6.UAP Thesis: Story Analogies with Structure Mapping and Plot Units

Jesse Dunietz

May 20, 2011

1 Introduction

Stories pervade our mental lives. We use stories to explain the state of the world, to provide context for objects and actions, to persuade others, and especially to predict the outcomes of our actions. If we wish to build programs that interact with and think about the world as intelligently as we do, we must allow them to understand and reason about stories.

For proper reasoning and inference, the ability to make analogies between stories is also crucial. For instance, political leaders often base their actions on precedent, inferring from a precedent's similarity to recent events the likely outcomes of their actions. To do this, they must be able to spot the similarity, map current entities and events onto those in the precedent, and hypothesize new events based on the precedent.

The Genesis group at MIT has developed a software system called Genesis that implements some of the requisite story comprehension and reasoning. It parses stories in simple English into a rich internal representation, using a common-sense knowledge

database to make basic inferences such as "If John killed Mary, Mary is dead." It can even extract "plot units" – high-level patterns such as revenges, mistakes, and successes – identifying what elements in the story matched each pattern. However, once it has this higher-level information, Genesis has historically not done very much with it. Genesis has also in the past included only minimal tools for comparing stories.

I therefore attempted to implement a subsystem to allow Genesis to take this next step: to make analogies between stories, using plot units (along with the rest of its information about a story) to guide the process. My basic approach was to use a version of the Structure Mapping Engine [Falkenhainer et. al. 1989], a general-purpose analogy-drawing system, to extract analogies between two stories. My results demonstrate that this technique, at least as I implemented it, is **not** an effective method for drawing powerful analogies between stories. Nonetheless, they also demonstrate that plot units contribute valuable information that can significantly improve even a generally ineffective method of analogy.

In this paper, I describe the components I used in the construction of the system, the design of the integrated system itself, and the results of my experiments running the system on a set of test stories. I also include some comments on the contributions of the work to the story understanding effort, and the implications this work may have for future directions.

2 Prior Work

My work incorporates two pre-existing software projects: the Genesis system and the Structure Mapping Engine.

2.1 Genesis

The Genesis system was developed by the Genesis group at MIT to push the state of the art in human-like story processing. It is designed to parse stories in simple English; interpret the story text to obtain an internal representation of the entities in the story, their relationships, and the events they participate in; and, most impressively, to derive the causal structure of the story and infer unstated facts about it.

Genesis is highly modular, built on a "propagator" architecture, an extension of the Observer Design Pattern that makes all entities equally publishers and subscribers. The system components communicate via metaphorical "wires" – named channels over which each object can broadcast, and to which other components may subscribe. This publish/subscribe-based loose coupling allows new modules such as mine to be added easily to the system, making it particularly easy to add new hooks for existing processing events. This feature was essential when attempting to add analogical processing as an additional step to be performed each time a new story was read.

Genesis uses an internal ontology consisting of four classes of items:

- A `Thing` is any item with an independent existence. This is the base class for the other three types of items.
- A `Derivative` is a `Thing` that exists only in terms of other `Thing`s – e.g., the top of a table.
- A `Relation` (a type of `Derivative`) represents a relationship between two other `Thing`s.
- A `Sequence` is a list (not necessarily strictly ordered) of other `Thing`s.

In large part, my work consisted of translating objects of these classes into the representations that the SME could work with.

An important feature of Genesis' story-processing apparatus is its ability to pick out "reflections," more commonly called "plot units." These are small patterns of events within stories that are given names. For instance, "A harms B, causing B to harm A" is the structure of a plot unit labeled "Revenge." Genesis can detect instances of patterns such as these in a story, even when the events that match the pattern were only implicit. It outputs a "reflection analysis," which consists of a list of descriptions of the plot units found in the story. Each plot unit description contains the name of the plot unit, the variable names used in the template that describes it, and the `Thing`s to which those variables have been bound. One of the goals of my project was to use these pieces of higher-level understanding of the stories to guide the analogical mapping process.

2.2 Structure Mapping Engine

The Structure Mapping Engine is a general-purpose analogy-drawing system, built by Falkenhainer and Forbus [Falkenhainer et. al. 1989] to implement Gentner's [1995] model of human analogical reasoning. Gentner's model characterizes people's preference for certain analogical mappings over others in terms of relational structure. When drawing an analogy between a "source" system A and a "target" system B, a mapping between an A item and a B item is considered good when those entities play similar roles in the larger relational structures of A and B. Inter-system mappings with higher degrees of "systematicity" – interconnectedness between relationships, such as relations between relations – will be preferred.

Forbus' SME is an implementation of this model. Forbus' mapping algorithm takes

as input two "description groups," or "dgroups" – graphs describing the entities and relationships of the two systems between which an analogy is to be drawn. The goal of the SME is to find a subgraph of each input that can consistently be mapped onto an identically structured subgraph in the other input. The algorithm attempts to maximize the systematicity of these subgraphs, as well as their size. Structure mapping theory has been widely accepted in the cognitive science community as at least a good approximation of the process of analogical reasoning, and the Structure Mapping Engine has been successfully applied in a variety of applications.

The SME's ontology is remarkably similar to that of Genesis. It allows for four types of items to appear in a dgroup:

- An `Entity` is any item with independent existence.
- A `Relation` represents a relationship between any two items.
- An `Attribute` is a feature of some item, and as such is inherently linked to the item; it cannot exist without it.
- A `Function` is essentially a unary relation. Conceptually, it is very similar to an `Attribute`, but the SME treats it slightly differently when performing a matching.

The SME will only map a relation, function or attribute from the source onto one in the target if they share a type, which is defined by a string (e.g., a `son-of` relationship in one system can be matched to a `son-of` relationship in the other, but not to a `daughter-of` relationship). Entities will be matched regardless of their name. Another important restriction that the SME algorithm imposes is that mappings between the source and target must be one-to-one; a mapping in which a single item in one system plays the roles of two items in the other is deemed inconsistent.

While Forbus' original SME implementation was written in LISP, I used a java port of it by Mark Finlayson. Debugging previously undiscovered problems in this implementation comprised a significant portion of my efforts.

3 System Design and Implementation

The design of my integrated system is fairly simple. My new module receives two inputs from the main Genesis system for each story read: a `Sequence` object containing the complete story and all the inferences that have been made about it, and an object containing the reflection analysis for the story. The primary tasks of my module are: to translate this information for each story into a dgroup which it can pass to the SME; to recover the original Genesis `Thing`s from the SME output; and to display the mappings between `Thing`s in some reasonable fashion.

To keep this project simple, analogies were restricted to be between the last two stories read. Initially, the system was intended to compare each newly read story against each of the previous stories and select the best match. However, the mappings output were of sufficiently poor quality that this would not have been a useful feature.

3.1 Building Dgroups from Genesis Output

To construct a description group for each story being compared, the system first iterates over all of the `Thing`s in the story output by Genesis. For each `Thing` t, it first recursively converts all of t's children to a set of dgroup items, then adds t itself. (A child in this context is either the subject of a `Derivative`, the subject or object of a `Relation`, or the elements of a `Sequence`.) The dgroup items are stored in a hashmap that maps `Thing`s to dgroup items. This allows the system to avoid adding the same `Thing` twice and to retrieve the same dgroup item each time

the corresponding `Thing` appears in the story graph. The actual dgroup object is constructed from the value list of the hashmap.

The actual Genesis-to-SME conversion process is straightforward. An undifferentiated Genesis `Thing` corresponds naturally to an SME `Entity` with the same name. `Relation`s likewise map directly to SME `Relation`s (with the caveats that, unlike an SME `Relation`, a Genesis `Relation` cannot have more than two arguments and its arguments are never commutative). A Genesis `Derivative` (that is not a `Relation`) is slightly harder to map: conceptually, it could translate either to an SME `Function` or an SME `Attribute`. The semantics of `Attribute`s are that they are properties inherent to a specific object, whereas `Function`s represent quantities that happen to be linked to a specific object but need not be. The semantics of `Derivative`s seem to match those of `Attribute`s more closely, so this translation was selected.

There is no equivalent in the SME to Genesis' `Sequence` objects. Treatment of `Sequence`s is further complicated by the fact that the actual meaning of a `Sequence` in Genesis depends on the tag associated with the `Sequence` – it may indicate a conjunction, an ordered sequence of events, or a variety of other meanings. To obtain the correct behavior, the system creates an `Entity` representing the entire `Sequence`, and creates a new SME `Relation` whose type is given by the `Sequence`'s tag. This has the effect of ensuring that the `Sequence` will only be matched against others of the same type.

The next step is to convert the plot unit descriptions into elements of the dgroup. The procedure for this is similar to that for converting `Sequence`s. Each plot unit was instantiated from a template – for example, the Revenge template might look like: `XX harms YY. XX's harming YY causes YY to harm XX.` The variables in this template have been instantiated to actual `Thing`s in the process of finding a match in the story for the plot unit. If Macduff takes revenge on Macbeth, we might say that

Macbeth is this particular revenge's `XX` – in other words, that there is an `XX` relation between Macbeth and this revenge. Following this line of reasoning, for each plot unit, the system creates a new `Entity`, and for each bound variable in the plot unit, a new `Relation` is added relating the plot unit entity to the variable's value, where the `Relation`'s type is given by the variable name.

3.1.1 Expected Effect of Plot Units on the SME

At first glance, it may seem that allowing each plot unit to be regular `Entity` like any other does not take full advantage of the plot unit. After all, plot units are supposed to be higher-level knowledge; intuitively, it would make sense to give them extra power in directing the mappings.

The expectation that led to the current translation mechanism was that a bias toward mapping the plot units well would simply fall out of the SME's algorithm. As elaborated above, the SME prefers analogies with dense networks of interrelated entities and relations. Each plot unit, when translated into dgroup items, is the nexus of a large cluster of relations between it and the items corresponding to its variable bindings. Each plot unit and its cluster therefore ought to be one of the most appealing subgraphs to be matched: it exhibits a high degree of systematicity.

3.2 Recovering Genesis Mappings from SME Output

The SME matching function returns a list of mappings between source items and target items. These objects are, naturally, drawn from the set of those provided as SME inputs. The project goal, however, was to produce a mapping of Genesis objects to Genesis objects. To facilitate translation of the mappings back into Genesis objects, I implemented derived classes for each of the relevant SME object types. The

new `Entity`, `Relation`, and `Attribute` objects store the Genesis objects from which they were generated, so that when these objects are included in a mapping, the corresponding Genesis object mapping can be immediately derived.

3.3 Displaying Results

In a more fully developed version of this project, it would make sense to include a full-fledged GUI to display the mappings. For the purposes of experimenting with the effectiveness of this mapping technique, however, simple text output sufficed. Indeed, the low quality of the mappings indicates that it probably would not have been worthwhile to produce a GUI for this system.

3.4 Greedy Algorithm Implementation

Performing an exhaustive search over all possible mappings is computationally infeasible for inputs of any reasonable size. Forbus therefore included in his original SME package a greedy algorithm that uses a beam search as it looks for the best ways to combine small groups of mappings into larger groups. This variant of the algorithm was not available in the Java SME library on which this project relies, but it had been in a previous version of the Java library. Part of my task, then, was to port over the algorithm from the old version of the SME package to the new version.

4 Results

The system was tested on several sets of stories from Genesis' story database. The most basic test was seeing what happened when a story was compared against itself. I also tried comparing several relatively similar stories against each other to see if

interesting mappings were produced. I tried the comparison with and without plot units.

Although the SME is technically capable of proposing inferences to be made about the target, it did not do so in any of my tests.

I present the data first; a discussion of their significance follows. For the sake of brevity, I have only retained the top three mappings from each output.

4.1 Comparing Stories with Themselves

Below are the mappings produced when the standard "Julius Caesar" plot from Genesis' story database is compared against itself without plot units (each SME run produces several proposed mappings, separated by blank lines):

```
Derivative: object-9906(anthony-9438, ) <--> Derivative: object-4957(brutus-4391, )
brutus-9429 <--> brutus-4391
Relation: harm-9904(roles-9905, brutus-9429, )
    <--> Relation: harm-4955(brutus-4391, roles-4956, )

Relation: murder-9634(roles-9653, cassius-9435, )
    <--> Relation: murder-4551(brutus-4391, roles-4584, )
Derivative: object-9654(caesar-9432, ) <--> Derivative: object-4585(caesar-4394, )
cassius-9435 <--> brutus-4391

Relation: want-9570(cassius-9435, appear-9571, )
    <--> Relation: want-4532(cassius-4397, appear-4533, )
cassius-9435 <--> cassius-4397
Derivative: appear-9571(property-9572, ) <--> Derivative: appear-4533(property-4534, )
dead-9573 <--> dead-4535
caesar-9432 <--> caesar-4394
Relation: property-9572(caesar-9432, dead-9573, )
    <--> Relation: property-4534(dead-4535, caesar-4394, )
```

The same comparison with plot units:

```
Relation: murder-9634(roles-9653, cassius-9435, )
    <--> Relation: murder-4551(brutus-4391, roles-4584, )
cassius-9435 <--> brutus-4391
Derivative: object-9654(caesar-9432, ) <--> Derivative: object-4585(caesar-4394, )

Derivative: object-9928(brutus-9429, ) <--> Derivative: object-4679(anthony-4400, )
```

```
Relation: harm-9926(anthony-9438, roles-9927, )
    <--> Relation: harm-4677(cassius-4397, roles-4678, )
anthony-9438 <--> cassius-4397

cassius-9435 <--> cassius-4397
Relation: property-9572(caesar-9432, dead-9573, )
    <--> Relation: property-4534(dead-4535, caesar-4394, )
caesar-9432 <--> caesar-4394
Derivative: appear-9571(property-9572, ) <--> Derivative: appear-4533(property-4534, )
dead-9573 <--> dead-4535
Relation: want-9570(cassius-9435, appear-9571, )
    <--> Relation: want-4532(cassius-4397, appear-4533, )
```

A similar self-comparison for the story of Macbeth, without plot units:

```
Relation: property-9966(dead-9967, lady_macbeth-9251, )
    <--> Relation: property-4766(sane-4608, lady_macbeth-4064, )
dead-9967 <--> sane-4608
lady_macbeth-9251 <--> lady_macbeth-4064

Relation: has-mental-state-9834(happy-9835, macbeth-9257, )
    <--> Relation: has-mental-state-4450(happy-4420, duncan-4061, )
happy-9835 <--> happy-4420
macbeth-9257 <--> duncan-4061

Derivative: appear-9750(position-9749, ) <--> Derivative: appear-4563(position-4562, )
lady_macbeth-9251 <--> lady_macbeth-4064
Relation: want-9702(appear-9750, macbeth-9257, )
    <--> Relation: want-4515(macbeth-4070, appear-4563, )
Relation: persuade-9701(lady_macbeth-9251, want-9702, )
    <--> Relation: persuade-4514(want-4515, lady_macbeth-4064, )
king-9441 <--> king-4254
Relation: position-9749(king-9441, macbeth-9257, )
    <--> Relation: position-4562(king-4254, macbeth-4070, )
macbeth-9257 <--> macbeth-4070
```

The same comparison, with plot units:

```
Relation: position-9747(macbeth-9255, king-9439, )
    <--> Relation: position-4560(king-4252, macbeth-4068, )
path-10263 <--> path-5099
Derivative: object-9785(duncan-9246, ) <--> Derivative: object-4598(duncan-4059, )
king-9439 <--> king-4252
```

```
Derivative: appear-9748(position-9747, ) <--> Derivative: appear-4561(position-4560, )
duncan-9246 <--> duncan-4059
Relation: murder-9767(roles-9784, macbeth-9255, )
    <--> Relation: murder-4580(roles-4597, macbeth-4068, )
macbeth-9255 <--> macbeth-4068
Relation: want-9693(appear-9748, macbeth-9255, )
    <--> Relation: want-4506(appear-4561, macbeth-4068, )
dead-9813 <--> dead-4626
Relation: property-9812(dead-9813, duncan-9246, )
    <--> Relation: property-4625(dead-4626, duncan-4059, )
path-10263 <--> path-5099
Derivative: appear-9811(property-9812, ) <--> Derivative: appear-4624(property-4625, )

macbeth-9255 <--> macbeth-4068
happy-9833 <--> happy-4646
path-10272 <--> path-5115
Derivative: appear-9831(has-mental-state-9832, )
    <--> Derivative: appear-4644(has-mental-state-4645, )
Relation: position-9747(macbeth-9255, king-9439, )
    <--> Relation: position-4560(king-4252, macbeth-4068, )
Relation: has-mental-state-9832(happy-9833, macbeth-9255, )
    <--> Relation: has-mental-state-4645(happy-4646, macbeth-4068, )
king-9439 <--> king-4252
Derivative: appear-9748(position-9747, ) <--> Derivative: appear-4561(position-4560, )
path-10272 <--> path-5115

king-9439 <--> king-4252
duncan-9246 <--> duncan-4059
Relation: position-9497(duncan-9246, king-9439, )
    <--> Relation: position-4310(king-4252, duncan-4059, )
```

4.2 Comparing Different Stories

Caesar and Macbeth were compared against each other, with Caesar as the source and Macbeth as the target:

```
Derivative: object-14983(cassius-14679, ) <--> Derivative: object-9699(duncan-9099, )
Relation: harm-14981(roles-14982, anthony-14682, )
   <--> Relation: harm-9697(macbeth-9108, roles-9698, )
anthony-14682 <--> macbeth-9108

Derivative: object-15239(brutus-14673, ) <--> Derivative: object-9699(duncan-9099, )
Relation: harm-15237(roles-15238, brutus-14673, )
    <--> Relation: harm-9697(macbeth-9108, roles-9698, )
```

```
brutus-14673 <--> macbeth-9108

cassius-14679 <--> macbeth-9108
```

The same comparison, with plot units:

```
Relation: property-9665(duncan-9099, dead-9666, )
    <--> Relation: property-4534(dead-4535, caesar-4394, )
duncan-9099 <--> caesar-4394
dead-9666 <--> dead-4535

Derivative: object-9706(macduff-9105, )
    <--> Derivative: object-5015(cassius-4397, )
macbeth-9108 <--> cassius-4397
Relation: harm-9704(macbeth-9108, roles-9705, )
    <--> Relation: harm-5013(roles-5014, cassius-4397, )

macbeth-9108 <--> cassius-4397
```

4.3 Analysis

This project was essentially an experiment attempting to answer two questions:

1. Is the Structure Mapping Engine a robust means of generating analogical mappings between stories?

2. Do the mappings produced by the SME improve as expected when plot units are included?

4.3.1 SME as a Story Analogizer

The above outputs demonstrate that the answer to the first question is unambiguously "no" – at least not when the SME input is generated by the method described above. Even when comparing a story against itself, in which case the algorithm should have produced a mapping of each element in the story onto itself, the mappings were very sparse, and contained largely useless mappings. For instance, several of the mappings

contained `Derivatives` whose names were simply "object" (plus some ID number). Upon closer inspection, the elements being mapped here have no role in the story; in fact, it is not entirely clear why Genesis generated them in the first place. Whatever their purpose, they are certainly not part of any overarching relational structure central to the story.

In addition, the highest-ranked mappings between stories and themselves usually fail to map entities onto themselves correctly. These correspondences do sometimes appear in mappings with lower rankings, but never with more than a few story elements; in fact, in several cases, a few mappings contained correct correspondences, but for complementary subgraphs of the input. Clearly, the mappings the SME produces are woefully inadequate.

In the case of the inter-story comparison, the SME generally picked up one or two useful correspondences per mapping; it did not fail completely. For instance, it correctly noticed that Duncan and Caesar both become dead. However, it did not then extrapolate and associate the rest of the associated correspondences – for instance, that the killers should then be placed in correspondence, as well.

Part of the underlying problem becomes apparent if we examine the outputs that included mappings of items whose names contained `roles`. Closer inspection of these cases revealed that Genesis was not using quite the representational scheme expected. Instead of the object of a murder being, for example, Caesar, the object of a murder was a "roles" `Sequence` containing the murdered person. This sort of indirection seems to be common in Genesis representations, and seems to prevent the SME matching algorithm from properly mapping the objects of relations.

Several other factors may be contributing to the poor match quality:

- The SME algorithm had a number of bugs which needed to be fixed. It is

entirely possible that there are still subtle problems with the implementation that are actually preventing it from producing good mappings.

- The greedy algorithm, by its nature, throws out potentially legitimate mappings in the name of execution speed. It is possible that it is throwing out the best matches. This seems unlikely, however, given that adjusting the beam width for the beam search over a wide range of values appeared to have no impact on the analogy results.

- My translation scheme does not accurately reflect the semantics of Genesis and SME objects. This seems to be the most likely cause: misunderstandings (such as that mentioned above with respect to "role" objects) of what the various sorts of Genesis objects mean in different contexts. It is quite likely that the "one-size-fits-all" approach to `Sequence`s, for instance, is too naive, and likewise for the uniform treatment of relations and their objects.

- The SME's one-to-one mapping restriction seems rather draconian in some cases. We humans are certainly willing to make analogies that allow one-to-many mappings, particularly with respect to stories. This seemingly ill-suiited restriction may be part of what prevented the SME from drawing correct analogies from Genesis-derived inputs.

- It may be that the SME's graph-based algorithm is simply not well-suited to story information. This would be somewhat surprising, given the generality of the SME, but perhaps that generality is also a weakness in the sense that the SME refuses to let itself exploit domain-specific information or representations that would help it compare stories more effectively.

4.3.2 The Effects of Plot Units

One positive result of these experiments is the strong effect of plot units in improving the match quality in some cases. Plot units did not always matter much; in the inter-story comparison, for instance, they simply caused a different set of weak mappings. In the Macbeth self-comparison, however, plot units made the mappings far richer (the number of correspondences per mapping shot up drastically), and their accuracy and relevance improved as well. In fact, with plot units, around half of the important entities and relations in Macbeth were correctly matched against themselves in the top mapping. In no case did adding in plot units decrease the quality of the match. Thus, although the positive effects were not as strong as hoped for, we can still say with some confidence that plot units add potentially valuable information to the analogizing effort.

5 Contributions and Future Directions

Despite its shortcomings as a useful implemented system, this project makes several contributions to the story-understanding effort. First, it suggests that the SME may not be a good tool for the purpose of story comparison. Second, it highlights the significance of plot units as potentially crucial information when drawing analogies between stories; if plot units can cause even as unreliable a story analogizer as the SME to improve dramatically in some cases, they can certainly contribute to more sophisticated analogy efforts. Finally, it demonstrates the difficulty of attempting to translate between even superficially similar representation schemes; it is important, it seems, to make sure that one's story representations are properly matched with story-processing algorithms, and it is a mistake to assume that translation between separately developed systems with very different capabilities will be a simple task.

In the future, several improvements could make the system more viable:

- Hand-crafting a comparison algorithm for stories may be necessary; it is at least worth experimenting with complete replacements to the SME for the purpose of story analogies.

- A more sophisticated translation scheme, with more careful attention to the precise meanings of variations in representation on the Genesis side, could drastically improve the matching results.

- A possibility that I began to explore but did not have the opportunity to develop fully is using the SME on a much smaller scale: aligning plot units to suggest potential mappings, then using the SME to propose more mappings consistent with those suggested by the plot units. This more explicitly gives greater power to plot units, which would likely be an improvement, and would mean that the SME would be working on smaller inputs, on which it seems to perform better.

Ultimately, the SME may prove to be at least a useful auxiliary tool in the story analogizing process. Plot units, meanwhile, will almost certainly be essential. Leveraging these tools, computer programs may soon be able to draw complex inferences from past stories and bring them to bear on more current ones, bringing the computers one step closer to human-like intelligence.

References

Gentner, Dedre. "Structure-Mapping: A Theoretical Framework for Analogy." *Cognitive Science*, 7, 1995, pp.155-170.

Falkenhainer, Brian, Kenneth Forbus, and Dedre Gentner. "The Structure-Mapping Engine: Algorithm and Examples." *Artificial Intelligence*, 41, 1989, pp.1-63.