Reading Between the Lines: Using Plot Graphs to Draw Inferences From Stories

Christopher Purdy and Mark O. Riedl

School of Interactive Computing, Georgia Institute of Technology, Atlanta, GA, USA
{cpurdy3,riedl}@gatech.edu

Abstract. Intelligent agents designed to interact with humans need to be able to understand human narratives. Past attempts at creating story understanding systems are either computationally expensive or require a vast amount of hand-authored information to function. To combat these difficulties, we propose and evaluate a new story understanding system using plot graphs, which can be learned from crowdsourced data. Our system is able to generate story inferences much quicker than the baseline alternative without significant loss of accuracy.

Keywords: intelligent narrative technologies, computational understanding of narratives, narrative intelligence, artificial intelligence

1 Introduction

Human beings use stories to describe routine or exciting experiences, communicate with each other, and make sense of the world. The ability to understand stories told by others is an important part of human intelligence. If we want digital agents to communicate effectively with human beings, they need to understand the narratives that humans communicate with.

Story understanding is difficult to model largely because of what humans *don't* say: for example, if somebody tells us, "I went to the grocery store and then went back home," most of us will recognize that the speaker also had to park the car at the grocery store, enter the store, and purchase groceries at the store, despite none of this being mentioned in the narrative. The ability to make these inferences is critical in understanding human-authored stories.

Story understanding is hard to define but easy to evaluate if treated as a question-answering problem: given a story with events $e_1...e_n$, what is the probability that predicate p_x is true? As an example, we can have a virtual agent examine a human-authored story: *"John entered the bank. John approached the teller and pulled out a gun. John left the bank with ten thousand dollars. John went to jail."* To test its understanding, we might ask it some questions about the story:

(a) Did John ask the teller for the money?
(b) Was the teller afraid of John?
(c) Did the police arrest John?

We can use the answers it supplies to assess its understanding capabilities.

Humans require a large amount of commonsense knowledge (knowledge about the facts and procedures shared by a human population) in order to perform accurate reasoning. Existing story understanding systems rely either on hand-authored scripts or large commonsense knowledge bases to capture this kind of information. Scripts are useful, structured tools for reasoning, but requiring human authors to hand-engineer granular information for a virtual agent is mentally taxing and time-consuming. Commonsense knowledge bases suffer the same difficulty, have little procedural knowledge, and are also computationally expensive to search through: knowledge bases can contain millions of facts which must be iterated through whenever the agent is asked a question. We want a story understanding system that is computationally efficient and does not rely on hand-authored scripts.

To address these difficulties, we built off of research on *Scheherazade* [4], an open story generation system, in order to build a system that can make commonsense inferences (inferences requiring commonsense knowledge) about stories. This work made two main contributions. First, we present an inference technique that is more efficient than the alternative brute-force inference technique. Our technique accepts stories in natural language questions, unlike prior story understanding systems that assume questions are posed in symbolic form. Second, we compare our technique to the alternative brute-force inference technique in terms of computational complexity, efficiency, and accuracy. We find that our inference technique is significantly more efficient at the expense of some loss in accuracy.

2 Background and Related Work

Efforts in story understanding date back to the mid-1970s with the *Script Applier Mechanism* (SAM) system [3], which provided a script-based framework to produce inferences from natural-language stories. The *Plan Applier Mechanism* (PAM) [11] focused on goals and explanations in order to aid in the understanding of never-before-seen stories. The AQUA system [9] introduced meta-reasoning to question-answering. Mueller [8] applied commonsense knowledge bases and templates to understand script-based news stories. Recent work by Cardona-Rivera et al. [2] applied cognitive models and computational models to reason over computer-generated stories as opposed to human-authored ones. These approaches require hand-built knowledge bases. Deep learning has been used in order to answer questions with a new kind of learning model that combines long short-term memory with inference operators [10].

Scheherazade [4, 5] is a story generation system that learns to tell novel stories from example stories crowdsourced from the general public. This alleviates the limitation of many story generation systems' reliance on knowledge engineering. Of particular interest to us is the knowledge representation used by *Scheherazade* to generate stories: the plot graph. The plot graph data structure describes all of the possible sequences of events that could occur in a specific scenario (including sequences not contained in the training corpus): for example, a plot graph constructed from crowdsourced bank robbery stories can be used to generate its own stories about bank robberies. While the plot graph's original purposes were to aide in story generation, the plot graph is itself a knowledge representation which encapsulates narrative information about scenario-specific stories. We thus posit that plot graphs can be used to demonstrate story understanding.

To draw inferences using a plot graph, the naive approach would be to adopt a sampling or brute-force approach. Given events A and B, what is the probability that event C happened? The system would either have to sample stories generated using *Scheherazade's* story generation algorithm and calculate relative frequencies (while sacrificing accuracy) or compute *all* stories and search through each of them. In the worst case this would result in searching $n!$ stories, where n is the number of nodes in the plot graph. We present an inference algorithm that can intelligently use the structure of the plot graph to draw inferences from input stories.

3 Plot Graphs

A plot graph is a compact representation of the space of possible stories that can describe a certain scenario. Plot graphs are mainly composed of three kinds of elements [4]:

– Event nodes: These are the vertices of the graph. Each of these nodes represents exactly one event (e.g. "John drove to the store" or "John opened the door"). Each node contains a set of sentences that semantically describe the event, which are learned from the crowdsourced exemplar stories.
– Temporal orderings: These are unidirectional edges of the graph. They are a kind of partial ordering that indicate a necessary order for events in a story: if event A is ordered before event B, B may not occur until A has occurred in the story (symbolically, we notate this as $A \prec B$)
– Mutual exclusions: These are bidirectional edges of the graph. They indicate situations where two events cannot take place in the same story (symbolically, we notate this as $A \bigotimes B$). Practically speaking, mutual exclusions result in story branches.

Additionally, plot nodes can be *optional*—they do not need to occur in a story—or *conditional*—they only occur if a linked optional event does not occur first. When $A \prec B$, we call A a *parent* of B and B a *child* of A. The transitive enclosure of the parent relation is the *ancestor* relation and the transitive closure of the child relation is the *descendant* relation.

A story generated from a plot graph is a sequence of events that obeys all of the constraints given by the temporal orderings and mutual exclusions. In Fig. 1, legal stories for the left plot graph are:

– C1, C2, C3, C4
– C1, C3, C2, C4

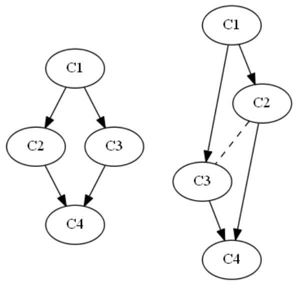

Fig. 1: Two plot graphs. Solid, directional arrows correspond to temporal orderings and dashed lines correspond to mutual exclusions.

The mutual exclusion in the right plot graph constrains the space of legal stories:

– C1, C2, C4
– C1, C3, C4

Fig. 2 is an example of a plot graph showing the procedure for going to a fast-food restaurant.

4 Inference Drawing with Plot Graphs

The purpose of our system is to answer questions of the form: given a story in natural language and a relevant plot graph, what other events (called inferences) could also have occurred in the story, and how likely are each of these inferences? In other words, it fills in gaps in human-authored stories with events left out and lists probabilities of each of its results. This gap filling demonstrates story understanding because the story could contain as few as two events and check for a third in the set of generated inferences. Additionally, and unlike prior story understanding systems, we start from natural language inputs.

The next sections describe the three components of the inference algorithm: (1) Matching input sentences with event nodes in the graph; (2) finding all candidate inferences to be drawn based on the matches; and (3) determining the confidence in each of those inferences.

4.1 Plot Event Matching

Our system first attempts to find matches between input sentences and event nodes in the plot graph. Each event node is composed of multiple sentences, so we select one representative sentence from each node to compare with the input sentences. Given sentence X and sentence Y, our system needs a numerical representation for the semantic similarity between the two. Our system employs the Stanford Parser [6] to find specific parts of speech to compare. Specifically, our system uses the parser to find the nouns and verbs of the sentence. It then uses

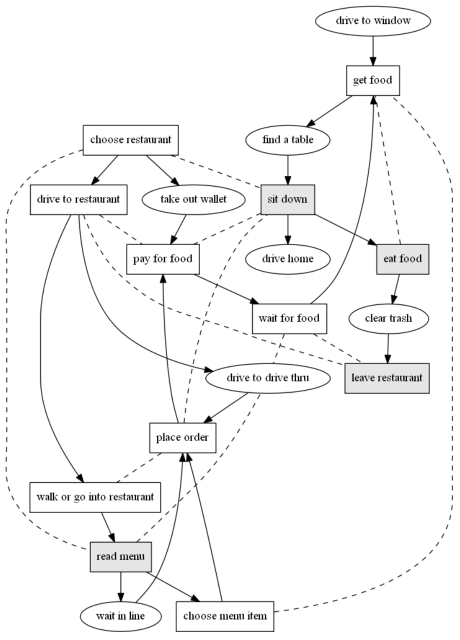

Fig. 2: The plot graph describing a trip to a fast-food restaurant. Ovals and boxes are events (the white boxes are optional and the gray boxes are conditional). Solid lines are temporal relations and dashed lines are mutual exclusions.

a Word2Vec [7] model trained on the Google News corpus to produce word embeddings (vector representations) of the selected tokens. From there, summation vectors are created from these Word2Vec vectors:

$$NounVector_X = \sum_{i=1}^{n_1} Noun_{X_i} \tag{1}$$

$$VerbVector_X = \sum_{i=1}^{n_2} Verb_{X_i} \tag{2}$$

where $Noun_X$ is the set of noun Word2Vec vectors from sentence X and n_1 is the number of nouns in X, and $Verb_X$ is the set of verb Word2Vec vectors from sentence X and n_2 is the number of verbs in X. With cumulative noun and verb vectors from both sentences, we can compute $NounSim$ and $VerbSim$ as the cosine similarity between X and Y, resulting in numbers in the range $[-1.0, 1.0]$.

For our purposes, different parts of speech are more important to the meaning of the sentence than others. In a sentence describing an event, the verbs are the most important tokens to analyze, so we assign numerical weights α and β to $NounSim$ and $VerbSim$ such that $\alpha + \beta = 1$:

$$Sim = \alpha * NounSim + \beta * VerbSim \tag{3}$$

We have assigned $\alpha = 0.4$ and $\beta = 0.6$ to give slightly stronger weight to verb similarity.

By comparing two sentences this way, we are left with a single scalar value. However, every sentence will match to every other sentence with some similarity. By comparing the scalar against an empirically determined threshold, the system is able to filter out false matches. For our purposes, a threshold value of $\gamma = 0.6$ performs adequately. If one input sentence matches (with similarity above γ) with multiple event nodes in the plot graph, we select the top match. Multiple event nodes may have an equal highest match to an input sentence, in which case the following steps are performed with each top match.

4.2 Locating Candidate Inferences

At this stage, our system has a list of event nodes, E, that were matched with sentences from the input story. The system then begins searching for candidate inferences in the plot graph.

For every pair of events a and b contained in E, our system records all other events in the graph that can occur after a and before b. This list, I_{ab}, represents possible inferences that can be drawn from information contained in a and b. Our system must then assign a confidence measure to these inferences. Given a single inference I_{ab_i}, our system employs the following heuristics to determine how likely that inference is correct given a and b:

1. If I_{ab_i} is an ancestor of b, then I_{ab_i} is guaranteed to have occurred unless I_{ab_i} is mutually excluded with another possible parent of b. If the latter is the case, the probability is inversely proportional to the number of parents mutually exclusive with I_{ab_i}.
2. Otherwise, if there are no temporal orderings between a and I_{ab_i}, I_{ab_i} has a very high chance of occurring, since it can be inserted anywhere into the story without consequence, unless a mutual exclusion prevents it.
3. Otherwise, if a is a parent of I_{ab_i}, the likelihood of I_{ab_i} being selected is inversely proportional to the distance between a and I_{ab_i}, since additional mutual exclusion constraints reduce the probability of an event being selected.

Finding nodes' ancestors and successors is performed with breadth-first search on the graph of temporal links.

In case multiple heuristics apply for the inference, the one highest on the list is selected. Depending on which heuristic is selected for I_{ab_i}, the inference is given confidence according to a confidence category (since assigning specific numerical values is difficult with such limited information). These confidence categories are given numerical intervals ahead of time; for example, an inference that falls under the conditions of heuristic 3 might be determined to have a "Near 50% confidence". The complete list is given in Table 1.

Table 1: List of confidence categories and their confidence intervals.

Confidence Category	**Confidence Interval**
100% confidence	[1.00, 1.00]
High confidence	[0.7, 1.00]
Near 50% confidence	[0.3, 0.7)
Low confidence	[0.0, 0.3)

4.3 Arriving at an Answer

The above approach produces sets of "local" inferences, but does not consider the larger scope of the story. An inference drawn from two events in the input story might be disproven by evidence given by another set of matches. For example, upon hearing that "John loves Sally" and "John treated Sally to dinner", a reasonable inference might be that "John and Sally are married," but if we learn that "John took Sally to the vet", our original inference should be discarded. To remove globally inconsistent references, we must also compute all "impossible" inferences from each match and compare those with candidate inferences from other pairs. We can accomplish this by looking at the mutual exclusions of every event match.

We first consolidate all of the inference sets I_{ab} into one larger set:

$$I_{candidate} = \bigcup I_i \tag{4}$$

where I_i is the ith event pair and $I_{candidate}$ is the resulting superset. If two equivalent inferences get merged in the set union, the higher confidence measure is kept. We then can consolidate all of the learned "impossible" events by performing set union:

$$I_{impossible} = \bigcup MutExc_i \tag{5}$$

where $MutExc_i$ is the set of mutually exclusive events with event E_i in the matched event set. Finally, the set of legal inferences is given as the set subtraction of the impossible inferences from the candidate inferences:

$$I_{final} = I_{candidate} - I_{impossible} \tag{6}$$

To determine the probability of event e_q one merely has to look up the confidence interval of the event in I_{final}.

5 Evaluation

The motivation for using plot graphs was to harness the power of crowdsourced knowledge engineering for use in story understanding. Without our inference technique, the only known way to produce inferences with plot graphs is to perform an exhaustive search. This brute-force story understanding method is computationally inefficient (considering that the number of possible stories from a single plot graph is, in the worst case, proportional to $n!$, where n is the number of event nodes in the graph) but completely captures the information of the plot graph. For the purposes of this evaluation, we treat the results of this method as ground-truth. We thus evaluate our system's inference algorithm's complexity and accuracy against the brute-force baseline. We do not include the similarity matching in the evaluation to control for variance in human-written sentences.

For these evaluations, we test on two crowdsourced plot graphs: one graph describing a restaurant scenario (shown in Figure 2) and one truncated graph describing a bank robbery. Both plot graphs were generated using the data and algorithms from [4]. It can produce 21,016 unique trajectories (stories). The bank robbery plot graph had six event nodes and all related links manually removed from it to produce a simplified version for tractability purposes in evaluation. It was found to be intractable to generate all stories for the unmodified bank robbery plot graph, which had 34 nodes and could produce over 30 million unique stories. The modified plot graph can produce 127,116 unique trajectories.

In this section, we first provide a description of the brute-force method, follow with an analysis of the computational complexities of our inference method and the brute-force baseline, and conclude with analysis on the inference method's accuracy with respect to the brute-force baseline.

5.1 Brute-Force Baseline

This method provides an exhaustive answer to the following question:

> Given events $e_1...e_n$, what is the probability of event e_i, with $e_i \notin \{e_1...e_n\}$, also having occurred?

The answer is determined by generating all possible stories in the plot graph containing $e_1...e_n$ and counting the frequency of e_i in-between e_1 and e_n (since we are not concerned with what occurs before or after the story). We can break down this method as follows:

1. Generate all possible stories from the plot graph.
2. Find the set of all stories, E_n, that contain each event $e_1...e_n$.
3. Count the number of stories within E_n that contain e_i in-between them and then divide by $\|E_n\|$ to obtain a relative frequency.

5.2 Complexity Analysis

Both the brute-force method's and the inference method's complexity depend on four factors:

1. The set of events in the plot graph, N, with size n
2. The set of temporal links in the plot graph, L, with size l
3. The set of mutual exclusions in the plot graph, M, with size m
4. The set of events in the input story, S, with size s

Below we describe how computational complexity of the method is affected by these steps.

Brute-Force Runtime Complexity The majority of the complexity of this algorithm comes from generating all possible stories from the plot graph. The complexity is directly proportional to the number of stories to be generated. The worst-case number of stories represented by a plot graph is $\mathcal{O}(n!)$, since legal stories are permutations of each event in the plot graph. The mutual exclusions and temporal orderings place additional constraints on this worse-case scenario.

It is difficult to compute an average-case complexity with l temporal links, since the exact configuration of links affects the graph structure differently; subsets of nodes can be arranged in sequence reducing the number of story permutations, or in trees increasing the number of story permutations. Determining the number of legal stories is equivalent to finding the number of topological sortings of a partial order graph (which, in itself, is equivalent to counting the number of linear extensions of a partial-ordered set), which has been shown to be a $\sharp P$-complete problem [1]. The effect of the number and placement of edges is unknown, but we denote it as $f(n, l)$.

Analyzing the effect of mutual exclusions is an easier problem. A single mutual exclusion causes branching in the plot graph. If we treat this as partitioning the graph, we can approximate the number of stories and thus the complexity of generating all stories as:

$$Complexity \approx \mathcal{O}(f(n,l) * (n-m)!) \tag{7}$$

which is at best $\sharp P$-complete and at worst factorial. The remainder of the algorithm's runtime is eclipsed by the generation step; subsequent steps of the brute-force baseline technique pertain mostly to scanning over the set of all possible stories.

Inference Algorithm's Complexity To reiterate, the main steps in our system's inference algorithm are as follows: (1) match input events with event nodes in the graph, (2) find all candidate inferences to be drawn based on the matches, (3) determine the confidence in each of those inferences, and (4) provide global analysis of the local inferences. We address each of these complexities separately.

The first step is of $\mathcal{O}(n * s)$ complexity, where s is the number of input sentences, since s story events are compared against every node in the plot graph. The second step requires lookup in the table of mutual exclusions, resulting in complexity $\mathcal{O}(m)$. The complexity required for the third step depends on which heuristics are applied, but we can provide an upper bound. The most difficult heuristic to apply is Heuristic 3, wherein our system must attempt to find the shortest distance between two nodes guaranteed to be connected by temporal edges. Since this is accomplished with breadth-first search, this step has worst-case time complexity $\mathcal{O}(l+n)$. The final step, which involves set union and difference, depends on the number of event nodes and number of inference sets (which, in turn, depends on s), resulting in an upper-bound complexity of $\mathcal{O}(n * s)$.

Composing the individual complexities of these steps together, we get a final complexity of:

$$Complexity = \mathcal{O}(n * s + m + l) \tag{8}$$

Thus, our inference algorithm runs in linear time versus the brute-force baseline's $\sharp P$-complete or factorial time.

Runtime Comparison In terms of runtime, there is a noticeable improvement over the brute-force method. In practice, when run on a standard desktop/laptop machine with an Intel Core i5-4200H CPU and 8.00GB RAM, our system requires only around 1% of the time required by the brute-force method on average, resulting in a 60x–100x increase in performance. This performance increase occurs in both plot graphs tested. This is important for conversational virtual agents where a user might expect a relatively quick response to a question that seems to the user to be trivial but it in fact computationally expensive.

If the brute-force method is allowed to perform the story generation step in advance and save data to disk, this improvement is less pronounced: our system performs only 0.5x–8x faster on average for the restaurant plot graph and 7x–20x faster for the bank robbery plot graph, but this still confirms our intuition that the brute-force's complexity grows rapidly with respect to the number of event nodes in the plot graph. When we consider that brute-force computation on large plot-graphs is likely to be intractable, this result is promising.

5.3 Accuracy Analysis

We randomly sampled 2000 legal stories from the fast-food and simplified bank robbery plot graphs and randomly removed events from each of them. We pit the brute-force baseline versus our system's inference algorithm to see how well each of them was able to correctly infer the removed events. We treat the brute-force method as the ground truth since it exhaustively iterates over all possible stories contained by the plot graph. Accuracy is determined by the coalignment between both method's results. Specifically, the brute-force method will return a value between 0 and 1 which indicates the relative frequency of each removed event: this value is compared against the numerical interval associated with our system's inferences' confidence categories. Accuracy is computed as the percentage of times our inference technique agrees with the brute-force technique as to which confidence interval an event belongs to over the 2000 trials.

Table 2 displays the results from the accuracy evaluation. With four likelihood classes, a random baseline would achieve 25%. When evaluating the brute-force method, we had cached the complete list of stories in advance, so the

Table 2: Results of evaluation with two plot graphs.

Plot Graph	**Graph Size (# Nodes)**	**Accuracy**	**# of Brute-Force String Matches**	**# of Heuristic String Matches**
Restaurant	19	0.7395	430,931,917	54,060,547
Bank Robbery	28	0.6980	5,705,264,000	285,024,000

number of computations only pertains to those done with string matching. The number of brute-force computations thus describes the number of string matching operations. The number of computations in the inference method relates to the number of heuristics called. Specifically, since each of the implementations of the heuristics relies on finding nodes in the graph, the number of computations in the inference method is equal to the number of nodes examined in breadth-first search.

The accuracy decreases slightly when the size of the graph increases, but this is expected since the complexity of the graph increases vastly with insertion of event nodes. This accuracy can be improved by developing more intelligent heuristics. Importantly, our inference method outperforms the brute-force method in speed by a factor of ten with only a 30% tradeoff in accuracy.

6 Conclusions

Story understanding systems have previously relied on hand-built knowledge-bases or models, making their utility constrained by the ability of human authors to hard-code scripts or commonsense knowledge bases. Plot graphs learned from crowdsourcing provides a means of acquiring commonsense procedural knowledge for story understanding. Plot graphs compactly encode a space of possible stories about a given situation, but exhaustively searching this space in order to make question-answering inferences can be intractable. Our heuristic technique makes question-answering inferences from plot graphs without the extensive computation, reducing the runtime by a scale of 10x–100x for only a 30% tradeoff in total accuracy. We take the additional step of performing question-answering about stories that are provided in natural language as a means of bringing story understanding closer to real world applications such as virtual agents and chat bots. Our story understanding system is the first of its kind to rely on crowdsourced story knowledge instead of human-authored knowledge-bases, increasing the potential for understanding of a large range of story scenarios in real time.

7 Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. IIS-1350339. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

1. Brightwell, G., Winkler, P.: Counting linear extensions. Order 8(3), 225–242 (1991)
2. Cardona-Rivera, R., Price, T., Winer, D., Young, R.M.: Question answering in the context of stories generated by computers. Advances in Cognitive Systems 4, 227–245 (2016)

3. Cullingford, R.: SAM and micro SAM. In: Schank, R., Riesbeck, C. (eds.) Inside Computer Understanding. Erlbaum (1981)
4. Li, B., Lee-Urban, S., Johnston, G., Riedl, M.O.: Story generation with crowdsourced plot graphs. In: Proceedings of the 27th AAAI Conference on Artificial Intelligence. Bellevue, Washington (July 2013)
5. Li, B., Thakkar, M., Wang, Y., Riedl, M.O.: Data-driven storytelling agents with adjustable personal traits and sentiments. In: Proceedings of the 7th International Conference on Interactive Digital Storytelling (2014)
6. de Marneffe, M.C., MacCartney, B., Manning, C.: Generating typed dependency parses from phrase structure parses. In: Proceedings of the 5th international conference on Language Resources and Evaluation (2006)
7. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Burges, C.J.C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 26, pp. 3111–3119. Curran Associates, Inc. (2013)
8. Mueller, E.: Understanding script-based stories using commonsense reasoning. Cognitive Systems Research 5(4) (2004)
9. Ram, A.: AQUA: Questions that drive the explanation process. In: Schank, R., Kass, A., Riesbeck, C. (eds.) Inside Case-Based Explanation. Erlbaum (1994)
10. Weston, J., Chopra, S., Bordes, A.: Memory networks. In: Proceedings of the 2015 International Conference on Learning Representations (2015)
11. Wilensky, R.: PAM and micro PAM. In: Schank, R., Riesbeck, C. (eds.) Inside Computer Understanding. Erlbaum (1981)