Driving Story Generation with Learnable Character Models

by

Matthew Paul Fay

B.S., Purdue University (2009)
S.M., Massachusetts Institute of Technology (2012)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2014

Author…….....
Department of Electrical Engineering and Computer Science
August 7, 2014

Certified by……..
Patrick H. Winston
Ford Professor of Artificial Intelligence and Computer Science
Thesis Supervisor

Accepted by…….
Leslie A. Kolodziejski
Chair of the Department Committee on Graduate Students

Driving Story Generation with Learnable Character Models

by

Matthew Paul Fay

Submitted to the Department of Electrical Engineering and Computer Science
on August 7, 2014, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Stories are a ubiquitous part of the human experience and an essential component of human intelligence. Stories serve countless functions, working in tandem with our language facilities to enable communication, problem solving, imagination, and more. Stories are eminently cross disciplinary, fueling how people think about everything from chemical reactions and novels to social encounters and politics. If we're to develop a comprehensive computational system mirroring human intelligence, the system must be able to both understand stories and go beyond that, using stories to empower how it thinks about the world.

On a high level, a key component of story understanding is story generation. In human thought generation enables us to imagine, solve problems and make decisions. Therefore, to take computational story understanding to the next level, this research makes a number of contributions centered on story generation. In particular, this includes the development a novel story generation system that is capable of creating new and interesting stories by learning from a human written corpus of stories. The design of robust learnable character models facilitated this development. The character models are capable of handling characters that span a number of genres including romantic comedies, fairy tales, warfare, and Shakespearean tragedies. The work also includes methods for automatic learning of characters traits in both an unsupervised way using topic modeling and a semi-supervised way using a new method of alignment-based trait learning.

Through my research these algorithms were tested using the Genesis system for story understanding. The results from the research illustrate how the robust methods are able to handle a variety of situations such as internally conflicted characters, characterization spanning genres, and conflicting character goals. Critically, the research demonstrates how story generation is important to story understanding through an example of using generation to better understand an existing story.

Dissertation Supervisor:

Patrick H. Winston
Professor, Electrical Engineering and Computer Science

Dissertation Committee:

Boris Katz
Principal Research Scientist, MIT Computer Science and Artificial Intelligence Laboratory

Nick Montfort
Professor, Comparative Media Studies

Acknowledgements

Without the support of people and organizations, this work would not have been possible. My advisor Patrick Winston was instrumental to my success. The mentorship, guidance, and education he provided shaped the course of my graduate experience. I also received valuable mentorship from my committee members, Boris Katz and Nick Montfort.

The Intelligence Advanced Research Projects Agency supported early stages of this work through the "Integrated Cognitive-Neuroscience Architectures for Understanding Sensemaking" project under IARPA-BAA-10-04. Funding and collaborations under this project helped spur ideas and development.

The Defense Advanced Research Projects Agency provided continued support through the "Narrative Networks" project under DARPA-BAA-12-03. This support enabled significant progress on this work.

Collaborations and discussion with colleagues was essential. Work and discussions with Gabriel Zaccak helped lay the foundations for my research. Additionally discussions with members of the Genesis group were extraordinarily valuable. They include Mark Finlayson, Adam Kraft, Eren Sila Sayan, Susan Song, Robert McIntyre, and Dylan Holmes.

Finally, I would not have been able to complete this work without the support of my wife and editor, Emily Fay.

Table of Contents

List of Figures

"All the world's a stage,
and all the men and women merely players"
- William Shakespeare

Chapter 1 – Vision

A long sought goal of artificial intelligence has been successfully capturing the essence of human intelligence. Since its inception as a field, AI has made many steps towards developing machines with the intellectual capabilities of people. However, despite great strides in areas such as chess, computer vision, and data analysis, we are still nowhere near creating systems able to think as humans do nor even to carry on human-like conversations able to pass the Turing test (Turing, 1950). A key missing ingredient is the capacity to understand stories.

Therefore, in order to contribute towards the goal of advancing artificial intelligence, I have focused my research efforts on improving the state of the art in story understanding. I am interested in how humans not only understand but also create stories that are understood by others. I'm interested in how the combined capability of both understanding and creating stories plays a fundamental role in learning, problem solving, and more.

Story Generation in Intelligence

If we are to make progress in making artificial intelligences behave on par with humans, we need to consider how we can advance computational story understanding (Winston, 2011). Specifically, I target the human ability to generate stories because story generation plays a vital role in intelligence, fueling our ability to imagine, solve problems, and make decisions. When people are faced with choosing a course of action, they are able to imagine possible outcomes of each action in a sense telling themselves a story of what would happen. Feelings such as surprise are often triggered when reality conflicts with our imagined story. These ideas are expanded upon in Chapter 2.

In addition to being useful on its own, story generation also empowers story understanding. When trying to understand a story people can ask and answer questions such as "What would have happened if…" For example, in the story of *Hansel and Gretel* we know that Gretel rescues her brother from being eaten by the witch. However, we can easily imagine what might have befallen Hansel, had Gretel been unsuccessful or even removed from the tale entirely. This ability to reimagine variations on existing stories bolsters our ability to understand them. In the case of *Hansel and Gretel* imagining the story without Gretel serves to emphasize Gretel's heroism.

This example also illustrates the importance of characters to stories. I posit that if we are to develop comprehensive models of story understanding, we need to develop robust character models.

Characters in Stories

Characters in stories are ubiquitous being the focal points around which the events of a story's plot unfold. When people read or hear stories, they automatically interpret the actions of characters and predict both what the character may do and what may happen to the character.

A story is comprised both of the events that occur and the entities that are affected or causing the events. Events require entities to exist, but entities can exist independent of events. Entities themselves can be almost anything: people, animals, plants, books, inanimate objects, animate objects, and the weather. Characters are a subcategory of entities composed of entities capable of acting with intention. Some theorists argue that a narrative requires characters, whereas others accept narratives consisting solely of non-character entities (Abbott, 2008; Bal, 1997) When considering how humans use stories to understand the world, even inanimate things such as chemical compounds could be considered characters with intention when understanding stories about chemistry. For example, an acid acts with intent to react with a base.

Intent

In the light of the importance of stories to human intelligence, if we are to build truly intelligent machines then we need to tackle the problem of story understanding. A key component of how people understand stories is how we imagine. Because of the importance of characters to stories, a large piece of this work will be the development of robust character models that can be learned by reading a corpus of stories. These models are flexible, making them useful both for the generation and analysis of stories.

In the rest of this document I build upon my thesis. First, I discuss the importance of stories to human intelligence and review the current state of computational story understanding and composition. Then, I describe my advancements in creating learnable character models which can improve the ability to understand stories on the level of character. These models can be used to enable character traits to be automatically learned from a corpus of stories. These character models and learned traits fuel story generation. A key component of the story generation process is a plot weaving algorithm which is capable of rapidly searching a combinatorial space in order to successfully weave the plots of characters together. Finally, I discuss how the models and methods are capable of handling a variety of situations and how story generation can improve the state of the art in story understanding.

Applications

In this thesis I focus primarily on story understanding and story generation as they relate to human and machine intelligence. However, the techniques are not bound to a specific domain and could be used in a number of related fields.

In the world of game design, procedurally generated gameplay aspects are becoming increasingly popular (Hendrikx & Meijer, 2013). For example, in Minecraft, the players can explore an entirely new, randomly generated world every time they play. The world is vast but nearly entirely lacks plot. Story generation could help to push procedural generation of game

content in a new domain, enabling games to dynamically create new stories, goals, and characters for players to interact with.

In the domain of interactive drama, systems such as *Façade* drive a unique narrative experience by allowing a user to interact with characters over the course of an unfolding drama (Mateas & Stern, 2003). *Façade* allows the user to influence the outcome of a story by modeling character interactions and having a higher level drama management system. My work in learning character models from experience could help to enable such systems to learn from other stories in order to construct a wider variety of possible narratives.

In the field cognitive science, researchers are trying to uncover how the mind works. A somewhat recent area of interest is how humans think about stories (Dodell-Feder, Koster-Hale, Bedny, & Saxe, 2011). Recent work has shown that people's brain activity noticeably varies when someone reads a coherent story versus an incoherent one. Through the use of a sophisticated story generation system, you could generate stories with specific attributes such as level of coherence, sophistication of word choice, etc. These tailored stories could then be used to test exactly how the brain responds to specific changes between stories.

This small set of examples illustrates how my techniques and story generation in general can be a valuable both in artificial intelligence but many other areas of interest.

Samples of Results

In this section I provide a number of samples of the capabilities of my work.

The development of the character models has enabled new ways for story understanding system to understand stories. One example of this is the ability to compare characters from across stories, even stories from different genres. Figure 1 shows an example of a similarity matrix that can be automatically generated using the techniques I've developed. These character models are discussed in more detail in Chapter 5.

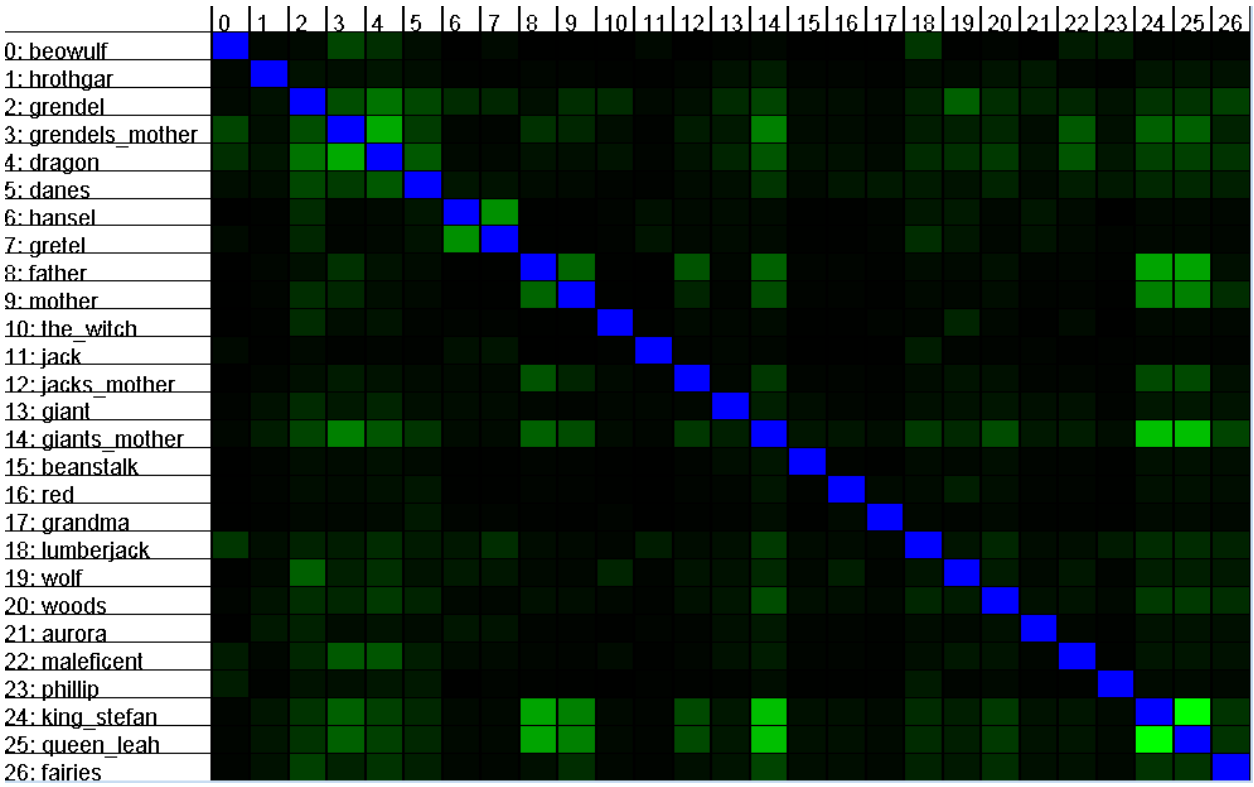
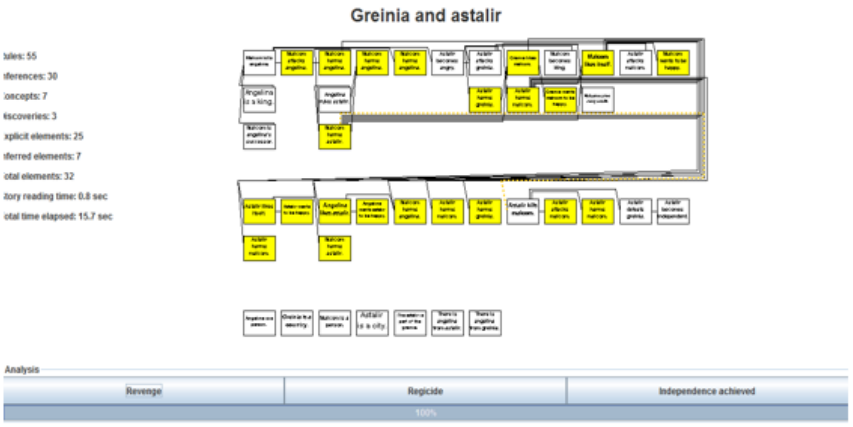


Figure 1 - Similarity Matrix of Character Comparisons
Using my developed character models, a large number of characters can be easily compared. Similarity matrices such as the one shown above can be automatically generated after the system reads one or more stories by doing pairwise comparisons between all the characters in a collection.

In an effort to make character models learnable, I investigated the concept of traits and characterizations. I present in Chapter 6 methods for learning such information directly from a corpus of stories. Examples of traits include the concept of *Vicious Ambition*, which is a trait that embodies the idea of a character that wants to gain power through murder.

An important part of the story generation process is being able to simulate character based on learned experience. These character simulations serve to drive the plot of the story forward. For example, in simulating character such as Scar from *The Lion King*, the system may decide that Scar kills Mufasa so Scar can become king. The character simulations are combined with an algorithm called plot weaving which allows characters' plot threads to be tied together into a single consistent story.

The story generation process on the whole can create a vast number of new stories even from a small corpus of stories. An example of a generated story is shown in Figure 2. This story was generated by combining information learned from across a number of stories and genres such as a story of *The American Civil War* and the genre Shakespearean tragedies.



Greinia is a country.
Astalir is a city in Greinia.
Angelina is from Astalir.
Angelina is from Greinia.
Angelina becomes the Queen of Greinia.
Angelina rules Greinia.
Malcom is Angelina's successor.
Malcom kills Angelina.
Malcom becomes King of Greinia.
Astalir rebels.
Astalir attacks Greinia.
Malcom scares away Astalir's forces.
Astalir attacks Malcom.
Astalir kills Malcom.
Astalir defeats Greinia.
Astalir becomes independent.

Figure 2 - Sample Generated Story
Using the story generation techniques I've developed, stories can be generated by learning from a preexisting corpus. This figure shows an example of a generated story which combines knowledge learned from across stories and genres such as a telling of *The Civil War* and Shakespearean tragedies.

In addition to generating stories for purely creative purposes, stories can also be created in order to better understand an existing story. For example, in the original story of *Hansel & Gretel*, Gretel saves herself and her brother by killing the witch. The system can now imagine an alternative telling of the story without the character of Gretel. In this case Hansel dies at the hands of the witch. This reimagining of the story reinforces the idea of Gretel as the heroine in the original tale. I discuss the details of my story generation process in Chapter 7.

Hansel & Gretel

Mother dislikes Hansel and Gretel.
Mother wants to kill Hansel and Gretel.
Mother convinces Father to abandon Hansel and Gretel in the woods.
Father abandons Hansel and Gretel in the woods.
Hansel and Gretel are lost.
Hansel and Gretel become hungry.
Hansel and Gretel find the house made of candy.
Hansel and Gretel begin eating the house made of candy.
The witch lives in the house made of candy.
The witch enslaves Hansel and Gretel.
The witch becomes hungry.
The witch wants to eat Hansel and Gretel.
The witch fattens up Hansel.
The witch tries to push Gretel into the oven.
Gretel frees herself.
Gretel pushes the witch into the oven.
Gretel kills the witch.
Gretel frees Hansel.

Hansel w/o Gretel

Mother dislikes Hansel.
Mother wants to kill Hansel.
Mother convinces Father to abandon Hansel in the woods.
Father abandons Hansel in the woods.
Hansel and is lost.
Hansel becomes hungry.
Hansel finds the house made of candy.
Hansel begins eating the house made of candy.
The witch lives in the house made of candy.
The witch enslaves Hansel.
The witch becomes hungry.
The witch wants to eat Hansel.
The witch fattens up Hansel.
The witch pushes Hansel into the oven.
The witch kills Hansel.
The witch eats Hansel.
Father discovers Hansel died.
Father kills himself.

Figure 3 - Example of Story Generation fueling Understanding
Story generation can serve as a powerful tool for improving story understanding. This figure shows an existing rendition of *Hansel & Gretel* on the left and a generated story which is similar except that the character of Gretel has been left out of the story. The generation improves understanding of the original story by highlighting Gretel's role as Heroine.

Chapter Summaries

Chapter 2 – Stories in Human Intelligence

The chapter reviews highlights from the literature on the study of human intelligence. I discuss work from anthropology, linguistics, psychology, and artificial intelligence to illustrate the importance of stories to intelligence.

Chapter 3 – Related Work in Story Generation

This chapter contains a literature review of work in computational story generation. The goal of this chapter is to highlight trends, successes, and failures of previous story generation approaches. The purpose of this review is to both find key aspects of story generation system that could be useful in improving story understanding as well as to find areas where improved artificial intelligence and story understanding can be used to bring new ideas to the field of story generation.

Chapter 4 – The Genesis Story Understanding System

In this chapter, I describe the Genesis system for story understanding. The Genesis system represents over a decade of work by a large group of researchers with the goal of modeling how humans understand and interpret stories.

Chapter 5 – Character Modeling

I discuss the capabilities of my novel character modeling techniques implemented into the Genesis system. The models enable Genesis to achieve an array of new abilities in better understanding individual characters and how characters are related through traits. This improved level of character understanding also enables new methods of analysis and comparison. Additionally these models set the stage for trait learning and plot generation.

Chapter 6 – Learning Character Traits

This chapter contains a core contribution of my work, analogical trait learning. The work draws on techniques from story understanding, artificial intelligence, machine learning, and natural language processing. The result of this work allows for the automatic unsupervised learning of characters, traits and motifs from a corpus of stories. This work is scalable allowing ever larger corpuses to be analyzed.

Chapter 7 – Generating Stories based on Past Experiences

I describe the development of my new story generation system, which is capable of combining the Genesis story understanding system and my character modeling techniques to create a powerful yet adaptive story generation system. This includes the creation of the plot weaving algorithm which represents a core contribution of my work. This work leverages the character models and learning techniques that I've developed. Additionally I present and discuss results of my work.

Chapter 8 – Contributions

I review the major contributions of my work and discuss future directions.

Chapter 2 – Stories in Human Intelligence

If we are to develop a comprehensive artificial intelligence capable of human level thought, we need to understand the powerful story understanding capabilities that humans have. In this chapter I discuss why story understanding, and more specifically story generation are important parts of human intelligence. Naturally this will be a brief summarization, as the challenge of understanding human intelligence is an entire field unto itself. The works I discuss are limited to key/core concepts that inform how I approach the problem of creating a system capable of modeling, understanding, and generating stories. I reviewed several of these concepts in a different context in my earlier work on story comparison and alignment (Fay, 2012a). This chapter builds upon my earlier project by focusing on the importance of generation in human imagination, which fueled my decision to create a computational story generation system as a means to improve computational story understanding in particular and artificial intelligence in general.

Importance of Language

Language is an extremely versatile tool that helps fuel high level cognition. People use language in a vast number of ways from communication to abstract thought, archiving knowledge, learning chemistry, and writing books. Some suggest that language plays a key role in enabling human intelligence (Tattersall, 2008; Winston, 2011).

Language has a number of important aspects. Language is expressive, enabling limitless ideas from even a small selection of words. Additionally, human language is productive, meaning that even language with a small set of phonemes can be used to create novel words and sentences (Trask, 2003). On a similar note, Noam Chomsky, has theorized that it is not language and symbolic reasoning alone that enable our intellectual prowess. Humans seem to be unique in our ability to combine two ideas to form a novel one, potentially ad infinitum. This closely mirrors the linguistic "merge" operation. This ability enables us to combine knowledge in incredibly important ways. For example, applied to mathematics, it allows us to combine that 3+1 is 4, which is one greater than 3, and 5+1 is 6, which is 1 greater than 5, into the general statement x+1 is 1 greater than x. This knowledge can then be applied to any value, e.g., we know that if x is 7 then x+1 is 8 (Chomsky, 2010; Winston, 2011). These capabilities likely fuel our ability to generate new stories by combining pieces of old ones.

According to Roger Schank, language seems to be integral to our ability to think in general. He suggests that language enables our ability to index, retrieve, and process information. To Schank, the core of intelligent behavior is the ability to recall relevant information that will help an individual make sense of the current situation. Beyond language itself, however, stories are a form of knowledge and it is stories that enable people to operate in the world (Schank, 1990).

What is a story?

Before proceeding, it is important to establish critical terminology that I use throughout the course of this thesis. Defining a seemingly straightforward term such as *story* is by no means

trivial. Even within the fields of literature and narrative theory there is no clear cut agreed upon definition. (Forster, 1962; Prince, 2003). To make thing even more complicated, the colloquial use of the term *story* is often interchangeable with *narrative*. However, to understand the scope of this research it is necessary to distinguish between them. In general, narrative is a particular representation of events, typically having two major components: the story (or *fabula*), and the narrative discourse (Abbott, 2008). The story is the content of the narrative, essentially it is a chronological collection of one or more events. This is distinct from the narrative discourse that is the way in which those events are represented, ordered, told, stylized, and expressed. (Abbott, 2008; Prince, 2003) For my work in particular, I focus on computational systems able to understand and create the content of the narrative. The discourse is incredibly important and difficult task on its own. While parallel work in the Genesis group has worked to approach problems in discourse, it is outside the scope of my own research (Sayan, 2014).

There is also a secondary use of *story* in this work, specifically in phrases such as *story understanding* and *the story level*. These phrases in particular, refer to the ability of humans or machines to derive meaning from beyond a surface level interpretation. This inevitably includes the human ability to draw upon past experience and knowledge to draw conclusions about events that are not present in the language alone. For example, if "Mary kissed John" and then "John smiled" we may infer that "John smiled because Mary kissed John" or perhaps "John likes Mary".

A similarly complex term is the word *plot*. Most definitions of plot are similar to the content definition of story. Meaning that *plot* is the events of the story (Abbott, 2008). However, for clarity it is worth noting that sometimes *plot* is given a slightly different meaning aligning it with the causality of events and in some cases even how the events are represented (Abbott, 2008; Prince, 2003). For my work I will stick with the more common definition of plot being an ordered collection of events.

Another important consideration is where stories come from. While events occur around us every day, they do not immediately become a story. For a story to exist, an author must come up with a selection and representation of those events to tell. This creation of the representation distinguishes the reality from the story. Choosing events to include in a story is not a simple task either even when retelling what happened over the course of a day, as the author must decide what constitutes an event and which events are worth mentioning. Once the author makes this decision, the events that the author chose form the story while how these events are told forms the discourse. This concept is important to understanding human intelligence, as putting information into the context of story and narrative seems to enable us to understand it more deeply. As filmmaker Brian De Palm puts it, "People don't see the world before their eyes until it's put in a narrative mode." (Abbott, 2008)

Similarly another idea about storytelling and narrative from narrative theory is the idea that there should have a point, or else the narrative elicits the question of "So what's the point?" (Polanyi, 1979). Many of the stories that people recall and tell themselves are to help them communicate and idea or to make sense of a situation (Schank, 1990). Therefore, a story understanding system should be able to come up with narratives that have a point or serve a

particular purpose. For my work in enabling story generation this goal takes on the form of being able to answer "What if?" questions such as "What if Gretel was removed from the story of *Hansel and Gretel*?"

Human Story Understanding

Many researchers believe that story understanding is at the heart of human intelligence. Patrick Winston suggests the bold idea of **The Strong Story Hypothesis**. The hypothesis states that the ability to understand and manipulate stories is the distinguishing mental feature that separates humans from other primates. A similar and connected idea is **The Directed Perception Hypothesis,** which states that knowledge, understanding, and reasoning stem from our symbolic reasoning system's ability to control our perceptive apparatuses. These can be used on both real and imagined perceptions, in order to answer questions about the world (Winston, 2011).

There is evidence from brain studies that supports these bold ideas about stories. In neuroscience, there has been many studies investigating how humans respond to stories that encode various cultural elements, causal connections, and emotional appeals. Rebecca Saxe's work has identified various brain regions consistent between participants which respond to story stimuli. Of particular interest for relating this to general human intelligence is how human brain response varies in response to story and non-story stimulus. Specifically, some recent work has shown that certain areas of the brain respond strongly to a set of sentences that from a coherent story, whereas those areas do not respond when the sentences are unconnected or form an incoherent story (Dodell-Feder et al., 2011).

Clinical investigations conducted by Kay Young and Jeffrey Saver suggest that many brain dysfunctions may be more accurately described as dysnarrativia. Dysnarrativia has been an area of interest to a number of neurologists and psychologists alike, but in Young and Saver's work they demonstrate how specific deficiencies in a person's ability to tell stories seem to be linked to specific areas of the brain. The wide array of dysfunctions caused by damage across a variety of areas of the brain suggests that the ability to understand and tell stories is a complicated process involving many of the brain's capabilities (Young & Saver, 2001).

A number of other efforts have approached questions about the involvement of stories in human intelligence from across disciplines. In instructional design, Jonassen and Hernandez-Serrano research methods for analyzing, organizing, and presenting stories in order to enable better learning and problem solving (Jonassen & Hernandez-Serrano, 2002). Their efforts demonstrate the importance of stories and storytelling to learning. Dedre Gentner has demonstrated how analogy and similarity differ and how these differences affect people's understanding of the world (Gentner & Markman, 1997). Her early work laid the foundation for a powerful system called the Structure Mapping Engine (Falkenhainer, Forbus, & Gentner, 1989). Their work demonstrates the power for similarity and analogy to fuel human understanding. Later work by Finlayson and Winston has highlighted a similar importance of analogy in the story domain where analogy is important to efficient and intelligent information and story recall (Finlayson & Winston, 2006).

These ideas on the importance of stories and analogies align with Roger Schank's work in understanding human intelligence. In *Tell Me A Story*, Schank goes as far as to say that "Storytelling and understanding are functionally the same thing". Much of Schank's ideas focus on the idea that stories are the format of memory. Seen in this light, choosing an action or making any other decision becomes recalling the most relevant stories and following a script. One way that Schank says that humans differ from other somewhat intelligent animals is that while many animals can imitate or copy a scripted story, only humans can create them (Schank, 1990).

Imagination and Story Composition

While typically thought of as a conscious process, creativity, especially in the sense of story composition, seems to the strongly connected to problem solving apparatuses of human intelligence. Specifically, imagination, a key ingredient for creative tasks, is extraordinarily important for day to day reasoning. In a number of works, George L. S. Shackle identifies how imagination fuels decision making (Shackle, 1964). To Shackle, decisions are not made based purely on knowledge about the state of the world and real experience, but also from imagined experiences. In this context, imagined experiences are specifically grounded in previous knowledge and experience. These imagined experiences are essentially stories constructed in order to answer questions like "What would happen if I did X?" and "What will happen in an hour?" In order to answer these sorts of questions the brain must employ the same tools used to construct works of fiction, simply applied to real world stimuli (Augier & Kreiner, 2000).

Artificial Intelligence

My research goals are to make step toward more comprehensive artificial intelligence system. Based on this brief overview of the importance of stories to human intelligence, my research focus is story understanding. My work builds upon Genesis, an existing story understanding system, which I discuss in detail in Chapter 4. Two important parts of my contributions are teaching Genesis to learn from past experience and to incorporate story generation into the story understanding process. Schank identified the importance of past experience; a large part of what makes humans intelligent is our ability to learn from experience and to recall that experience when appropriate. As such, my work in story generation builds on the ability of the system to learn from past experience. As discussed, integrating story generation into the story understanding process is important because imagination plays a key role in human decision making.

Summary

In this chapter, I have discussed how stories play an integral part in human intelligence. Additionally I have identified that learning, recall, and imagination all play important roles in human intelligence and need to be considered by anyone working on improving the state of artificial intelligence. In the next chapter, I review the literature in the related field of creative computing, specifically looking at the history of tools for story generation.

Chapter 3 – Related Work in Story Generation

Over the course of my thesis research I have been working to take computational story understanding to the next level. An important facet of this work is the incorporation of imagination through story generation. Methods for computational story generation have been of interest to researchers in the field since at least the 1970s with the development of two of the earliest story generation systems: Klein's novel writer and Meehan's Tale-Spin (Gervás, 2009).

In this chapter I discuss the history of systems for creative story composition. Most importantly I highlight some of the strengths and weaknesses of various approaches. I explain how each system approaches computational creativity and provide a brief summary of each system's key features. As character modeling is a key component of this thesis, I also demonstrate how character modeling has been used in story generation in the past. In doing so, I set the stage for how I use character modeling and story generation to improve on state of the art computational story understanding.

From a high level, nearly all story generation systems so far fall into one of the three informal groups: author models, story models, and world models (Bailey, 1999). The author models attempt to emulate the strategies of human authors. The story models rely on structural story representations such as rule grammars. The world models attempt to set up a world and simulate forward from that point. My approach would primarily fall in the world modeling category stemming from its focus character modeling and simulation. However, I also draw from the other two categories. Using the Genesis system as a base allows me to use its already-available rich story rule, analysis, and comparison features, which mirrors some of the capabilities of author and story models of generation.

Novel Writer (1973)

The earliest known story generation system was created in 1973 by Sheldon Klein (Klein, Aeschlimann, & Balsiger, 1973). *Novel Writer*, as it was called, produced murder mystery stories intended to be used as the basis for weekend social parties. Klein's approach was to use a probabilistic story grammar combined with a simple linguistic model and rigid domain.

Specifically, *Novel Writer*'s input is a detailed stochastic rule set, an initial scene description, and a simple language dictionary and grammar. The scene description is used to initialize the semantic network that tracks the story elements. The rule set is used to generate plot events probabilistically. Event and other sorts of story information are represented as changes to the underlying semantic network. Finally, the language dictionary and grammar are used to generate English sentences for story output. The bulk of the story generation for *Novel Writer* takes place in the form of world simulation. The simulator uses the probabilistic rules to propagate new events forward in time based on the current world state and the set of rules available.

Novel Writer set a good baseline for computational composition methods as it included many important basic components. It was the first system to include character and language models that work in tandem.

Tale-Spin (1977)

Tale-Spin was story generation system by James Meehan that produced interesting stories in a very different manner than *Novel Writer* (Meehan, 1977). While *Novel Writer* is restricted to the domain of social parties, *Tale-Spin* is restricted only by the knowledge base that it is provided. *Tale-Spin*'s primary technique is a goal and problem solving oriented approach to creative story generation. Tale-Spin incorporates a number of interesting information types, including physical space, relationships, and basic personalities.

In order to generate a story, a user inputs a number of characters and objects and then gives these characters specific goals that they want to achieve. *Tale-Spin* then proceeds with generation by determining what course of action the characters would want to take in order to achieve their goals. *Tale-Spin* relies on a large database of rules and goals in order to model the world and the outcomes of character actions. Using backward chaining, *Tale-Spin* decomposes goals into simpler goals where possible and attempts to find chains of actions that will lead characters to the appropriate goal.

Additional constraints of the backward chaining planner include elements of personality, physical space, and relationships. These are expressed in terms of a large set of hard-coded rules that influence what actions characters will want to take, will not take, and things out of the characters' control, such as falling into a river. The methods and routines used by *Tale-Spin* for rule based generation closely parallel of number of methods in Genesis for story analysis such as rules for understanding the outcomes of actions.

Meehan acknowledges that while ideally you want your story generator to create good works, it's often the bad results that are the most interesting. One such example is this ending of a *Tale-Spin* story "Gravity drowned." (Wardrip-Fruin, 2006)This is an example of a sentence you'd never expect a human to produce, which demonstrates that there must be a dichotomy between a person's and the system's understanding of the world. In this case the system modeled gravity as an actor in the story, but gravity had no friends nor legs and so couldn't escape drowning.

As Wardrip-Fruin notes in his discussion of that story, the errors are often the most interesting because they give insight into how the system works. In *Tale-Spin*'s case, the system carries out a great deal of complex processing during its reasoning process, but this often doesn't come through in its stories because the thought process doesn't make it into the finished story. This suggests that future story generation systems could benefit from the ability to explain the choices they make, especially when anomalies are encountered. This would allow a more transparent look at how the system works and allows a user to understand why errors are made which can suggest how the system can be improved. While my system does not have the capability to explain its choices, future work to use *Genesis*'s story analysis tools in a self-reflexive way in order to identify why events happen.

Author (1981)

The *Author* story generation system was created by Natalie Dehn in 1981. The system was the first to have a strong focus on authorial goals (Dehn, 1981). In order to compose its stories, *Author* proceeds through an iterative process that continually works on improving the plot to fulfill the goals of the author while also occasionally modifying the author's goals by reflecting upon the state of the in-progress story. Additionally, the system contains a mechanism for reminder, which allows for the author to be periodically reminded of previous goals to see if they have become more relevant again in light of the direction the story has been constructed. For example, originally an author may expect a character, John, to have a violent conflict near the end of the story. However, as the story progresses, the author changes its mind because John meets Mary and the story is seeming more like a love story. Next the character Kyle is introduced as Mary's former boyfriend. At this point while the system is reminding the author of previous goals, the violent conflict seems relevant again because John might be heading for a fight with Kyle.

Similar to *Tale-Spin*, one of the important components of *Author* is a planning mechanism, except in this case the planning is used to fulfill authorial goals instead of character goals. However, *Author* also introduces some interesting concepts. First, *Author* is among the first creative systems to introduce the idea of reflection into the generative process. The system frequently reflects back on the generated story in order to update author's goals. Thus the author's goals are flexible, allowing the goals to change depending on the way the story is progressing. Related to this, the system uses what Dehn calls conceptual reformulation. In conceptual reformulation, *Author* takes a high level goal, such as demonstrating that main character is shy, and reformulates it into a number of lower level sub-goals. As examples, the high level goal of a kernel episode (e.g., Shy person needs a job and becomes employed as a sales person, but can't face the everyday challenges.) can be split into a sequence of sub goal episodes (e.g., Anecdotes demonstrating shyness, main character gets hired in sales, and main character gets fired).

The design decisions behind *Author* seem to be driven by Dehn's take on the creative process. As stated by Gervás, the creative process should capture both deliberateness and serendipity (Gervás, 2009). From this core concept, Dehn came up with the important idea of reflection in story generation, illustrated by the strategy to fulfill your goal while simultaneously trying to find a better goal. The concept of reflection is of critical importance to my work because of its relation to story understanding. The Genesis story understanding system uses reflection to understand stories from a higher level. Additionally, I demonstrate in chapter 7 how story generation itself can be a means for improving a story understanding system's ability to understand an existing story.

Universe (1983)

Universe creates universes of characters for telling stories. Created by Michael Lebowitz, the aim of the system was to generate a series of episodes akin to a soap opera from the same collection of characters. The main goal was a focus on consistency and coherence by

improved modeling of characters using interpersonal relationships and character traits (Lebowitz, 1984).

In constructing *Universe*, one of Lebowitz's goals was dealing with the problem of showing and not telling. For example, if an author's intent for an episode is for two characters to break up but remain friends, then the system must set up a story where this characterization plays out naturally rather than being told explicitly. In order to achieve this, Lebowitz focused not on planning and simulated actions of a few characters, but instead on techniques for creating casts of consistent and coherent characters. One import aspect of Lebowitz's character model is the explicit inclusion of a relationships as a key aspect of a character. Most of the story content that *Universe* generates is specifically targeted to demonstrate and modify the relationships between the characters in a never-ending story.

For generating plot, *Universe* uses a provided library of plot fragments. These plot fragments are similar to the plot units found in Genesis (Lehnert, 1981; Winston, 2011), and in Lebowitz's case they serve to represent authorial goals. These plot fragments provide plot elements that are used to fill in character activity. Interestingly, Lebowitz includes some techniques in *Universe* for the spontaneous generation of new plot fragments. This would occur by mix and matching elements from existing plot fragments. I believe idea of mixing elements to create new ones is an important example not just of a method of generation but also as a step towards how a system can learn to generate, which is important to my research in enabling a system to learn how to generate plot from reading a corpus of stories.

Minstrel (1993)

Minstrel was a story generation system created in 1993 by SR Turner (Turner, 1993). The system is able to tell stories about the knights of Camelot. The generation methodology of *Minstrel* is very similar to Dehn's *Author*. *Minstrel* begins the generation process with high level author goals and then tries either to solve via planning these goals or to break them down into one or more lower level goals. One point of distinction however, is *Minstrel's* aim to tell stories that encode specific morals.

Minstrel has the interesting capability to query, search, and modify story memory using a process of imaginative recall. Essentially, the system has a number of search and transformative functions that match and modify events in order to create new situations and tell a coherent story. These transform-recall-adapt methods (TRAMs) allow *Minstrel* to modify sentences to alter their meaning in specific ways. For example, "John slew a dragon" can be matched with "Someone slew something with a sword" which in turn allows for the creation of "John slew the dragon with the sword" (Tearse, Mawhorter, Mateas, & Wardrip-Fruin, 2012). Turner claims that this ability to discover and apply these transformations to a story is an important aspect of computational creativity.

Matching and modification methods like *Minstrel*'s TRAMs are important because they both make it easier for the system to recall and identify relevant story fragments but also because they allow such fragments to be modified to fit new situations. Similar types of problem are solved within Genesis through the use of previously developed semantic matching functions

(Fay, 2012a; Winston, 2012b). Additionally, in my work similar techniques are used to aid the plot generator in creation of novel plot elements.

Mexica (1999)

Mexica was developed by Rafael Pérez y Pérez in 1999 and took a number of important steps forward in story generation. On a high level, *Mexica* generates stories by switching iteratively between two distinct modes of operation: engagement and reflection (Pérez y Pérez & Sharples, 2001). During engagement mode, *Mexica* expands the current story state by generating plot elements based on a set of constraints, a set of possible actions, and the current story state. In reflection mode, *Mexica* evaluates the story so far on factors such as novelty and coherence and then can modify the story via modification or backtracking if it needs improvement in an area. *Mexica* currently tells stories based on Mexican folklore, but could be extended with additional rules, actions, and stories to tell stories in other domains.

The reflective stage in particular sets *Mexica* apart from previous generation systems because it incorporates both in depth introspective analysis of the constructed story and successfully incorporates a library of previous stories for the first time. During its analysis, *Mexica* compares the current story to other similar stories in order to ensure that the new story is neither too similar nor too different from previous tales. The incorporation of a set of previously known stories makes *Mexica* have commonalities with my own research. However, while *Mexica's* library is for the purpose of evaluation, my own work leverages a library of stories for the purposes of learning and generation.

On an interesting side note, *Mexica* also is one of the first systems to attempt to tell generated stories using visual media. Using a secondary program *Mexica* is able to create illustrated versions of its stories in the style of Mayan artwork (Pérez y Pérez, Morales, & Rodríguez, 2012). This advancement demonstrates that story generation can extend beyond simply text which is important in working toward a high level goal of human intelligence which combines both visual and linguistic information. It would be an interesting extension to my own work to branch into this sort of visual intelligence.

Virtual Storyteller (2003)

Virtual Storyteller is an interactive narrative and story generation system created by Theune et al in 2003 (Theune, Faas, Heylen, & Nijholt, 2003). Virtual Storyteller does multi-agent simulation in order to generate stories. In order to try to incorporate parameters of what may make a good story, *Virtual Storyteller* tries to make sure its stories are well structured, containing a beginning, a climax, and an ending.

Additionally, the system tries to strike a balance between having purely character-based simulation and having scripted plot lines. In order to accomplish this task, Theune et al. take an interesting approach: they introduce non-character agents with knowledge on narrative theory. For example, a director agent has knowledge related to how a plot should flow and uses this to constrain the actions that other character agents may take. Similarly, a narrator agent has knowledge for how a story should be told to a user, translating it from internal representations into a specific ordering of events.

The approach taken by *Virtual Storyteller* is interesting because of the independence and variety of agents which construct the story. My approach taken in creating and simulating characters is somewhat similar to this idea in *Virtual Storyteller*. However, the big difference is that *Virtual Storyteller* seeks to have agents that construct a story whereas my work has agents that are simulated within a story.

As future work, an interesting approach would be to see if multiple levels of agents could be used in conjunction to produce more interesting stories. For example, combining the character agents existing within a story with agents that exist around the story. Such a combination could combine many of the positives of world simulation and author simulation.

Façade (2003)

Façade is a game that explores interactive drama. It was created in 2003 by Michael Mateas and Andrew Stern. In *Façade*, the player interacts with two married characters by walking around in their three dimensional virtual apartment and communicating with them by typing English statements for the player character to say. The player's actions affect the unfolding events that take place over the course of the story, including determining how all the characters' emotions and relationships end up (Mateas & Stern, 2003).

Internally *Façade* incorporates characters that interact with each other and the user over the course of the drama. *Façade* uses A Behavioral Language (ABL), to define parallel, sequential, and joint behaviors of the characters and the user. Behaviors defined by ABL are similar to goals with certain conditions or actions needing to be met for success. When a behavior fails due to its conditions, *Façade* can attempt to find alternative behaviors in order to carry out the higher level goals.

While *Façade* is more of an interactive narrative than narrative generation system, it is worth describing here because it succeeds in modeling characters and making those characters' actions believable in the context of a story in which the user has control over the flow of the plot. Despite the different domain, the idea of simulating character behavior is an important commonality between my research and the work done for *Façade.* Additionally, the plot weaving algorithm that I develop shares some characteristics with the drama management ideas from *Façade*. This suggests potential future work in exploring how my models could be extended from the domain of story generation into other areas such as interactive narrative.

Fabulist (2004)

Fabulist is a narrative planning system for doing story generation created by Mark Riedl in 2004. In developing *Fabulist*, Riedl wanted to enable story generation with little prior knowledge built into the system from a design standpoint. This would potentially allow Fabulist to generate stories of types that were not anticipated by the system's creator. *Fabulist* achieves story generation by applying search-based planning algorithms to the domain of story understanding with the goals of plot coherence and character believability (Riedl & Director-Young, 2004).

Fabulist is an interesting approach to story generation because the research itself is grounded in artificial intelligence methodologies. Specifically, much of the work is grounded in planning algorithms which are useful for developing control systems. It demonstrates how heavily rule based system can be used to create interesting stories using cross-disciplinary algorithms. The work itself however, works quiet differently from my own area of research because while both incorporate a rule based system, *Fabulist* uses this for the planning whereas *Genesis* uses this for learning and story analysis. The methods in *Fabulist* could potentially be adapted to enable *Genesis*'s rule based story analysis tools to be leveraged more directly in the story generation process.

Curveship (2011)

Curveship, developed by Nick Montfort in 2007, tackles computational creativity in story telling from a very different angle from the previously discussed story generation systems. In particular, while most of the previously discussed systems focus on generating the plot, *Curveship* attempts to tell stories in many different styles without changing the plot. For example, a single story could be told from different perspectives, with different levels of attention to detail, and a wide number of different methodologies. *Curveship* has been developed with interactive narrative and storytelling in mind and thus provides methods for doing progressive world simulation that can take user input during the course of a story to direct how the story unfolds and is told (Montfort, 2011).

Additionally, through a collaboration between Pérez y Pérez and Montfort, *Mexica* and *Curveship* have been adapted to work together through a system called *Slant*. *Slant* uses a combination of techniques including a story whiteboard concept to allow various story generation and storytelling systems to work together to produce interesting narrative better than any of the individual components (Montfort, Pérez y Pérez, Harrel, & Campana, 2013). This collaborative work is valuable because it suggests a means by which future iterations of work done by myself, the Genesis group, and others could be improved via integration. Such integration of individual systems mitigates weaknesses, combines strengths, and adds value.

Other Works

There have been a number of other efforts in the domain of story generation. The ones discussed in the chapter represent both an overview of the history of the field as well as an illustration of those projects that have been most influential on my own work. Other developments in the areas of plot generation and simulation for interested parties include *Tailor* (Smith & Witten, 1991), *The Oz Project* (Bates, 1992), *DEFACTO* (Sgouros, 1999), *MAKEBELIEVE* (Liu & Singh, 2002), and *America's Army: Soldiers* (Nieborg, 2004).

Summary

In this chapter, I've given an overview of the field of computational story generation. Specifically, I've highlighted works that have made important contributions and that helped shape my own research. In the next chapter, I discuss the *Genesis* system for story understanding. This system serves as the baseline for my own work and implementations.

Chapter 4 – The Genesis Story Understanding System

A major contribution of my research is pushing the capabilities of computational story understanding forward. My work builds upon the existing Genesis system for story understanding. Genesis represents a state of the art platform for conducting research in artificial intelligence with a focus on language, vision, and, of course, stories.

In this chapter I detail the workings of the Genesis system and demonstrate the most important capabilities of its functionality. Additionally, I illustrate areas for improvement that I have been targeting with work in character modeling, learning, and story generation.

Genesis, an overview

The Genesis system (hereafter, Genesis) reflects the accumulation of many years of work by a large body of researchers. The project is in active development in the Computer Science and Artificial Intelligence Laboratory at MIT. The explicit goal is to provide a comprehensive story understanding artificial intelligence with a strong focus on modeling how humans think.

The focus on human intelligence means that Genesis has benefited from the experience of people interested in story analysis, cognitive science, visual understanding, language, and more. The wide spread of research topics has enabled frequent collaborations with other groups to explore new territory.

Currently, Genesis is able to read stories written in English that can be parsed through the START Natural Language Processor (Katz, 1997). Once parsed, Genesis is able to use background knowledge to expand on the understanding beyond the information provided explicitly in the text. Finally, Genesis provides an array of analysis tools such as the elaboration graph for understanding aspects of the story and how it compares to previously read stories (Winston, 2011).

Modular Integration through the Wired-Box Paradigm

At the core of Genesis's implementation is the Wired-Box design paradigm (Winston, 2011). The paradigm draws from the idea of computational propagators created by Gerald Sussman. In the propagator model, valued cells are updated via propagation connections. These propagators represent constraints and relations on the values of these cells. As more information is supplied to a propagator network the cell value ranges are continually updated to incorporate the information where applicable (Radul & Sussman, 2009). A key feature of propagators is that the connections are bidirectional, which allows information to be used both in a top down and bottom up fashion.

Genesis contains a loose implementation of the propagator ideology. Within Genesis, there are a large number of independent modules, known as boxes. These boxes can be connected via any number of bidirectional connections, or wires, across which information can be transmitted as input and output from the boxes. Each box can listen for relevant information to come through on its wires and act on it accordingly. In practice, this communication style

bears resemblance to the popular "Observer/Observable pattern" programming style (Johnson, Helm, Vlissides, & Gamma, 1995).

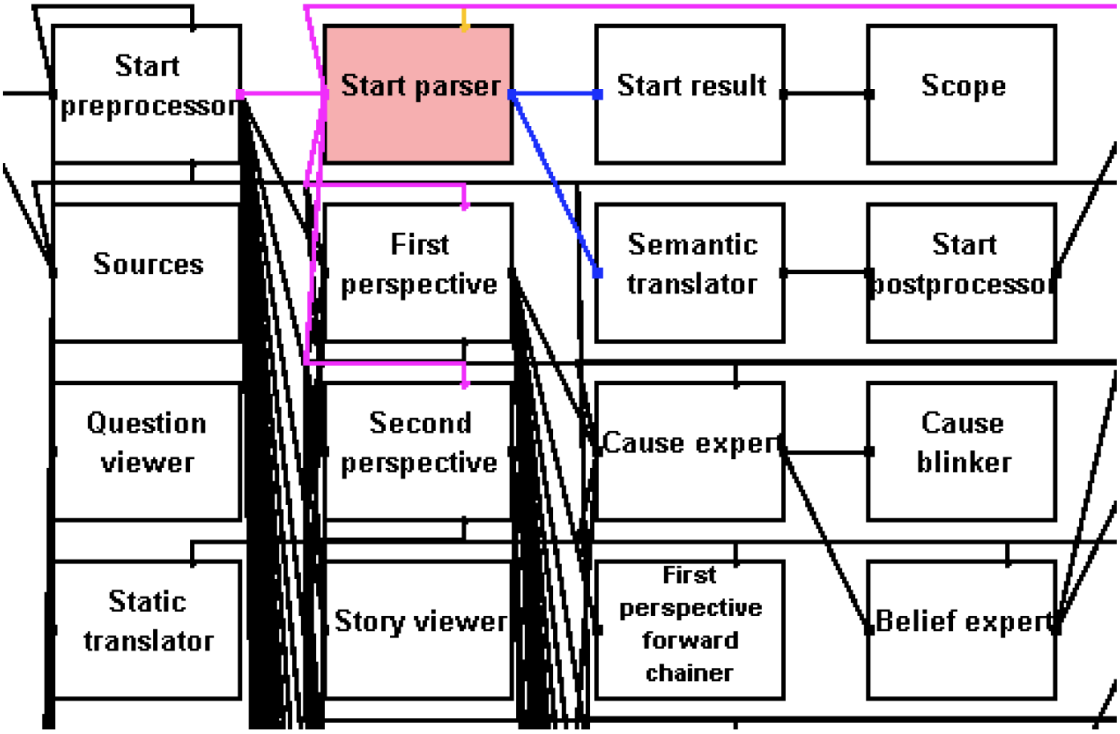


Figure 4 – Wired box connections between modules in Genesis
Genesis consists of a large number of independent modules that interact via the wired box paradigm. Each box is a single module that can have its inputs and outputs wired to any number of other boxes. The modules each serve different purposes and work together to enable Genesis's story understanding capabilities. This figure shows a snap shot of an interactive view of Genesis's internal connections. The pink box currently in focus has color-coded wires indicating the flow of information. Blue wires are outputs and pink wires are inputs.

This wired-box paradigm underlying Genesis grants a number of advantages to the research. First, from a development standpoint, boxes do not need to know about the source nor implementation of connected boxes. This means boxes can be seamlessly updated or even replaced without having to rewrite any of the connections. Additionally, it is easy for new researchers to use and augment Genesis because it lowers the amount of knowledge necessary to begin working on the system. Secondly, from a design standpoint, that the overall model matches some human functionality in terms of its modality and bi-directionality. Finally, the infrastructure supporting the wired-box paradigm is abstracted to allow it to function seamlessly regardless of the physical locations of any particular boxes. Boxes can span many processes on one or more physical machines across any number of locations and wires behave identically regardless as to whether they remain within a single process or are connecting boxes across a network.

Semantic Language Processing

Genesis has been designed to be capable of both reading and storing its knowledge bank in natural English because language is an incredibly important part of human intelligence. In order to facilitate this, Genesis has been integrated with a number of different language processors including the Stanford parser (Cooper, 2009) and an in house language system (Bender, 2001). Currently, Genesis's primary language provider is the START Natural Language Processor (Katz, 1997; Winston, 2012b).

The START natural language processor is a powerful semantic parser and question answering system. START is available on the web and can answer English questions with natural language responses. It does this by parsing input questions into an inner representation, using that representation to search its data sources for relevant information and then returning the result to the user (Katz, Borchardt, & Felshin, 2006).

START has a number of advantages that make it a good choice for the needs of the Genesis research project. First it provides a two-way transformation route. This means that Genesis can use START for both parsing and generating English sentences. Second, rather than simply creating a most probable parse tree like most language packages, START creates a semantic network that more directly conveys the meaning of sentences. START also attempts to track agents between sentences, which can help deal with problems such as pronoun ambiguity. Finally, START is provided as a web service which means it does not require a complex API or source libraries to access its functionality (Katz, 1997).

After parsing a sentence, START returns its semantic parse as a set of inter-related triples. For example the primary semantic information parsed from the sentence "Mary liked her little brother very much" would be output in triples shown in Figure 5.

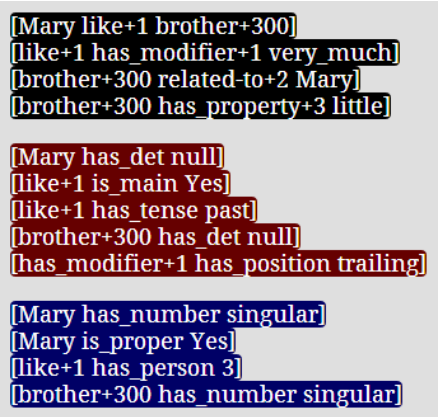


Figure 5 – START's triples for the sentence "Mary liked her little brother very much"
START is a powerful semantic parser for the English language. When parsing sentences, START is capable of drawing in information from a variety of sources such as Omnibase (Katz, Felshin, Yuret, & Ibrahim, 2002) and IMPACT (Katz et al., 2006). Using this information and its sophisticated internal methods, it can extract both the important syntax and semantics of the sentence. The output of START's parsing comes in the form of triples like those seen in this figure. These triples can also be used by START in a reversed process to generate English sentences. This figure shows the triples produced from the sentence "Mary liked her little brother very much".

Triples are capable of representing semantic information, syntactic information, and a mix of both. For example, the triple [Mary kiss+1 John] represents mostly semantic information whereas [kiss+1 has_person 3] represents mostly syntactic information and [kiss+1 has_tense past] represents a solid mix of both.

For generation, START accepts in a set of triples from which it can generates an English sentence matching the semantic and syntactic constraints provided. Conveniently, START does not need a full set of triples to generate a sentence. [Mary kiss+1 John] is sufficient to generate the English "Mary kisses John" while more triples can be provided to add more information or to tailor the syntax appropriately.

English Knowledge

As previously mentioned, Genesis stores its knowledge bank in natural English. We believe this is an important characteristic for a number reasons. First, it allows both Genesis and human readers to understand exactly the same text; no intermediary is necessary for mutual understanding. Second, it makes Genesis's knowledge bank robust: The knowledge remains unchanged and usable regardless of system changes and the knowledge can easily be leveraged by other interested systems. Finally, we feel that being able to learn knowledge from human language is an important and powerful contributor to human intelligence and learning, so Genesis being able to learn about concepts from the same source as a human is a vital design characteristic.

Genesis Innerese: The Inner Language of Genesis

All information within Genesis's memory is stored as Genesis Innerese, the inner language of Genesis. Within Innerese, Genesis stores semantic information in a hierarchical form that is easily searchable, comparable, and highly expressive. Innerese consists of four basic frames which form the building blocks of the representation: *Entities, Functions, Relations, and Sequences.*

- An *Entity* is the base unit, consisting of a single character, object, place, or other thing. Examples of things that would be represented as entities include "Emily", "a keyboard", "Indiana", "a duke", and "happiness". Entities are often expressed in a short hand of (e Emily).
- A *Function* builds upon an Entity and consists of a modifier, action, or function and a single child entity. Examples of expressions that would be represented as functions include "under the desk", "the front of the class", and "a fork in the road". Functions are often expressed in a short hand of (f under (e desk)).
- A *Relation* builds upon a function and expresses the relationship between two entities. Examples of relationships that would be represented as relations include "Mary kissed John", "Stephen King is an author", and "The river flowed south". Relations are often expressed in a short hand of (r kissed (e Mary) (e John)).
- A *Sequence* builds upon a relation by expressing an abstract relationship between an unbounded number of entities, functions, and relations. Examples of expressions that would be represented as sequences would include "Matt and Mark", the plot of a story, and the set of objects and modifiers of an action. Sequences are often expressed in a short hand of (s and (e Matt) (e Mark)).

Using just this set of frames, Genesis is capable of representing ideas of arbitrarily high complexity. This means that the system can easily handle knowledge of many levels of complexity varying from representational knowledge to high level reflective understanding (Winston, 2012a).

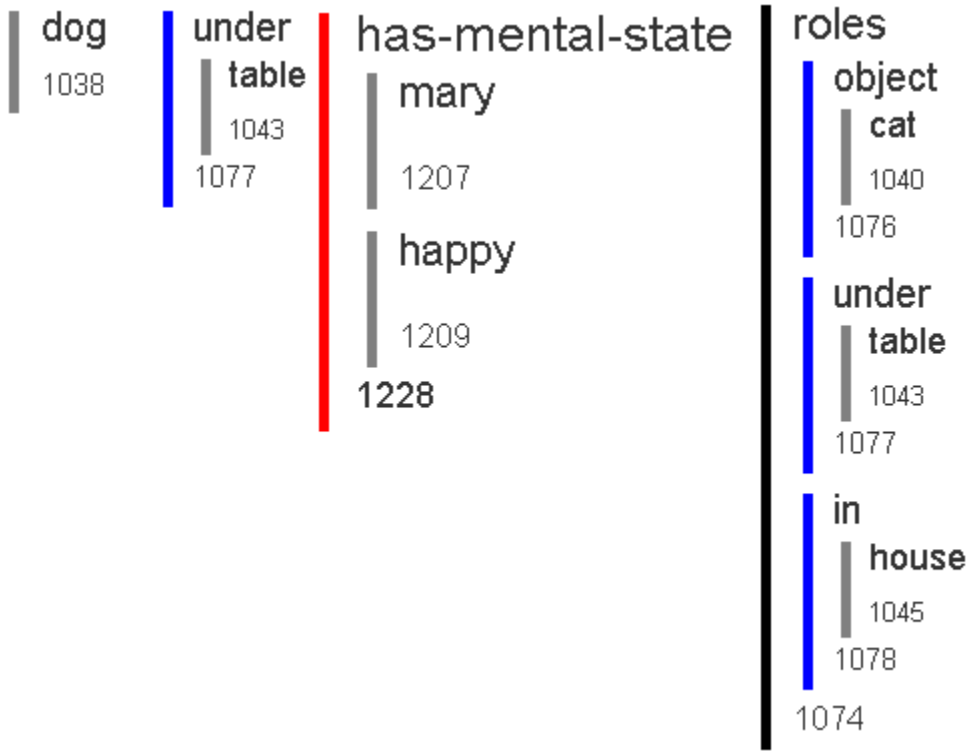


Figure 6 – The building blocks of Genesis's Innerese
In Genesis there are four main frames for representing semantic information. On a high level they structure how the semantic information is connected. Entities are the most basic, corresponding to information about things, usually nouns. Entities are shown in grey. Functions are Entities that modify one entity, such as the concept of "under". Functions are shown in blue. Relations are Entities which relate two other Entities, often taking on the form of verbs. Relations are shown in red. Sequences are an Entity that connects an unbounded number of other Entities. Sequences often relate to lists of things. Sequences are shown in black. All of these types of semantic building blocks can be embedded within one another, for example, there are a number of functions embedded within the "roles" sequence.

Representational Knowledge

A large number of general knowledge representations can be represented within Innerese and understood by Genesis. This includes but is not limited to *classes*, *transitions*, *social relations*, *properties*, and *trajectories*. All of these knowledge types can be easily expressed both in Genesis's inner language and natural English. The table in Figure 7 shows a variety of examples of this type of information.

Representation	Example
Cause	Alpha killed Bravo because Bravo angered Alpha.
Class	A keyboard is a type of input device.
Goal	Charlie wanted to be a pilot.
Job	Delta is a pilot.
Mood	Echo became unhappy.
Persuasion	Foxtrot persuaded Golf to commit the crime.
Possession	Hotel had the prize.
Property	India is large.
Role frame	Juliet killed herself with the dagger.
Social relation	Kilo was Lima's brother.
Time	Next, Mike looked around.
Trajectory	Oscar went quickly through the city.
Transition	The current month became November.

Figure 7 - Representational Knowledge in Genesis
Genesis contains a variety of types of representational knowledge. Such knowledge is used to classify, categorize, and better understand various components of a story. This figure shows a subset of representations understood by the Genesis system. The representations are all encoded into Genesis Innerese (Fay, 2012a).

Because of the expressiveness of Innerese, as seen through the variety of possible representations, Genesis is capable of handling a vast array of types of information all using the entity, function, relation, and sequence structural frames.

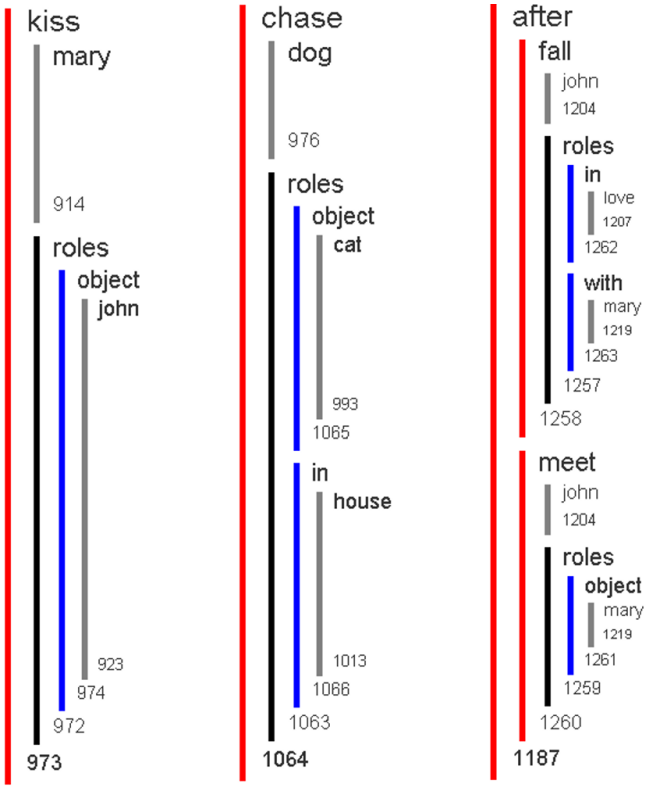


Figure 8 – Genesis's Innerese frames for representational knowledge
Genesis stores all of its knowledge bank in English. However, whenever that information is needed, Genesis must read the English and convert the information to its internal representation. First, sentences are given to the START system, which produces semantic triples. These triples are then processed by internal Genesis modules in order to produce Genesis Innerese, as shown above. The figure shows three sentences which have been processed by Genesis. From left to right the original sentences are "Mary kissed John", "The dog chased the cat in the house", and "John fell in love with Mary after John met Mary."

Common Sense Knowledge

In addition to the low level types of information mentioned thus far, Genesis is also capable of handling higher more powerful levels of knowledge using the same systematic framework, including common sense knowledge (Winston, 2011). Within the Genesis group, we define common sense knowledge to be the sorts of rule based knowledge that allows people to understand relationships between things, events, and the world. Importantly, commonsense knowledge embodies what a particular person or read might bring to the table when understanding a story. Different collections of commonsense knowledge could therefore represent the background knowledge of people from different cultures, world views, and ages. Examples of common sense knowledge include everything from knowing that if a king dies then

his successor becomes king to knowing that if someone is hungry she will want to eat. This sort of knowledge allows Genesis to draw connections between events and thus understand how stories can be more than just a string of events.

In general, Genesis uses causal rules to understand inevitable outcomes of actions. For example a simple causal rule would be "If XX is killed, then XX becomes dead." If provided with such a rule, Genesis would insert a "XX becomes dead" whenever it encounters someone killing someone and it would connect these events with a causal connection. It's worth noting that XX is simply a generic entity, Genesis is capable of using powerful abstraction and matching techniques in order to apply common sense knowledge if and only if it is relevant to a situation. Two common types of connections are *explanative* and *predictive*.

Genesis uses predictive rules to make guesses about the future and understand the outcomes of events. For example if Genesis knows that "If XX is killed, then XX becomes dead", Genesis would be primed to look for events matching XX is killed. Upon finding such an event Genesis would imagine the event XX becomes dead and connect these two events with a predictive link.

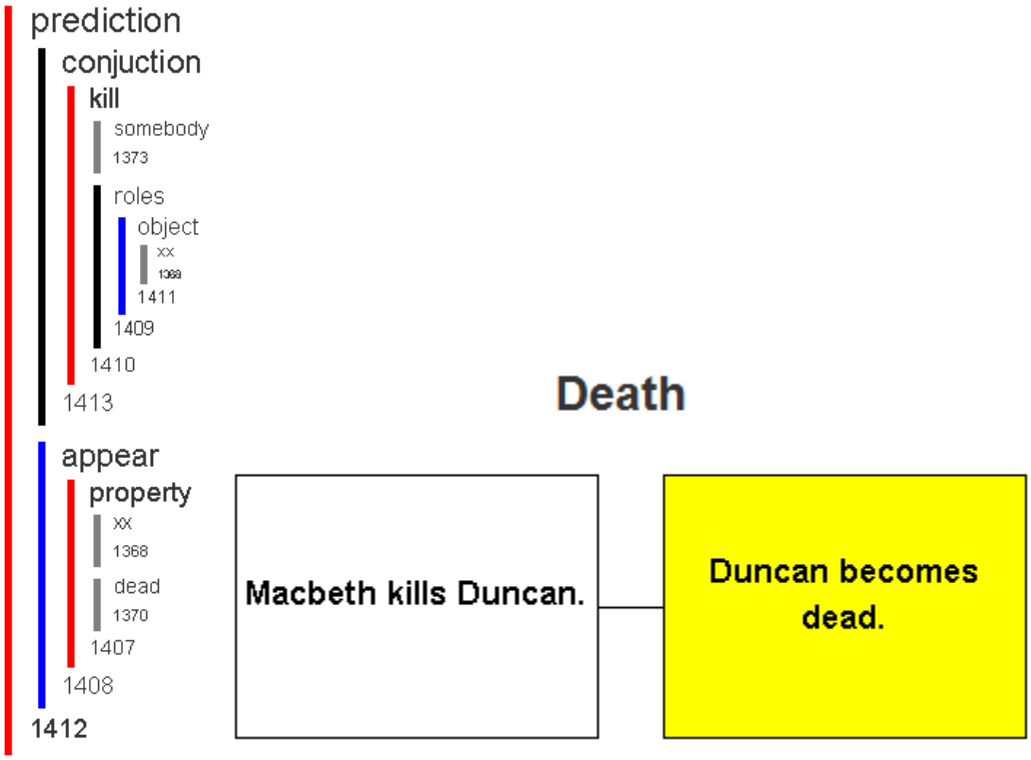


Figure 9 - Predictive rules in Genesis
Genesis has a library of available commonsense knowledge, which allows it to better understand stories. One type of element in the commonsense library is the predictive rule. Predictive rules allow Genesis to make guesses about things that would occur in the story but are not explicitly stated. For example, one such predictive rule is "If XX is killed, then XX becomes dead," which is shown on the left. From this bit of knowledge Genesis is able to infer the event shown in yellow, "Duncan becomes dead," when it encounters the event "Macbeth kills Duncan" in a story. The solid black line between the boxes indicates a direct connection between the events.

On the other hand, Genesis uses explanative rules to understand how multiple events that occur in a story may be connected. For example a simple explanative rule would be "If XX is hungry, XX may buy food." If Genesis encounters someone who is hungry and then later buys food, it will connect the events with an explanative connection.

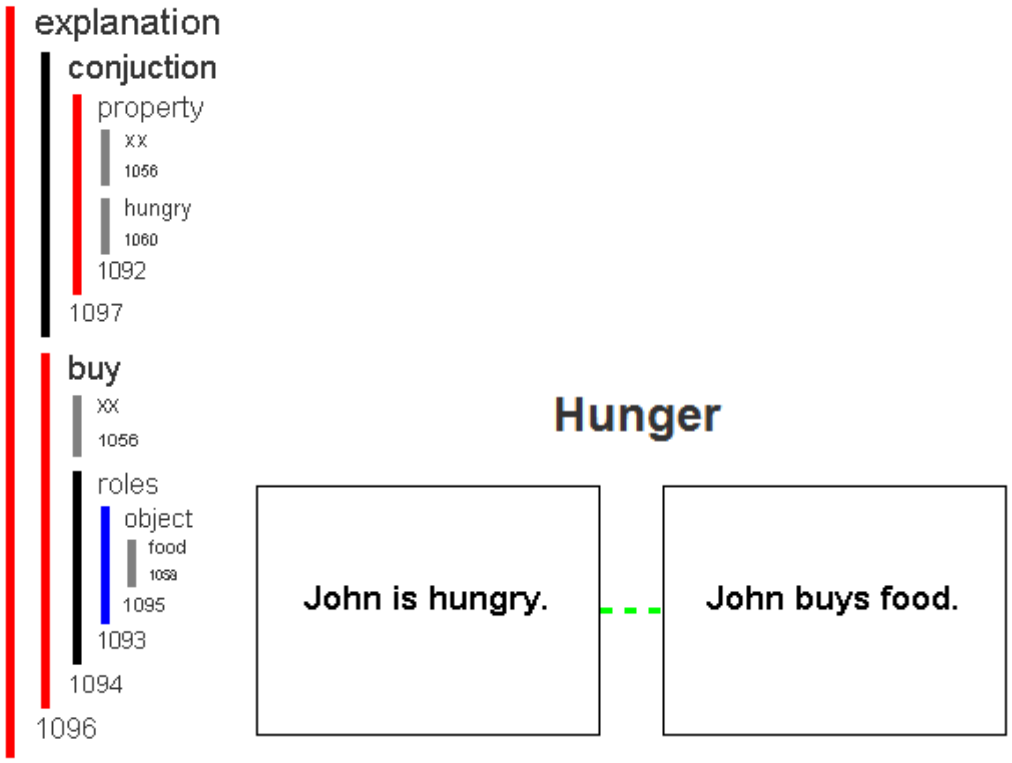


Figure 10 - Explanative rules in Genesis
Genesis has a library of available commonsense knowledge, which allows it to better understand stories. One type of element in the commonsense library is the explanative rule. Explanative rules allow Genesis to make guesses about how various events in the story may be connected. For example, one such predictive rule is "XX may buy food because XX is hungry," which is shown on the left. From this bit of knowledge Genesis is able to connect the event "John is hungry" with the event "John buys food". The dotted green line between the boxes indicates that Genesis believes there is likely a connection between the events.

Reflective Knowledge

Another common and important type of knowledge that Genesis is capable of dealing with is Reflective Knowledge. Reflective knowledge mostly comes in the form of concept patterns, which can be thought of themes, motifs, and other higher level plot elements that occur within stories. Having a knowledge base of concept patterns allows Genesis to analyze stories and provide more in depth insights about the plot (Winston, 2011).

Many concept patterns are available within the Genesis knowledge base simple English representations. For example the idea of *Revenge* can be taught to Genesis with the statement "John's harming Mark leads to Mark's harming John." Genesis generalizes these sorts of mini-stories and can use them to search for occurrences of them within any story being analyzed.

While the majority of concept patterns used in Genesis are a part of common data set, additional work has been done in the group to automate the learning of reflective knowledge. Given a collection of stories, Genesis can attempt to find common plot patterns and extract them. This capability allows Genesis to expand its knowledge without being explicitly taught by a user (Krakauer, 2012).

Elaboration Graph

When working with Genesis we find it useful to be able to rapidly visualize what the system is thinking. One extremely important tool is the elaboration Graph, which represents Genesis's interpretation and understanding of a story. The elaboration graph combines the story chronology, common sense understanding, and reflection analysis into a single powerful visualization containing not only events explicitly occurring in the story but also what Genesis is able to infer from what it knows. An example of the elaboration graph is shown in Figure 11.

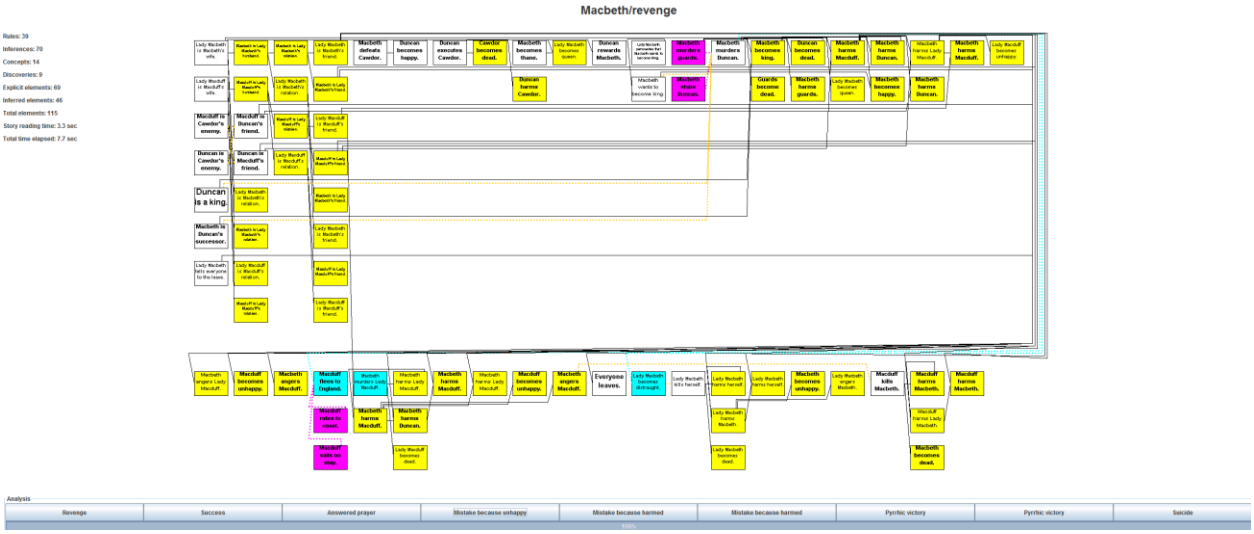


Figure 11 - The Genesis Elaboration Graph
Genesis is capable of creating powerful visualization called the elaboration graph from any story it reads. The elaboration graph shows how all the events, both explicit and inferred, are interconnected within a story. In this graph, white boxes are important plot elements which are explicitly stated within the story text, yellow boxes are plot elements which have been inferred from commonsense rules, magenta boxes are mechanisms of action, and cyan boxes indicate events are related through a leads to statement. Beneath the graph are the high level plot themes that Genesis has detected in this rendition of Macbeth, including "Revenge" and "Pyrrhic Victory".

Using the elaboration graph a user can quickly understand the concept patterns that Genesis detects in a story. For example, in the case of the revenge in the story of Macbeth, Genesis can find and highlight the chain of events starting from the event of Macbeth harming Macduff to the event of Macduff harming Macbeth. A simplified excerpt of the revenge discovered in Macbeth by Genesis is shown in Figure 12.

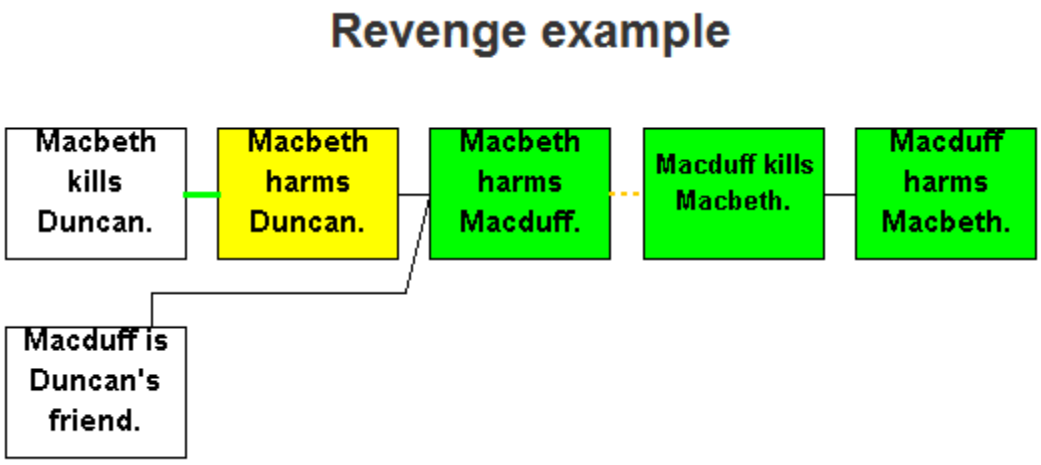


Figure 12 - An example of a revenge plot unit in Genesis from Macbeth
Plot units are a high level of knowledge representing patterns, themes, and analysis over multiple events that can be extracted from a story by Genesis. An example of a plot unit is the idea of revenge where one character harms another character and later the harmed character reciprocates this harm on the original character. This figure shows a simplified excerpt of the act of revenge that occurs within Macbeth between the characters of Macbeth and Macduff. Genesis uses its commonsense knowledge to determine the connections between the events of the story and then finds a chain of events leading from Macbeth's harming of Macduff to Macduff's harming of Macbeth. The main plot elements of the discovered revenge are marked in green.

Story Summarization

Genesis provides a number of capabilities for summarizing the stories. Most importantly, unlike naïve summarization tools, which may simply omit sentences or phrases in order to reduce the length of a document, Genesis is capable of actually summarizing based on its understanding of the plot. This means Genesis is capable of producing much more targeted and human-like summarizations.

On the simple level, Genesis can create a summary similar to how a naïve tool would, by removing extraneous elements and others at random until it cuts the story down to the appropriate size. Increasing in complexity, summaries can focus just on a specific character or collection of characters. In a more impressive show of Genesis's capabilities, the system can produce summaries containing only the events that are relevant to the actual plot as demonstrated by their connection to other events in the plot. Genesis can even tailor its summaries to target the most important high level plot features of a story, for example it can note that Macbeth is a story about Pyrrhic victory and tell the events related to that concept pattern. Finally, regardless of how the chosen, Genesis can then render a textual summary of the story.

Story Comparison

The ability to compare stories and their plots is a powerful capability in Genesis. Stories can be compared via a wide array of criteria ranging from word counts all the way through high level concept comparison. For example, Genesis can determine that one reason *Macbeth* and *Hamlet* are similar is because both include instances of revenge. One common usage of

comparison within Genesis is modeling and comparing readers of different cultural backgrounds. Cultural backgrounds within Genesis are treated as independent sets of background knowledge. Given multiple readers each with a different culture background, Genesis can evaluate the text of a single story and delve into the differences between the interpretations.

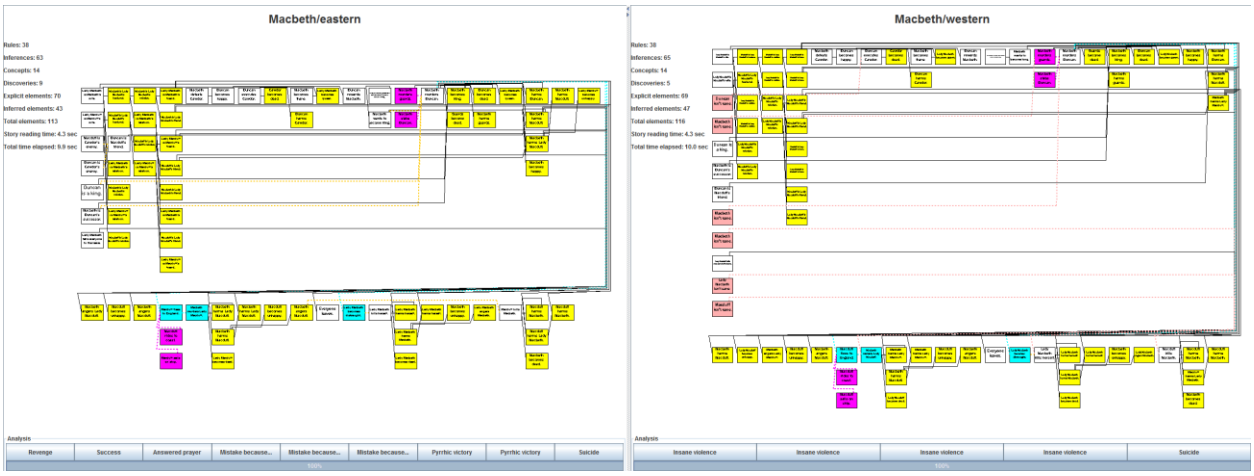


Figure 13 – Analyzing a story from two perspectives with Genesis
An important functionality of Genesis is its ability to do comparisons between stories. One powerful comparison enabled by Genesis is the ability to analyze a story from two different perspectives. In Genesis, a reader's perspective can be modeled as a specific collection of commonsense knowledge which contains the rules, definitions, and reflective knowledge that Genesis uses to understand a story being read. The figure above shows Genesis reading a single version of Macbeth under two different cultural backgrounds. The difference in the knowledge bases of these backgrounds significantly change the elaboration graph and discovered plot units. For example, in one interpretation Macbeth murders Duncan because he is greedy for power. In another, Macbeth is thought to have gone insane because he murders his friend Duncan.

The most pertinent method for plot comparison leveraged in my thesis research is the technique of simultaneously matching and aligning stories. This technique enables rapid and robust comparison between stories by finding the best possible alignment between the plot lines of any two stories. The alignment comparison method takes into account semantic differences, word definitions, event gaps, and a variety of other differences between the stories. In the past I have used this method to enable semantic gap filling during video analysis. The advantages of this method include its ability to find both the best pairings between characters between stories and the best pairings for events (Fay, 2012a, 2012b).

Story	Element 0	Element 1	Element 2	Element 3
Story B Score: 3.01	Israelis know that Egyptians are preparing to attack them.	Egyptians are preparing to attack Israelis.	Israelis defeat Egyptians.	Israelis know to defeat Egyptians.
Story A Score: 3.01	USA knows that Viet Cong is preparing to attack it.	Viet Cong is preparing to attack USA.	USA defeats Viet Cong.	USA knows to defeat Viet Cong.

Element 4	Element 5	Element 6	Element 7	Element 8
Israelis know that Egyptians know that they defeat Egyptians.	Egyptians know Israelis defeat them.	Egyptians don't attack Israelis.	Israelis believe that Egyptians don't attack them.	Gapfilled, Egyptians attack Israelis.
Gapfilled, USA knows Viet Cong knows it defeats Viet Cong.	Gapfilled, Viet Cong knows that USA defeats it.	Viet Cong doesn't attack USA.	USA believes that Viet Cong doesn't attack it.	Viet Cong attacks USA.

Figure 14 - Alignment of stories within Genesis
An important functionality of Genesis is its ability to do comparisons between stories. A powerful technique for comparing the events between two stories is simultaneous matching and alignment. Using this algorithm, Genesis is capable of doing an ordered pairwise comparison of all the events between two stories. Using this technique Genesis can also attempt to fill in gaps of information that may be missing in the one of the stories. This figure shows how some of the events from the Yom Kippur War align with events from the Tet Offensive.

Leveraging Genesis

Genesis provides a powerful computational framework for working on the problem of artificial intelligence in the domain of story understanding. By using Genesis as the baseline for my research in character modeling and story generation, I use a variety of tools allowing me to focus on the problems at hand while taking advantage of decades of research and development. Additionally, it means that my work is available to all future projects that build off of the Genesis system and thus I am contributing to the greater research effort

Summary

In this chapter, I've discussed in detail the Genesis system for story understanding. Genesis is an ongoing research project investigating how computational methods for story understanding can be made to match human intelligence. My thesis research uses the Genesis system as a baseline for working with English, commonsense knowledge, and representing story level information. In the next chapter, I discuss the development of the character modeling approach I've implemented into the Genesis system.

Chapter 5 – Character Modeling

Characters play a crucial role in how humans understand stories and because of this understanding their role in stories is important to human understanding as a whole. Therefore, an import part of my thesis research is developing computational models of characters to enable improved story understanding and generation. In this chapter I detail the character models I have designed and implemented and I describe how they can be used to empower Genesis's story understanding capabilities.

What are characters?

For the purpose of my work, I have chosen to use a very broad and general definition of character. I define a character to be any agent capable of taking action. This aligns with the view of many in narratology (Abbott, 2008; Prince, 2003). As I previously discussed I consider characters can be anything from a person, to an animal, a country, or even a molecule in a chemical reaction. This definition of character is useful because it allows us to consider deep comparisons that might otherwise be missed by a more rigid definition. For example, both countries and individual people may fight, send messages to each other, or support one another. Two molecules with sympathetic properties may combine to form a new larger entity not unlike two companies during a merger.

In working on computational character modeling, I've identified three important features of a robust character model. First, characters are the source of action in a story and a character model should organize a character's actions appropriately. Second, we associate abstract aspects or character traits with people and characters, which allows us to effectively generalize aspects of characters personalities and actions; thus, character traits are an important component of the model. Finally, characters frequently rely on the capacity to think about other characters and objects. Therefore, it's important for characters to be able to form mental models of both their environment and other agents a story. Even for characters that are non-living or non-thinking, we tend to anthropomorphize them to the point of thinking and wanting. For example, we commonly say that electrons want to move to positive charge.

Expanding the Genesis Corpus

Genesis currently has selection of stories available in its corpus notably including a number of Shakespearean tragedies such as Macbeth and Hamlet. In order to facilitate my research, I expanded on the original corpus by incorporating over 15 new stories. Specifically, I added stories across three genres: comedies, fairy tales, and war stories. Each genre was chosen because it offered unique properties which make them interesting from a story understanding standpoint while additionally making them useful to test the capabilities of my work. The comedies genre was chosen because they generally contain a mix of interesting relationships between characters. The fairy tales often have unusual actions and occupancies. The war stories have characters which are non-human entities. Each of these properties requires and strains different aspects of the Genesis system as whole, making it a useful selection.

Characters are sources of action

As I mentioned, the most important aspect of characters is the fact that they are drivers of action. As stated by Henry James, "What is character but the determination of incident? What is incident but the illustration of character?" (James & Edel, 1956). This is an important idea as it both identifies characters as the source of action and also actions taken as characterizing the characters that take them. From this an important part of modeling a character is the actions and events that the character takes part in.

When Genesis reads and understands a story, every action a character takes part in can be stored as a discrete Genesis Innerese event. However, Genesis's event representation can store more than just actions; properties, state changes, relationships, and more can be seamlessly represented in a standardized form. Therefore, the set of events in the character model is actually more powerful than a simple list of actions because it is able to track state, properties, and other information about the character as well. Importantly, the events stored in the character model are time-ordered, meaning that each character model contains a chronological copy of all the plot events experienced by that character.

Having character models track all events that a character takes part in gives Genesis the powerful ability to begin making comparisons between characters regardless of whether those characters are from the same story or are from entirely different genres. In order to enable character comparisons I've developed two different algorithms, both of which are implemented within Genesis.

Comparing Characters with Event Vector Angles

The first comparison method I developed allows for extremely fast pairwise comparison of characters, taking only $O(n)$ time per comparison where n is the number of event types. The high level goal of this method is to create an event vector for each character that tracks how often a character takes part in every type of event and then to compare the angles of these event vectors to determine similarity. However, creating these comparable event vectors is a non-trivial task.

There are three main problems to overcome in creating event vectors for this purpose. First, the features of the vector cannot be too simple. For example, having each feature be simply the appropriate verb would not yield good results because it would cause unwanted conflation of certain actions. For example, Macbeth killing Duncan would be added to the kill feature of both Macbeth's and Duncan's vector based on verb alone, which is obviously undesirable, since killing and being killed are not at all the same thing. Second, events cannot be too specific. For example, we would like for Macbeth killing Duncan to be the same sort of feature as Scar killing Mufasa, even though the characters are from different stories and the characters are different species. Finally, we also need to make sure that the features are not too general. Macbeth being happy and Macbeth being unhappy should map to different features.

In order to tackle this problem I developed the standardized event generalization algorithm. The goal of this algorithm is to generalize events so that after generalization the events take on common forms that are both countable and comparable between characters. The

process takes advantage of the standard hierarchical structure of the Entities, Functions, Relations, and Sequences that form the Innerese events.

The algorithm proceeds as follows. First, a depth first search occurs in order to find all entities in the semantic structure. As it is found, each entity is placed into an ordered set. Importantly, entities are included in this set only once, the first time the entity is encountered. Next, the algorithm iterates through the entity set. Each time a character is encountered, the character is mapped to a new generic entity token with sequential numbering. The first character encountered is mapped to *generic_0*, the second is mapped to *generic_1* and so on. When the character whose events are being generalized is encountered, the entity is instead mapped to a special entity token called *focus*. Once the map has been created, there is a final depth first search in which all of the entities are replaced with their appropriate mapping. This process is applied to each event in the character's event list and the results are stored in the character event vector by incrementing each features value every time that generalized event occurs. For an example of what these generic events and vectors look like, see Figure 15.

	Macbeth	Duncan
Focus kills GEN_0	3	0
GEN_0 kills Focus	1	1
Focus executes GEN_0 for GEN_1	1	0
GEN_0 is focus's wife	1	0
Focus becomes angry at GEN_0	0	0
Focus becomes happy with GEN_0	1	1
GEN_0 is focus's friend	0	1
Focus becomes king	1	0

Figure 15 - Event Vectors for Character Comparison
In order to enable rapid character comparison, first characters' plots are used to create generic event vectors. Each plot event in a character's plot is first made generic. For example the event "Macbeth kills Duncan" in Macbeth's plot would become "Focus kills GEN_0". Macbeth's event vector is then incremented by one in the "Focus kills GEN_0" dimension. The same event "Macbeth kills Duncan" in Duncan's plot would become "GEN_0 kills focus". Duncan's event vector therefore would be incremented by one in the "GEN_0 kills focus" dimension instead.

Once the generalized event vector for each character has been created, characters can be compared rapidly. By taking the cosine of the angle between each pair of character's event

vector, we can even generate useful graphs for visualizing the similarity between large sets of characters. Figure 16 shows a similarity matrix generated by Genesis from event vector comparisons.

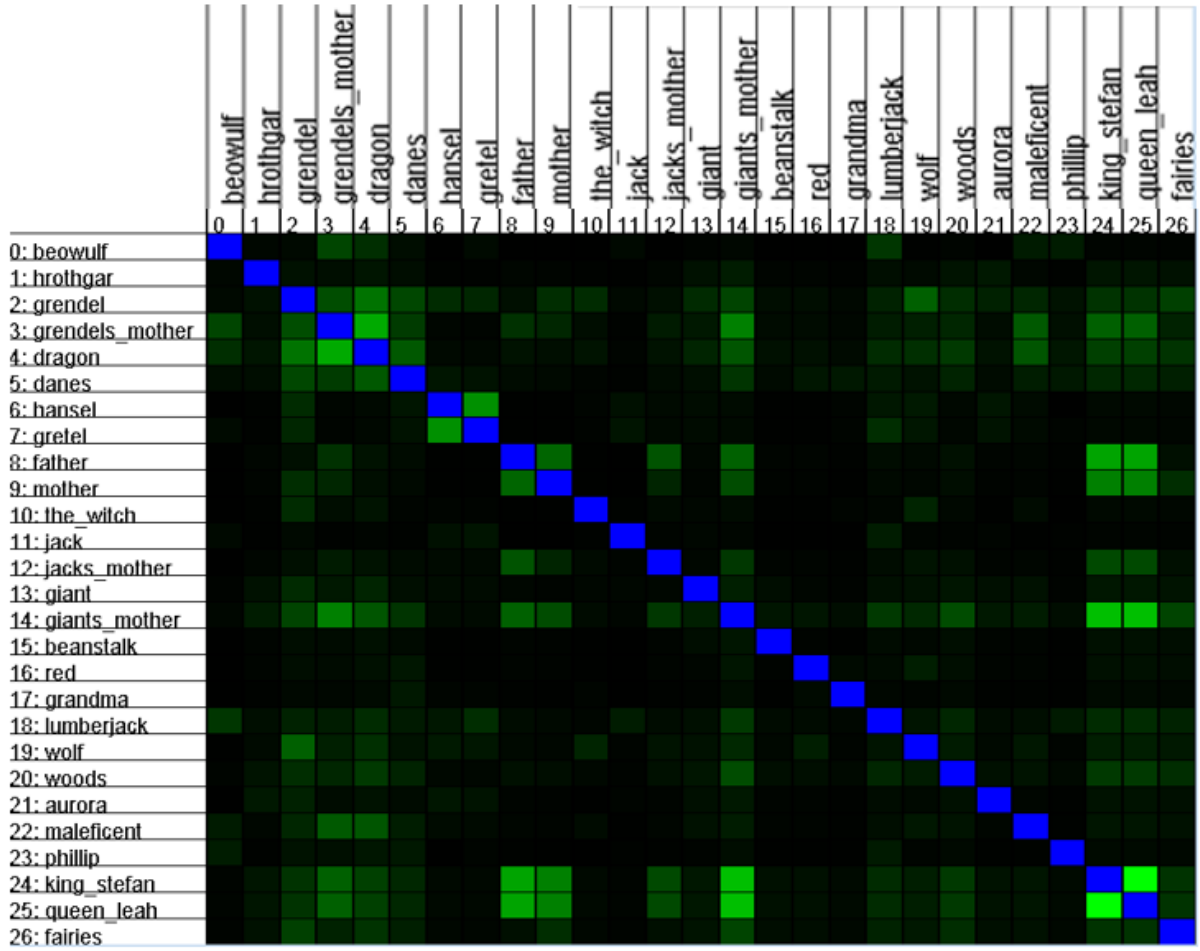


Figure 16 - Comparing character via event vectors
Using the character models that I've developed, Genesis is capable of analyzing characters from across stories. The event vector comparison method calculates a vector of events for each character that represents each character's propensity to take part in various actions. The distances between these vectors' angles are then compared in order to determine the similarity of characters. My extension in Genesis can then create analysis matrices on the fly. This figure shows a bulk comparison of many of the characters from the Fairy Tale corpus via event vector comparison.

While this approach gives us a fast and powerful means for comparing characters, it does have some disadvantages. Most importantly, it does not leverage all of the story level information available in the character model. Specifically, this approach cannot take into account any temporal or causal information, meaning that the ordering and connections of the events in character's stories is completely lost. In the next section, I discuss a second method for character comparison that attempts to give a more plot-level comparison between characters.

Comparing Characters on the Plot Level

While the event vector comparison of characters is powerful and useful, I also wanted to enable a more plot-focused method for determining the similarity between characters. In order to achieve this goal, I developed a comparison method based on aligning the plots of characters. In previous work, I developed the Simultaneous Matching and Alignment algorithm for doing story comparison (Fay, 2012a) that is capable of aligning stories using the Needleman Wunsch algorithm for doing fast alignments (Needleman & Wunsch, 1970). Using this algorithm as a baseline, I was able to rapidly develop a character alignment algorithm.

Given two characters, the alignment comparison attempts to find the best alignment between their plots of those characters. It allows for extra events, gaps, and varying levels of semantic similarity between the plot events. An important and difficult aspect of alignment is keeping the associations between entities of the two plots consistent. Fortunately, most of these problems are solved by the Simultaneous Matching and Alignment algorithm (Fay, 2012a). For the purpose comparing characters, I added an additional constraint to the alignment process that requires that the characters that are being compared match during the alignment of their plots.

The algorithm proceeds by iteratively finding new bindings, or matching constraints, between the entities of the two plots and finding the best alignment for the current set of entities. The algorithm is able to rapidly search through the exponential space of all possible pairings by using a branch and bound philosophy when applying new constraints. An example of the search space is shown in Figure 17. A brief overview of the algorithm follows. First, a root node is created which has a list of unpaired entities from each character's plot and an empty set of pairings. Then the algorithm chooses one entity from a character's list of unpaired entities and pairs it with each entity from the other character's list; for each pairing a new node is created with that pairing appended the list of paired entities. These nodes are then each scored by aligning the characters' plots using the nodes set of paired entities as a set of constraints for the matching. This process of creating child nodes then repeats starting with whatever node currently has the highest alignment score. Once a leaf node is found the algorithm knows that set of pairings produces the best alignment between the character's plots having both consistency of matched characters and the highest number of aligned plot elements.

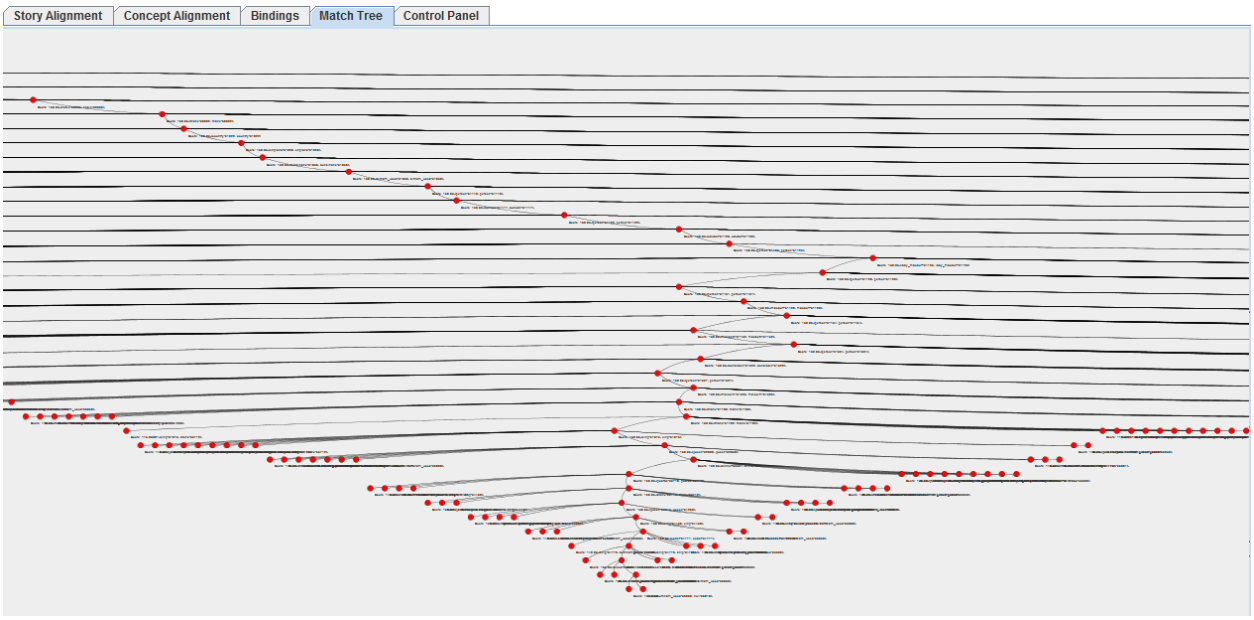


Figure 17 - A partial match tree constructed during story alignment
The simultaneous matching and alignment algorithm within Genesis allows for the efficient and optimal alignment between stories to be found. This problem is difficult because it requires searching for the optimal set of parings between entities in the stories which is an exponential search space. The simultaneous matching and alignment algorithm is able to rapidly prune and search this space. A visualization of part of the search tree from aligning Hamlet and Macbeth is shown in this figure.

After finding the best alignment between the plots of the characters I apply a score function to the alignment. Depending on a the comparison wanted, different score metrics are available, but the most simple and still effective metric is summing the number of matching events in the alignment and then dividing by the number of events in the longer plot. Using this metric allows for the comparison and analysis of large numbers of characters based on the similarity of their plots. Similar to the event vector comparison method, this allows for generation of useful visualizations comparing characters. Figure 18 shows an example of a similarity matrix that Genesis can generate using alignment comparisons.

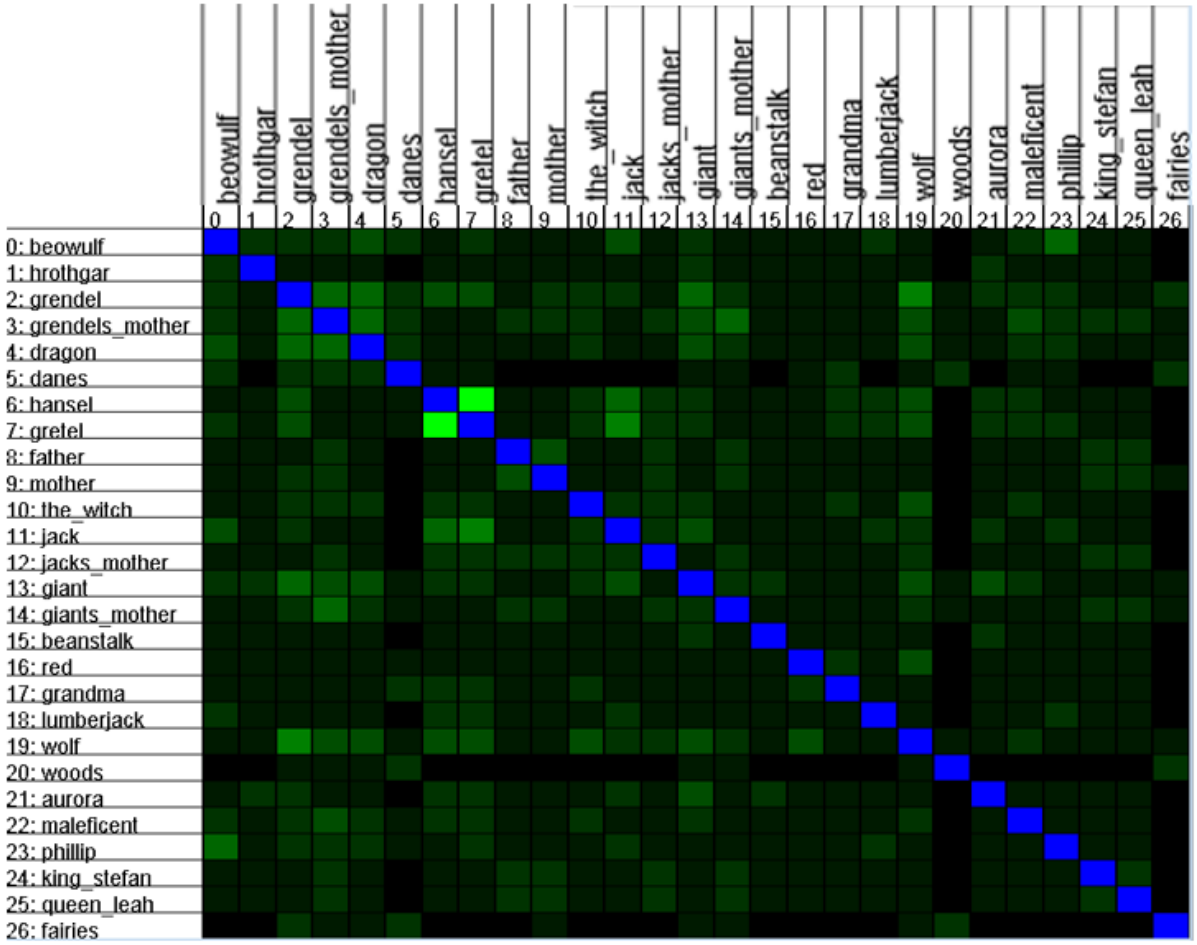


Figure 18 - Comparing a corpus of characters via alignment

Using the character models that I've developed, Genesis is capable of analyzing characters from across stories. The alignment comparison method uses an improved version of the simultaneous matching and alignment algorithm within Genesis to align pairs of character models. The resulting scores from doing this alignment are used as the comparison metric for comparing characters. My extension in Genesis can then create analysis matrices from this data on the fly. This figure shows a bulk comparison of many of the characters from the Fairy Tale corpus via alignment comparison.

Additionally, this type of comparison enables us to delve in and see how the plots of the characters align, easily accessible by clicking on a matrix cell. An example of visualizing character alignments is shown in Figure 19

Wolf eats Grandma.	Wolf disguises as Grandma.	Red thinks wolf is a Grandma.	Wolf eats Red.	Lumberjack finds wolf.	---	Lumberjack kills wolf.
Grendel eats Danes.	---	---	---	---	Beowulf attacks Grendel.	Beowulf kills Grendel.

Figure 19 - Comparing a pair of characters via alignment

The alignment method for comparing characters is enabled by an extensions of the simultaneous matching and alignment algorithm within Genesis. In addition to doing comparisons across a corpus of characters, Genesis can now also drill down and inspect the pairwise comparisons between characters. Shown above is the excerpt of a character alignment between Grendel from Beowulf and the lumberjack from Red Riding Hood. The plot elements in green are aligned between the characters' plots.

Comparison of Methods

The two methods of comparing characters that I have discussed in this chapter both have unique advantages and disadvantages which can make one more or less useful depending on the situation at hand. In this section, I will describe to simple scenarios in order to illustrate how the methods compare.

First, consider the two simple stories "John angered Mark. Mark killed John." and "Jane angered Mary. Mary killed Sally." When examining the characters of Mark and Mary, the event vector method and the alignment method will yield different results. Specifically, the event vector comparison will give Mark and Mary a perfect match. On the other hand, the alignment method shows that the characters only match on half of their events. This is because the alignment method can retaining consistency between which characters are matched while the event vector method cannot. This means that during the comparison, alignment method cannot match Mark's complete story with Mary's complete story. This is because while Mark becomes angry at a character, John, and then kills that same character, Mary becomes angry at one character, Jane, but kills a different character, Sally. As shown by this example, the alignment comparison method is better than the event vector comparison method when dealing with cases involving differences on the story level as opposed to the event level. A visual representation of this example is shown in Figure 20.

Story 1	Story 2
John angered Mark	Jane angered Mary
Mark killed John	Mary killed Sally

	Mark	Mary
GEN_1 angered Focus	1	1
Focus killed GEN_1	1	1

Event Vector Similarity Between Mark & Mary	1.00

	Element 1	Element 2	Element 3
Mark	John angered Mark	Mark killed John	---
Mary	Jane angered Mary	---	Mary killed Sally

Alignment Similarity Between Mark & Mary	0.33

Figure 20 - Alignment Comparison Advantage
The alignment comparison method for comparing characters has the advantage of maintaining consistency between how characters and plot elements of a story are matched. This means that it often gets more accurate results for character comparisons which involve the correlation between multiple events. Shown in this figure are the comparisons between Mark and Mary using both the event vector and the alignment comparison methods. In this example the alignment comparison result is more favorable.

Second, consider the two simple stories "John is generous. John gives aid to Mark. John gives aid to Sam. John gives aid to Tom," and "Jane is generous. Jane gives aid to Mary." When examining the characters of John and Jane, we would expect the characters to be very similar. However, the two comparison methods give diverging results, with the alignment method determining that the characters are less similar and the event vector comparison determining that the characters are more similar. This is because the event vector comparison method is able to better handle repeated actions and out of order events for determining the similarity of characters. The alignment method on the other hand determines that only one out of three of the giving aid events can be matched, leading it to decide that the characters are dissimilar. A visual representation of this example is shown in Figure 21.

Story 1	Story 2
John is generous	Jane is generous
John gives aid to Mark	Jane gives aid to Mary
John gives aid to Sam	
John gives aid to Tom	

	John	Jane
Focus is generous.	1	1
Focus gives aid to GEN_1	3	1

Event Vector Similarity Between John & Jane	0.89

	Element 1	Element 2	Element 3	Element 4
John	John is generous	John gave aid to Mark	John gave aid to Sam	John gave aid to Tom
Jane	Jane is generous	Jane gave aid to Mary	---	---

Alignment Similarity Between Mark & Mary	0.5

Figure 21 - Event Vector Comparison Advantage
One of the advantages of the event vector comparison method for comparing characters is that it is better able to handle repetition of events and actions of characters. It also better handles stories in which the order of events does not matter. Shown in this figure are the comparisons between John and Jane using both the event vector comparison and the alignment comparison methods. In this example, the event vector comparison's result is more favorable.

Character Traits inform Character Actions

The next major component of the character model is the incorporation of character traits. In everyday life people ascribe traits to people, characters, and other entities frequently in order to help organize and understand their experiences. Character traits allow the grouping of thoughts, personalities, and actions under general labels and then to reuse those labels to refer to qualities in new encounters.

For the purpose of my thesis I've decided to use a definition of character trait that combines the idea of personality with the idea that characters are drivers of action. To that end, I define a character trait as being a collection of generalized Genesis Innerese events that are given a label describing the collection. For example, the "violent" character trait may be made up of the set of events "XX attacks YY", "XX kills YY", "XX fights YY", "XX angers easily", etc. It's important to note that just as event arrays store a character's entire plot, the character traits are collections of Genesis Innerese events. Thus, character traits can similarly embody actions, relations, state changes, and plot fragments.

The character model I've developed contains a collection of character traits. By default characters can be assigned either a trait or the negation of a trait. For example, the character of Macbeth can be assigned both "Macbeth is violent" and "Macbeth is not a pacifist". Using character traits, Genesis is capable of leveraging its knowledge base to attempt to learn more information about the character. Within the Genesis knowledge base there are a collection of pre-defined character traits that can be used to better understand characters in stories. These pre-existing traits can be actuated in two different ways, explicitly and implicitly. An explicit trait assignment consists simply of a statement such as "Macbeth is violent." An implicit assignment on the other hand is based on Genesis's observations of action in an ongoing story. When Genesis detects an event such as "Macbeth defeats rebels," it is capable realizing that this matches examples of various forms of violence that it knows about and so applies the violent character trait to Macbeth. In both cases, when a new character trait is detected, Genesis loads in any knowledge that it has about the character trait, including common sense and reflective knowledge. This knowledge can then be applied throughout the story whenever the character with the trait is involved.

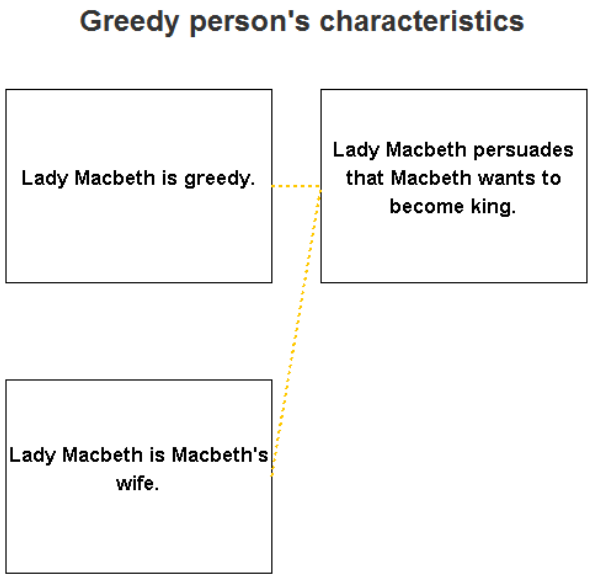


Figure 22 – Pre-defined Character traits within Genesis
Genesis is capable of handling pre-defined character traits both explicitly and implicitly. For a trait to be considered explicitly, Genesis must see an explicit plot event indicating that a character has a trait, for example "Lady Macbeth is greedy". For a trait to be considered implicitly, Genesis must detect a key event linked to a trait such as stealing from someone to the greedy trait. Either way, once a trait has been detected, Genesis loads in commonsense knowledge relevant to that trait and associates to that character. That information allows Genesis to causally connect events that it otherwise would have been able to such as the ones shown in this figure.

Mental Modeling

The final component is the mental model of the character. Genesis's initial mental modeling capabilities were created in order to allow Genesis to reason about the various entities in a story in a semi-independent way. Essentially, Genesis is able to draw in specific background knowledge based on what it knows about an entity and then apply that background knowledge in a selective manner. An example of this is how Genesis uses character trait background knowledge. When Genesis detects a character trait, it selectively loads all of the information it has on that character trait. However, that information is only loaded into the mental model of the entity that has that particular trait.

Genesis's early mental modeling worked on the system level. Essentially, the reader would create mental models of the entities it encountered and then load those mental models selectively with knowledge. In order take this idea to the next level, I've developed the Cascading Character Model which allows a character's model to contain not only the information on itself but also internal models of how that character understands the other characters with which it interacts.

The core idea of the Cascading Character Model is to allow each character's model to contain partial character models of each other character. These mental character models could then have their own internal models of other characters as well. This process can proceed

recursively for many levels. This is akin to how a human may think about another person, and then think about how that person may perceive the original human (e.g., Matt thinks about Mark and also about what Mark thinks about Matt). In general people tend to be able to handle only a few recursions easily and so I've imposed an artificial limit within Genesis to cap the depth of mental models at three.

This sort of character model is most advantageous for improving Genesis's capability in story understanding. Enabling an understanding of how characters perceive one another is a step towards enabling Genesis to understand stories with complex character motives, in which characters actions can be analyzed based not only on the true state of the world but also on what a reader suspects a character's internal representation of the world to be. Consider this simple example with one level of recursion: in the story of Romeo and Juliet, Juliet takes a drug that produces a temporary deathlike state. Romeo, however, believes Juliet has died and acts accordingly, killing himself out of grief. Thus this model would allow Genesis to make sense of Romeo's actions, because even though Genesis knows Juliet is not dead the system also understands that Romeo is acting according to his own false perception that she is dead.

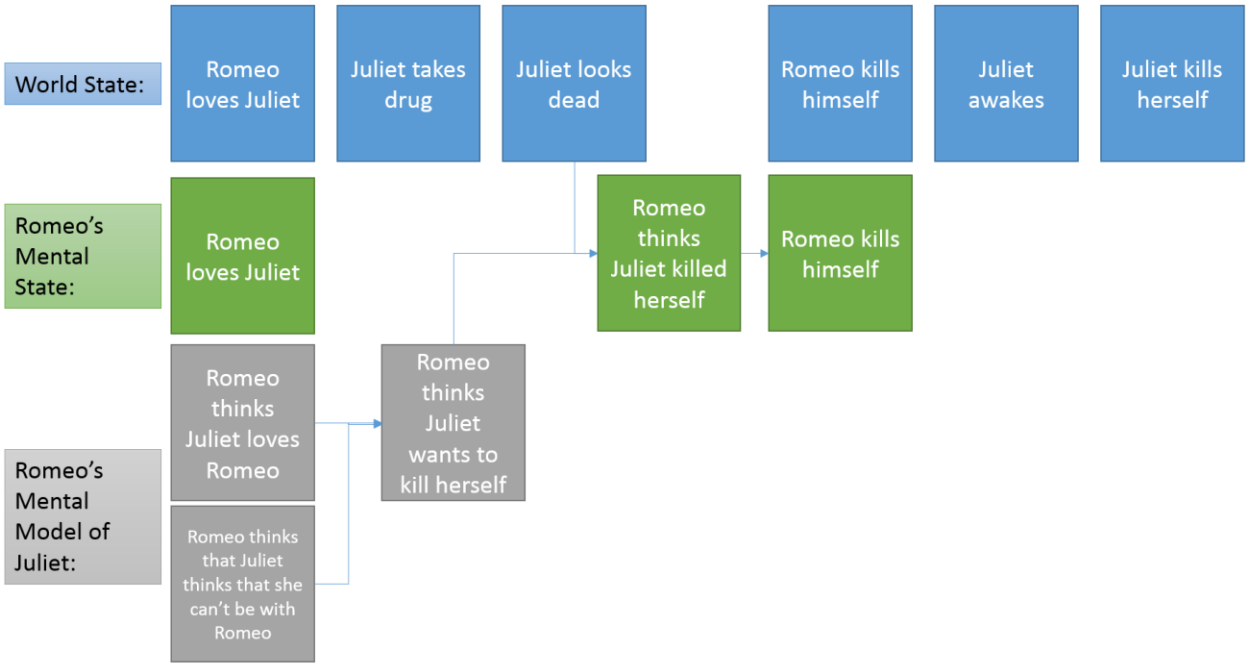


Figure 23 - Cascading character mental models
A cascading character model allows characters to have recursive levels of knowledge about themselves, other characters, and the world. This figure shows an example of how Romeo's flawed model of Juliet's mental state causes the tragic end of Romeo and Juliet. Romeo's model of Juliet does not know about the drug that Juliet took, which causes Romeo to misinterpret events.

Summary

In this chapter, I have discussed my work in developing powerful models for storing character information. I implemented these character models within Genesis that are capable of tracking characters' actions, traits, and mental models. Additionally, the models can be compared via both the event vector comparison method and the character alignment method. In the next chapter, I discuss how Genesis can use a collection of stories to automatically learn about characters and their traits. In subsequent chapters, these models will be used to fuel story generation.

Chapter 6 – Learning Character Traits

In the previous chapter, I discussed how I have developed power character models to empower computational story understanding to better understand characters and their actions. An important component of these character models is character traits. A character's set of traits influences what sorts of actions a character is willing to take. For example, a character that is violent may be prone to attacking others and a character that is generous would often give things to others. I've developed methods for learning character traits from a corpus of stories autonomously in order to demonstrate how Genesis and other story understanding systems can learn from past experience.

In this chapter I present two parallel methods for trait learning, one that can learn the traits in a completely unsupervised manner and one that works in a supervised manner.

Learning traits in an Unsupervised Way

A major goal of this work was to discover a way for an artificial intelligence to learn about commonalities between characters in autonomous way. In order to succeed in this, a system should be able to read a collection of stories and draw systematic connections between the characters. Ideally this will allow the system to learn traits which are possessed by one or more characters and can be used to understand the actions characters take. For example the system should be able to detect that kind characters tend help others and donate their time or belongings while violent characters tend to attack and steal from others. It should also recognize that characters can have overlapping traits, for example Robin Hood might demonstrate both kindness and violence.

In my research I've developed a technique to model this sort of relationship between characters, traits, and actions by leveraging recent work in the realm of topic modeling.

Topic Modeling

Topic modeling is an area of machine learning and natural language processing that is most commonly used to learn what topics are shared among a corpus of documents. An early example of topic modeling approaches is Latent Semantic Indexing (Papadimitriou & Tamaki, 1998), which attempts to find the hidden semantic topics of documents for the purpose of information retrieval. Since then a number of alternate approaches have been developed including latent dirichlet allocation (Blei, Ng, & Jordan, 2003) and Hierarchical dirichlet processes (Teh & Jordan, 2006).

However, regardless of the approach, the goal of learning topic models remains the same: given a set of documents made up of collections of words, learn the topics shared between documents that determine word choice and learn the distribution of these topics in the documents. Figure 24 is a visual representation of topic modeling.

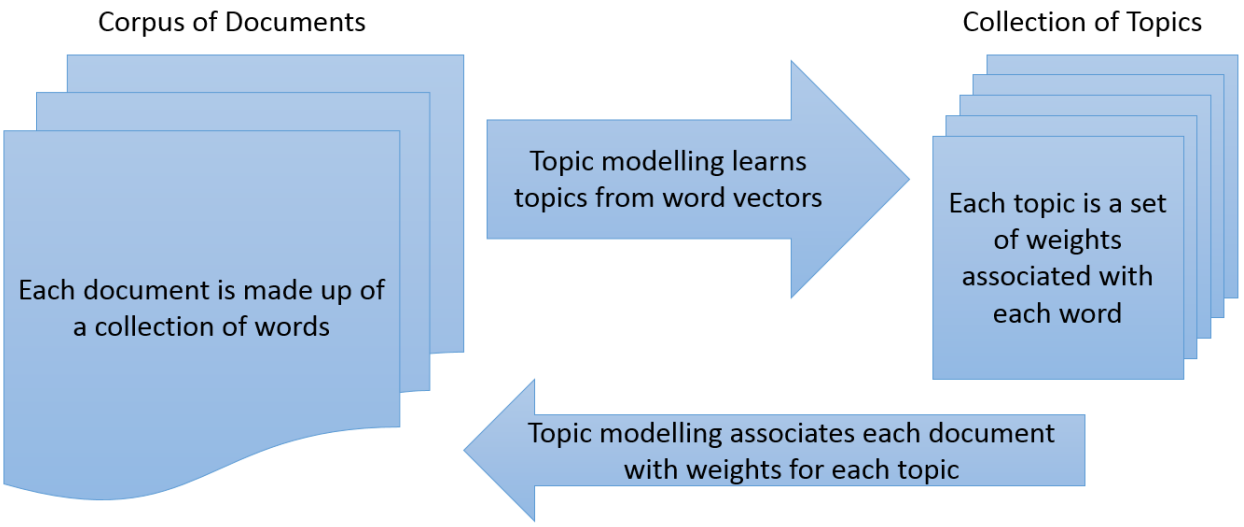


Figure 24 - Visual explanation of topic modeling
Topic modeling is a method for automatically learning about and categorizing documents. In this figure, the relationship between documents, topics, and words is illustrated. Each document in topic modeling is assumed to be made up of one or more topics. Each topic in turn is made up of a collection of related words. Topic modeling algorithms attempt to learn the topics that make up each document given only the documents and the words they contain. The output of a typical topic modeling algorithm is a collection of topics with the likelihood of each topic containing each word and a set of likelihoods of each document containing each topic.

As an example, consider using topic modeling to learn about the topics of encyclopedia articles. In this example, each encyclopedia article represents a document. Let's consider encyclopedia pages about computer science, organic chemistry, and bioinformatics. The bioinformatics and chemistry pages may contain words about organic compounds, but only bioinformatics may mention genes. Organic chemistry and computer science may share certain terms about proteins because numerous papers have been published on efficient folding algorithms. All three documents may contain a lot of scientific jargon. From this we could hypothesize that these documents likely share certain common topics, while each also has unique topics. A visualization of this example is shown in Figure 18.

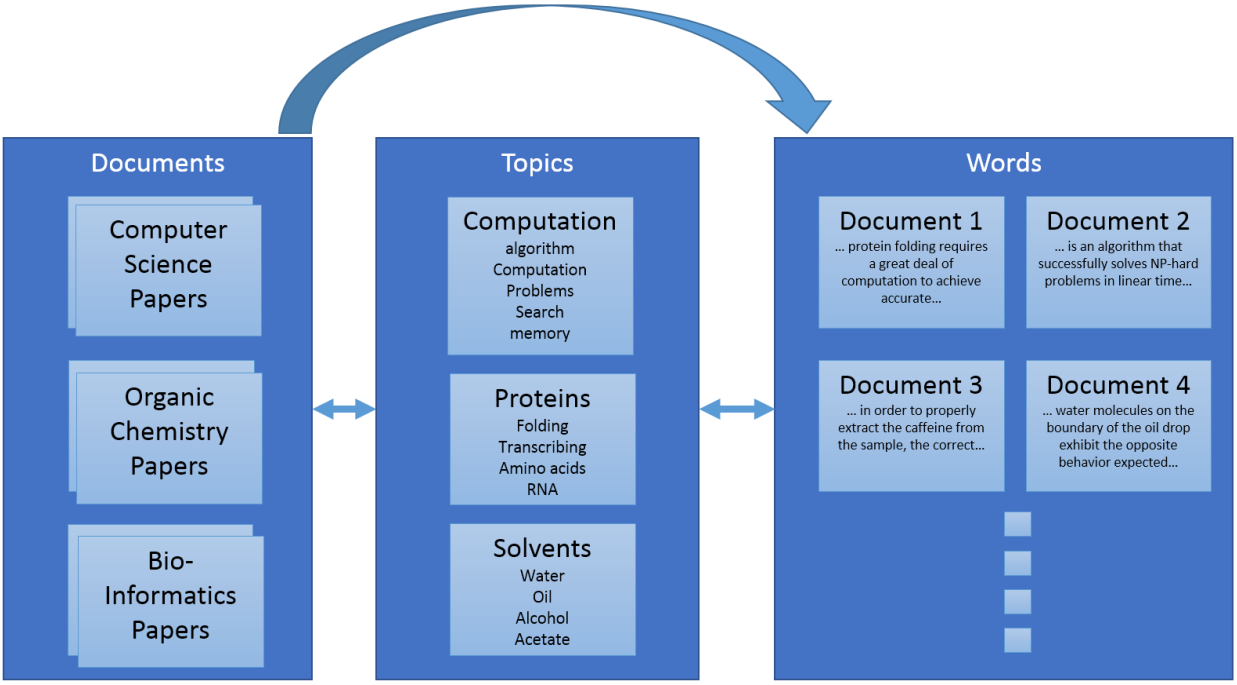


Figure 25 - Topic modeling on scientific documents
This figure shows an example of the sorts of topics that can be learned from analyzing a set of scientific documents. The documents might come from a number of origins such as scientific journals in fields such as computer science, organic chemistry, and bioinformatics, shown on the right. Each of these documents is made up of a set of words, example snippets of which are shown on the right. The topic modeling algorithm converts the documents to vectors of word counts and then analyzes them to produce topics like those shown in the center. The algorithm also determines the mixture of topics present in each document. It is worth noting that topic modeling does not provide names to traits, the ones shown would need to be added manually later.

Learning Character Traits via Topic Modeling

Metaphorically, the structure of topic modeling seems like a powerful candidate for learning about characters. Each character model contains an initially unknown collection of character traits. These traits include a wide variety of descriptors such as good, evil, violent, charitable, energetic, happy, unlucky, rich, young, and old. Depending on the collection of traits that make up a character, we would expect that character to take part in various types of events. For example, an energetic person may be prone to exercise and a person who is rich and charitable may be likely to give money to charity. Using a variation of topic modeling, I show how Genesis is able to learn this collection of traits simply from observing the actions of a set of characters that exhibit them.

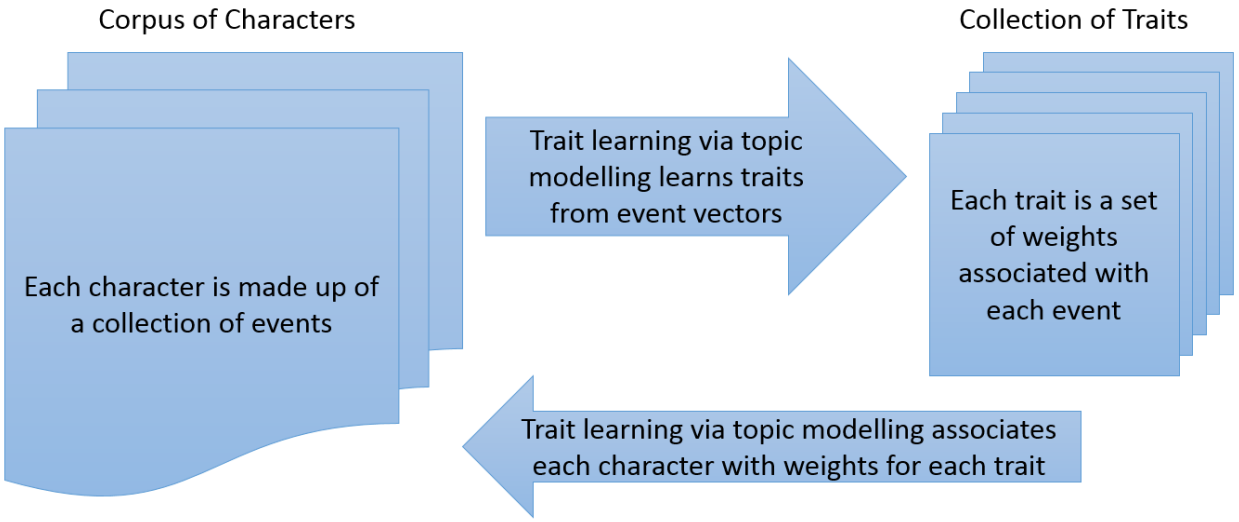


Figure 26 – Modeling character traits with topic modeling
Using an idea analogous to topic modeling allows character traits to be automatically learned from a collection of characters. This method leverages the characters models that I discussed in the previous chapter. Each character is assumed to have a collection of one or more associated traits and each trait is made up of a collection of related actions and inactions. Using this model of characters, traits, and actions allows us to leverage techniques from topic modeling in order to automatically learn both which traits exist and also which traits each character has.

In order to model characters using topic modeling, Genesis reads a corpus of stories to learn about specific characters and observe what those characters do. As previously discussed, Genesis can interpret stories and collect the story elements for all the characters. By leveraging the character models I implemented, Genesis is capable of looking at characters independent of the specific story the character is from.

Once all the characters and associated events for each character have been collected, they still need to be processed in order to make use of topic modeling approaches. This can be done using the technique for making events generic that I described in the previous chapter. This technique was previously used for the event vector comparison of characters in Genesis. Once the events no longer contain character specific information, each character document simply becomes the vector of the generic events, counting the number of occurrences of each event for the character. For an example of what these event vectors look like see Figure 27.

	Macbeth	Duncan	Macduff	Claudius
Focus kills GEN_0	3	0	1	3
GEN_0 kills Focus	1	1	0	1
Focus executes GEN_0 for GEN_1	1	0	0	0
GEN_0 is focus's wife	1	0	1	1
Focus becomes angry at GEN_0	0	0	2	0
Focus becomes happy with GEN_0	1	1	0	1
GEN_0 is focus's friend	0	1	2	1
Focus becomes king	1	0	0	1

Figure 27 - Event Vectors used for topic modeling
During the preparation process for learning character traits for topic modeling, the characters' plots are converted into a generic vector form. Each plot event, such as "Macbeth kills Duncan" becomes a generic event based on the character whose plot the event came from. For Macbeth, the event "Macbeth kills Duncan" becomes "Focus kills GEN_0". For Duncan, the same event would become "GEN_0 kills Focus". This preserves the overall meaning of the event while allowing it to easily match against events from other characters' plots. This figure shows a subset of the vectors that would come from processing characters from Macbeth and Hamlet.

Once the characters' events are in vector form, the approach becomes agnostic to the particular algorithm for topic modeling being used. This is because for all intents and purposes, each event's representation can be treated as a single symbol just like a word from traditional topic modeling procedures. Therefore, it becomes possible to use well tested algorithms, methods, and tools for the computation of the topic modeling process. Two common methods for this are Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA) (Blei et al., 2003; Deerwester, Dumais, & Landauer, 1990).

The techniques used in LSA draw from linear algebra. The core of the technique is as follows. The complete set of occurrence vectors (whether word or event) are combined into a single matrix that then contains all of the frequency data of the corpus. Once this matrix is constructed the algorithm uses singular value decomposition on the matrix (Golub & Reinsch, 1970). The number of dimensions used in the decomposition determines the number of topics. The left-hand matrix of the decomposition will contain the correlation of each word or event to each topic. The right-hand matrix of the decomposition contains the correlation of each document or character to each topic.

The techniques used in LDA draw from Bayesian statistics. The goal of LDA is to learn the set of topics or traits using Bayesian inference. The generative model set up is that each document has a distribution of topics and each topic has a distribution of words. There are also two priors introduced, the prior on the probability to see any particular topic and a prior on the probability to see any particular word. Once initial parameters have been set, Gibbs sampling is

used to determine the set of parameters, in effect the topic to document correlations and the word to topic correlations, for the model that produces the highest likelihoods for the input dataset.

For the purposes of my research, I primarily used LSA to perform the topic modeling on character traits (Deerwester et al., 1990). This is due to the fact that it is deterministic which made it more useful for integrating and testing within the context of the larger system. However, LDA and other probabilistic approaches should perform equally well (Blei et al., 2003; Girolami & Kabán, 2003). In order to not reinvent the wheel, I used well tested implementations of LSA and LDA provided by the gensim library (Řehůřek, n.d.).

Examples from Topic Modeling for Trait Learning

This approach can now be used to learn character traits automatically from corpuses of stories that have been processed by Genesis. Exploring this approach has yielded interesting results. An example of how the overall process looks, along with some examples of discovered traits is shown in Figure 28. Traits learned included the ideas of vengeance and ambition.

One amusing discovery was the concept of doomed royalty, in which a character who has this trait is likely both to be king and to be killed. The discovery of this trait was particularly interesting because it is not something I noticed nor is explicit in the corpus. However, despite it not having simple English term, I immediately recognized it as a pattern that existed I in a vast array of stories – even beyond stories in the Genesis corpus. This demonstrates how topic modeling is capable of discovering novel concepts and thus a promising example of the power of topic modeling for this problem.

Trait				Example Character
Vicious Ambition	Focus is GEN_0's successor	Focus kills GEN_0	Focus wants to be king	Macbeth
Romantic	Focus falls in love with GEN_0	Focus meets GEN_0		Miranda
Doomed Royalty	Focus is the king	Focus dies.		King Hamlet
Mutual Attraction	Focus loves GEN_0	GEN_0 loves Focus		Hero
Vengeful	GEN_0 angers Focus because GEN_0 kills GEN_1	Focus kills GEN_0	Focus is GEN_0's friend	Macduff
Spiteful Sibling	Focus is GEN_0's brother	Focus wants to kill GEN_0		Alonso
Violent	Focus dislikes GEN_0	Focus kills GEN_0	Focus attacks GEN_0	Beowulf

Figure 28 - Learning character traits using topic modeling
Using the topic modeling to learn character traits allows for a number of characteristics to be discovered. The system first extracts the plot elements from the character models and converts them into frequency vectors. Then the system uses LSA to learn a set of traits and character associations. The traits shown are interesting examples learned from a small corpus of characters from Shakespearean tragedies, fairy tales, and romantic comedies. On the right side of the table are examples of characters strongly associated with the traits.

Strengths and Weaknesses of using Topic Modeling to Learn Traits

Topic modeling provides a number of benefits as a form for learning traits. Most notably, it can proceed in an entirely unsupervised manner on unlabeled data. This gives it great flexibility in terms of the types of datasets that it is capable of learning traits from. Additionally, topic modeling scales very well. Topic modeling for documents has been used successfully on datasets as large as Wikipedia (Chaney & Blei, 2012). Therefore, it will likely be of continued interest as Genesis and other story understanding systems progress to analyze characters and stories from very large corpuses.

While using topic modeling to learn character traits is a powerful method, it is not without its disadvantages. Its primary weakness is that modeling traits via topic modeling will produce a large number of traits that are not human understandable. That means that many traits will be discovered that correlate events even though there is no logical connection between them. For example, a trait might be discovered that draws a correlation between "jumping", "killing", and "donating". This flaw stems from topic modeling itself and cannot be readily avoided, as given a corpus it is likely there will be countless coincidental correlations with statistical significance found between the elements (Chang & Gerrish, 2009).

The discovery of nonsensical character traits feeds into a second problem, which is that no discovered trait will have a label initially. This is undesirable because in the common usage of character traits, we humans rely on the ability to refer to traits by name and the names themselves carry meaning. This means that at some point detected traits will likely need to be annotated either by hand or by a secondary system in order to be evaluated, reused, and further analyzed.

Finally, this approach shares a weakness with the event vector approach for character comparison. Namely, it is unable to take into account any temporal information relating the events that make up the discovered traits.

Despite these issues, modeling traits via topic modeling is a very powerful idea. Additionally, despite some of the oddities of traits that are detected by this method, the fact that the traits are statistically significant means that they can still be useful for improving the capabilities of computational story understanding.

Learning traits in a Supervised Way

While the topic modeling approach is an effective and useful way of modeling character traits, I wanted to investigate a second method for doing trait learning that might be able to overcome some of the weakness of that method. In my research I developed a supervised method of learning traits inspired by the idea of near miss learning (Winston, 1970, 1986). This method differs from the topic modeling approach because it requires a bit of additional information either from the user or within the story texts. Specifically, the method requires labeled, positive examples of characters that have the trait of interest. Optionally, the methods can also leverage negative examples to obtain a more precise trait definition.

Supervised trait learning integrates well with Genesis's language processing capabilities. For example, the system is now able to learn traits passively while reading stories by watching for explicit trait definitions such as "Macbeth is violent" and "Lady Macduff is not violent." This automatically integrates with directed learning as a user could provide Genesis the same sorts of sentences and the system could immediately combine that active training with what it learned passively.

Alignment-Based Trait Learning

My method for doing supervised trait learning leverages the character alignment techniques that I created and discussed in the previous chapter. As such, the technique is referred to as alignment-based trait learning. The method proceeds as follows. First, the positive examples of the trait are collected. From an initial pair of positive examples, the characters are compared via alignment. In order, a generalized version of each event from the alignment is stored in a trait event list. Matching events are scored with a 1 and each non matching event is scored as 0. Then, each other positive example is compared to this trait event list. Whenever events of the character align with those of the trait event list, those event's score is adjusted towards 1 with the Function 1.

1. $Score = \frac{Score * Count + 1}{Count + 1}$

Events that don't match are adjusted towards 0 as shown in Function 2.

2. $Score = \frac{Score * Count}{Count + 1}$

Once all positive examples have been exhausted, the method goes through each negative example. Similarly, each negative example character has its plot elements aligned with the trait event list. However in this case, for every match between a plot element in the negative example and the trait event list results in that event being strongly adjusted towards negative values using the function shown in Function 3.

3. $Score = \frac{Score * Count - 2 * \frac{Positive\ Count}{Negative\ Count}}{Count + 1}$

In each of these functions, count refers to the total number of examples. Positive count is the number of positive examples, and negative count is the number of negative examples. Score is the current weighting of the plot element.

Example of Alignment-Based Miss Trait Learning

Using this method, interesting traits can be learned quickly and with very low numbers of examples. For example, consider a character trait "Vicious Ambition", which we may want to encompass the sorts of actions that power hungry murderers may take. On a high level we will first tell the system that Macbeth and Claudius have vicious ambition. Given these two initial positive examples, the system can begin constructing the trait. This causes the system to come up with an initial idea of what makes up the trait based upon how the two characters align.

The algorithm proceeds as follows. First the algorithm does an alignment between the initial two characters of Macbeth and Claudius. From this alignment, the algorithm records all the plot elements, whether aligned or unaligned, in the trait definition. Plot elements that are aligned are assigned a score of 1 indicating that they should be considered as part of the trait definition. For example the events "Macbeth is Duncan's successor" and "Claudius is King Hamlet's successor" align and so the corresponding element in the trait is marked with a 1. Plot elements that do not align are marked with a zero indicating that they currently should not be considered as part of the trait. For example, "Macbeth angers Macduff" does not align with another event and so the corresponding element in the trait is marked with a 0. Figure 29 shows an abbreviated version of this process and the result.

Character	Plot Element 1	Plot Element 2	Plot Element 3	Plot Element 4
Macbeth	Macbeth is a person.	---	Macbeth is Duncan's successor	Macbeth wants to be king
Claudius	Claudius is a person	Claudius dislikes King Hamlet's son, Hamlet	Claudius is King Hamlet's successor	Claudius wants to be king

Character	Plot Element 5	Plot Element 6	Plot Element 7	Plot Element 8	Plot Element 9
Macbeth	Macbeth kills Duncan	Macbeth becomes king	Macbeth angers Macduff	---	Macduff kills Macbeth
Claudius	Claudius kills King Hamlet	Claudius becomes king	---	Claudius drives Hamlet insane.	Hamlet kills Claudius

Vicious Ambition	Focus is a person	Focus is GEN_0's successor	Focus wants to be king	Focus kills GEN_0	Focus becomes king	GEN_1 kills Focus

Figure 29 - Learning vicious ambition from positive examples
After learning about Macbeth and Claudius from reading the stories of Macbeth and Hamlet, Genesis can begin learning the trait of vicious ambition. This figure is a summary of the learning that occurs in the alignment-based learning. The plots of Macbeth and Claudius are aligned, and then the system uses the alignment to determine which elements are likely part of the trait. The green elements in this figure show the aligned elements which become the basis for the trait. The system's understanding of vicious ambition at this point is shown as the bottom set of plot elements.

From just the first two examples, Genesis already has a reasonable sense of what the trait for vicious ambition looks like. However, there are notably two events that don't fit well with what one might expect to be a part of the trait. Specifically, the fact that "the character is a person" and "someone kills the character". One way the system is capable of improving its concept of the trait is via a negative example. Therefore, the system is next informed that Duncan does not have vicious ambition. This causes the system to go back and update its concept of vicious ambition by adjusting the weights of elements that align with the negative example's plot. Figure 30 shows a visual example of this process.

Character	Plot Element 1	Plot Element 3	Plot Element 4	Plot Element 5	Plot Element 6	Plot Element 9
Vicious Ambition	Focus is a person.	Focus is GEN_0's successor	Focus wants to be king	Focus kills GEN_0	Focus becomes king	GEN_1 kills Focus
Duncan	Duncan is person	---	---	---		Macbeth kills Duncan

Vicious Ambition	Focus is GEN_0's successor	Focus wants to be king	Focus kills GEN_0	Focus becomes king

Figure 30 - Learning vicious ambition from a negative example
Genesis can learn an initial trait from solely positive examples. However, negative examples help the system refine its understanding of a concept. Previously, Genesis learned the concept of vicious ambition from the positive examples of Macbeth and Claudius. This figure shows a summary of how Genesis refines this concept through the negative example of Duncan. Duncan's plot aligns with the existing trait as shown and this causes the matching elements to become negatively weighted. The result is the refined trait shown in the bottom of the figure.

As seen, using the provided negative example, Genesis is able to refine its initial vicious ambition trait. The system proceeds by aligning the negative example against the existing trait concept. All of the matches that come out of this alignment become negatively weighted. For example, from vicious ambition trait learned from Macbeth and Claudius, there is a plot element where "Macduff kills Macbeth", generalized to "GEN_1 kills Focus" that aligns with an event from Duncan's plot, "Macbeth kills Duncan." Because of alignment the element "GEN_1 kills focus" is down weighted towards -1 in the trait, essentially removing it from the current trait definition. Elements that do not align with the negative example, such as "Focus wants to be king," are unaffected.

This process successfully removes the more questionable plot elements that were a part of the original concept are matched which leaves a succinct understanding of the trait within Genesis. The final understanding of the trait *Vicious Ambition* is now "Focus is GEN_0's successor. Focus wants to be king. Focus kills GEN_0. Focus becomes king."

Summary

In this chapter I have discussed two powerful methods that I have developed for learning character traits from a corpus of stories. The methods have been successfully implemented using Genesis story information, expanding on its ability to understand character actions and motives and the relationships between characters. Specifically, I've identified how topic modeling techniques can be applied to learning character traits in the story domain. I've also developed an alignment-based trait learning algorithm that can learn traits from annotated characters.

I have found both techniques to be powerful methods for story understanding. As I approached the process of development of the plot generation system, it became apparent that the problems with topic modeling made it less ideal for my purposes due to the noisiness of its data. The alignment-based approach tended produce cleaner data and its ability to preserve more story structure was valuable. That said I believe that topic modeling has a lot of potential and future

work could focus on refining the technique and using its advantage of being able to process very large datasets.

In the next chapter, I discuss my work in creating a method for plot generation that leverages these traits and character models.

Chapter 7 – Generating Stories based on Past Experiences

As I've discussed, the ability to create new stories is an important part of being able to understand stories. Therefore, I developed a novel plot generation module within the Genesis story understanding framework. This plot generation module ties together my work in character modeling, trait learning, and story understanding in addition to solving challenges specific to the domain of creative generation. One of the inspirations for this approach is trying to imagine what would happen if you were to put characters from across a collection of stories into one plot. This is a tactic for story creation that has been used by many authors; for example, characters from multiple stories are put into a single story in comic books such as *The Avengers* and video games such as *Kingdom Hearts*.

My method for doing automatic plot generation differs from the previous methods for doing story generation outlined in Chapter 3 in two main ways. First, my work is grounded in a story understanding system, which gives supplies capabilities and adds interesting depth. It also leads to some additional integration challenges and rewards, as I discuss in this chapter. Such a combination of combining story understanding and story generation has not been a goal of previous systems. Second, my approach emphasizes learning about the world from previous stories. Essentially everything my system knows about character, traits, and stories will come from what it learns from training data, particularly the character modeling and simulation. This aspect is similar to other story generation systems previously discussed such as Mexica, Virtual Storyteller, and Minstrel because they all do some level of world modeling and simulation (Pérez y Pérez & Sharples, 2001; Theune et al., 2003; Turner, 1993). My novel take on the problem is the focus specifically on using learned data to fuel the character simulation.

The plot generation process proceeds through a number of specific phases which I cover in this chapter. They include learning from the training corpus, setting the stage for generation, simulating the characters, and weaving the plot together. Finally, I discuss the results of this work using example generations.

Learn from the Library

An initial step, before plot generation can begin, is that the system must read in a corpus of stories in order to learn about characters. Using this process, Genesis learns about characters and their traits as well as the domain of possible actions and interactions that can occur. The library corpus is always stored in English and read in as needed to do story generation. This means that the plot generator can automatically leverage advancements made to Genesis's story understanding capabilities, character modeling, and trait learning. Also the system can take advantage of any expansion to the Genesis story corpus.

Set the Stage

To begin an instance of plot generation, first the stage must be set. This means setting up a number of initial characters that will play a role in the story. Importantly, characters to be simulated are marked with similar characters and possessed traits. Optionally, the stage can

include information about the setting, relational information about the characters, or any other information that Genesis is capable of handling.

An example a very simple stage setting is shown in Figure 31. In this instance, the stage is set to generate a simple rendition of *The Lion King* based on some of the characters from *Hamlet,* which is the inspiration for Disney's film (Minkoff & Allers, 1994).

```
Insert file Hamlet.

Clear story memory.

Start Experiment.

Start story titled, "The Lion King".

Mufasa is a character.
Mufasa is similar to King Hamlet.

Scar is a character.
Scar is similar to Claudius.

Simba is a character.
Simba is similar to Hamlet.
Simba is heroic.

Simulate characters.
Weave character plots.

The end.
```

Figure 31 - A simple staging for generating Lion King from Hamlet
Each plot generation in Genesis starts with a staging of the characters. Currently, the easiest way to set the stage is with a text file consisting of English text such as the one shown in this figure. The main characters of the story are introduced with the English statement of "XX is a character." Once a character has been introduced the character can be assigned traits and other characterization with sentences such as "XX is similar to YY" and "XX is TT". When doing a character similarity, the system draws in trait and characterization knowledge about the character into the staged character's model. Similarly, any explicit traits are also loaded into the staged character model. The "Simulate characters" and "Weave character plots" statements signal to Genesis to activate various modules.

While such stage files are typically created by a user, the structure and syntax is simple so that automatically generated stage files can also be generated either by GUIs within Genesis or in a random fashion with names being derived from a list or generator and the staging being done through random selection of available traits and characters from the library.

Simulate the Characters

Once the stage for generation has been set, the next step is simulating the actions of the characters. Characters' actions are simulated in a semi-independent manner. That is, each character is considered individually to determine what actions will be taken. The simulator leverages a number of sources of information including previously seen characters, learned traits,

semantic knowledge from WordNet, and Genesis commonsense knowledge (Fay, 2012a; Miller, 1995). This process is computationally expensive, but has been optimized to run quickly.

The algorithm's goal is to generate plot elements for each character until his/her/its individual plot thread comes to an end. The generation proceeds as follows. Each character is selected in turn to be the focus character. The algorithm will first analyze the focus character compare it to all known characters that Genesis believes are similar to the focus character. This determination can be made on the basis of similarity in character traits, similarity in plot elements, and explicit statements of similarity in the stage file. Once a collection of similar characters is available, the algorithm uses sequence alignment to align the plot elements of the focus character against each of the collected similar characters. Recall that this sort of alignment is able to draw in a number of data sources within Genesis including commonsense knowledge and definitional information from WordNet. A sample of the sort of result of these alignments produce is shown in Figure 32.

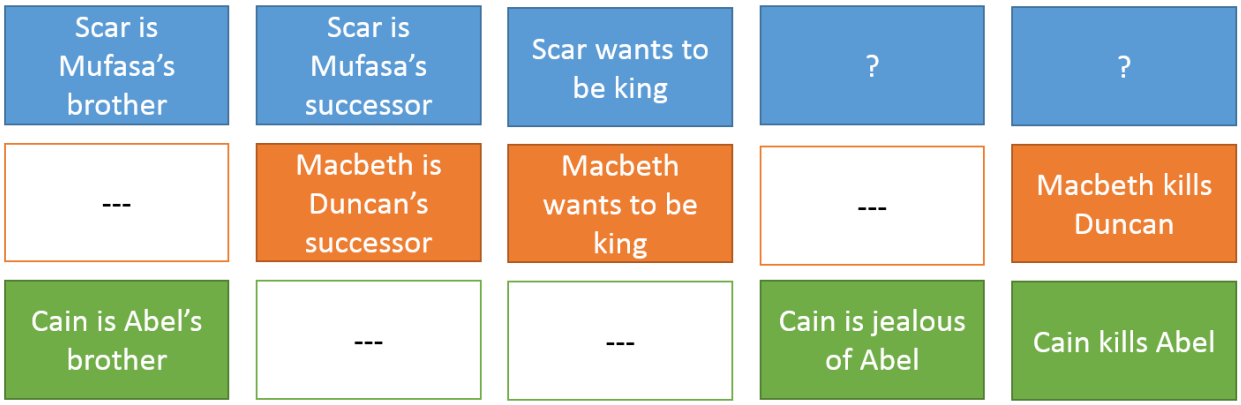


Figure 32 - Aligning characters to find potential plot elements
In order to generate a new plot, Genesis must simulate each character's plot progression. The first step of the algorithm is to determine which plot elements a character might enact. This is done by aligning a character's plot to other characters. The alignments enable Genesis to consider the different directions the character's plot can go. Alignment also produces scores, which can be used to weight potential options. This figure shows the example of generating a plot for the character of Scar. In this case, two characters with similar characteristics to Scar include Macbeth from the tragedy *Macbeth* and Cain from the story of Cain and Abel. In this case, potential plot elements include becoming jealous and killing, but the killing plot element will be more likely because it occurs in close proximity from multiple sources.

Next, for each alignment, the algorithm looks for plot elements that occur in aligned characters' plots but not the plot of the character being simulated and which occur after the last element in the simulated character's plot. Each such element is stored in list of potential elements along with the alignment score from aligning the plots. It is worth noting that the elements making up this potential candidates list are all in generalized form so that corresponding events will match regardless of their source. After the list is created, elements that have multiple entries are combined by adding together their associated scores. For example, from the simulation in progress shown in Figure 32, Scar's current plot has been aligned to two

similar characters, Macbeth and Cain. From these alignments, the potential candidate list is created by collecting the set of events that occur next, in this case "Scar is jealous of Mufasa" from the Cain alignment and "Scar kills Mufasa" from the Cain alignment and the Macbeth alignment.

Next, for each potential plot element, the system will iterate through the traits associated with the character. If the potential plot element matches any elements that make up the trait, the score for that potential plot element is adjusted by the association of that element with the trait by Formula 4 shown below. This results in all the potential plot elements being adjusted in terms of how well the event in question fits the personality traits of the character.

4. $Score = Score * (1 + Association)$

Once all of the potential plot elements have been processed for each trait, the algorithm must choose which action the character takes. Typically, the plot element that has the highest score will be chosen, which will result in a deterministic simulation of the character. However, the potential events can also be chosen from probabilistically, with the elements being given weighted probabilities correlated to their relative scores.

For example, assume that Scar has a trait *Vicious Ambition* that has a score of 0.9 attached to an event of "Focus kills GEN_0" and a score of 0.4 attached to an event of "Focus is jealous of GEN_0". Also assume that the score for the potential event "Scar is jealous of Mufasa" is 0.8 from the alignment and the score for "Scar kills Mufasa" is 0.7 from the alignment. The algorithm will update these scores from the alignment with equation 4. This will give an updated score of 1.12 for "Scar is jealous of Mufasa" and a score of 1.33 for "Scar kills Mufasa"

This process of character simulation proceeds iteratively, repeatedly choosing new plot elements in succession until the plot thread for the character is compete. The end of the plot thread is determined by a *null* story event being chosen as the next element, which occurs when there are no reasonable actions for the character to take. Once the full plot threads for all characters in the new story have been simulated, the system proceeds to the plot weaving phase.

Plot Weaving

Plot weaving is my method for tying the plot threads of all of the characters together to create the new story. Plot weaving is important because it makes sure that characters' plots are compatible and because it creates a consistent timeline for all of the plot elements of the story. However, plot weaving is computationally difficult because it requires selecting the best set of pairings of characters to generic entities to create the best possible combination for the story. Figure 33 shows a collections of plots before plot weaving and Figure 36 shows the same characters after plot weaving as a visual example.

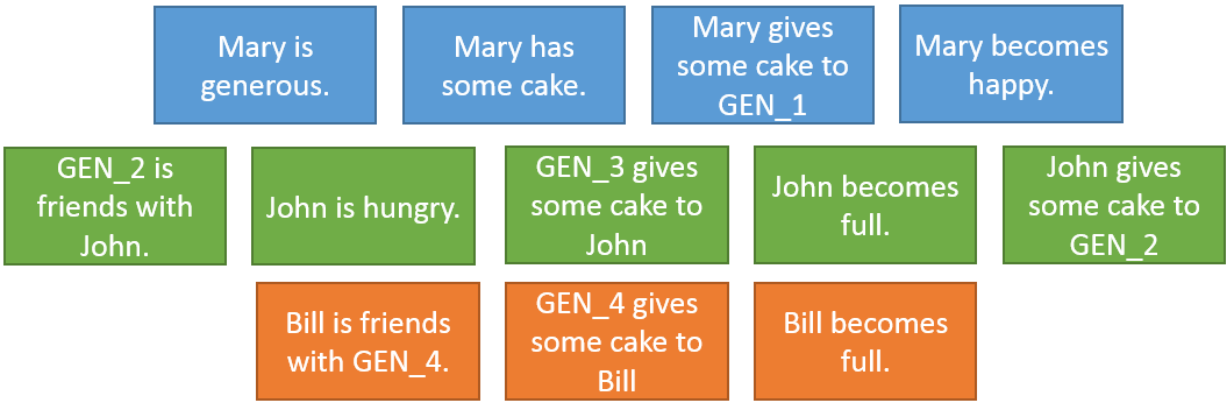


Figure 33 - Characters' plots before plot weaving

The goal of the plot weaving methods within Genesis is to take all the plots of the characters in a generated story and weave them together into a consistent, coherent plot. In order to accomplish this goal, the plot weaver needs to determine the best way to pair the generic entities – GEN_X in the figure – with actual characters. This is very difficult problem computationally and is non-trivial even for simple examples like the one shown. In this example, GEN_1 could map equally well to John or Bill. The plot weaving algorithm uses a combination of efficient search and heuristics to determine the set of bindings which produces the best possible weave that has the fewest unmatched events.

Plot weaving requires selecting the best set of pairings of characters to generic entities. While there exist only $O(c * g)$ (the number of characters times the number of generics) possible pairs, each pair comes with its own unique constraints on all other pairs. Thus, the total number of possible sets of pairings is exponentially high. Of a set of *n* characters, each character will have on average *(n-1)* generics and each of these generics can be paired with *(n-1)* possible characters. This gives the search space for all possible pairs a size of $O(n^n)$. Therefore, the plot weaving algorithm must be able to quickly and reliably find the best set of pairings without searching through the entire space of pairings, which would be computationally intractable.

In order to do this search efficiently, my plot weaving algorithm uses a method of simultaneously choosing pairings and partially weaving the plots. The goal of the algorithm is to construct a search tree over the set of possible pairings. The search tree has two important properties. First, all the leaf nodes of the tree represent unique sets of pairings. Second, nodes will be scored in such a way that the score of a child node will always be less than or equal to its parent node. This approach uses similar principles to the concept of branch and bound algorithms (Land & Doig, 1960).

Plot Weaving Algorithm and Implementation

Set up

The input for the algorithm is a collection of characters, each with its own collection of plot elements to be woven into the story. The output of the algorithm is a chronological ordering of plot elements making up the complete story involving the set of characters. The first step is to collect all of the generic entities from the characters' plots. The optimal set of binding pairs between these generic entities and the set of characters must be found in order to create the best weave. In this case the best weave is the weave with the fewest unwoven plot elements that

maintains consistency between all of the individual character plots. Pseudocode versions of these components of the plot weaving algorithm are shown in Appendix B.

To construct the set of generic entities, the algorithm iterates through the list of characters. For each character, the algorithm iterates through the character's plot elements. Whenever a generic entity is encountered, the entity is added to the set of all generic entities. The entity is also marked as originating from the current character as this will be important later during the pairing process.

The plot weaving process requires the creation of a search tree with nodes that hold information vital to the search for the best set of binding pairs. Each node contains a set of unmatched generic entities, a set of binding pairs between generic entities and characters, a map of each character to a list of available characters for that character's generic entities, and a score value. Initially a root node is created. This node is initialized as follows. The set of unmatched generic entities contains all of the generic entities from across all the characters. The set of binding pairs, is empty. For each character, *C*, the map of available target characters is set initially to contain the set of all characters except *C*. Finally, the score is set to infinity. The root node is added to a priority queue that keeps added nodes in order of their scores with the property of highest score in, first out.

Search for the Bindings

The search begins by obtaining and removing the highest scoring the node from the queue. This node becomes the current node of the search. Next, an unmatched generic entity, current generic, is chosen arbitrarily from the list of unmatched generic entities. Then, list of target characters for pairing with the current generic is obtained by looking up that entity's originating character in the current node's mapping of available target characters.

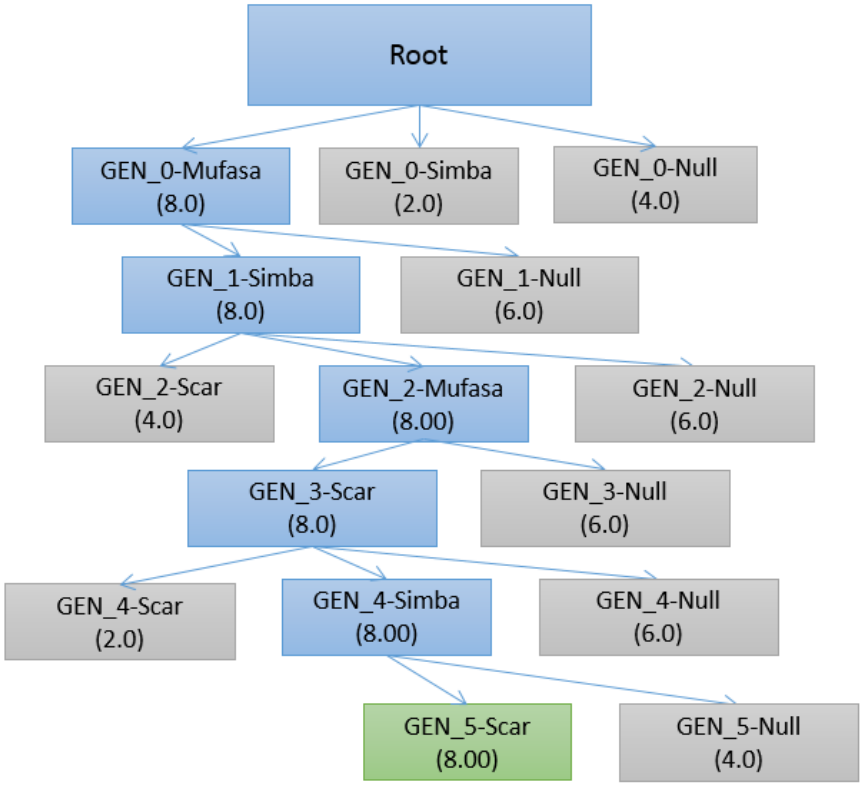


Figure 34 - Plot Weaving Binding Search
In order to find the best set of binding pairs to produce the best weave, the plot weaving algorithm searches efficiently through the space of all sets of binding pairs. The algorithm proceeds by iteratively adding new pairs to the list of bindings and testing how the additional constraint affects the score of the weave. The search proceeds in the direction that produces the highest weave scores. In this figure, each node is labeled with the pair that has been added in that node, with all nodes' binding lists having the set of nodes of all ancestor nodes. Nodes are also labeled with their score from weaving. The characters and entities in this figure are from the simplified version of *The Lion King* shown in Figure 35.

For each target character, a new child node of the current node is created. The child node is initially created as a copy of the current node. A new binding pair is created between the current generic and the target character. The pair is added to the list of binding pairs on this child node. The current generic is then removed from the child node's list of unmatched entities and the target character is removed from the current generic's originating character's list of available target characters. Next, the new child node is scored with the weaving scoring method, which is discussed in the next section. Finally, the child node is added to the priority queue.

After all the child nodes that pair the current entity to target characters have been created, one final child node is created. This child node pairs the current entity to a Null character marker, but otherwise proceeds in the same fashion as described for the other child nodes.

This search for the optimal set of bindings for weaving proceeds by obtaining the best node from the queue, creating child nodes, scoring them, adding them to the queue and then repeating this process until a leaf node of the search tree is found. A leaf node will be a node that

has an empty set of unmatched generic entities. Such a leaf node is guaranteed to have the best set of bindings for weaving the characters' plots as its set of pairings.

Scoring a Weave

In order to guarantee an efficient search, each node must be scored quickly and accurately. The score method essentially checks how well the characters' plots can be woven given a current set of binding pairs between generic entities and characters. Specifically, the method needs to guarantee that adding binding pairs, which act as constraints, can only ever maintain or decrease the score of the node. This enables the search to know that each node's score is an upper bound on the scores of all child nodes, which ensures that the first leaf node discovered has the best set of bindings for weaving. An example of two weave scores are shown in Figure 35.

Figure 35 - Plot Weaving Score Example

In order to produce the best final weaving of the story, the plot weaving algorithm repeatedly scores how good the current weaving of the story is. The score is calculated by counting the number of plot elements that can be aligned between all the characters based on the current set of constraints. Two examples are shown in this figure. The top example shows a non-optimal binding pair between GEN_3 and Mufasa. This is non-optimal because while it allows the event Simba kills GEN_3 to match, it prevents other matches between Scar and Simba. A better pairing is shown in the bottom example that is binding GEN_3 to Scar.

The method proceeds by iterating through the set of characters and calculating partial weave scores by aligning each character to each other character and seeing how this performs given the current set of constraints. These partial scores are then summed in order to obtain the final score for the weave.

The algorithm begins with an initial total weave score value of zero. Then it iterates through the set of all unique pairings between characters. For each pair of characters, the characters' plots will be aligned using the alignment algorithm previously discussed. The alignment will be fed the current set of constraints, which will affect how well the plot elements from the characters can align. Recall that for each generic entity the algorithm retains information about which character that generic entity originated from—that is, its originating character. A generic entity, *gen1,* without a binding pair is allowed to match any and all characters that are not already paired to a different generic entity, *gen2*, that shares the same originating character as the generic entity, *gen1*. A generic entity that has a binding pair is allowed to match only against its own paired character. Finally, generic entities bound to the Null character marker will not be allowed to match to any characters.

From this constrained alignment, the partial weave score will be the number of successfully aligned plot elements. Each partial weave score from these pairwise alignments will be added to the total weave score. Once all the pairs of characters have been scored, the final total score represents the weaving score for that particular set of constraints.

The Final Plot Weaving

Once the search for optimal bindings has finished, the algorithm can proceed to construct the final woven story of all the characters' plots. This process takes the collection of characters and the set of bindings between generics and characters in order to produce the final chronological ordering of the events of the story.

First, the algorithm maps each character to a plot position iterator that indicates how much of that character's plot has been added to the final story. These iterators are all initially set to zero. Additionally a new empty list of plot elements is created to store the woven story. Then the process proceeds as follows. The algorithm goes through each character's plot and replaces all incidents of generic characters with their respective characters from the bindings list. Plot elements containing the Null character marker are removed.

Next the algorithm produces the output story. The process repeatedly iterates through all the characters until the plot position iterator for each character is at the end of all characters' plots. For each character, the algorithm checks what plot element is next using the plot position iterator for that character. If the character is at the end of its plot, the character is skipped. If the next plot element contains only the current character, then the plot element is added to the story and the character's plot position is incremented. This process repeats for that character until either the end of the character's plot is reached or an event containing more than one character is reached. If an event containing more than one character is encountered, the algorithm marks the current character as blocked and proceeds to the next character.

Whenever a blocked character is encountered, including the act of marking it as blocked, the algorithm checks to see if the character can be unblocked. To do so it inspects the plot element blocking the character to obtain a list of all characters that are a part of that plot element. Then, each of these other characters are checked. If all these characters are blocked on this same plot element then this element is added to the story, all of these characters' plot positions are incremented, and these characters are marked as unblocked.

When this process completes, the algorithm will have a complete chronological account of the woven story of the characters' plots.

Sample of Weaving

The simple example shown in Figure 33 and Figure 36 illustrates why this process is necessary to find best set of all bindings. GEN_1 in Mary's plot line interacts with Mary in the event "Mary gives some cake to GEN_1" therefore we would like GEN_1 to match with a character that has an event of the form "GEN_X gives some cake to Focus". As we can see both John and Bill would be able to satisfy this constraint. From the perspective of weaving Mary's plot efficiently, both are good fits. However, assigning GEN_1 to Bill will cause problems later in the weaving process because this binding assignment will later cause the algorithm to be unable to find a match for GEN_2 from John's plot line.

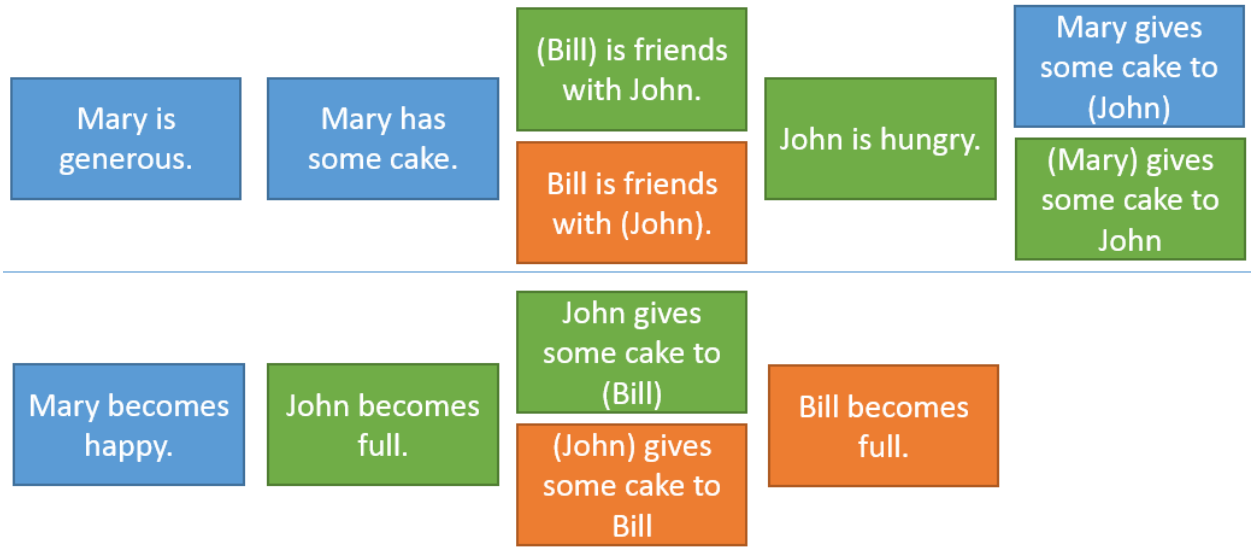


Figure 36 - Characters' plots after plot weaving

The goal of the plot weaving methods in Genesis is to take the plots of all characters in a generated story and weave them together into a consistent plot. In order to accomplish this goal, Genesis performs the computationally difficult task of finding the best way to match the generic entities with characters. After the process of selecting the best binding pairs is complete, the plots of the characters can be woven together into a consistent story as shown here. The pre-woven plots of these characters can be seen in Figure 33.

Fill the Gaps

After the plot weaving process is complete, the system knows the best way to weave together the characters' plot threads. However, an additional step, gap filling, can occur at this

stage in order to allow additional character interactions and improve the cohesiveness of the story as a whole. Gap filling is done by comparing two characters stories in the final weave and determining if any events from one character's plot thread should be inserted into another character's thread.

Sometimes after gap filling, the plot weaver will end up with situations like the one shown in Figure 37. In this example, John is the best match for GEN_1 from Mary's plot and Mary is the best match for GEN_2 from John's plot line. However, there are events in both Mary's and John's plot lines that have no corresponding event in the alignment between their plots. When this happens the gap filler can add events missing slots in the plot lines to make the alignment consistent. So in this case the event "Mary dates John" will be added to John's plot line and "John loves Mary" will be added to Mary's plot line.

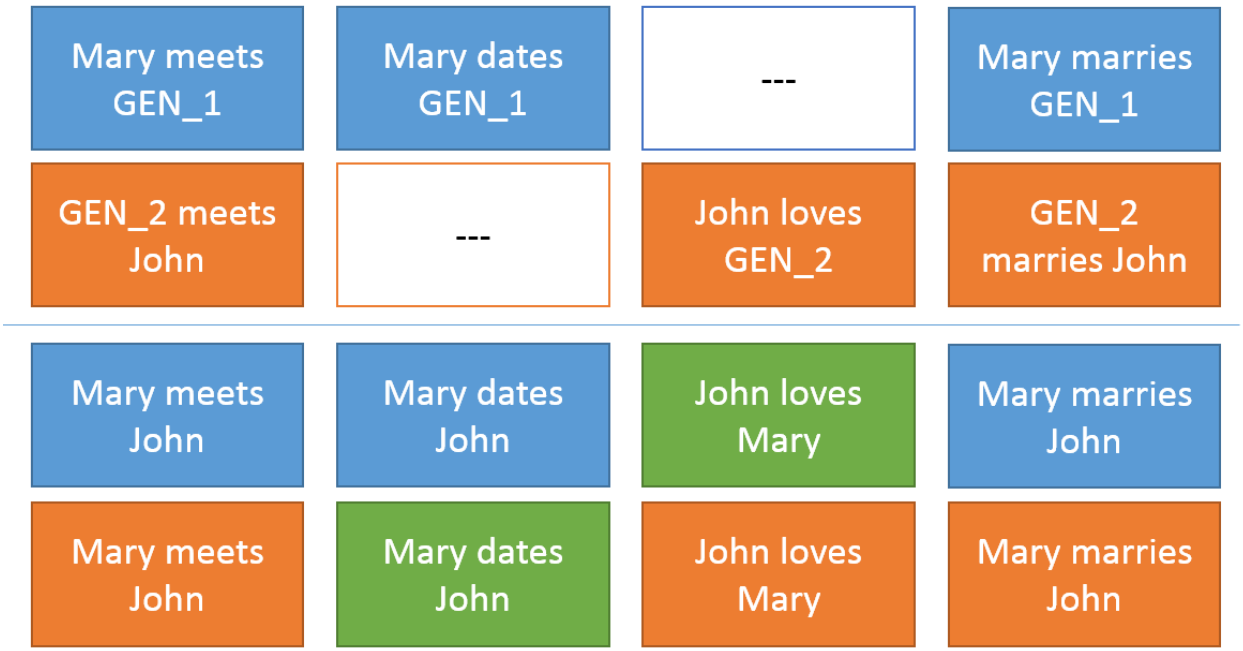


Figure 37 - Gap filling a story after weaving
Genesis is capable of weaving together the plots of characters to form a consistent story. However, sometimes it turns out to be impossible to do a perfect weave because matches cannot be found for all the plot elements that occur between characters. In these cases, Genesis is capable of filling the gap in order to attempt to either add in new elements or remove conflicting elements. As shown in this figure, the gap filler is able to take the plots of Mary and John and fill in the gaps in their plots to make them consistent.

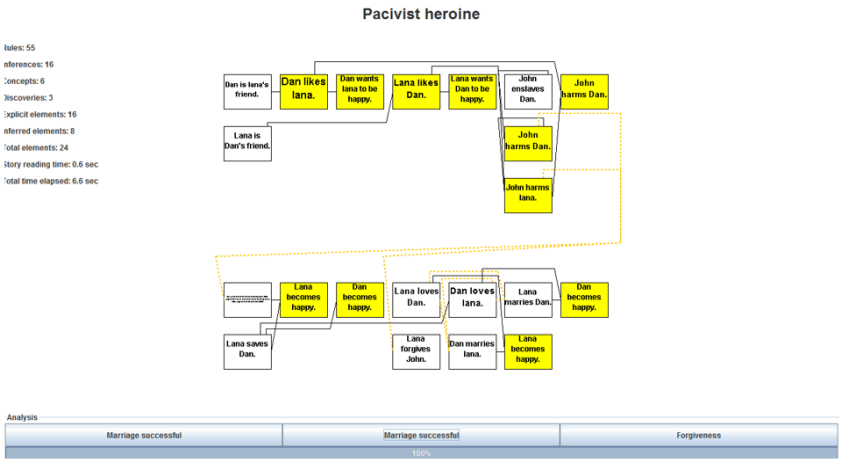
The gap filler has to decide to either remove the event from the original plot thread or to add the event to the aligned plot thread. Typically, the event can be added to the aligned thread without issue. However, the gap filler does validation against the characters in question in order to make sure that the new event would not break any constraints set by either the binding pairs or the character traits of the characters. If a constraint would be broken then the element is instead removed from the original plot thread.

Results

One of the powerful advantages of my approach is that it is able to generate a wide variety of interesting plots even when using only fairly small libraries. In this section I detail some of the interesting scenarios that my plot generation methods are capable of handling through a variety of examples.

Conflicting Character Traits

In many stories, characters with internal conflict are the most interesting because this tends to result in situations in which characters are faced with a difficult choice. Within Genesis and my character modeling extension, such characters can be modeled as having a set of character traits that are in conflict with each other. One example would be someone who is both greedy and generous. The plot generation methods I've discussed in this chapter are capable of handling such characters well. As an example, consider the story shown in Figure 38.




```
Dan is Lana's friend.
Lana becomes queen.
Pete enslaves Dan.
Lana wants to help Dan.
Lana saves Dan.
Lana wants to kill Pete.
Lana forgives Pete.
Dan loves Lana.
Dan marries Lana.
Lana marries Dan.
Lana dies of old age.
```

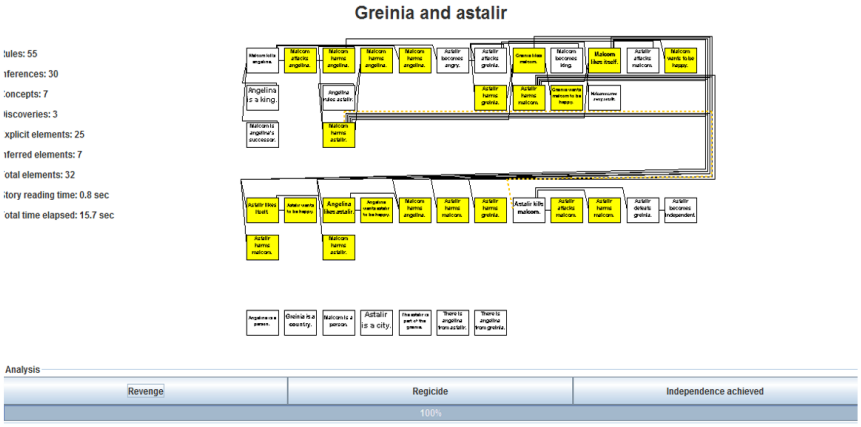
Figure 38 – Generated story with conflicting character traits
This figure shows a sample story generated by Genesis that demonstrates how the system handles characters with conflicting character traits. In this generation, the character Lana was modeled after both violent and peaceful characters. During the generation of this story Lana's friend Dan is harmed by Pete, which makes Lana upset with Pete. Part of her wants to kill him, but in the end her more peaceful side triumphs and she instead forgives him. This figure shows both the elaboration graph Genesis creates and a summary of the explicit plot elements for this story.

In this story the critical character is Lana. In the initial staging of this generation, Lana was modeled to be both violent and forgiving. In the generated story, this conflict plays out as Lana wants to help her friend Dan who has been enslaved by Pete. Part of Lana wants to get revenge and kill Pete, but in the end her forgiveness overcomes her aggressiveness and she ends up forgiving him.

Cross-Genre Characterization

In order to show the robustness of my work, I have been testing my methods on stories from multiple genres that have varying types of characters, traits, and plots, such as conflict stories and Shakespearean tragedies. However, in addition being able to handle analysis and

generation of stories within a genre, my work is also capable of merging characterizations from across genres and still generate interesting plots. This capability is important for two reasons. First, it demonstrates that Genesis and my work appropriately handle a wide array possible stories. Second, it shows that the system as a whole can combine information from very different sources based on their commonalities much in the way that humans are able to do. As an example, consider the story shown in Figure 39.




```
Greinia is a country.
Astalir is a city in Greinia.
Angelina is from Astalir.
Angelina is from Greinia.
Angelina becomes the Queen of Greinia.
Angelina rules Greinia.
Malcom is Angelina's successor.
Malcom kills Angelina.
Malcom becomes King of Greinia.
Astalir rebels.
Astalir attacks Greinia.
Malcom scares away Astalir's forces.
Astalir attacks Malcom.
Astalir kills Malcom.
Astalir defeats Greinia.
Astalir becomes independent.
```

Figure 39 - Generated story with cross-genre characterization
This figure shows a sample story generated by Genesis that demonstrates how the system handles characters and stories with traits and characterizations spanning genres. In this generation, all of the characters and countries involved have been modeled as having a mix of traits and characterizations from multiple genres, including war stories and fairy tales. Specifically, the characters in this story have been modeled of countries from World War I, characters from Beowulf, and characters from Macbeth. The overall story successfully combines elements from across genres, creating an interesting story of revenge and revolution. This figure shows both the elaboration graph created by Genesis and a plot summary of the explicit plot elements.

In this story, all of the characters, both human and country alike, have been modeled off of a combination of Shakespearean characters and countries and factions from wars and conflicts. Specifically, the characters in this story have been modeled of countries from World War I, countries from the American Revolution, characters from Beowulf, and characters from Macbeth. The generator is able to successfully combine these stories to produce this tale of Astalir's rise to independence.

Conflicting Goals between Characters

An important aspect of many stories is how characters seek to fulfill their goals. However, there are many situations in which multiple characters' goals are in conflict with one another such that not all can be successful simultaneously. My system can handle and resolving this sort of conflict, as demonstrated by the story in Figure 40.

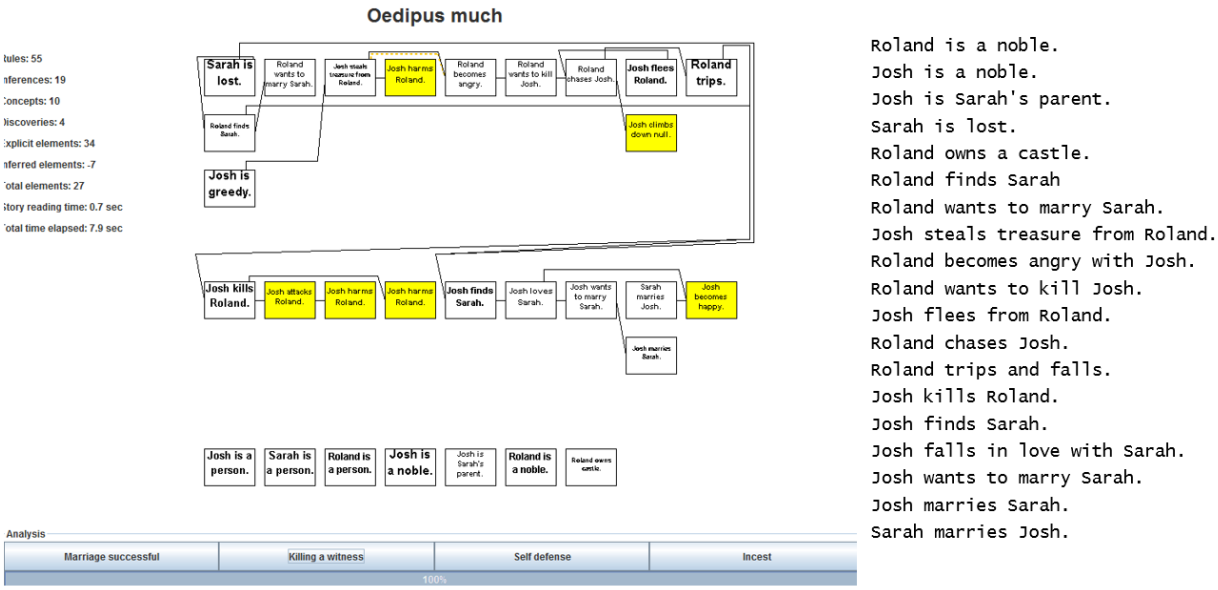


Figure 40 - Generated story with conflicting goals between characters
This figure shows a sample story generated by Genesis that demonstrates how the system handles characters that have goals that are in conflict with each other. In this generation, the characters of Roland and Josh end up having goals that cannot be simultaneously achieved. Specifically, the character of Roland wants to marry Sarah and later, after being robbed by Josh, wants to murder the thief. However, because Josh succeeds in killing Roland, Roland is unable to complete either of these goals. This figure shows both the elaboration graph created by Genesis as well as a summary of the explicit plot elements for this story.

In this story the characters of Josh and Roland end up having goals that are in conflict with one another. Roland decides he wants to marry Sarah after discovering her. However after Josh steals Roland's treasure, he has a secondary goal of wanting to kill Josh. Neither of these goals are achieved because Josh kills Roland before Roland can kill him or marry Sarah.

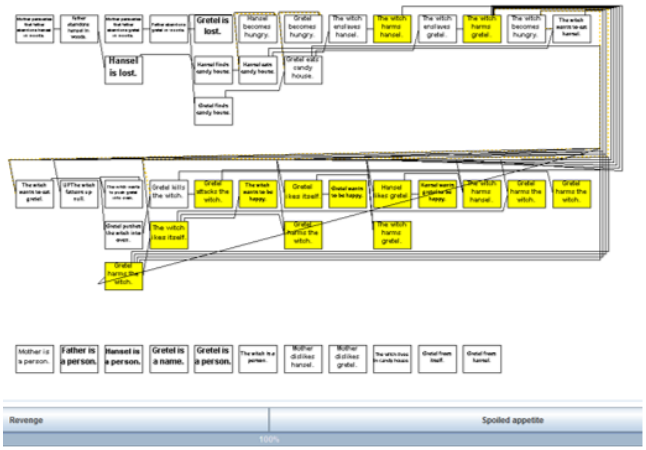
This story also demonstrates some of the unintended points of interest that can occur within plot generation. Specifically, in the ending of this story Josh finds his lost daughter but the two of them end up falling in love and getting married in an off-putting act of incest. While this sounds like an odd reversal on the Oedipus story, no such story exists in the Genesis corpus and the marriage is actually an unexpected use of knowledge gathered from the romances in the Shakespearean comedies. While this sort of plot twist is interesting, key aspects such as the unexpected marriage are not explicitly known by the system. In future work it would be useful to model these sorts of plot twists and other surprising elements more explicitly in order to create ways for the system to be aware them. One potential method would be to integrate Genesis's reflective knowledge system into the generation process.

Taking Story Understanding to the Next Level

As I've mentioned previously in Chapter 2, the human ability to imagine is strongly tied to concepts such as story generation. Therefore, if we are to improve the state of the art in story understanding in order to better model human intelligence, the system needs to have an aspect of

story generation. In this section I detail a compelling example of how story generation can be used to fuel better story understanding.

First consider the rendition of Hansel and Gretel shown in Figure 41 which is a typical telling of the tale.



Hansel & Gretel

Mother dislikes Hansel and Gretel.
Mother wants to kill Hansel and Gretel.
Mother convinces Father to abandon Hansel and Gretel in the woods.
Father abandons Hansel and Gretel in the woods.
Hansel and Gretel are lost.
Hansel and Gretel become hungry.
Hansel and Gretel find the house made of candy.
Hansel and Gretel begin eating the house made of candy.
The witch lives in the house made of candy.
The witch enslaves Hansel and Gretel.
The witch becomes hungry.
The witch wants to eat Hansel and Gretel.
The witch fattens up Hansel.
The witch tries to push Gretel into the oven.
Gretel frees herself.
Gretel pushes the witch into the oven.
Gretel kills the witch.
Gretel frees Hansel.

Figure 41 - Hansel & Gretel in Genesis
Hansel & Gretel is a classic fairy tale in which two siblings are abandoned in the woods and captured by a witch while eating her house made of candy. The witch is cannibalistic and decides to fatten up the children to eventually eat them. However, when the witch decides to push Gretel into the oven, Gretel manages to push the witch into the oven instead. Thus, Gretel is able to save herself and her brother. The plot summary and elaboration graph from Genesis is shown in this figure.

In this story, Hansel and Gretel are abandoned in the woods by their Father. Then after trying to eat a witch's house they're imprisoned by the witch. The witch decides to fatten them up so she can eat them. However, when the witch decides it's time to eat them, Gretel manages to push the witch into the oven instead and then frees herself and Hansel. We know Gretel is the heroine in this story because she defeated the witch and saved herself and her brother.

We can also imagine alternative outcomes in which Gretel is not so successful, and doing so can help a reader understand how important her success is to the outcome of the story. In a system like Genesis, such imagination could be handled by story generation. In order to demonstrate the effectiveness of this approach, I had Genesis generate a version of Hansel and Gretel without Gretel. This a comparison of the generated story to the original is shown in Figure 42.

Hansel & Gretel

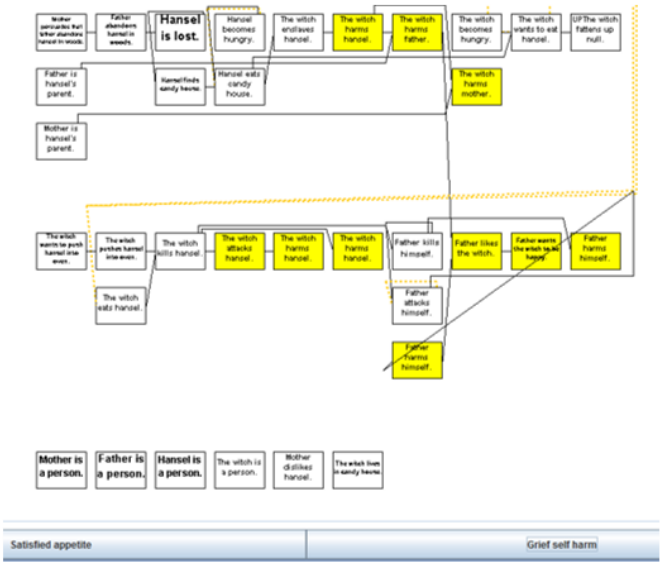
Mother dislikes Hansel and Gretel.
Mother wants to kill Hansel and Gretel.
Mother convinces Father to abandon Hansel and Gretel in the woods.
Father abandons Hansel and Gretel in the woods.
Hansel and Gretel are lost.
Hansel and Gretel become hungry.
Hansel and Gretel find the house made of candy.
Hansel and Gretel begin eating the house made of candy.
The witch lives in the house made of candy.
The witch enslaves Hansel and Gretel.
The witch becomes hungry.
The witch wants to eat Hansel and Gretel.
The witch fattens up Hansel.
The witch tries to push Gretel into the oven.
Gretel frees herself.
Gretel pushes the witch into the oven.
Gretel kills the witch.
Gretel frees Hansel.

Hansel w/o Gretel

Mother dislikes Hansel.
Mother wants to kill Hansel.
Mother convinces Father to abandon Hansel in the woods.
Father abandons Hansel in the woods.
Hansel and is lost.
Hansel becomes hungry.
Hansel finds the house made of candy.
Hansel begins eating the house made of candy.
The witch lives in the house made of candy.
The witch enslaves Hansel.
The witch becomes hungry.
The witch wants to eat Hansel.
The witch fattens up Hansel.
The witch pushes Hansel into the oven.
The witch kills Hansel.
The witch eats Hansel.
Father discovers Hansel died.
Father kills himself.

Figure 42 - Hansel with and without Gretel
This figure demonstrates the power of generation to fuel story understanding and artificial intelligence. In this figure, two variations of the classic Hansel & Gretel tale are shown. On the left is a typical telling from the Genesis library in which Gretel saves the day by killing the witch before the witch eats Gretel and her brother. However, to understand the story more deeply, Genesis can now leverage plot generation in order to imagine a version of the story in which Gretel does not exist. In this generated story, shown on the left, Hansel is killed and eaten by the witch, which later causes Hansel's father to kill himself in grief. This recreation of the story proves how important Gretel is as a character. Thus, this sort of generation demonstrates how plot generation can play a powerful role in story understanding.

This generation to fuel understanding is now possible in Genesis. The system reads collection of tales, and then uses its library of knowledge in the generation of alterations to the original stories. In this darker version of the tale, Hansel is unable to escape the witch and ends up being devoured. Later, after Hansel's father discovers that Hansel has been killed, he kills himself in grief. A Genesis elaboration graph representation of the story is shown in Figure 43.



Hansel w/o Gretel

Mother dislikes Hansel.
Mother wants to kill Hansel.
Mother convinces Father to abandon Hansel in the woods.
Father abandons Hansel in the woods.
Hansel and is lost.
Hansel becomes hungry.
Hansel finds the house made of candy.
Hansel begins eating the house made of candy.
The witch lives in the house made of candy.
The witch enslaves Hansel.
The witch becomes hungry.
The witch wants to eat Hansel.
The witch fattens up Hansel.
The witch pushes Hansel into the oven.
The witch kills Hansel.
The witch eats Hansel.
Father discovers Hansel died.
Father kills himself.

Figure 43 - Hansel without Gretel in Genesis
Genesis can use its plot generation methods to generate variations of existing stories by generating a story based on modifications of the original characters. This figure shows generated a version of Hansel & Gretel in which Gretel has been removed from the story. Genesis uses what it's learned from other stories in order to imagine the new plot. In this version, without Gretel to save him, Hansel is killed by the witch and in the end his father takes his own life in grief. This figure shows both the elaboration graph and plot summary from Genesis.

This sort of generation is important because it allows Genesis to ask powerful "What if…" style questions and then generate a hypothetical story to answer them. Such stories could be generated both by adding and removing characters and also by adjusting the characterization of characters within the story. Using this new technique, Genesis can now understand stories at a much deeper level. Alterations to stories can be made, the plot regenerated, and then the new story compared to the old. The comparison between the original and generated tales will allow Genesis start to truly understand why stories unfold the way they do.

Summary

In this chapter I have discussed how my work allows Genesis to leverage what it learns from reading collections of stories in order to generate novel plots. Most importantly, I described the details of the Plot Weaving algorithm which is capable of combining multiple plot lines into a single coherent story. I've also demonstrated some of the interesting capabilities of the system and identified how story generation can be used as a component in a larger system to improve story analysis and understanding.

Chapter 8 – Contributions

The high level vision for my thesis research is to work towards artificial intelligence by tackling problems in the domain of computational story understanding. My work highlights the importance of characters, traits, and plot generation as components of a greater story understanding system. An outline of the main contributions of my work follows.

I identified the importance of character modeling and plot generation to story understanding. Specifically, I reviewed the literature in both human intelligence and computational story generation. I discussed how symbolic reasoning, language, and story understanding play important roles in human intelligence. I also explored how creativity in the form of story generation enables humans to engage in challenging problem solving tasks. Throughout the thesis I have made the case for the inclusion of story understanding and generation as important goals for creating a robust artificial intelligence.

I demonstrated the utility of good character models for doing story analysis and generation. The character models that I described have been successfully implemented into the Genesis Story Understanding System. These models enable improved story analysis by allowing cross story comparisons to be done on the level of characters in addition to the levels already available. The models are robust allowing Genesis to consider characters from across a variety of genres. In order to demonstrate their utility I added a new corpus spanning 3 genres and over 15 works. These character models also enable the plot simulation techniques I developed which drive the plot generation processes.

I developed novel methods for learning character traits from a corpus of stories. The ability to learn character traits from a corpus of stories mirrors the important human ability to learn from past experience. Traits allow Genesis another dimension of information on which it can organize its knowledge. In addition, traits broaden the system's story understanding capabilities by allowing it classify characters by their actions and group actions by co-occurrence. These traits help to fuel the character simulation process, acting as the backend for the story generation system.

I developed the plot weaving algorithm for generating stories. The plot weaving algorithm I developed enables my story generation system to combine the plot lines of multiple characters into a single, consistent story. The algorithm is capable of efficiently searching over a combinatorial search space to find the best possible weave. This technique leverages the capabilities of *Genesis* allowing it to use information from multiple sources to guide the process. The speed and effectiveness of this algorithm allows complex stories to be generated in a small amount of time.

I created a plot generation system capable of generating interesting stories by learning from an existing library. The story generation methods I've developed have been implemented into Genesis. Requiring only a library of English stories for background knowledge they illustrate how creativity can be successfully drawn from a small corpus without highly structured knowledge. The method is also robust to a number of situations including conflicted characters, cross-genre characterization, and characters with conflicting goals. Importantly, I've

demonstrated how story generation can fuel story understanding. The ability to consider how alterations to a situation affect outcomes is a powerful and valuable ability which takes Genesis's story understanding to the next much higher level.

Future Work

My thesis research has been successful both in its goals but also in its ability to illuminate directions for new and continued research. A low hanging fruit would be the integration of additional story generation techniques with my own approach. In my work I wanted to demonstrate how a focus on learning from past experience makes for a good basis for plot generation. However, I believe such an approach could be augmented by combining it with other parallel work. For example, many previous approaches integrated an authorial component to direct the story. Such a component could theoretically work in tandem by directing characters to take actions which help the story as a whole achieve high level plot goals, teach specific morals, or incorporate motifs. From another angle, an integration with systems like Curveship and Slant with Genesis would allow for generated stories to have narrative guidance. This would allow the stories to be told in a greater variety of ways including shifts in perspective, chronology, and style.

My work in trait learning also shows how story understanding systems like Genesis can begin to take advantage of much larger datasets than ever before. Techniques such as topic modeling have been shown to be very capable of processing very big data in an effective way. By integrating such techniques with a system like Genesis, a much deeper analysis and understanding of story based data can be achieved.

References

Abbott, H. P. (2008). *The Cambridge Introduction to Narrative* (2nd ed.). Cambridge, UK: Cambridge University Press.

Augier, M., & Kreiner, K. (2000). Rationality, imagination and intelligence: some boundaries in human decision-making. *Industrial and Corporate Change*.

Bailey, P. (1999). Searching for storiness: Story-generation from a reader's perspective. *Working Notes of the Narrative Intelligence Symposium*.

Bal, M. (1997). Narratology: Introduction to the theory of narrative. *Trans. Christine van Boheemen. 2nd Ed. Toronto: U of …*.

Bates, J. (1992). *The nature of characters in interactive worlds and the Oz project*.

Bender, J. (2001). Connecting language and vision using a conceptual semantics.

Blei, D., Ng, A., & Jordan, M. (2003). Latent dirichlet allocation. *The Journal of Machine Learning Research*.

Chaney, A., & Blei, D. (2012). Visualizing Topic Models. *ICWSM*.

Chang, J., & Gerrish, S. (2009). Reading tea leaves: How humans interpret topic models. *Advances in …*.

Chomsky, N. (2010). Some simple evo devo theses: How true might they be for language. *The Evolution of Human Language*.

Cooper, H. (2009). Learning meaning in Genesis.

Deerwester, S., Dumais, S., & Landauer, T. (1990). Indexing by latent semantic analysis. *JASIS*.

Dehn, N. (1981). Story Generation After TALE-SPIN. *IJCAI*.

Dodell-Feder, D., Koster-Hale, J., Bedny, M., & Saxe, R. (2011). fMRI item analysis in a theory of mind task. *Neuroimage*.

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

Fay, M. P. (2012a). *Enabling Imagination through Story Alignment*.

Fay, M. P. (2012b). Story Comparison via Simultaneous Matching and Alignment. In *The Third Workshop on Computational Models of Narrative* (pp. 100–104).

Finlayson, M., & Winston, P. H. (2006). Analogical retrieval via intermediate features: The Goldilocks hypothesis.

Forster, E. (1962). Aspects of the Novel. 1927. *Ed. Oliver Stallybrass*.

Gentner, D., & Markman, A. (1997). Structure mapping in analogy and similarity. *American Psychologist*.

Gervás, P. (2009). Computational Approaches to Storytelling and Creativity. *AI Magazine*, *30*(3), 49. doi:10.1609/aimag.v30i3.2250

Girolami, M., & Kabán, A. (2003). On an equivalence between PLSI and LDA. *Proceedings of the 26th Annual International …*.

Golub, G., & Reinsch, C. (1970). Singular value decomposition and least squares solutions. *Numerische Mathematik*.

Hendrikx, M., & Meijer, S. (2013). Procedural content generation for games: A survey. *ACM Transactions on …*.

James, H., & Edel, L. (1956). The future of the novel: essays on the art of fiction.

Johnson, R., Helm, R., Vlissides, J., & Gamma, E. (1995). Design Patterns: Elements of Reusable Object-Oriented Software.

Jonassen, D., & Hernandez-Serrano, J. (2002). Case-based reasoning and instructional design: Using stories to support problem solving. *Educational Technology Research …*.

Katz, B. (1997). Annotating the World Wide Web using Natural Language. *RIAO*.

Katz, B., Borchardt, G., & Felshin, S. (2006). Natural Language Annotations for Question Answering. *FLAIRS Conference*.

Katz, B., Felshin, S., Yuret, D., & Ibrahim, A. (2002). Omnibase: Uniform access to heterogeneous data for question answering. *Natural Language …*.

Klein, S., Aeschlimann, J., & Balsiger, D. (1973). Automatic novel writing: A status report. *Wisconsin University*.

Krakauer, C. (2012). Story retrieval and comparison using concept patterns.

Land, A., & Doig, A. (1960). An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*.

Lebowitz, M. (1984). Creating characters in a story-telling universe. *Poetics*.

Lehnert, W. G. (1981). Plot Units and Narrative Summarization*. *Cognitive Science*.

Li, B., Lee-Urban, S., Appling, D. S., & Riedl, M. O. (2012). Crowdsourcing narrative intelligence. *Advances in Cognitive …*.

Liu, H., & Singh, P. (2002). MAKEBELIEVE: Using Commonsense Knowledge to Generate Stories.

Mateas, M., & Stern, A. (2003). Façade: An experiment in building a fully-realized interactive drama. *Game Developers Conference, Game …*.

Meehan, J. (1977). TALE-SPIN, AN INTERACTIVE PROGRAM THAT WRITES STORIES. *Citeseer*.

Miller, G. (1995). WordNet: a lexical database for English. *Communications of the ACM*.

Minkoff, R., & Allers, R. (1994). *The Lion King: Platinum Edition* (p. Origins (DVD)). Disney.

Montfort, N. (2011). Curveship ' s Automatic Narrative Style. *Proceedings of the 6th International Conference on the Foundations of Digital Games (FDG '11)*, 211–218.

Montfort, N., Pérez y Pérez, R., Harrel, D. F., & Campana, A. (2013). Slant: A Blackboard System to Generate Plot, Figuration, and Narrative Discourse Aspects of Stories. *Proceedings of the …*.

Needleman, S., & Wunsch, C. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*.

Nieborg, D. (2004). America‟ s Army: More than a game.

Paolacci, G., Chandler, J., & Ipeirotis, P. (2010). Running experiments on amazon mechanical turk. *Judgment and Decision Making*.

Papadimitriou, C., & Tamaki, H. (1998). Latent semantic indexing: A probabilistic analysis. *Proceedings of the …*.

Pérez y Pérez, R., Morales, N., & Rodríguez, L. (2012). Illustrating a Computer Generated Narrative. *International Conference on …*.

Pérez y Pérez, R., & Sharples, M. (2001). MEXICA: A computer model of a cognitive account of creative writing. *Journal of Experimental & Theoretical Artificial Intelligence*, *13*(2), 119–139. doi:10.1080/09528130010029820

Polanyi, L. (1979). So what's the point? *Semiotica*.

Prince, G. (2003). *A dictionary of narratology*.

Radul, A., & Sussman, G. (2009). The art of the propagator. *Proceedings of the 2009 International …*.

Řehůřek, R. (n.d.). gensim: Topic modelling for humans. Retrieved July 28, 2014, from http://radimrehurek.com/gensim/

Riedl, M., & Director-Young, R. (2004). *Narrative generation: balancing plot and character*.

Sayan, E. (2014). Audience Aware Computational Discourse Generation for Instruction and Persuasion.

Schank, R. (1990). *Tell me a story: A new look at real and artificial memory*.

Sgouros, N. (1999). Dynamic generation, management and resolution of interactive plots. *Artificial Intelligence*.

Shackle, G. L. S. (1964). General Thought-schemes and the Economist. *Woolwich Economic Paper 2*.

Smith, T., & Witten, I. (1991). A planning mechanism for generating story text. *Literary and Linguistic Computing*.

Tattersall, I. (2008). An evolutionary framework for the acquisition of symbolic cognition by Homo sapiens. *Comparative Cognition & Behavior Reviews*.

Tearse, B., Mawhorter, P., Mateas, M., & Wardrip-Fruin, N. (2012). Lessons Learned From a Rational Reconstruction of Minstrel. *AAAI*.

Teh, Y., & Jordan, M. (2006). Hierarchical dirichlet processes. *Journal of the American …*.

Theune, M., Faas, S., Heylen, D., & Nijholt, A. (2003). The virtual storyteller: Story creation by intelligent agents.

Trask, R. (2003). *Language: the basics*.

Turing, A. (1950). Computing machinery and intelligence. *Mind*.

Turner, S. (1993). Minstrel: a computer model of creativity and storytelling.

Wardrip-Fruin, N. (2006). Expressive Processing: On Process-Intensive Literature and Digital Media.

Winston, P. H. (1970). Learning structural descriptions from examples.

Winston, P. H. (1986). Learning by augmenting rules and accumulating censors. *Machine Learning: An Artificial Intelligence …*.

Winston, P. H. (2011). The Strong Story Hypothesis and the Directed Perception Hypothesis. In P. Langley (Ed.), *Technical Report FS-11-01, Papers from the AAAI Fall Symposium* (pp. 345–352). Menlo Park, CA: AAAI Press.

Winston, P. H. (2012a). The Next 50 Years: A Personal View. *Biologically Inspired Cognitive Architectures*, *1*, 92–99.

Winston, P. H. (2012b). The Right Way. *Advances in Cognitive Systems*, *1*, 23–36.

Young, K., & Saver, J. (2001). The neurology of narrative. *SubStance*.

Appendix A – Evaluating Story Understanding: Stories, Traits, and Beyond

One of the difficulties in working on the development of automated systems attempting to match human intelligence in learning and creativity is evaluating progress. The only way we currently have to truly evaluate such work is via manual human analysis. However, doing human participant studies often can be both very expensive and time consuming. A popular rising alternative is the use of online forms of data collection and powerful tools such as Amazon's Mechanical Turk. Because of the need for human evaluation of both my work and future Genesis projects, I created a set of tools that work with Genesis information in order to deploy surveys to Amazon's Mechanical Turk.

What is Amazon's Mechanical Turk?

Amazon's Mechanical Turk (often called simply Mechanical Turk) is an online system for doing "Human Intelligence Tasks" or HITs (Paolacci, Chandler, & Ipeirotis, 2010). These tasks are created by requesters, often researchers, and then fulfilled by workers from around the world for a nominal fee. HITs generally have participants doing tasks so that human knowledge can be applied to a problem that is difficult or near impossible to solve computationally. As such, popular tasks include transcription of audio into text, rewriting paragraphs, commenting, and filling out surveys. Some groups have even looked into using Mechanical Turk to help them discover the scripts underlying certain social situations such as ordering food at a restaurant (Li, Lee-Urban, Appling, & Riedl, 2012). An advantage of Mechanical Turk for this type of work is the low cost and high speed of results. Typical rates are often as low as $0.05 per minute of work and many HITs can collect results over a meaningful population less than a day.

Creating the tools for Story Evaluation

In order to leverage the capabilities of Mechanical Turk, I created a web application that enables evaluation of Genesis story information via human surveys. Human participants visit the web application, fill out a survey and then are given a completion code that they submit to Mechanical Turk in order to receive payment.

The goal of the web application is to load the information file from Genesis, generate a set of survey questions, and then record human responses in its database. One requirement is the inclusion of data validation methods to the web server. This is necessary because some users of Mechanical Turk attempt to game the system and get through HIT assignments as quickly as possible without following directions. In order to prevent such users from being paid and making it into the final dataset, the web server automatically validates responses for red flags. If a red flag is detected, the user is given a completion code with some embedded information indicating faulty responses. For normal users, the validation will be successful and they will receive a completion code with a flag indicating the user should receive payment and their data should be accepted.

User Interface

An important part of using Mechanical Turk is to have a good user interface. The interface needs to fulfill a number of goals. First, it needs to direct the user's attention to the relevant information to help them stay on task. Second, it needs to keep the user informed of progress in order to decrease the likelihood that a user starts the survey but then quits. Finally it needs to provide training and instructions to the users in order to ensure that the users are doing the task correctly. Figure 44 below shows a sample of the user interface that a user of my evaluation tool would see.

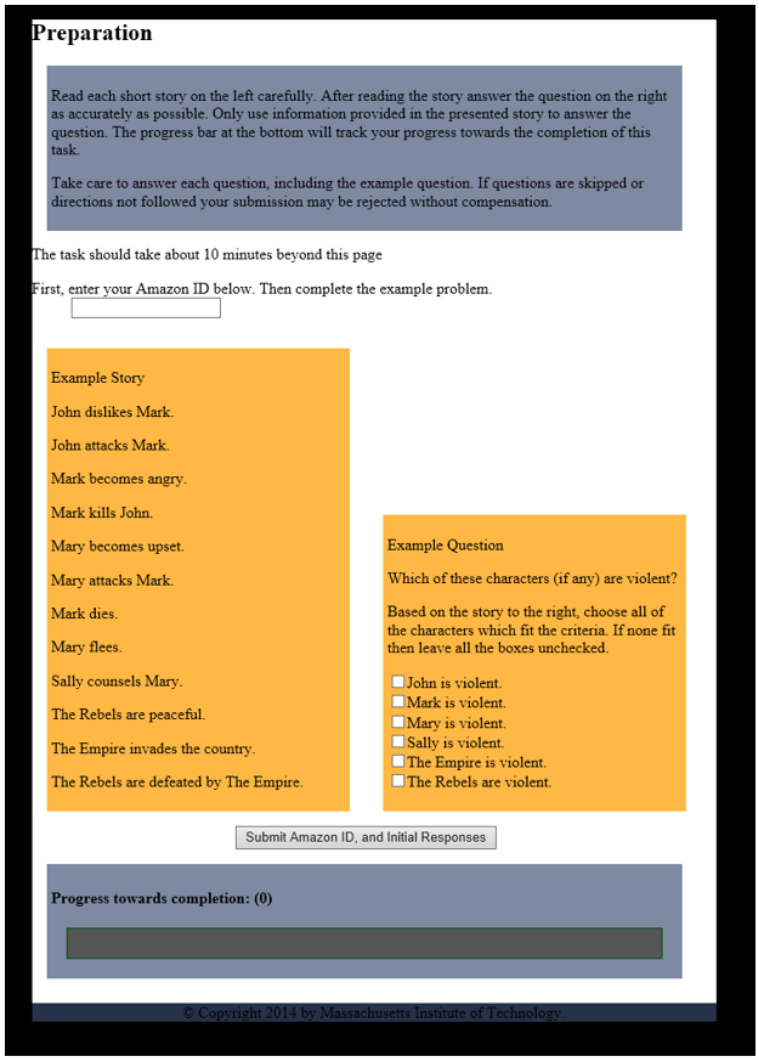


Figure 44 - Web application for conducting surveys about Genesis stories

In order to collect survey data from Amazon's Mechanical Turk, I've created a web application where participants can easily input answers to questions. The system is capable of loading data exported from Genesis and automatically generating the survey for use on Mechanical Turk. The application provides participants with ways of tracking their progress and of reporting completion back to Amazon after taking the survey.

Reusability

While developing this tool I wanted to ensure that this tool would be re-useable for future tasks. I created additional utilities within Genesis and within the web application. I extended Genesis to be able to export stories, characters, and traits into a portable JSON format. Specifically, Genesis now can take the entire library of stories currently in memory and then from that generates the English language version of the stories along with associated characters and traits. All of this information is then output to a JSON file. The survey application is able to work with this JSON format to rapidly and automatically generate the survey from the data. Second, the web page generation is based on a simple template engine, which allows it to be easily adapted to a new type of survey centered on story analysis. Finally, most of the back end code for interacting with the database, recording users' progress, and validating input is easily extendable to fit a variety of tasks while still making it easy for Mechanical Turk users to interact with new surveys.

Creating and Running the Experiment

During my thesis research I ran a pair of initial pilot studies on Mechanical Turk asking participants evaluate stories and the characters within them. The procedure was as follows. The web application was provided with a set of stories and characters to be evaluated. The application generated the survey which was then posted to Mechanical Turk for 30 participants per study. Each participant was asked to read stories and after each reading to answer a question about the characters that the story contains.

In these pilot studies, I focused on the fairly simple character trait: being a violent character. For each story, the participants were asked to identify which characters in the story were violent characters, if any. The first of the two experiments I performed used only stories from the Genesis library (that is, existing stories originally created by humans). The survey included 15 of these stories. The second experiment used only stories generated by Genesis and included 10 such stories. The goal of the pair of experiments was to analyze how people responded to the stories they were reading. Ideally, we would like to see the human responses to the generated stories be similar to the responses of the non-generated stories, as this would indicate that the AI-generated stories are similar to human-generated stories in important ways, at least in terms of the trait being evaluated.

Results

The collected results demonstrate that the Mechanical Turk participants responded to the stories similarly which is promising because it supports the idea that my system is doing a reasonable job at plot generation. Figure 45 below shows some of the high level results analyzing how consistently people marked characters as violent or nonviolent in general.

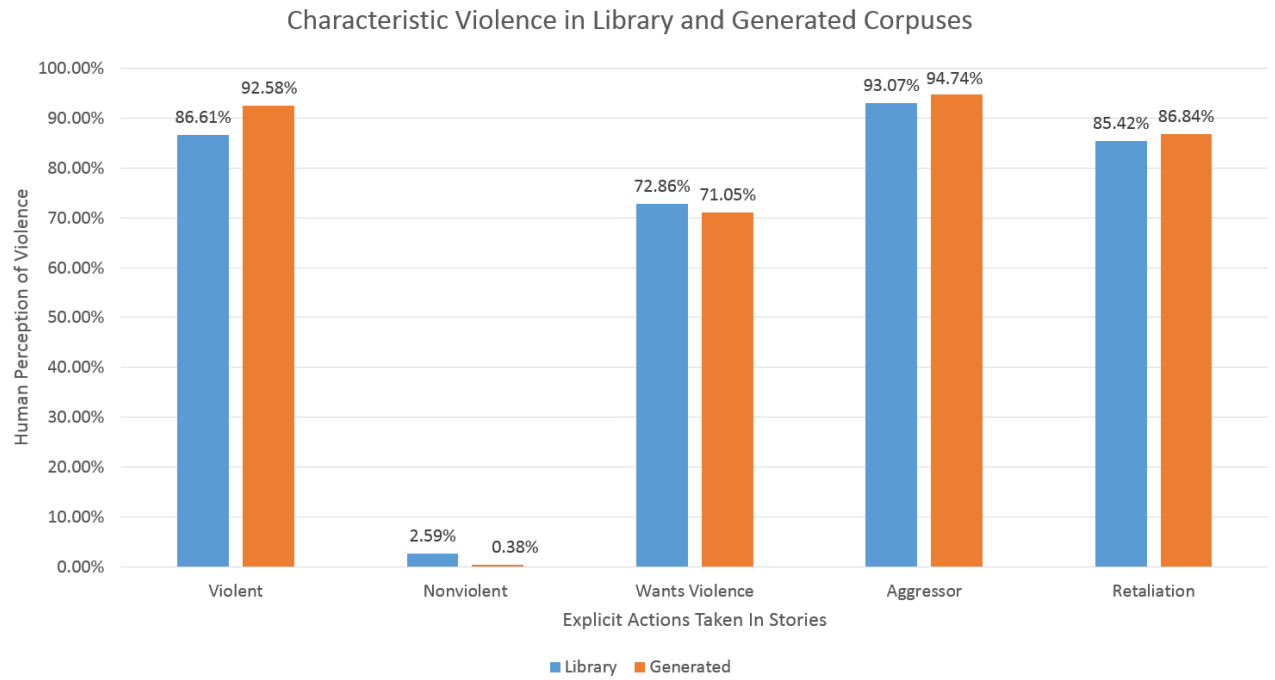


Figure 45 - High level analysis of Mechanical Turk survey
Two experiments were run on Mechanical Turk in order to analyze human response to both AI-generated and human-generated stories. In the chart above, blue columns correspond to results from stories in the Genesis library and orange columns correspond to results from an AI-generated corpus. Each category relates to how often participants marked characters as violent (Except for the *Nonviolent* category, which indicates how often participants marked characters as not violent.) The *Violent* and *Nonviolent* categories are, respectively, for characters who explicitly commit at least one violent act and for characters who do not commit any violent acts at all. The *Wants Violence* category is for characters who want to commit violence, but do not. The final two categories, *Aggressor* and *Retaliation*. *Aggressors* are characters who strike first. *Retaliators* are characters which only commit violent actions after they have been attacked.

In this chart, blue columns represent participant responses to stories from the Genesis library while orange columns represent participants' responses to generated stories. The first category, *Violent*, shows how often characters that commit at least one explicitly violent action are marked as being violent characters by participants. Similarly the second category, *Nonviolent*, shows how often character that do not commit any violent actions, implicitly nor explicitly, are not marked as being violent by participants. The graph shows that the responses to the human-generated stories and AI-generated stories are quite similar.

The next category, *Wants Violence*, shows how often characters that wanted to perform violent actions but ended up not doing so were marked as violent characters by participants. The closeness of the results here helps to support the success of the plot generator while also demonstrating an interesting measure of how people think about violence and violent characters in general. Clearly, a majority of the people surveyed believe that wanting a violent action should be considered violent, but a significant percentage of people do not. The most crucial pattern, however, is that the number of participants who believe that a character who wants violence has the "violent" trait is extremely close whether the stories are AI-generated or not.

This means that people are responding to the AI-generated stories in the same way they respond to the human-generated stories, which is exactly what the goal of my system has been.

The final two categories, *Aggressor* and *Retaliation*, are of high importance. The *Retaliation* column indicates how often participants marked a character as violent when the character did not take violent action until a violent act was committed against the character earlier in the story. Similarly the *Aggressor* column indicates how often characters who commit an unprovoked act of violence are marked as violent by participants.

The results show that for these stories, people tend to be a little more conflicted when decided whether or not retaliatory action makes a character violent. This is a critical result because it demonstrates that the context of violence actions have a noticeable effect on the human responses. The similarity in human response to the human-generated and AI-generated stories supports the idea that the AI-generated stories have believable contexts. Further it suggests that the methodology employed in modeling characters and traits as strings of actions is valuable as it allows the system to maintain and work with the complexities of the scenarios as they are both read and generated.

Additionally, the varying response of humans to different types of actions and scenarios that are consistent between AI-generated and human generated stories suggests that future work could investigate how people think about stories. Specifically questions like "What is violence?" and "What do violent actions mean to different people?" would be of interest to people across fields including the humanities and psychology. As the goal of story generation, story understanding, and artificial intelligence on a whole is to model human intelligence, future experiments could be envisioned in which the goal is to generate stories that elicit similar responses both in a human reader and an AI reader. Such experiments could push the state of the art forward in a number of disciplines.

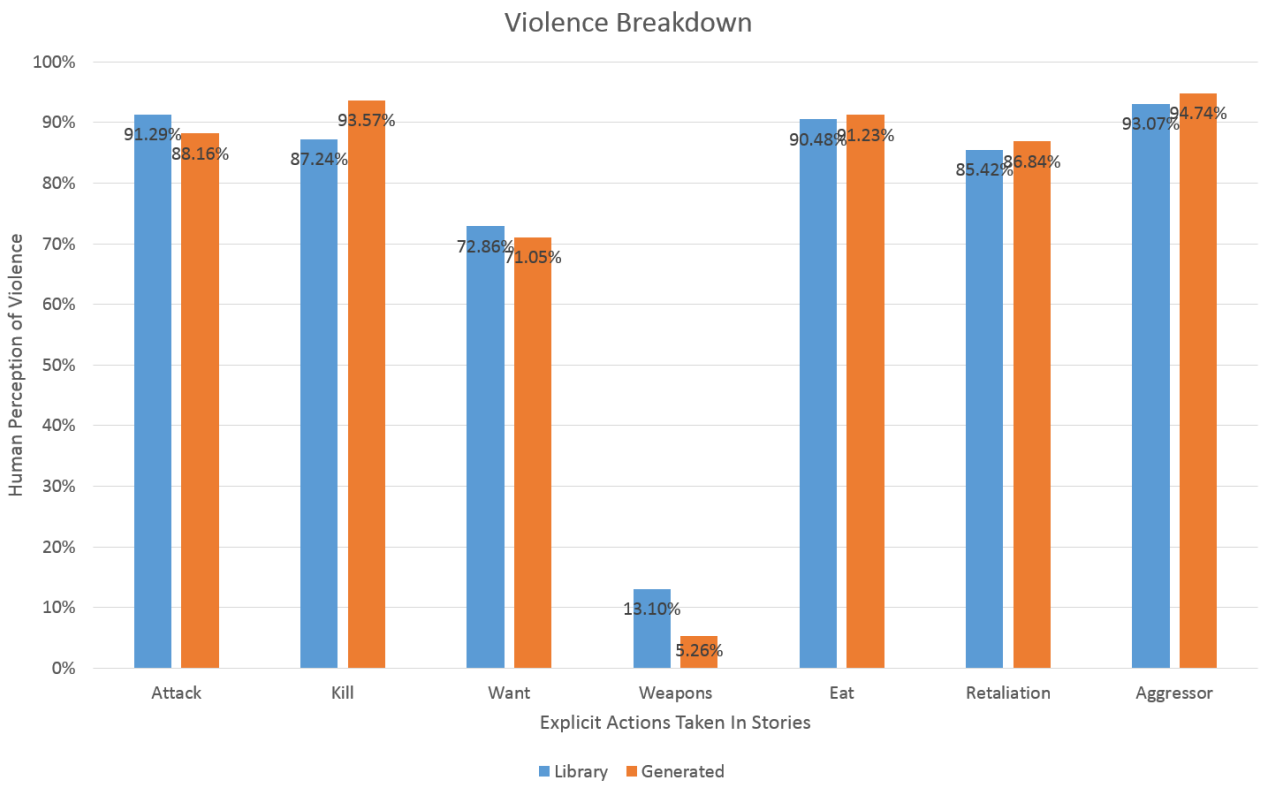


Figure 46 - Breakdown of violent actions and participant responses
Characters with a violent personality may commit violence in a variety of ways. In this chart, the participants' responses to characters who commit various types of violent actions are broken down by type of action. For example, how often characters who attacked or killed others were marked as violent are shown in in the first two categories. The *Weapons* category is an odd, but interesting category. That category corresponds to characters that gave weapons to a violent character, but did not personally commit an act of violence.

A bit less relevant to evaluating my work, but interesting nonetheless is a breakdown of how people responded to different sorts of violent actions that were occurring within the stories. Figure 46 shows that, for the most part, the violent events elicit fairly similar responses whether they were committed in AI-generated or human-generated stories. An interesting side note however, is the *Weapons* column which indicates that the character did not participate in explicitly violent actions but did supply weapons to a character that used them in a violent manner. This is another example that illustrates how context dictates what concepts like violence mean to different people.

Appendix B: Algorithms

B.1 Plot Weaving Binding Search

```
Initialize(RootNode)
PriorityQueue.Add(RootNode)
While(!Queue.isEmpty())
     CurrentNode = PriorityQueue.GetNext()
     If IsLeaf(CurrentNode)
          Return CurrentNode.Bindings
     CurrentGeneric = CurrentNode.UnpairedEntities[0]
     TargetCharacters =
     CurrentNode.AvailableTargetsFor[CurrentGeneric.OriginatingC
     haracter]
     For TargetCharacter in TargetCharacters
          ChildNode = Copy(CurrentNode)
          ChildNode.Bindings.Add(CurrentGeneric,
          TargetCharacter)
          ChildNode.UnpairedEntities.Remove(CurrentGeneric)
          ChildNode.AvailableTargetsFor[CurrentGeneric.Orginatin
          gCharacter].Remove(TargetCharacter)
          ChildNode.Score =
          WeaveScore(Characters,ChildNode.Bindings)
          PriorityQueue.Add(ChildNode)
     ChildNode = Copy(CurrentNode)
     ChildNode.Bindings.Add(CurrentGeneric, NULL)
     ChildNode.UnpairedEntities.Remove(CurrentGeneric)
     ChildNode.Score = WeaveScore(Characters,ChildNode.Bindings)
     PriorityQueue.Add(ChildNode)
```

B.2 Plot Weaving Scorer

```
WeaveScore = 0
For Each Unique Character Pair (C1, C2) in Characters
     PartialWeave = WeaveAlign(C1.plot, C2.plot, Bindings)
     PartialScore = PartialWeave.AlignedElements
     WeaveScore += PartialScore
```

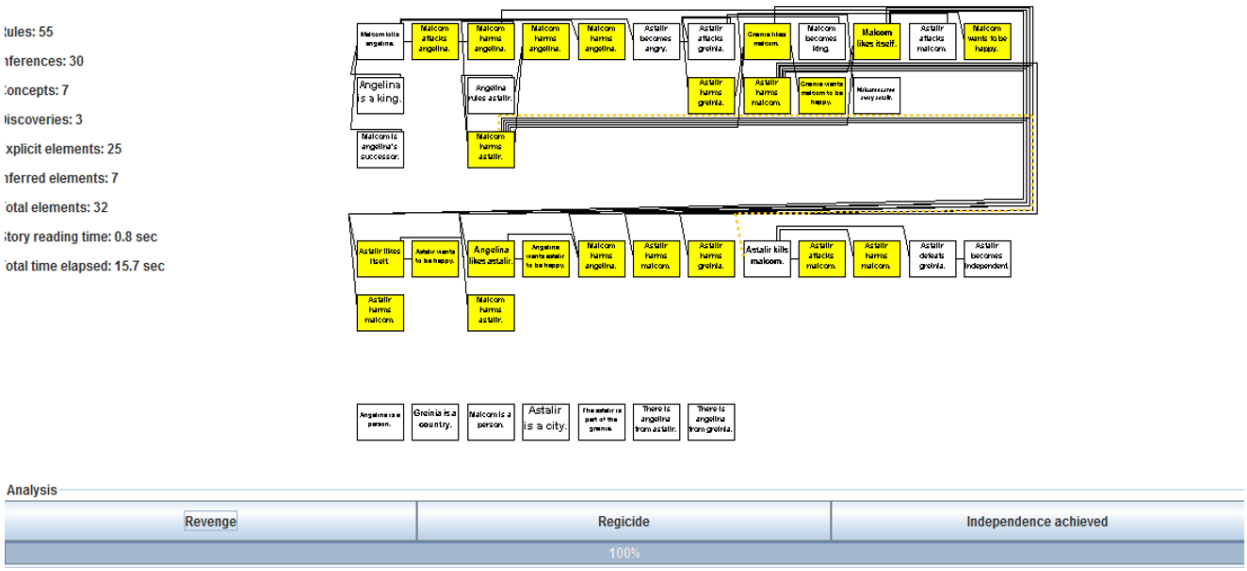
B.3 Plot Weaving Finalization

```
WovenStory = []
PlotPosition = {}
For Character in Characters
     PlotPosition[Character] = 0
While any PlotPosition[Character] < Character.Plot.Length
     For Character in Characters
          While PlotPosition[Character] < Character.Plot.Length
               If
               Character.Plot[PlotPosition[Character]].ContainsO
               nly(Character)
                    WovenStory.Add(Character.Plot[PlotPosition[C
                    haracter]]
                    PlotPosition[Character] += 1
               Else
                    Character.Blocked = True
                    BlockedElement =
                    CharacterPlot[PlotPosition[Character]]
                    CanUnblock = True
                    For OtherCharacter In BlockedElement
                         If
                         OtherCharacter.Plot[PlotPosition[OtherC
                         haracter]] != BlockedElement
                              CanUnblock = False
                    If CanUnblock
                         WovenStory.Add(Character.Plot[PlotPosit
                         ion[Character]]
                         PlotPosition[Character] += 1
                         For OtherCharacter In BlockedElement
                              PlotPosition[OtherCharacter] += 1
```

Appendix C: Collection of Generated Stories

C.1. Greinia and Astalir

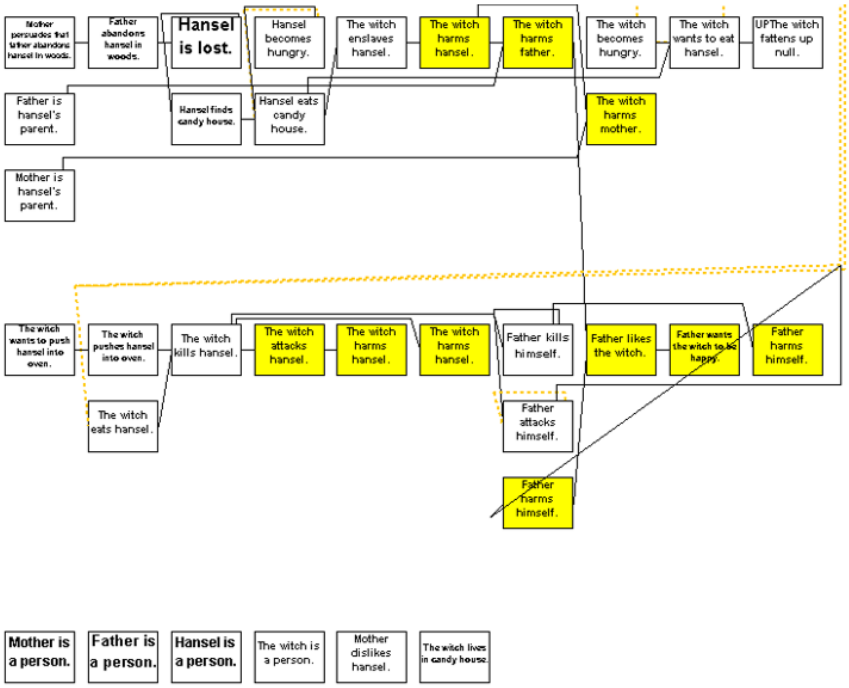
This story was generated by combining traits and characterizations from Shakespearean tragedies and war stories.




```
Greinia is a country.
Astalir is a city in Greinia.
Angelina is from Astalir.
Angelina is from Greinia.
Angelina becomes the Queen of Greinia.
Angelina rules Greinia.
Malcom is Angelina's successor.
Malcom kills Angelina.
Malcom becomes King of Greinia.
Astalir rebels.
Astalir attacks Greinia.
Malcom scares away Astalir's forces.
Astalir attacks Malcom.
Astalir kills Malcom.
Astalir defeats Greinia.
Astalir becomes independent.
```

C.2. Hansel without Gretel

This story was generated by removing the character of Gretel from the original story of Hansel and Gretel.

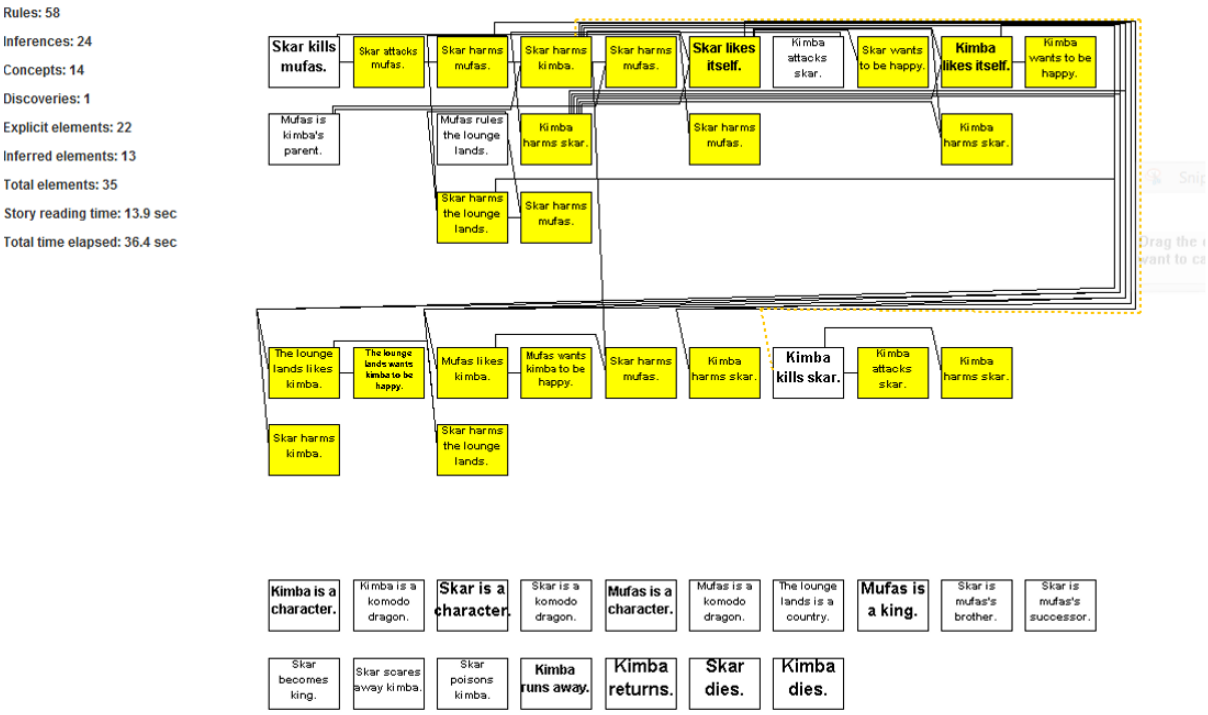


Satisfied appetite	Grief self harm

```
Mother dislikes Hansel.
Mother wants to kill Hansel.
Mother convinces Father to abandon Hansel in the woods.
Father abandons Hansel in the woods.
Hansel and is lost.
Hansel becomes hungry.
Hansel finds the house made of candy.
Hansel begins eating the house made of candy.
The witch lives in the house made of candy.
The witch enslaves Hansel.
The witch becomes hungry.
The witch wants to eat Hansel.
The witch fattens up Hansel.
The witch pushes Hansel into the oven.
The witch kills Hansel.
The witch eats Hansel.
Father discovers Hansel died.
Father kills himself.
```

C.3. Komodo King

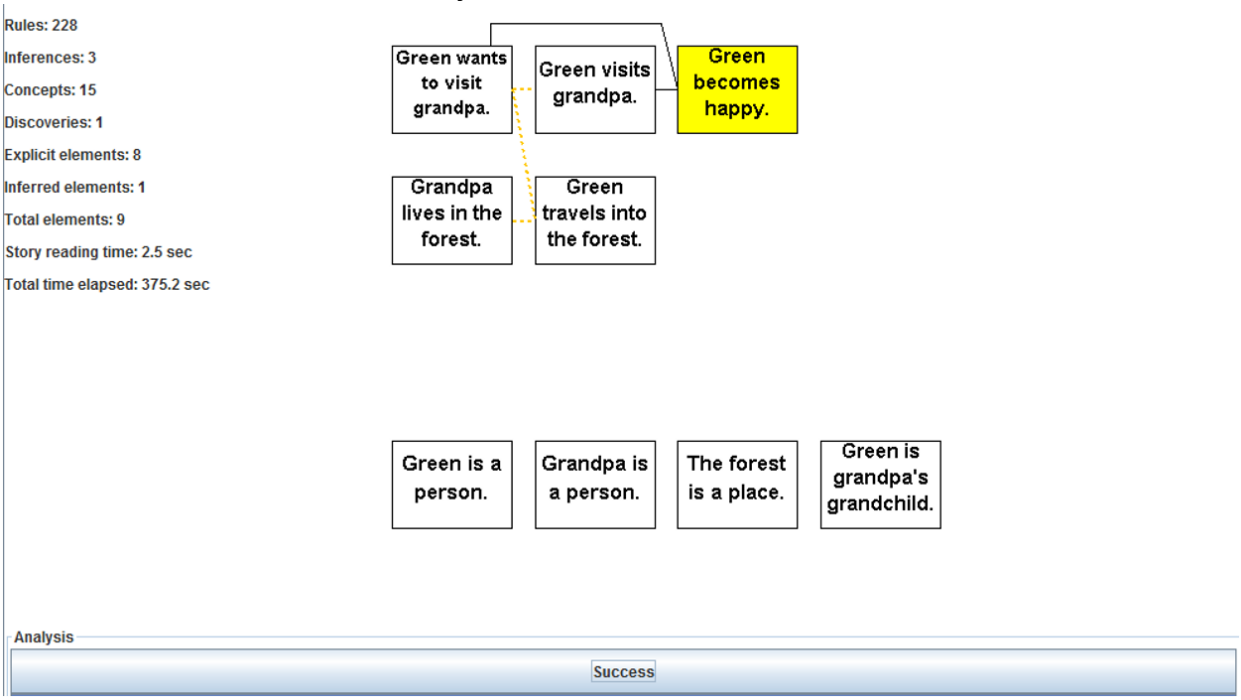
This story was generated primarily from traits and characterizations from *Hamlet* in order to try to emulate the story of *The Lion King*.




```
Mufas is the king.
Mufas rules the Lounge Lands.
Skar is Mufas's brother.
Skar is Mufas's successor.
Mufas is Kimba's parent.
Skar kills Mufas.
Skar becomes King.
Skar scares away Kimba.
Skar poisons Kimba.
Kimba runs away.
Kimba returns.
Kimba attacks Skar.
Kimba kills Skar.
Skar dies.
Kimba dies.
```

C.4. Little Green Riding Hood

This goal of this generated story was to explore what would happen in Little Red Riding Hood if the wolf was removed from the story.

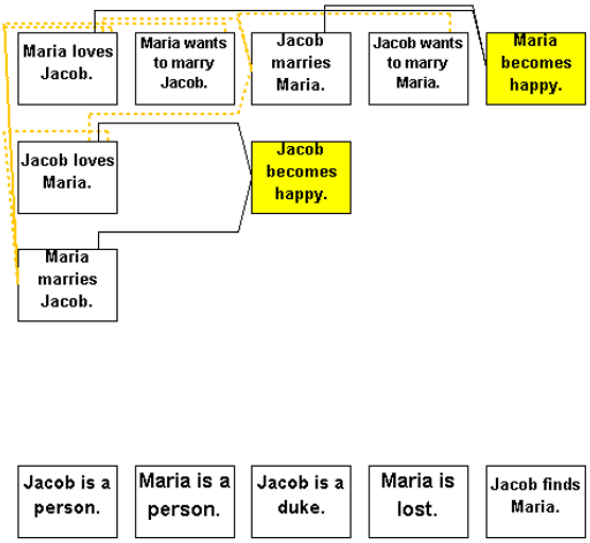



```
Green is Grandpa's grandchild.
Grandpa lives in the forest.
Green wants to visit Grandpa
Green travels into the forest
Green visits Grandpa.
```

C.5. Lost Love

This story was generated from combinations of characteristics from romantic comedies.

Rules: 117
Inferences: 6
Concepts: 14
Discoveries: 2
Explicit elements: 11
Inferred elements: 2
Total elements: 13
Story reading time: 0.6 sec
Total time elapsed: 25.8 sec



Analysis

Marriage successful	Marriage successful

Jacob is a duke.

Maria is lost.

Jacob finds Maria.

Jacob loves Maria.

Maria loves Jacob.

Jacob wants to marry Maria.

Maria wants to marry Jacob.

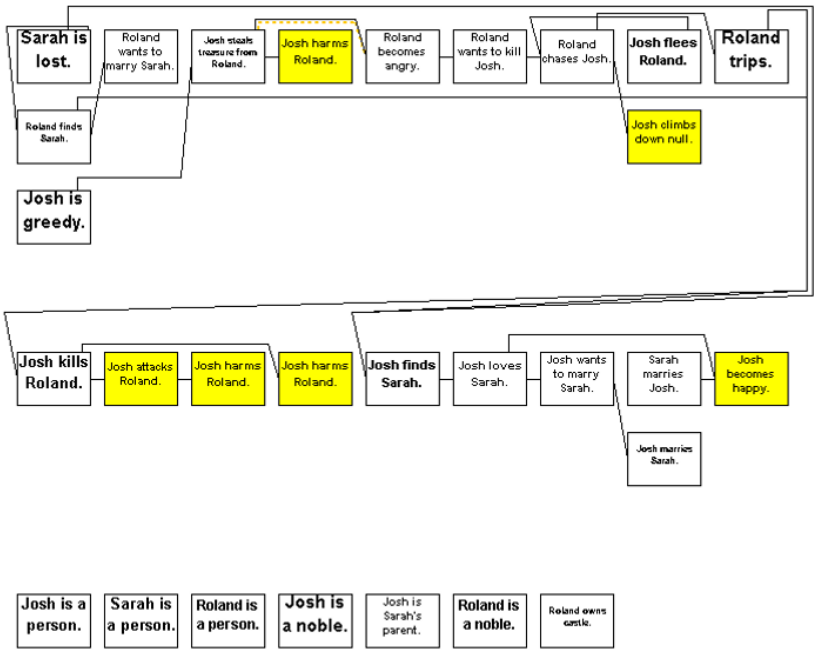
Maria marries Jacob.

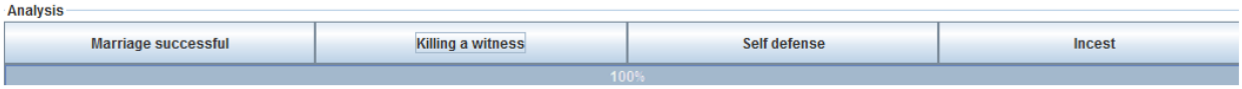
Jacob marries Maria.

C.6. Oedipus Much

This story was generated from a combination of characteristics from romantic comedies and fairy tales. It exhibits an unexpected romantic pairing.

tules: 55
nferences: 19
:oncepts: 10
)iscoveries: 4
:xplicit elements: 34
nferred elements: -7
'otal elements: 27
Story reading time: 0.7 sec
'otal time elapsed: 7.9 sec

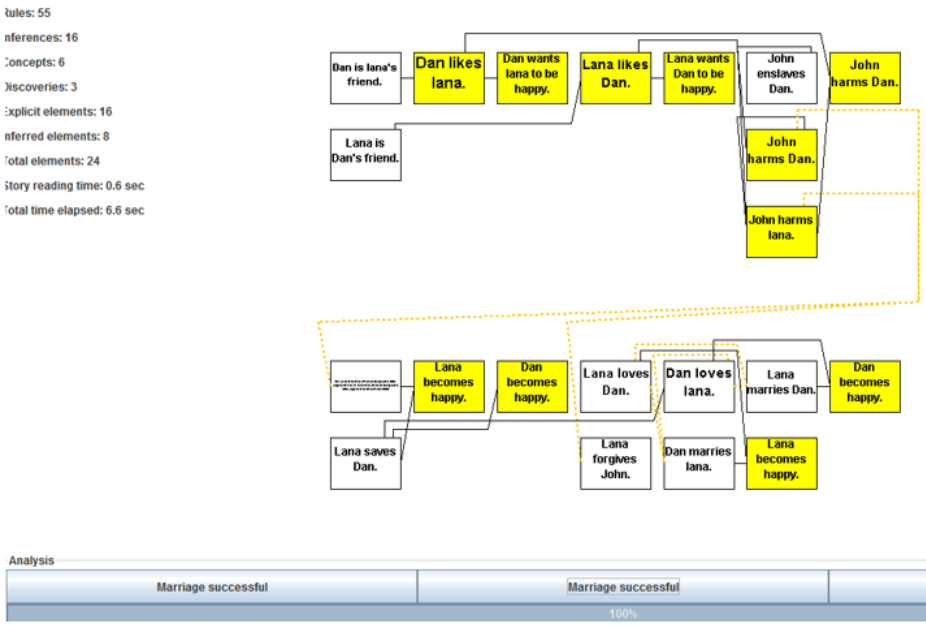





```
Roland is a noble.
Josh is a noble.
Josh is Sarah's parent.
Sarah is lost.
Roland owns a castle.
Roland finds Sarah
Roland wants to marry Sarah.
Josh steals treasure from Roland.
Roland becomes angry with Josh.
Roland wants to kill Josh.
Josh flees from Roland.
Roland chases Josh.
Roland trips and falls.
Josh kills Roland.
Josh finds Sarah.
Josh falls in love with Sarah.
Josh wants to marry Sarah.
Josh marries Sarah.
Sarah marries Josh.
```

C.7. Pacifist Heroine

This story was generated to show how a character could have conflicting characterizations, in this case Lana was generated with qualities of violence and pacifism.

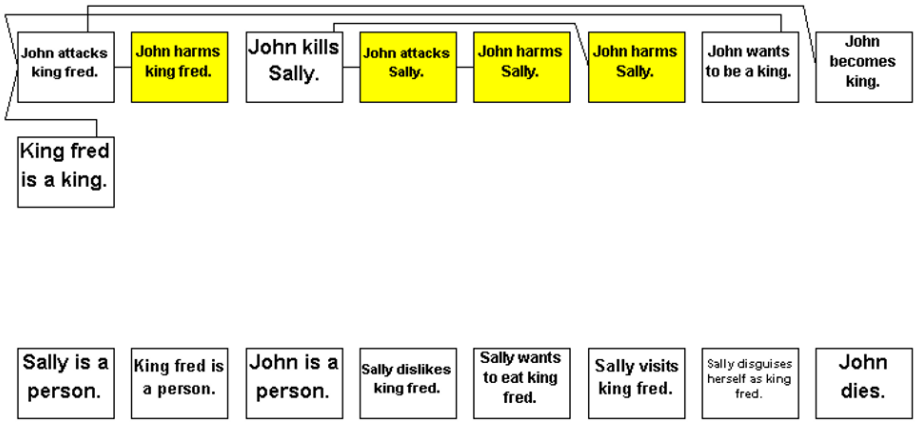


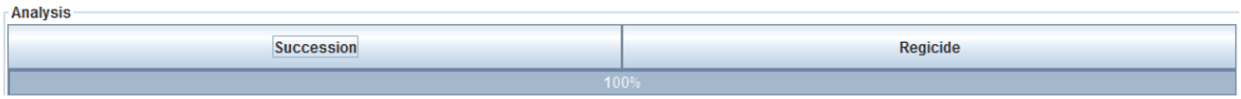

```
Dan is Lana's friend.
Lana becomes queen.
Pete enslaves Dan.
Lana wants to help Dan.
Lana saves Dan.
Lana wants to kill Pete.
Lana forgives Pete.
Dan loves Lana.
Dan marries Lana.
Lana marries Dan.
Lana dies of old age.
```

C.8. Regicides

This story was generated using a combination of characteristics from fairy tales and tragedies.

Rules: 530
Inferences: 7
Concepts: 16
Discoveries: 2
Explicit elements: 16
Inferred elements: 0
Total elements: 16
Story reading time: 0.8 sec
Total time elapsed: 1387.0 sec





King Fred is the king.

Sally dislikes King Fred.

Sally wants to eat King Fred.

Sally visits King Fred.

Sally disguises herself as King Fred.

John wants to be king.

John attacks King Fred.

John kills Sally.

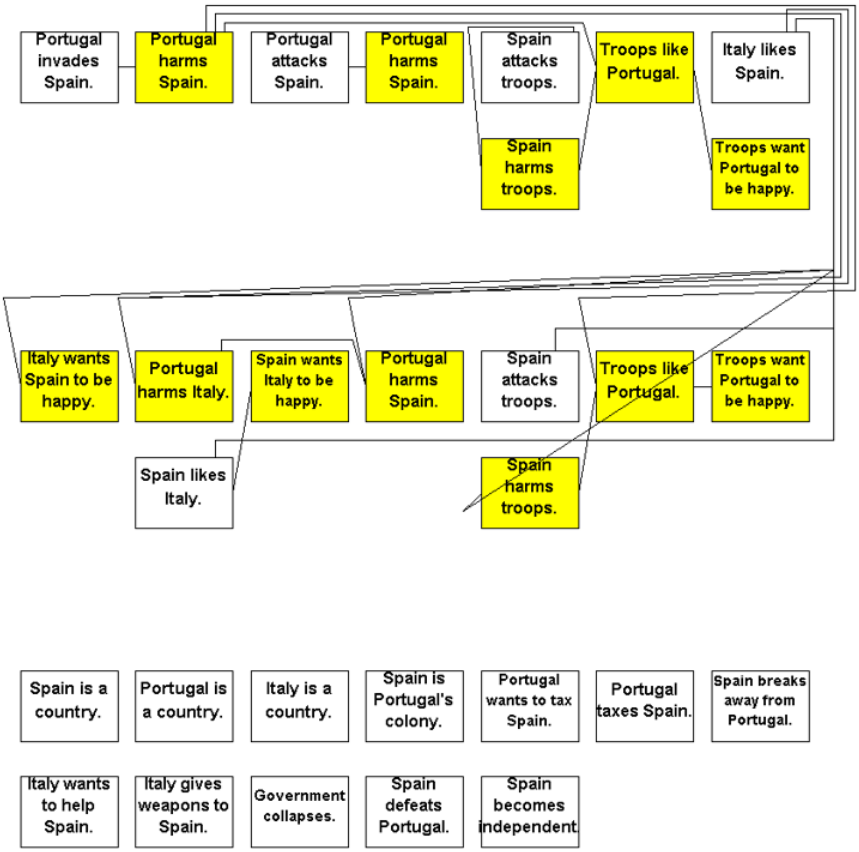
John becomes king.

John dies.

C.9. War for Western Europe

This story was generated by from characteristics of countries in war stories.

Rules: 589
Inferences: 12
Concepts: 16
Discoveries: 2
Explicit elements: 18
Inferred elements: 10
Total elements: 28
Story reading time: 8.3 sec
Total time elapsed: 2045.1 sec

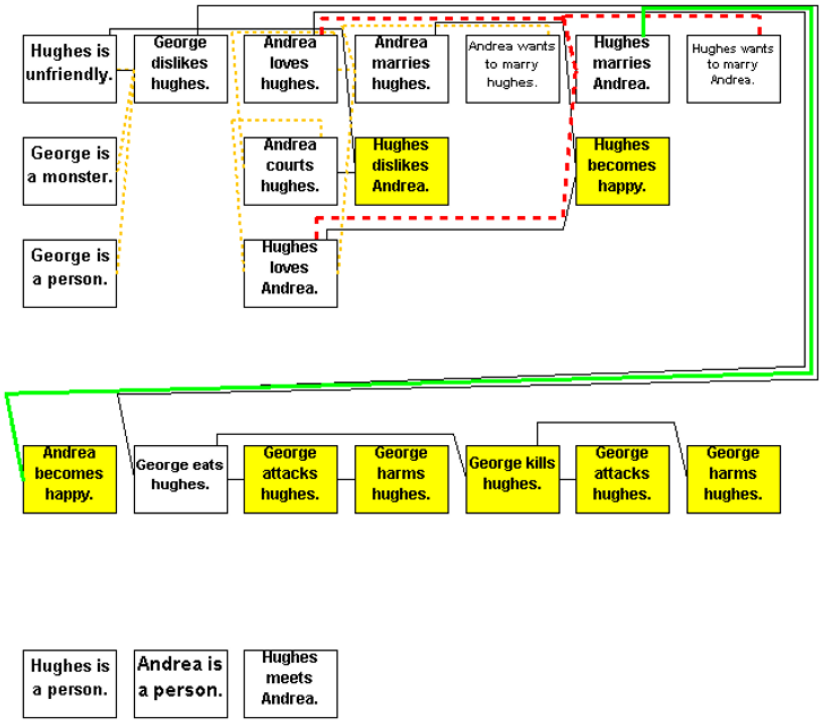


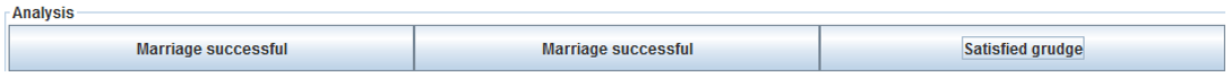

```
Spain is Portugal's colony.
Portugal wants to tax Spain.
Portugal taxes Spain.
Spain breaks away from Portugal.
Portugal invades Spain.
Portugal attacks Spain.
Spain attacks Portugal's troops.
Italy likes Spain.
Italy wants to help Spain.
Spain likes Italy.
Italy gives weapons to Spain.
Italy's government collapses.
Then, Spain attacks Portugal's
troops.
Spain defeats Portugal.
Spain becomes independent.
```

C.10. Wedding Crasher

This story was generated from a combination of characteristics from fairy tales, comedies, and tragedies.

Rules: 707
Inferences: 18
Concepts: 17
Discoveries: 3
Explicit elements: 17
Inferred elements: 4
Total elements: 21
Story reading time: 1.4 sec
Total time elapsed: 2946.6 sec





George is a monster.

Hughes is unfriendly.

George dislikes Hughes.

Hughes meets Andrea.

Andrea loves Hughes.

Andrea courts Hughes.

Hughes loves Andrea.

Andrea wants to marry Hughes.

Hughes wants to marry Andrea.

Andrea marries Hughes.

Hughes marries Andrea.

George eats Hughes.