Modifying Explanations to Understand Stories*

Alex Kass
Yale University Department of Computer Science

ABSTRACT

We describe a system that learns new schemas by modifying old ones, in order to understand anomalous events in stories that it reads. We discuss how these schemas (called Explanation Patterns [Schank 86]) are structured in order to make them modifiable, and how the understanding process applies and modifies them. This model bridges the gap between two previous models of understanding, which were based on either application of prestored schemas or understanding-time inference chaining. By employing modifiable schemas, our model is more flexible than the former and more efficient than the latter.

Introduction

The crucial problem in story-understanding is inference. Inference is needed, among other things, to tie different sentences together, to fill in ellipses, and to resolve ambiguity. The story-understanding literature contains several models of the inference process, but for the purposes of this paper we can divide these models into two principal categories: One group operates via the application of prestored schemas; the other builds inference chains from scratch at understanding time. Both models have something to offer, but each has serious drawbacks as well. In this paper we argue that in order for a system to overcome these drawbacks it must be able to adapt its schemas to new situations. We discuss a new model (implemented in a computer program called SWALE) that is based on modifiable schemas, called Explanation Patterns (henceforth XPs).

Previous Work — Inference Chaining and Schema Application

The inference-chaining approach to story understanding is exemplified by Rieger's Conceptual Inferencer [Rieger 75] and Wilensky's PAM [Wilensky 78]. Rieger's system was completely bottom up; his program chained forward from each input sentence and noticed where the chains intersected. Although this simple model was able to generate many useful inferences it was too unconstrained. The combinatorial explosion of inferences caused the irrelevant to overwhelm the useful. Wilensky was able to constrain things somewhat by having knowledge of plans and goals guide the inference process. PAM would attempt to match inputs to known goals of the actor, and to backward chain to those goals if they couldn't be matched directly. Guiding the inference process in this way helps matters considerably but it remains the case that PAM had to do a lot of work to understand each story. Furthermore, because it didn't store the inference chains it built, it had to do as much work to understand a story it had seen a hundred times as it did to understand it the first time.

A response to the inference-chaining school is the schema-application approach, as practiced, for example, by Charniak's Ms. Malaprop [Charniak 72], and Cullingford's SAM [Cullingford 78]. These programs avoid most understanding-time inference by using prestored schemas that contain the expectations needed to understand the story. For these programs, the inference process is reduced to matching input from the story against a schema in memory. They are therefore very efficient at handling stories that closely match their schemas, but fail badly unless there is a very close match.

*This work is supported in part by the Air Force Office of Systems Research under contract 85-0343

Our Approach — Schema Modification

Given that schema-based programs tend to be brittle but efficient, while the inference-chainers are more flexible but less efficient, one might be tempted to propose that there are two different modes of story-understanding; that a complete model would combine SAM and PAM, using scripts when they were applicable, and resorting to inference-chaining when necessary. Such an either or system would be a good idea if stories were generally totally novel or totally old-hat. The fact is that most are neither; interesting stories we read are usually reminiscent of things we understand well, but not exactly like them — they are near misses and we don't want our model to have to understand these from scratch. Rather, we want to be able to tweak our old schemas to make them applicable. This way the understander learns new schemas incrementally when old ones fail. Old schemas serve as the starting points for creating the new, and the wheel does not need to be reinvented in order to build a slightly different schema. SAM-like scripts, however, are not good candidates for modification because they don't encode enough of the causal reasoning that went into building them to make it clear to a tweaking mechanism what modifications are reasonable to make. In order to make the tweaking idea work, we need to specify what these modifiable schemas should look like and what the process will be that modifies and applies them. These are exactly the goals of the SWALE project, which we will spend the rest of this paper describing.

XPs — Modifiable Schemas

XPs differ from scripts in that scripts are meant to provide an overall view of some group of events (such as a doctor's visit or a meal at a restaurant) while XPs are designed to be a detailed trace of the reasoning that was used to resolve a particular problem that could arise in such a group of events (such as the waiter not bringing your food). An XP is a set of beliefs and a set of belief-support relations which link the beliefs together in an inference network. The belief-support links specify which beliefs depend on which others, and what the type of the dependency is. This is what makes XPs modifiable. The belief supports indicate *why* a given belief is in the XP. This tells the modification process what the effect of deleting or changing beliefs will be.

Some of SWALE's XPs

The best way to give a feel for what XPs are like is to describe some the XPs we have equipped the SWALE program with. Consider the following story: "Swale, a successful 3-year old race horse, was found dead in his stall a week after winning the Belmont Stakes race." Most people who read this story find the death to be an anomalous event, which they feel the need to explain. What follows are some death-related XPs that we have built into SWALE, Some of these XPs are clearly relevant to Swale's death while others are connected only in a more fanciful way. Our goal is to have SWALE propose as many interesting explanations as possible by applying the XPs we have indexed in its XP library and tweaking them to create new variations of these XPs; we are much more interested in having the system develop interesting hypotheses than in having it avoid bad ones.

The Jim Fixx XP: Joggers jog a lot. Jogging results in physical exhaustion because jogging is a kind of exertion and exertion results in exhaustion. Physical exhaustion coupled with a heart defect can cause a heart attack. A heart attack can cause death.

The Janis Joplin XP: Being a star performer can result in stress because it is lonely at the top. Being stressed-out can result in a need to escape and relax. Needing to escape and relax can result in taking recreational drugs. Taking recreational drugs can result in an overdose. A drug overdose can result in death.

Too Much Sex XP: Too much sex can kill you.

Preoccupation XP: Being preoccupied about something can cause you to be inattentive. Being inattentive can result in walking into traffic. Walking into traffic can result in you being hit by a vehicle. Being hit by a vehicle can cause death.

Despondent Suicide XP: Thinking about something you want but that you don't have can make you despondent. Being despondent can result in suicide.

The idea is that when an anomaly is encountered while reading a story or having an experience in the real world, the understander uses features of the anomaly to retrieve XPs that might explain the anomaly. For any anomaly an understander encounters, there are four possible states of readiness its XP library might be in:

1. An XP can be retrieved from memory that applies perfectly to the anomaly. In this case it is easy

to explain the anomaly. It is a simple case of schema-application.

2. None of the XPs retrieved from memory apply directly, but some relevant XPs can be modified (we call this tweaking) to create new XPs that are applicable. This case is trickier, but it is also more important, since, at the end of the process the system has created and learned a new XP.

3. No XP is retrieved that can be tweaked to apply. An XP must be built from scratch out of the primitive inference rules. This is the kind of situation for which PAM was designed. Under our current model, understanders often give up on problems for which they are so unprepared unless the anomaly is quite important. Although a complete model should include a component to handle this case, the SWALE program currently does not.

4. The system does not even have the inference rules to allow it to build an appropriate XP. In this case the system simply is not equipped to understand the anomalous experience.

The SWALE Process Model

My claim is that most interesting experiences fall into category 2; we understand them by retrieving an existing XP and modifying it to fit the new situation. This is what the SWALE program does. Thus SWALE is both an understanding program and a learning program. Its actions are driven by the goal of discovering an explanation that will help it understand an anomalous event, and in doing so it learns new explanations, and stores them for future use. After an anomaly is detected the algorithm involves the following:

XP SEARCH: Search the XP library for an XP that may apply to the anomaly.

XP EVALUATION: Attempt to apply XPs. If successful then skip to XP INTEGRATION.

XP TWEAKING: If unable to apply XPs directly, attempt to tweak them into XPs that might apply better. If successful send tweaked XPs back to XP EVALUATION.

XP INTEGRATION: If results accepted, integrate into memory making appropriate generalizations.

The program is divided into three modules. The main driver is called the Accepter and is responsible for detecting anomalies, evaluating explanations, and generalization. One sub-system is devoted to retrieving XPs and another to tweaking them. David Leake is working on the Accepter, Chris Owens on the Retriever, and I am working on the Tweaker. Figure 1 presents a simple description of the SWALE architecture.

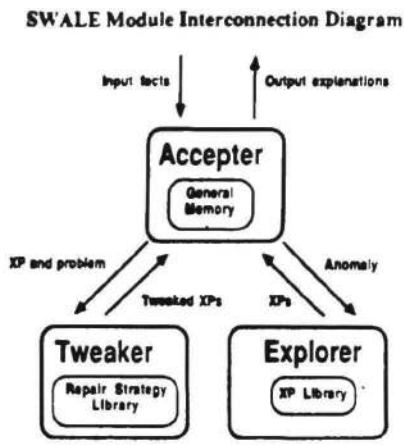

Figure 1: SWALE Module Interconnection diagram

Unfortunately, there isn't enough room here to describe the entire SWALE program. There are, of course, important issues that arise throughout the processing of the story. For example, the reader might wonder how SWALE notices anomalies, how it searches for relevant XPs and how it evaluates explanations. These are all interesting issues that we simply don't have room to touch on here — this paper is really only about how XPs that have been retrieved and evaluated as near misses can be modified. Some of the other issues are discussed in [Leake and Owens 86]. A somewhat more detailed (albeit out of date) discussion of the entire SWALE program appears in [Kass, Leake and Owens 86].

Tweaking XPs to Make Them Fit

When the Accepter rejects an XP as a near miss it passes the XP and the reason for rejection to the Tweaker. The goal of the Tweaker is to generate a new XP (or set of XPs) that might fit better. These are then sent back to the Accepter for re-evaluation.

The high level control structure of the Tweaker is very simple. There are two main sub-steps:

Strategy Retrieval: Retrieve XP REPAIR STRATEGIES from a library of such strategies maintained by the Tweaker. Retrieval of an appropriate set of XP

REPAIR STRATEGIES relies on using the XP FAILURE TYPE, created during the evaluation stage, as an index into the XP REPAIR STRATEGY library. Each XP REPAIR STRATEGY is stored in the library along with a failure pattern. The retrieval step essentially involves matching the XP FAILURE TYPE, against these patterns and collecting the strategies associated with patterns that successfully match. These are the strategies that will be applied in the application step.

Strategy Application: Each strategy is a program designed to map an XP that suffers from some problem to a set of modified XPs that don't suffer from the problem. Apply each retrieved strategy in turn, collecting and returning any resulting XPs. Of course, the details of what goes on during this phase of processing is completely determined by the nature of the XP REPAIR STRATEGY. Some of the XP FAILURE TYPEs generated in the current version of the program, and some of the associated XP REPAIR STRATEGIES are discussed below. This list not intended to be exhaustive by any means; we've just begun to build up SWALE's library of strategies. However, it should convey the basic form that these things take.

In a sense, the SWALE Tweaker represents the application to understanding of the same philosophy that Hammond's CHEF program [Hammond 84] applied to planning. CHEF uses goal-failure configurations to index plan-repair strategies; SWALE uses XP FAILURE TYPEs to index XP REPAIR STRATEGIES. Hammond's theoretical goal was a content-theory of plan repair; ours is a content-theory of explanation repair. Of course, the idea of using failures to index repair strategies traces back at least to HACKER [Sussman 75].

Some XP FAIURE TYPES

NORMATIVE-FILLER-VIOLATION: This failure indicates that the explanation hypothesizes that a role in an action description be filled by an actor who is not a member of the categories that the Accepter expects to fill the role.

When the **JIM FIXX XP** is retrieved it generates this failure description because it calls for SWALE be a jogger, but the program expects joggers to be humans.

SCRIPT-LINE-VIOLATION: This failure is similar, but not identical to the one above. It indicates that the explanation called for an actor to fill a role in a script that it is not valid for it to fill because one of the lines in the script is an action that the actor is actually incapable of performing.

When evaluating the **JANIS JOPLIN XP** this failure is generated because the XP hypothesizes that Swale was the actor in the recreational drugs script, but this script involves injecting oneself, which Swale is not capable of.

SCHEDULING-VIOLATION: This failure complains that an explanation hypothesizes that an action will occur at a particular time but the program has reason to believe that it should have happened at a different time (earlier or later) instead.

For example, the **TOO MUCH SEX XP** calls for Swale to be having sex around the time of his death. The program knows, however, that race horses are kept celibate until sent to stud after their racing days are over. It therefore generates this failure.

UNCONVINCING-SUPPORT-LINK: This failure is quite different from those above. The Accepter isn't complaining about any paritcular belief, but rather about the jump from one to another. The Accepter expects to have inference rules to back up each link in an XP. When a link isn't satisfactorily supported this failure is generated.

Some XP REPAIR STRATEGIES

Associated with each XP FAILURE TYPE is one or more XP REPAIR STRATEGIES. The best way to explain these is to describe some examples. We give some brief descriptions below. Examples of how some are used appears in the description of the SWALE run near the end of the paper.

SUBSTITUTE ALTERNATIVE THEME: This is a fairly general strategy. It is a candidate for fixing any **INVALID-ACTION** failure.

The notion here is to find out which line (or lines) in the inappropriate script was actually important in the XP (*ie.* which one supports other beliefs in the XP), and to search for a theme associated with the actor that can substitute. The new theme must be one that is appropriate for the current actor, and one which has the necessary line(s) in it. In other words, the XP carries within it the knowledge of why a particular belief is important to the XP, and this strategy uses this information to find another theme that can fit in in an analogous way.

SUBSTITUTE EQUIVALENT ACTION: This is quite like the above strategy, and can fix the same set of failures.

Sometimes there are no themes associated with the actor that can substitute for the action that he Accepter has found objectionable. In this case, another way to search for a substitution is to look at generalizations of the objectionable action, and then find other specifications of those generalizations. Any of these other specifications that support the inferences supported by the original action are candidate substitutions.

SUBSTITUTE ANTICIPATION: This rather specialized strategy applies to some **SCHEDULING-VIOLATION** failures when the action in question was expected to happen later than called for in the XP.

When this happens it is reasonable to entertain the notion that thinking about the future event might play a role in the explanation. Thus this strategy substitutes the belief that the actor was thinking about the event for the belief that the event actually occurred. Sometimes this makes sense, although often it doesn't. It is up to the Accepter, rather than the Tweaker, to decide.

FIND CONNECTING XP: This provides a way to fix **UNCONVINCING-SUPPORT-LINK** failures.

It attempts to repair such a problem by finding another XP that will connect the two beliefs in question. It works by calling the Explorer. If no XP can be found to connect the beliefs directly it will try a limited amount of causal chaining. After each chaining step it will call itself recursively, using the newly inferred belief. This is used to find connections between thinking about sex and death.

A Brief Description of a SWALE Run

SWALE is actually a running computer program. There is no room here to present the actual output from the system (which is rather verbose). The following is a brief paraphrase of a SWALE trace in which it develops some variations on explanations contained in its library when it is set to work on the story about Swale's death. This is intended to give the reader a rough idea of the way that SWALE's processing proceeds:

- Use routine search to find XPs concerning premature death in animals. Find **DEATH FROM ILLNESS XP**. This XP can't apply, since Swale wasn't sick. Because this is a severe failure do not try to tweak.
- Look for XPs indexed by unusual features of Swale. Racehorses are in top physical condition; death + top condition retrieves **JIM FIXX XP**. The Evaluator rejects **FIXX XP** because Swale can't be a jogger.
- Try to Tweak. The problem was a **DEFAULT-FILLER-VIOLATION**, so try **SUBSTITUTE ALTERNATIVE THEME**. Swale's known themes are horse-race and eat-oats. The horse-race theme is selected because it involves running, which was the aspect of jogging playing a role in the XP. The tweaked XP is: Since Swale had a heart defect, the exertion from running overtaxed his heart.
- Evaluate the new XP. It's reasonable, but since the heart defect can't be confirmed, continue looking for other XPs. Other strategies fail to find more XPs, so try folkloric explanations of death. Pull up the old wives' tale **TOO MUCH SEX XP**. The evaluator notices that racehorses aren't allowed to have sex while racing, but they do have a lot of sex when they retire to the stud farm. This is a **SCHEDULING-VIOLATION**
- Tweak. The tweaking strategy, **SUBSTITUTE ANTICIPATION** applies to this fault. Could Swale have died just from thinking about life on the stud farm?
- The new XP is unconvincing. There's no link from thinking about sex to death. Tweak.
- Fault is **UNCONVINCING-SUPPORT-LINK**, use the strategy **FIND CONNECTING XP**. Possible effects of thinking about sex are, excitement, and depression (if you're thinking about not having it). Distraction can be linked to death by two XPs, Excitement can cause death by heart-attack. Depression can cause death from suicide.
- Search continues but no more XPs are found. Each of the new explanations depends on conditions which can't be confirmed. Since the Fixx XP was the possibility located most directly by the Explorer, the tweaked version of Fixx is accepted as the most likely explanation. The causally-significant feature Fixx and Swale shared was that they did physical exertion. The XP is generalized to apply to actors who have an exertion theme and this is installed in memory for future use.

Some Explanations Created by SWALE

The Jim Fixx Reminding Explanation: Swale had a congenital heart defect. The exertion of running in horse races strained his heart and brought out the latent defect. He had a heart attack and died.

The Drug Overdose Explanation: Swale's owner was giving him drugs to improve his performance. He accidentally gave him an overdose, which killed him.

The Stud Farm Pair

SWALE's processing brings it to consider the idea that thinking about sex too much caused Swale's demise. It then attempts to imagine ways in which this might have occurred and develops the explanations that follow:

The Sexual Excitement Explanation: Swale was thinking about his future life on a stud farm. Since he was an excitable creature, thinking about the prospects proved to be too much strain for his heart. He had a heart attack and died.

The Despondent Suicide Explanation: Swale was thinking about his forced chastity during his racing career. He became despondent and killed himself.

Conclusion

Explanation Patterns represent frozen inference chains in a way that preserves the reasoning pattern used to develop them. This allows a model using XPs to bridge the gap between schema appliers and inference chainers. SWALE understands stories in terms of its preestablished schemas, but those schemas are not rigid in the way that scripts are. When a schema is a near miss, SWALE it in in order to make it apply. Of course, the further the story strays the more tweaking will be needed, so this model predicts that there will be a continuum of difficulty, from straight application through more and more major tweaking, to building the explanation from scratch.

SWALE is still in its formative stages but we are very excited by the preliminary results. The program possesses an important trait that previous understanders did not: When its knowledge structures fail, the program attempts to build new hypotheses for understanding the events and then stores these for future use. This is an important component of flexible, human-style understanding.

Acknowledgements

All of the other members of the SWALE team, Roger Schank, Chris Riesbeck, David Leake and Chris Owens helped develop the ideas in this paper. Chris Riesbeck and David Greenberg provided helpful comments on preliminary drafts.

References

[Charniak 72] Charniak, E., *Towards a Model of Children's Story Comprehension,* Technical Report 266, MIT Artificial Intelligence Lab, 1972.

[Cullingford 78] Cullingford, R., *Script Application: Computer Understanding of Newspaper Stories,* Ph.D. Thesis, Yale University, 1978. Research Report #116.

[Hammond 84] Hammond, K., *Indexing and Causality: The organization of plans and strategies in memory,* Technical Report 351, Yale University Department of Computer Science, December 1984.

[Kass, Leake and Owens 86] Kass, A. M. and Leake, D. B. and Owens, C. C., *SWALE: A Program that Explains,* 1986. In [Schank 86].

[Leake and Owens 86] Leake, D. B. and Owens, C. C., Oragnizing Memory for Explanation, *Proceedings of the Eighth Annual Conference of the Cognitive Science Society,* Cognitive Science Society, Lawrence Erlbaum Associates, 1986.

[Rieger 75] Rieger, C., Conceptual Memory and Inference, *Conceptual Information Processing,* North-Holland, Amsterdam, 1975.

[Schank 86] Schank, R.C., *Explanation Patterns: Understanding Mechanically and Creatively,* 1986. Book in press.

[Sussman 75] Sussman, G.J., *Artificial Intelligence Series,* Volume 1: *A computer model of skill acquisition,* American Elsevier, New York, 1975.

[Wilensky 78] Wilensky, R., *Understanding Goal-Based Stories,* Ph.D. Thesis, Yale University, 1978. Research Report #140.