A *Twenty Questions* Player Using Semantic Networks

Joel Michelson, Rajagopal Venkatesaramani
EECS, Vanderbilt University

Abstract

20 Questions is a cognitive task that relies on players having background knowledge of the world, and being able to efficiently search the knowledge-space, given prior information from the chain of questions asked at any point. In a typical game, the answerer chooses some subject, and the questioner asks yes-or-no questions which the answerer then answers. The questioner's goal is to identify the subject in under twenty questions. The game thus lends itself well to crowdsourcing of common-sense knowledge. Existing automated questioner solutions use pre-defined sets of questions to play the game and to fill in gaps in the dataset. Using ConceptNet - a semantic network - we attempt to create a questioner that will generate meaningful questions on its own given its knowledge of such semantic edges throughout the concept graph.

Introduction

Twenty Questions is an excellent example of a cognitive task based on semantic inference. In typical human gameplay, one player thinks of a subject - usually common objects, animals, plants, places, etc. - and the second player tries to guess the subject by asking up to 20 yes-or-no questions. The objective is to ask questions that help eliminate large sections of the knowledge-base and narrow potential guesses down to the correct category within the limit of 20 questions. Although the rules of the game are fairly specific, it has the potential to generalize to linguistic reasoning.

Efficiently playing the game from an algorithmic standpoint requires search space optimization over a well-represented knowledge-base. An obvious representation of common-sense knowledge is a semantic network - where vertices represent concepts (typically nouns), and each edge represents a semantic relationship between concepts. Naturally, such a network contains multiple types of edges, thus making our network heterogeneous. Forming questions from these relational links involves rudimentary natural language processing as well. In order to win a game of Twenty Questions, the questioner must contain a large database of semantic information and the ability to generate questions that divide the database optimally during runtime.

The most successful existing approach uses a neural network architecture, and is online at *20q.net*. This implementation is a variant of the original 20 questions implementation that made its way into hand-held video games, produced initially by Radica in 2004, and later on licensed to Techno Source in 2011. Whereas the hand-held version had a limited subset of the web version's knowledge-base, the web implementation allows dynamic expansion of the knowledge-base using information gathered both during game-play, as well as from the Internet. In contrast, our approach is an illustration of search-space optimization on existing semantic networks, where 20 questions is used as an evaluation metric - the problem is to minimize the number of steps taken to correctly guess what the player is thinking of using only the relations embedded in the network.

In our implementation, we use ConceptNet - a crowdsourced weighted directed semantic network of common-sense knowledge. ConceptNet (Liu and Singh 2004) consists of a large number of common-sense relationships between words and phrases based on the Open Mind Common Sense database (Singh et al. 2002). Its data comes from many sources, including Wiktionary and Open Multilingual Wordnet (Bond and Foster 2013). Its data consists of approximately 28 million Edges between nodes and the Relations that label them. The edge-relations of ConceptNet are well suited for Twenty Questions, as they present binary properties about subjects for the questioner to use (e.g., UsedFor, IsA, PartOf, etc.). ConceptNet also supplies a weight for each edge as a rough approximation of that edges credibility. Its data is open and available in a JSON-LD API.

Model

Our approach to building a 20 Questions player relies on using the relation-specific edges in semantic networks, and exploiting network structure to ask questions similar in nature to those that a human player would ask, while also optimally searching the database. Searching involves two aspects for the purpose of 20 Questions - when yes is received as an answer, we wish to identify vertices in the network that are connected to the previous vertex - i.e. the one the question was asked about - by the relation using which the question was asked. We further want vertices selected from this set to maximize information gain, i.e. give us the best chance of landing on a correct answer. When a no is encountered as an answer, we eliminate all vertices connected to the previous vertex, and then identify the next potential vertex.

The semantic network may be interpreted either as a single layer with heterogeneity over the edges, or as a multi-layer graph, with homogeneous individual layers. Though these are equivalent in terms of the information they carry, the definition of vertex degrees changes slightly in the two cases. In a single layer interpretation, it is conventional to define degrees as being edge-relation agnostic, whereas in the multi-layer model, degrees are typically defined over each layer, and convey more specific information. It is worth noting that the sum of degrees over all layers for a vertex is the relation-agnostic degree. With the intuition that degrees

specific to relations may be useful, we choose to adopt the latter, a multi-layer implementation. Before building the actual graph, however, we do the following pre-processing on the dataset.

Pre-Processing

Restricting to English ConceptNet, having been constructed with Multilingual WordNet as one of its sources, contains words and relations from many major languages. In theory, each subset corresponding to a language can be separated and a multi-lingual player be formed, but we judge this to be beyond the scope of this project. We thus filter out relations that correspond to the English language. Language is denoted in each entry by the regular expression $\backslash c \backslash \langle lang \rangle$ where *lang* is a 2-letter abbreviation of the language name. The expression 'en' corresponds to English, and we use a Python script to filter these edges out.

Combining Relations Some relations in ConceptNet, though different in a lexicographic sense, may be combined for purposes of playing 20 questions. The relations *IsA* and *FormOf* for instance convey similar ideas - both are hierarchical in nature, and thus strictly one way. Similarly, we group the relations *HasFirstSubevent, HasLastSubevent* and *HasSubevent* all into *HasSubEvent*, and remove duplicate word/phrase pairs. For purposes of question generation, the order of subevent occurrence is almost always found irrelevant.

Expanding DBPedia Relations ConceptNet contains, among its diverse range of sources, information from DBPedia - a public infrastructure for semantic knowledge. DBPedia contains specific common-sense knowledge snippets as relations in a graph, which in ConceptNet have been categorized within a single relation type, with an additional string that signifies the specific relation. Examples of these are *DBPedia-Product, DBPedia-Occupation, DBPedia-Genre*. We expand each of these sub-relations into individual relations in our graph implementation.

Removing Irrelevant Relations The relation *Antonym* does not help much with question generation for a game of 20 questions. Rarely at best, if at all, is a question phrased with a negation, and given that in the English subset of ConceptNet there are 1397805 Synonym edges, as opposed to only 20041 Antonym relations, we consider it safe to discard them. Relations like *CausesDesire, DerivedFrom, EtymologicallyRelatedTo* etc. also do not provide any useful information for purposes of our game.

Adding Transitive Edges For the categories *Plant, Animal, Person, Object* and *Place*. The intuition behind doing so is that most human 20 Questions players typically think of subjects that would fall in one of these categories, and by itself, ConceptNet did not have sufficient number of links to consider these vertices as candidate start points for the game.

Constructing the Graph

We construct a multi-layered graph $G = (V, \{E_1, E_2, \ldots, E_n\})$, where edge-set E_i corresponds to the i^{th} relation. The vertex set is common, as two words may be related to one another by more than one relation. Edges in G are directed, but we discard the weights provided by ConceptNet, as they are not truly reflective of edge-strength. In ConceptNet, edge-weight is decided by the number of times a particular word-pair was linked using a certain relation, and as it is crowd-sourced, some topics are contributed to way more than others.

Layer Out-Degree For a vertex $u \in V$, we define vertex degree in layer i as the number of outgoing edges from u in that layer. Thus,

$$\delta_i(u) = |\{v|(u, v) \in E_i\}|$$

Layer In-Degree For a vertex $u \in V$, we define vertex degree in layer i as the number of incoming edges at u in that layer. Thus,

$$\delta_i^*(u) = |\{v|(v, u) \in E_i\}|$$

Question Generation

Given the graph representation, G of the semantic network, we store a list of positive answers,

$$P = \{(v_1, e_1^i), \ldots\}$$

and a list of negative answers

$$N = \{(v_1, e_1^i), \ldots\}$$

. Each entry in the answer lists is a vertex-relation pair. Then we define the following sets, to formulate selection of candidate questions that satisfy the conditions as posed by previous answers:

$$V_P = \{v \in V(G)|(v, u_k) \in E_k^i(G), \forall (u_k, e_k^i) \in P\}$$

$$V_N = \{v \in V(G)|(v, u_k) \notin E_k^i(G), \forall (u_k, e_k^i) \in N\}$$

Then candidate vertices to ask the next question are contained in

$$V_{candidate} = V_P \cap V_N$$

At each question, generating the next question thus involves solving the problem:

$$\arg \max_{v \in V, i \in [1,n]} \delta^i_{candidate}(v)$$

where $\delta^i_{candidate}(v)$ is the number of edges of the i^{th} relation type originating from a candidate vertex and terminating in v. Instead of only storing the max, we store all candidate question vertices in a list and sort them in order of degree. In case the answer to the best question is no, we simply move on to the next candidate. If the answer is yes, we move one level down in the relation-hierarchy and re-calculate candidate edges.

Algorithm

Version 1

In our first attempt, we use a naïve data structure to hold the graph G, where we store a set of two lists for each node in G. The first list contains all the other vertices the original node is connected to, and the second list holds the relation-type

by which they are connected at the corresponding indices. The following is an example for the node 'Dog':

G[dog][0] = [animal, mammal, house, ...] G[dog][1] = [IsA, IsA, AtLocation, ...]

In this case, solving the optimization problem involved an exhaustive search over the entire graph at each question, and thus was extremely slow - to the point where real-time game-play was impractical. Time complexity was made worse by having to count incoming edges at each vertex, as is required for question generation. The naïve approach is presented in Algorithm 1.

Algorithm 1 Version 1 Question Selection

Let $G = (V, E)$ be the semantic network.
Let P be the list of previous "yes" responses.
Let N be the list of previous "no" responses.
Initialize empty priority queue $counts$
for vertex v in G **do**
    if $\exists$ e in $G[v]$, $(v,e) \in P$ and $\forall$ $e \in G[v]$, $(v,e) \notin N$ **then**
        for $(item, rel) \in G[v]$ **do**
            if $(item, rel) \in counts$ **then**
                $counts[(item, rel)].priority+ = 1$
            else
                add $(item, rel)$ to $counts$
                $counts[(item, rel)].priority = 1$
Query regarding highest priority $(item, rel)$ in $counts$

An Efficient, Dynamic Multi-Layered Graph Data Structure

To overcome the problems encountered in our first implementation, we design a dynamic data structure that would hold the graph in a manner well-suited to searches, makes inference over particular relation-types easy by indexing them, and improves time complexity by drastically reducing the number of traversals required for condition verification. Consider the graph in Figure 1.

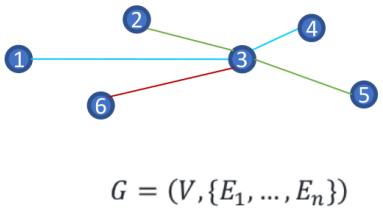

Figure 1: A graph, G with multiple edge-types

This graph is equivalent to the multi-layered graph structure in Figure 2.

We now store G as a hash-table (Python dictionary implementation), where the keys are nodes, and corresponding values are nested lists corresponding to $Neighborhoods$. Each sub-list in the neighborhood represents connectivity over a certain edge-type. So for the graph in Figures 1 and 2, the graph is realized as follows:

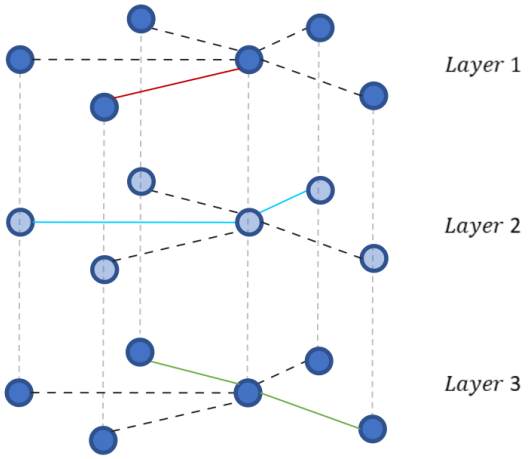

Figure 2: Multi-Layered Interpretation of G

1 : [(), (3), ()]
2 : [(), (), (3)]
3 : [(6), (1,4), (2,5)]
4 : [(), (3), ()]
5 : [(), (), (3)]
6 : [(3), (), ()]

Version 2

Our second implementation makes use of this optimized data structure, thus managing to significantly improve time taken for game-play by solving the optimization problem faster. In addition, we also flip all edges in G, so that counting the number of vertex-neighbors by a given edge-type is a constant time call to a function that returns the length of the corresponding list.

On the down side, this version starts asking very specific questions too soon, as the number of such root-leaf edges are fairly high in certain categories. This implementation does perform fairly well when trying to guess animals/plants, which we attribute to the well-defined hierarchical nature of edges in those categories, drawn from scientific databases.

Version 2.5

The previous versions would continue eliminating or selecting nodes by solving the optimization at each question. However, in a realistic scenario for a 20-questions player, it would start guessing specific instances of subjects that satisfy all conditions as inferred by having asked previous questions. To simulate this, we wanted the program to start guessing if the number of candidate questions left was below a certain threshold.

Given that ConceptNet is crowd-sourced, the distribution of edges over concepts is highly biased towards certain fields. Thus properly defining a universal threshold proved

to be impractical. In addition, random guessing would often pick objects that are not commonplace, yet exist in the database. Naturally, most players are more likely to pick a subject that they encounter frequently.

A third issue with the implementation thus far, is that while having answered yes to a question should satisfy the condition for all future vertices along that path, such transitivity is not explicitly exhibited by the edges of ConceptNet. For instance, we know that

$$dog \textbf{ IsA } mammal \textbf{ IsA } animal$$

Yet, ConceptNet does not contain the direct edge

$$dog \textbf{ IsA } animal$$

Thus, enforcing condition checks based on previous yes responses was wrongly eliminating large chunks of the graph that would by transitivity satisfy the condition. Checking for transitivity by repeated traversal would incorporate a significant decrease in performance in terms of speed. We thus choose to eliminate the condition checks over positive answers, with the intuition that a series of yes answers would definitely lead to a set of candidate vertices that do in fact satisfy all those conditions by implicit transitivity.

Version 3

Having accounted for the transitive satisfaction of positive conditions above, we now attempt to make the guessing more concrete, and natural - in the sense of bringing it closer to human game-playing strategies. We observe that for different categories, specific instances start to occur at varying depths starting from the highest level root node for the category. Thus, to allow for a universal implementation, whenever we encounter a yes answer over a particular relation-type for a node u, we store the vertices connected to u by that relation type in a list L. In case we run out of candidate questions, or are left with fewer than 5 questions (out of 20), we would start guessing items from the list L. Note that in case we descend down a hierarchy, and encounter a series of one or more no answers, we still have the last updated list L, which increases our chances of making a correct guess, given the information up to that point.

Having generated such a list L, when it does come to guessing, we have previously noted that random guessing is not particularly well-suited, given that human players are more likely to choose a subject they are more familiar with - or in other words, is more common in the everyday world. ConceptNet contains edges of the type RelatedTo, which by themselves did not prove very useful for question generation, as they were extremely noisy (incorrect, heavily biased towards certain topics, inconsistencies in the sense of being related, etc.). The most likely cause of such noise is ill-designed crowdsourcing methods used to create the dataset. However, we observed that even though the RelatedTo edges were not accurate, they were larger in number for subjects that would occur frequently in our daily lives. Thus, we use the number of RelatedTo edges terminating in and originating from a vertex as a measure of how common a concept it is.

Having generated a candidate set of guesses, L, we calculate the probability of guessing the i^{th} item in L as:

$$P(u_i) = \frac{\delta_{RT}(u_i) + \delta^*_{RT}(u_i)}{\sum_{u_k \in L}(\delta_{RT}(u_k) + \delta^*_{RT}(u_k))}$$

where δ_{RT} and δ^*_{RT} represent the out-degree and in-degree over the RelatedTo edge-type. Additionally, we smooth the counts of RelatedTo edges by adding 1, as some subjects do not have any RelatedTo edges connecting them. Then our final game-play strategy is summarized in Algorithm 2

Algorithm 2 20 Questions

Let $G = (V, E)$ be the semantic network.
Let $G* = (V, E')$, where $(v, u) \in E' \Leftrightarrow (u, v) \in E$
Let N be the list of negative answers (vertex-relation pairs).
Let $Visited$ be the list of previously visited nodes.
Let $ItemRel = \{(v, i, |N_i(v)|), \forall v \in V(G), \forall i\}$, where i indicates relation type.
Let $L = []$
Let $qC = 0$
Let $i = 0$
Sort $ItemRel$ by $|N_i(v)|$.
while Correct answer not found **do**
    $v_i = ItemRel[i][0]$
    $e_i = ItemRel[i][1]$
    if $len(ItemRel) \geq 1$ and $20 - qC \geq 5$ and $i < len(ItemRel)$ **then**
        if v_i in $Visited$ **then** $i++$
        else
            if v_i satisfies all conditions in N **then** $qC++$
                Ask question using e_i about v_i.
                Add v_i to $Visited$
                if Answer is NO **then**
                    Add (v_i, e_i) to N.
                else
                    Empty $ItemRel$.
                    for x in $N_i(v)$ **do**
                        Add x to L
                        if x not in $Visited$ **then**
                            Add $(x, i, |N_i(x)|)$ to $ItemRel$ for all relation types i.
                            Sort $ItemRel$ by $|N_i(v)|$
                    $i = 0$
            else
                $i++$
    else
        Guess from L where probability of guessing the i^{th} element in L is given as:

$$P(u_i) = \frac{\delta_{RT}(u_i) + \delta^*_{RT}(u_i)}{\sum_{u_k \in L}(\delta_{RT}(u_k) + \delta^*_{RT}(u_k))}$$

Results

To test the ability of our algorithm to play 20 Questions using ConceptNet, we asked three participants to play four games each. Each participant was familiar with 20 Questions and did not need the rules to be explained. Although the goal for the $Answerer$ in 20 Questions is typically to outsmart the $Questioner$, we admit that ConceptNet lacks the majority of topics that can be imagined by typical humans and is biased towards certain categories. Before each participant began, they were told the following: "Please restrict your item to common, nonfiction knowledge. This can include things like animals and people." Subjects were then read the questions produced by the algorithm and their responses were recorded. The results, grouped by participants, are as follows:

Target	Success	Reason/Number of Queries
Crowbar	Failure	not IsA $object$ in db
Printer	Failure	not IsA $object$ in db
Giraffe	Failure	not IsA $placental$ in db
Knife	Success	14
Peanut	Failure	User said not IsA $plant$
Gift Card	Failure	$gift\_card$ lacks most relations
Crab	Failure	IsA $person$ in db
Trumpet	Failure	not IsA $object$ in db
Earbuds	Failure	not IsA $object$ in db
Plate	Success	20
Cockroach	Failure	User said IsA $beetle$
Venus Flytrap	Failure	not IsA $flowering\_plant$ in db

Discussion

Overall, the system performed quite poorly, but every failure can be attributed to either the dataset or the user. In particular, $objects$ were quite difficult to find, and they were common among the subjects' targets. In this version, we only consider one level of hierarchical IsA relations under $object$, so many items that are indeed $objects$ are excluded. Otherwise, the dataset contained one incorrect edge in the case of $crab$, and lacked one necessary edge in the case of $giraffe$. In our game of 20 Questions, a single error in the graph structure or in the human responses causes an entire round to be a failure.

All subjects began by selecting $objects$, but their later rounds displayed more variety and included animals and one plant. If the target is an $object$, the first three questions are "Is it a person?", "Is it an animal?", and "Is it a plant?", so we believe the questions presented may have influenced targets chosen in later rounds. No participant ever used a person as a target, perhaps believing people to be outside of common knowledge.

One aspect of the system that became clear while playing with the subjects was the need for stochasticity in the questioning process. Although our final answers are queried about with a weighted probability distribution, all questions asked before them are deterministic. Although adding stochasticity would undoubtedly make the algorithm find target object in a less than optimal amount of questions, following the exact same route of questions as a previous round makes the game quite uninteresting.

Related Work

20Q

In 2005, Robin Burgener of 20qnet Inc. filed a patent for a method to play Twenty Questions using a neural network (Burgener 2005). In the 20Q variant, the electronic Questioner has access to a set number of pre-defined questions. Players may provide a number of answers besides "Yes" and "No", and each is given a different weight (e.g. "Probably" is positive with a weight of 3, and "Depends" has a weight of 0). After multiple example runs, each target-question combination is given an agreeability metric, or a weighted total of positive minus negative answers.

A neural network with two modes is trained upon this information. In the first mode, the inputs are answers to asked questions, and outputs are target objects. In the second mode, target objects are inputs and questions are outputs. During gameplay, both modes are used to find ideal questions. Initially, the first mode is used to rank target objects. The second mode is then used to find questions to provide maximal information regarding likely objects. For example, if the first network assigns high probabilities to fruits and vegetables, the question "Does it bite?" will not provide meaningful information. With this architecture, it is possible to repeatedly retrain the network using new information gathered across a number of games, at least for the online version. Such information could change the agreeability of certain target-question combinations, or it could conceivably be used to append the weight matrix with an entirely new target object.

Although our approach is quite different from Burgener's model, we wish to retain several of its favorable qualities. Mirroring the output of his first neural network mode, our model's search for a meaningful question begins at specific concept. The question our model then asks will be specific to that concept's neighbors. Additionally, our model should be able to operate relatively quickly at run-time to ensure that it can be played without an uncomfortable amount of waiting. Our model does not involve machine learning, but it has a time-intensive pre-processing stage that would need to be run whenever a new version of ConceptNet is released. Our model is also not limited to pre-defined questions as is Burgener's. Instead, meaningful questions will be constructed on the fly. For example, if our model is focused on an area of the graph featuring vegetables, it might ask "Is it green?", where *green* is an arbitrary adjective that happens to be connected to a large number of highly-rated concepts, according to our metric. We use only a small number of relations, so a handful of question types will cover all constructed questions.

The Large-Scale Structure of Semantic Networks: Statistical Analyses and a Model of Semantic Growth

(Steyvers and Tenenbaum 2005) analyze the structure of three related networks: word associations, WordNet, and Roget's Thesaurus. They find that all three exhibit the qualities of small-world structures, being sparsely connected, having short average path lengths between words, and having strong local clustering. A typical node in each network is connected to very few other nodes. In WordNet, the average path length is 10.56 and the diameter is 27. Its clustering coefficient is 0.0265, which is smaller than the other examined networks but significantly greater than it would be for a random graph of the same size and density. Additionally, they find that the degree distributions of all three networks have power law distributions, the exponent of WordNet's distribution being 3.11.

The results of this study give much needed insight that would be otherwise difficult to learn with our limited resources. While Steyvers et al. use all of WordNet, our model uses the nouns (and directly related adjectives and verbs) of the somewhat larger ConceptNet. Without performing such an analysis ourselves, it is not possible to know how similar our subset of ConceptNet might be. We assume that the structures of our network and WordNet resemble one another. Removing several relations might increase the diameter of our network, but certain properties such as being scale-free should remain unchanged as the data is of similar modality and source. The relatively short average path length and diameter of WordNet are encouraging results, as they indicate a traversal strategy such as ours could easily cover the entire network, given a well-selected starting node. Because WordNet (and most likely ConceptNet) is a scale-free network, it should be possible to find nodes that are particularly useful, both as a starting point for the search and as a tool in finding questions with high information gain.

Concept Description Vectors and the 20 Question Game

(Duch, Szymański, and Sarnatowicz 2005) create a knowledge representation by creating *glosses*, or lists of keywords used to describe words. They generate Concept Description Vectors using words from WordNet and their definitions in a number of dictionaries and encyclopedias. To improve computation time, only words relating to animals are used.

Duch et al. demonstrate the usefulness of their Concept Description Vectors using Twenty Questions. Despite using a semantic network to achieve the same goal as our model, theirs is distinct from ours in its use of definitions instead of relations. While Duch et al. build a graph using keywords as indicators of relatedness, our graph is built using the labeled relations between concepts. The relations of ConceptNet provide a basis for more explicit affiliations between words than a bag-of-words approach, and Duch et al. admit that their system can collect contradictory or confusing information, such as labeling a word as both "large" and "small". Although ConceptNet is largely the product of crowdsourcing and might contain a large amount of nearly useless information, its labeled relations should allow our system to interpret all of its information in a meaningful way.

Conclusions and Future Work

We find that 20 Questions is an excellent method for locating missing and incorrect edges in ConceptNet. Because of this, we intend to extend our program to allow for the modification of its graph through gameplay. Currently, when the algorithm fails to guess an item, one must search the dataset for that item to discover why it was never reached. This process could easily be automated, since all we must do is check the validity of the twenty relevant questions. If the item is missing from the dataset, we may add it and supply the dataset with twenty new edges connecting that item from the previous round. If an edge is missing, a "yes" answer on that edge indicates that it should be added. If not, the program will need to query the user to find out what went wrong, as at least one edge must be incorrect.

In order for such a system to be deployed, it will need to include the capabilities of other modern 20 Questions players. One such capability is the ability to answer "Unsure" or "Sometimes", as ConceptNet (and many other datasets) may have relations which are ambiguous in certain cases. Doing so would require edges to have weights representing likelihoods, which would also aid in crowdsourcing when people provide differing answers to the same question.

Besides deploying our system as a game with a purpose for ConceptNet, it will be possible to use our system on a wide variety of datasets to help people find errors. Such datasets could include medical records and other expert models with highly specialized relation types.

References

Bond, F., and Foster, R. 2013. Linking and extending an open multilingual wordnet. In *ACL (1)*, 1352–1362.

Burgener, R. 2005. Artificial neural network guessing method and game.

Duch, W.; Szymański, J.; and Sarnatowicz, T. 2005. Concept description vectors and the 20 question game. In *Intelligent Information Processing and Web Mining*. Springer. 41–50.

Liu, H., and Singh, P. 2004. Conceptnet — a practical commonsense reasoning tool-kit. *BT Technology Journal* 22(4):211–226.

Singh, P.; Lin, T.; Mueller, E. T.; Lim, G.; Perkins, T.; and Li Zhu, W. 2002. Open mind common sense: Knowledge acquisition from the general public. In Meersman, R., and Tari, Z., eds., *On the Move to Meaningful Internet Systems 2002: CoopIS, DOA, and ODBASE*, 1223–1237. Berlin, Heidelberg: Springer Berlin Heidelberg.

Steyvers, M., and Tenenbaum, J. B. 2005. The large-scale structure of semantic networks: Statistical analyses and a model of semantic growth. *Cognitive science* 29(1):41–78.