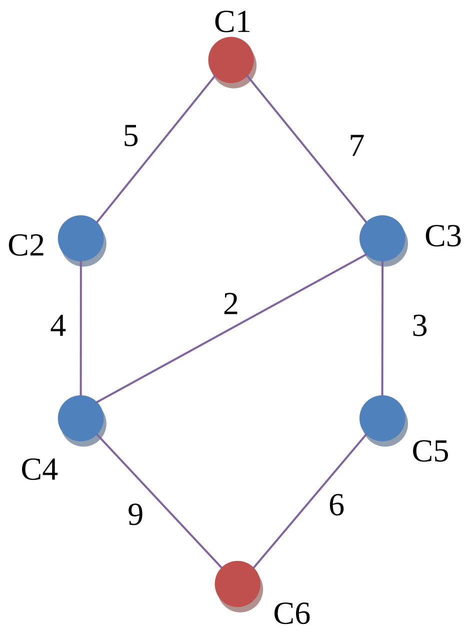

Figure 3.9: Example of a weighted search result graph

$$A = \begin{bmatrix} 0 & \frac{5}{9} & \frac{7}{12} & 0 & 0 & 0 \\ \frac{5}{12} & 0 & 0 & \frac{4}{15} & 0 & 0 \\ \frac{7}{12} & 0 & 0 & \frac{2}{15} & \frac{3}{9} & 0 \\ 0 & \frac{4}{9} & \frac{2}{12} & 0 & 0 & \frac{9}{15} \\ 0 & 0 & \frac{3}{12} & 0 & 0 & \frac{6}{15} \\ 0 & 0 & 0 & \frac{9}{15} & \frac{6}{9} & 0 \end{bmatrix} \qquad V = \begin{bmatrix} \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \end{bmatrix}$$

Figure 3.10: The transition matrix *A* and the initial node values *V* for the search result graph in Figure 3.9

3.2.2 Parsing the Search Graph

From the search component we obtain a search result graph object which contains a collection of all the nodes and their corresponding edges. Before we can rank the nodes we need to parse the information in this object to generate the transition matrix. We first obtain all the nodes from the result graph and create an empty matrix of the same size as the number of nodes in the graph. We then populate the matrix with the edge label combination as described in the previous section. For example, *A*[0,0] would be *C1C1*, *A*[0,1] would be *C1C2*, *A*[0,2] would be *C1C3* and so on. The edge labels from the result graph object follow the same pattern. This means we can get the set of edges from the graph object and iterate through it to see if a certain edge exists and if so, get the weight of that edge. We store the weights in a separate matrix. We then determine the total weight for each column and divide each value in the same column by this total weight to obtain the fractional values as seen in Figure 3.10. Each column's total value should now sum to 1. This is our transition matrix.

3.2.3 Calculating Rank

Now that we have our transition matrix we can compute the rank for our concept nodes. We simply multiply the transition matrix with itself some defined number of iterations (5 for this report) then multiply the result with the vector *V* to obtain the rank for the concepts in our search result graph. We should end up with a vector of fractional values that represent the rank for each concept in the search result graph.

Chapter 4

Potential Application

In section 2.3 we went over narrative theme, simile, and metaphors as the potential application of this tool. We will now describe in more detail how we can use the search and ranker to accomplish this. We'll begin with narrative theme then move on to simile and metaphors with each different application using the search and ranker component in a different way to achieve the desired goal.

4.1 NARRATIVE THEME

Recall from section 2.3 that a narrative theme is some concept with its characteristics applied across many different elements of the narrative. We have the tools necessary to do this with our search and ranker component. The search component will gives us the ability to find the characteristics of our theme concept. It will also allow us to find the potential elements that are relevant to our theme. The ranker component will help us determine which element is more relevant with our theme.

We first set the root to one of our search tree as the theme concept. Now say we want to find a location for our narrative that fits in with the theme. All we have to do is set the root concept for another search as "location" and perform an "IsA" relation search on that concept. This should return concepts that are locations. We then perform the search for both trees using all relations until we have enough layers in each tree such that we will have some intersecting concepts as seen in Figure 4.1. The idea is that if we do a search for the concept "location" with the relation "IsA" then we will get concepts that are actual locations and those locations have certain characteristics that define them. If we do the search for the theme concept for enough layers there will be a point where the theme characteristics and the many different locations characteristics intersect. We apply the ranker to the location concepts to get the rank each location's relevance to our theme.

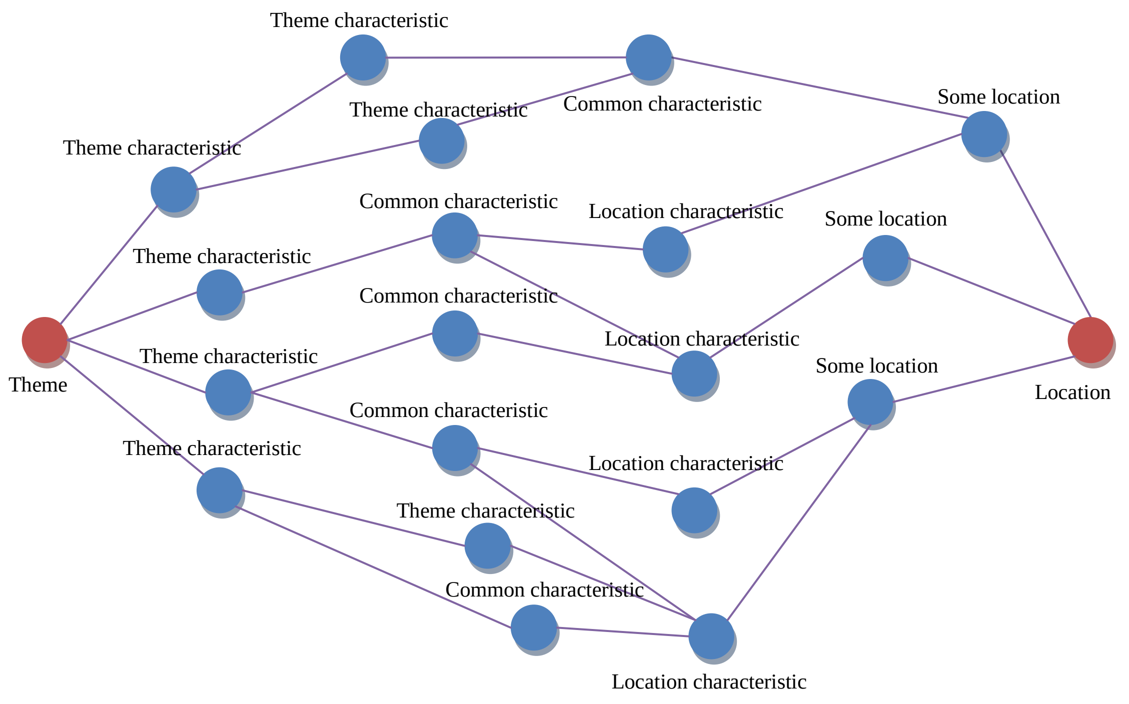

Figure 4.1: Example search graph of how a theme connects with a location

4.2 Simile and Metaphor

In a metaphor we do a comparison of one concept to another by using one in the other's place which suggests like-characteristics between them. For a simile we compare seemingly unlike concepts by inferring a characteristic from one concept exists in the other [2]. Both of these figurative language types make use of common characteristics between concepts. Discovering characteristics for a given concept is what our search component was designed for.

In both situations we know of one root concept and the other is unknown. Our goal is then to expand the search for this known root concept then use the ranker to find our unknown root concept of interest. We start by first doing a search for the known root concept for some number of layers depending on how distant you want the two root concepts to be. In Figure 4.2 we used two layers. We then search for one more layer on each of the leaf concepts as seen in Figure 4.3. We see in the same figure that each of the new leaf nodes can have multiple references to it. The ones that have the most are highlighted red and are top candidates for being the unknown root concept we're searching for. We prune the non-potential root concepts in Figure 4.4. If we apply the ranker to this graph then ideally the highest ranked leaf concept in red would be the most relevant concept to use in our simile or metaphor. That is the unknown root concept we're searching for.

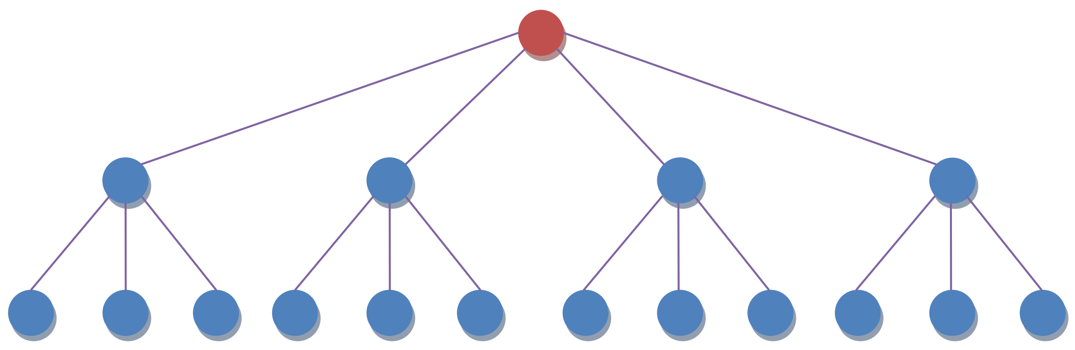

Figure 4.2: A 2-layer search graph for some known root concept

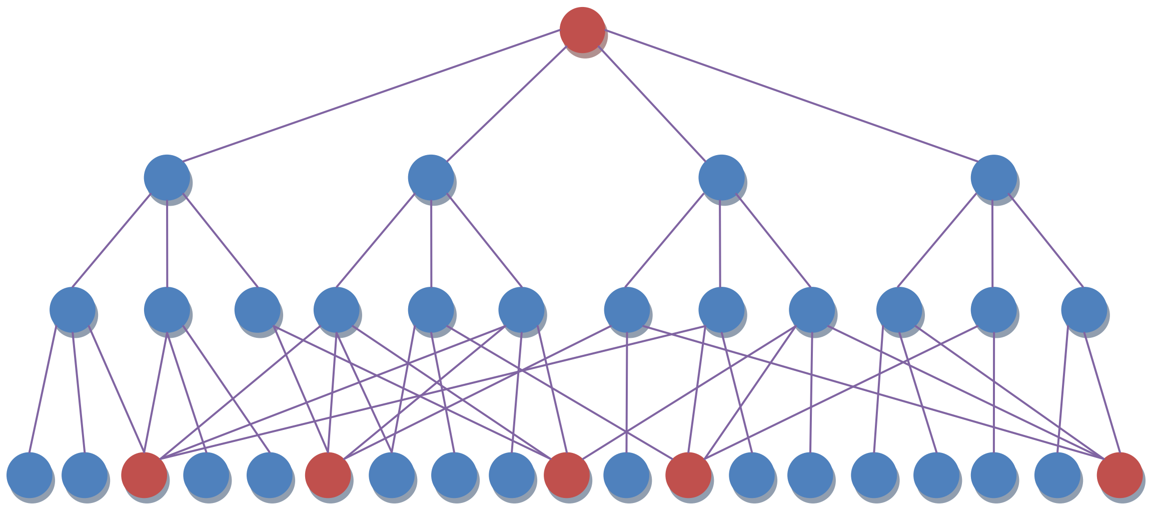

Figure 4.3: Expanding the search from Figure 4.2 to one more layer

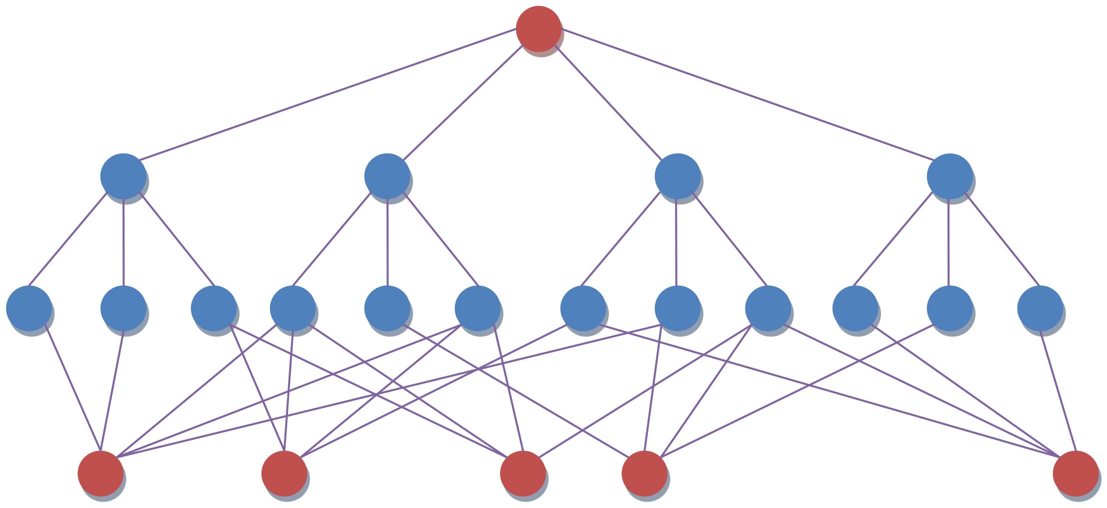

Figure 4.4: Search graph from Figure 4.3 with concepts with low reference count pruned

Chapter 5

Results

We presented some ideas on how to combine the search and ranker component in the previous chapter to achieve different narrative goals. Here we show that even using the two components by themselves can generate useful insight. To do this we use a random word generator to pick two root concepts. What our components will generate is a graph of the two root concepts and all the concepts that link them. Our two randomly generated concepts are "future" and "harness". The graph generated in Figure 5.1 shows the relationship between the two concepts and Table 5.1 shows the rank for the top 15 concepts in the graph. The search was limited to "RelatedTo" and "IsA" relations or else the resulting graph would be much more complicated and hard to decipher.

The blue nodes in Figure 5.1 represent the interface where the two root concepts are joined. The purple paths represent the paths between the root concepts that intersect the interfacing nodes. In this particular example we notice that the connection between the two root concepts is quite far apart so it may be difficult to infer a relationship between the two. However, if we were to expand the search to all available relations we might be able to find much closer connections. When analyzing the concepts and their relations from Figure 5.1 we can confirm from our semantic knowledge that the search component is working as designed. From Table 5.1, "time" and "control" are the highest ranked concepts and we can visually verify this from our graph by the fact that many edges are connected to these concepts. This confirms that our ranker is also working as designed.

From our knowledge, the concepts “future” and “harness” might appear to be seemingly unrelated but interestingly enough from Figure 5.1 there appears to be a distant connection. We see that “harness” is related to “control” which ultimately is related to “government”. For “future”, we see it connects to “tense” then to “state”. We then see that “state” is connected to “government” which is an interesting connection since “state” represents a different idea when connected to “tense” than when it is connected to “government”. This is a play on the word “state” since its two different meaning is bridging the concept “government” and “tense”. This can be useful information to a crafty writer trying to explore ways to connect the concept "future" and "harness" that he or she may not have thought of before.

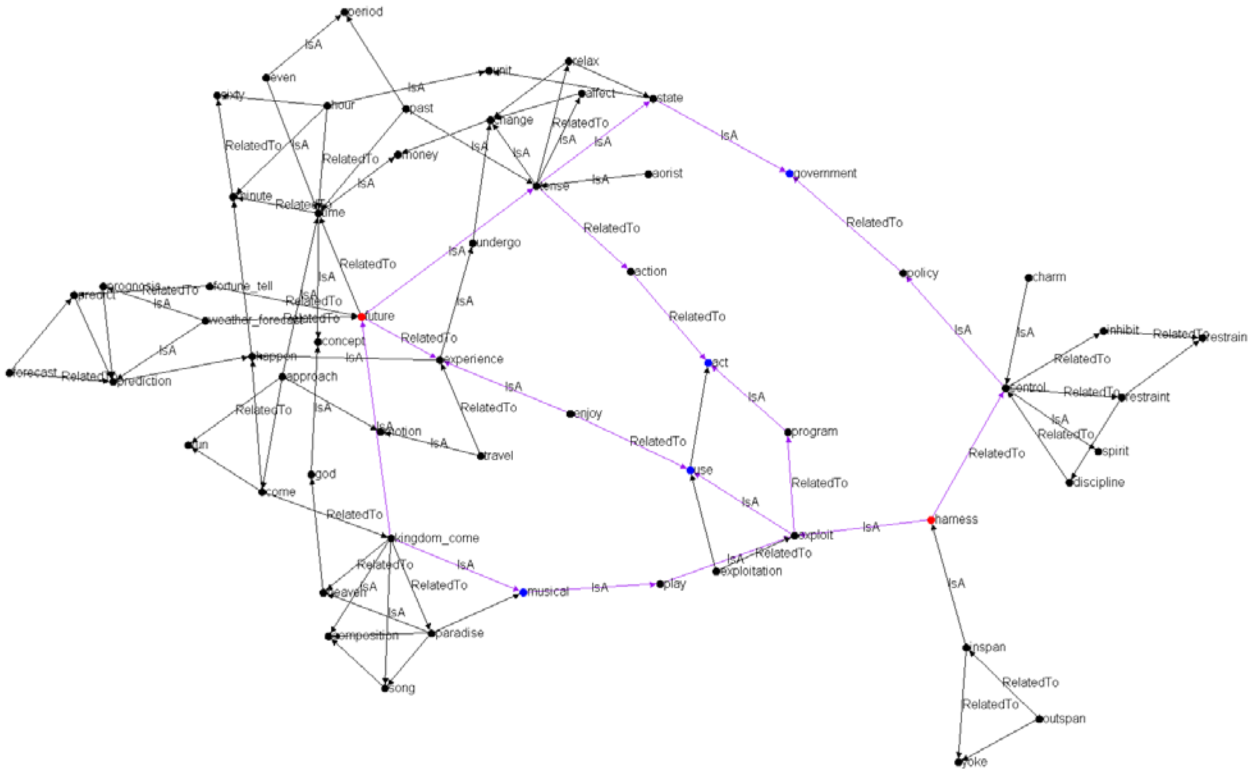

Figure 5.1: Search result graph for concepts "future" and "harness"

Concept	**Rank Value**
time	0.043433692
control	0.038548224
tense	0.0351923
hour	0.03351272
minute	0.029620389
kingdom_come	0.028674366
future	0.027036997
exploit	0.026471604
use	0.025771791
state	0.024508204
change	0.023734184
experience	0.0224625
come	0.021738226
prediction	0.021122461
happen	0.020617357

Table 5.1: Top 15 ranked concepts for Figure 5.1

Chapter 6

Future Work

In this report we presented a usage idea of the semantic network ConceptNet. We designed the search and ranker component to execute this idea and also presented some potential practical application of them. For future work we hope to implement these applications along with some performance optimization.

We discussed applying the search and ranker for the purpose of applying narrative theme and discovering simile and metaphors in Chapter 4. For narrative theme we start with two known concepts and for simile and metaphors we start with one known concept. A third idea is to start with no known root concepts but rather some known characteristic concepts. We use the known characteristic concepts we're interested in to find us two or more root concepts. This basically is the reverse process of what we've presented in Chapter 5. The practical application of this in terms of creative writing is currently not known but is an interesting idea nevertheless.

A future work mentioned at the beginning of this report is to extend the search component to except more than two concepts. The reason is that for the general case of any *n* concepts it is more complex to keep track of all the connections while pruning unnecessary dead end nodes to improve performance. Performance is a big issue since the search takes exponential time based on the number of layers we're performing in the search. As part of the future work we will devise an algorithm to prune uninteresting connections and nodes to cut down the number of concept searches. We can also add parallelization by making multiple API calls at the same time and spinning up multiple services to distribute the load.

Chapter 7

Conclusion

In this report we set out to explore a usage idea of the semantic network ConceptNet. We wanted to create a creative writing aid that can search the network and rank the relevancy of resulting concepts, effectively mimicking our own natural creative search process. We designed and implemented the main components for this which was the search and ranker. Potential practical application of these components was discussed. Even in its basic form we were able to utilize the components to show that we can generate useful insight on how concepts connect with one another and their relevance. This even works on seemingly unrelated concepts as long as the search is wide and deep enough. Such knowledge on the relationship between concepts can be valuable to a writer seeking new ways to craft the content of a narrative that was never thought of before. The cost of the search is exponential and wouldn't be feasible for a large search space but we can reduce this with careful pruning and parallelization. This might not be an issue since our goal is to keep the search space small anyhow. This is so that we can retain relevancy in the connections between concepts. An insight resulting from the search that is too distant to make any sense is not useful to a writer.

Appendices

Appendix A

Open Source Credits

Implementation of the search and ranker component would not have been possible without the following open source libraries.

- ConceptNet
- Google Guava
- Apache Commons
- GraphStream

Appendix B

Source Code

The source code for this report can be found at:

https://bitbucket.org/viet54/HarmonEv

Bibliography

[1] ConceptNet contributors. ConceptNet5. 10 Nov. 2014. 1 May 2016 <http://conceptnet5.media.mit.edu/>.

[2] "Examples of Figurative Language." YourDictionary, n.d. Web. 18 Jun 2016 <http://examples.yourdictionary.com/examples-of-figurative-language.html>.

[3] Page, Lawrence, and Brin, Sergey, and Motwani, Rajeev, and Winograd, Terry (1999) The PageRank Citation Ranking: Bringing Order to the Web. Technical Report. Stanford InfoLab, Stanford University, Stanford, CA, 1998.

[4] Tanase, Raluca, and Remus Radu. "The Mathematics of Web Search." PageRank *Algorithm - The Mathematics of Google Search*. 6 Feb. 2009. 18 Jun. 2016 <http://www.math.cornell.edu/~mec/Winter2009/RalucaRemus/index.html>.