

Story Creation from Heterogeneous Data Sources

Marat Fayzullin and V.S. Subrahmanian

Department of Computer Science, University of Maryland, A.V. Williams Building, College Park, MD 20742

Massimiliano Albanese, Carmine Cesarano and Antonio Picariello

Università di Napoli "Federico II", Dipartimento di Informatica e Sistemistica, Via Claudio 21, 80125 Napoli, Italy

Abstract. There are numerous applications where there is a need to rapidly infer a story about a given subject from a given set of potentially heterogeneous data sources. In this paper, we formally define a story to be a set of facts about a given subject that satisfies a “story length” constraint. An optimal story is a story that maximizes the value of an objective function measuring the goodness of a story. We present algorithms to extract stories from text and other data sources. We also develop an algorithm to compute an optimal story, as well as three heuristic algorithms to rapidly compute a suboptimal story. We run experiments to show that constructing stories can be efficiently performed and that the stories constructed by these heuristic algorithms are high quality stories. We have built a prototype STORY system based on our model — we briefly describe the prototype as well as one application in this paper.

Keywords: multimedia, heterogenous, databases, summarization, stories, storytelling, framework, algorithms

1. Introduction

There are thousands of applications where we wish to derive a *story* about a particular event, a person, an artifact, or a place. For example, consider a person walking through the archaeological site at Pompeii who encounters an painting labeled with a simple statement such as “Death of Pentheus”. Though a casual tourist may be satisfied with the knowledge that there is a beautiful painting depicting some unfamiliar event, a student or a person with a deeper interest in culture may be unsatisfied. She may want to know more about Pentheus, events surrounding his death, etc. In the same vein, consider a police officer who has to serve a warrant at a particular address. The police officer may want to get the quick story on this address — (i) who lives there? (ii) what can be said about these people? (iii) who lives in the neighborhood? (iv) what is their background? And so on.

What constitutes a story may vary dramatically from one example to another. In the case of Pompeii, users may be interested in cultural, historical, mythological, and artistic aspects of the entities about whom



© 2005 Kluwer Academic Publishers. Printed in the Netherlands.

stories are being woven. On the other hand, these aspects may not be of great interest to the police officer who instead may want to assess threats existing in the area as well as whether there are any other criminals to be arrested there. Thus, what goes into a story depends not only on the basic facts about the subject of interest, but also on the subject's domain and user's interests.

There are two other important aspects of stories: they must be *succinct* and they must allow the user to *explore* different facets of the story that are of interest to him or her. The police officer probably wants just the pertinent facts, not a long complex story about the genealogy of the residents of the house he is going to.

In this paper, we propose a formal framework for story creation. Furthermore, we present procedures to extract information about entities (persons, events, places, etc.) from the Web and other heterogeneous data sources and create stories from this information. The proposed story framework supports the goals of succinctness and exploration and creates stories with respect to three important parameters: the priority of the story content, the continuity of the story, and the non-repetition of facts covered by the story.

We have implemented our STORY system and applied it to the archaeological site of Pompeii. Our STORY system allows us to deliver stories over both wired and wireless networks to multiple heterogeneous devices such as computers, Internet terminals, and PDAs.

The organization and key contributions of this paper are as follows:

1. In Section 2, we present the concept of a *story schema* and *story instance*. Story schemas and instances can be applicable to diverse data sources. Informally speaking, a story is a set of facts obtained from a set of data sources.
2. Section 3 introduces optimal stories and the story computation problem and shows that optimal story computation is NP-complete.
3. In Section 4, we show how to extract data (i.e. facts) from heterogeneous data sources including text sources and relational/XML sources.
4. In Section 5, we develop the OptStory algorithm which is guaranteed to find an optimal story. As the optimal story computation problem is NP-complete, this algorithm is inefficient. We therefore develop three heuristic algorithms — OptStory⁺, GenStory, and DynStory.
5. In Section 6, we describe how an ordered collection of facts is rendered into an actual narrative in English or other language.

6. In Section 7, we present experiments that attempted to measure subjective qualities of the stories produced by our algorithms and discuss experimental results.

2. Story Schema and Instance

In this section, we describe concepts of a *story schema* and a *story instance*. We assume the existence of some set \mathcal{E} whose elements are called entities.

Intuitively, entities describe the objects of interest. In a museum, the objects of interest could be all the known people depicted by images or sculptures shown in the museum, as well as all people related to those people in some way. Additionally, the set of entities could include all places depicted. In the case of the police officer, entities of interest could include all people about whom the police have information. *Note that there is no need to explicitly enumerate this set of entities - they could, for example, be discovered by an algorithm seeking entities* (Callan and Mitamura, 2002).

DEFINITION 2.1 (ordinary attributes). *Suppose \mathcal{E} is a set of entities. We assume the existence of a universe \mathcal{A} whose elements are called ordinary attributes. Each attribute A has an associated domain $dom(A)$. We say that \mathcal{A} is a set of ordinary attributes associated with the set \mathcal{E} of entities if $\mathcal{E} \subseteq \bigcup_{A \in \mathcal{A}} dom(A)$.*

The above requirement merely ensures that each entity can be characterized by the values of ordinary attributes.

Intuitively, the story about Pentheus may have many ordinary attributes. One ordinary attribute might be `mother` - the domain of `mother` could be the set of all alphabetical strings. The *value* of this attribute could be the string “*Agave*”. Attributes do not need to be elementary types. An attribute such as `persons` could have as its domain, the powerset of the set of people known in Greek mythology. In the Pentheus example, the value could be $\{Pentheus, Agave, Maenads\}$ - note that Maenads is not one person, but rather a collective name for a group of people.

Notice that it is entirely possible that the value of an attribute could be an entity by itself and there may be a story about this entity which involves other entities as well. In the Pentheus example, Agave (his mother) could be an entity about whom many attributes have known values. However, the attribute `occupation` for Pentheus may have the

value “king” which is in $\text{dom}(\text{occupation})$ but is not an entity (so it has no attributes, etc.).

There are many cases where we may have multiple values for an attribute and want to generalize them into one.

DEFINITION 2.2 (generalization function).

Suppose A is an ordinary attribute. Then a generalization function for attribute A is a mapping Γ_A from $2^{\text{dom}(A)}$ to $\text{dom}(A)$.

For example, suppose we have a domain called `occupation` whose domain is the set of all strings. A generalization function Γ_{occ} may map a set of strings of the form “ X was King of . . .” to a single string “king.” This is just one example of a generalization function - many more are possible.

In the above discussion, attributes have invariant values. However, there are many situations where attributes may have time-varying values. For example, Pope Paul III may have an `occupation` attribute with the value `Cardinal` from 1493 to 1533 and `Pope` from 1534 to 1549 - we have ignored exact dates of ascension here and just approximated the years. In order to express this kind of information, we introduce the concept of a time-varying attribute.

DEFINITION 2.3 (time-varying attribute).

A time-varying attribute is a pair $(A, \text{dom}(A))$ where A is the name of the attribute and $\text{dom}(A)$ is the domain of values for the attribute.

A time-varying attribute looks just like an ordinary attribute. However, a value for a time varying attribute associates an interval.

DEFINITION 2.4 (timevalue). *A timevalue for a time-varying attribute $(A, \text{dom}(A))$ is a set of triples (v_i, L_i, U_i) where $v_i \in \text{dom}(A)$ and L_i, U_i are either integers or the special symbol \perp (denoting unknown). A timevalue is fully specified iff there is no triple of either the form (v_1, \perp, U_i) or (v_i, L_i, \perp) or (v_i, \perp, \perp) in it.*

Intuitively, if an object has a time-varying attribute A with a timevalue of $\{(v_1, 15, 20), (v_2, 25, 30)\}$ this means that attribute A has value v_1 between times 15 and 20 and value v_2 between times 25 and 30. In the case of Pope Paul III, the timevalue of `occupation` is given by $\{(\text{Pope}, 1534, 1549), (\text{Cardinal}, 1493, 1533)\}$.

NOTE 2.1. *Though we could have allowed timevalues to be more complex and expressive, the reader will see later (Note 3.1) that doing so creates a huge additional burden on the system implementor which we prefer to avoid.*

DEFINITION 2.5 (consistent timevalue). *A timevalue tv for a time-varying attribute $(A, dom(A))$ is consistent iff there is no pair $(v_1, L_1, U_1), (v_2, L_2, U_2)$ in tv such that $v_1 \neq v_2$ and $L_1, U_1, L_2, U_2 \neq \perp$ and such that the intervals $[L_1, U_1] \cap [L_2, U_2]$ intersect.*

Intuitively, consistency of a timevalue ensures that the attribute does not have two distinct values at the same time (e.g. Pope Paul III could not be both Pope and a cardinal at the same time). Thus, the timevalue $\{(Pope, 1534, 1549), (Cardinal, 1493, 1533)\}$ for Pope Paul III's occupation attribute is consistent.

Note, however, that had we wanted to allow a person to have multiple occupations at the same time, we could simply have defined the domain of `occupation` to be the *powerset* of the set of all strings rather than the set of all strings.

NOTE 2.2. *Throughout the rest of this paper, we will abuse notation and use the term attribute to refer to both ordinary and time-varying attributes. The context will determine the usage.*

Just as we have defined generalization functions for ordinary attributes, we can also define generalization functions for time-varying attributes.

DEFINITION 2.6 (generalization function (cntd.)). *Suppose $(A, dom(A))$ is a time varying attribute. A generalization function for A is a mapping Γ_A from a timevalue for attribute $(A, dom(A))$ to a singleton timevalue for attribute $(A, dom(A))$ such that if $\Gamma_A(X) = \{(v, L, U)\}$ then $[L, U] \subseteq [\min_{(v_i, L_i, U_i) \in X} L_i, \max_{(v_i, L_i, U_i) \in X} U_i]$ and there exists $(v_j, L_j, U_j) \in X$ such that $[L, U] \cap [L_j, U_j] \neq \emptyset$.*

For example, a generalization function may map the set of time values $\{("Bishop of Massa", 1538, 1552), ("Bishop of Nice", 1533, 1535)\}$ to the singleton timevalue $\{("Bishop", 1533, 1552)\}$.

Note that if the definition of generalization function for time varying attributes did not require that $\exists (v_j, L_j, U_j) \in X [L, U] \cap [L_j, U_j] \neq \emptyset$ then we would have a problem. The generalization function could, for example, return $\{("Bishop", 1536, 1537)\}$ even though its input said nothing about the time interval from 1536 to 1537.

The concept of a story schema below specifies the entities of interest, and the attributes of interest for a given story application.

DEFINITION 2.7 (story schema). *A story schema consists of a pair $(\mathcal{E}, \mathcal{A})$ where \mathcal{E} is a set of entities and \mathcal{A} is a set of attributes associated with \mathcal{E} . We will use \mathcal{A}^o (resp. \mathcal{A}^{tv}) to denote the ordinary (resp. time-varying) attributes in \mathcal{A} .*

For example, if the *Sito Archaeologico di Pompei* wishes to allow visitors to learn everything about archaeological ruins at Pompeii, then the set of entities could be defined as follows: (i) the set of all objects in Pompeii that are of interest (including paintings, sculptures, etc), plus (ii) the objects and events depicted in those paintings, plus (iii) any entities related to entities in the previous two categories. Clearly, the museum workers can only define the first two items in the above list, while the story creating system will automatically derive all other entities using the third criterion.

DEFINITION 2.8 (story instance). *A story instance w.r.t. story schema $(\mathcal{E}, \mathcal{A})$ is a partial mapping \mathcal{I} which takes an entity $e \in \mathcal{E}$ and an attribute $A \in \mathcal{A}$ and returns as output, a V of value $v \in \text{dom}(A)$ when A is an ordinary attribute, and a timevalue $\{(v, L, U) | v \in \text{dom}(A)\}$ when A is a time-varying attribute.*

We use the notation $\mathcal{I}(e, A) = \perp$ to indicate that $\mathcal{I}(e, A)$ is undefined. Attributes like *daughter* may be set valued. For example, Agamemnon had several daughters — amongst them Elektra and Iphigenia. In this case, the attribute *daughter* should have a set valued domain as its output type — and thus, an instance would say that the daughter attribute of the entity Agamemnon has the value $\{Elektra, Iphigenia\}$. Thus, requiring that each attribute have at most one value per entity leads to no loss of generality as the attribute can assume set values.

EXAMPLE 2.1 (Pentheus painting). *The table below shows other entities related to a Greek mythological character called Pentheus who is depicted in a stunning painting in Pompeii.*

<i>Entity</i>	<i>Attribute</i>	<i>Value</i>
<i>Bacchus</i>	<i>occupation</i>	<i>god</i>
	<i>enemy</i>	$\{(Pentheus, \perp, \perp)\}$
	<i>friends</i>	$\{(Maenads, \perp, \perp)\}$
<i>Maenads</i>	<i>occupation</i>	$\{(priestess, \perp, \perp)\}$
	<i>friends</i>	$\{(Bacchus, \perp, \perp)\}$

Note that it is often convenient to think of a story instance as a set of entity-attribute-value triples (possibly with additional associated time intervals). We will often abuse notation and switch between these two representations.

3. Story Computation Problem

The notion of a story instance does not allow us to include generalized tuples or to handle conflicts. To introduce these features, we will need to define several specialized instances.

In this section, we first define concepts of a valid instance and a full instance based on a set of data sources. Intuitively, these concepts are used to collect all facts reported by a set of data sources. We then define a closed instance by allowing generalization. However, given any topic or entity, there may be many stories that can be associated with that topic or entity. To address this, we define continuity, priority, and non-repetition criteria to test if one story is better than another. Later, in Section 5, we will present several algorithms for story computation.

3.1. VALID AND FULL INSTANCES

In order to create a story from a story schema, one may need to access a variety of sources. In the case of Pentheus, we may need to access electronic Greek texts to find out more about him. Let us assume that our data sources have an associated application program interface (this is a reasonable assumption as most commercial programs do have APIs). The *source access table* describes how to extract an attribute's value using a source's API.

DEFINITION 3.1 (source access table). *A source access tuple is a triple $(A, s, f_{A,s})$ where A is an attribute name, s is a data source, and $f_{A,s}$ is a partial function (body of software code) that maps objects to values in $\text{dom}(A)$ when A is an ordinary attribute, and to time values over $\text{dom}(A)$ when A is a time-varying attribute. A source access table is a finite set of source access tuples.*

The source access table does not, of course, need to be populated with a function for each source and each attribute. Some sources may provide some information, while others may not. The functions $f_{A,s}$ are *partial functions* because some sources may not have information about Pentheus. When implementing the STORY system, we created several f functions capable of extracting data from relational tables, XML hierarchies, and HTML documents returned by the Google search engine¹. In Section 5 of this paper, we will describe in detail our algorithms to extract entity-attribute-value triples from text (e.g. Web)

¹ Our algorithm can be used in conjunction with any algorithm for topic discovery (Agrawal et.al., 2000) and can be applied to any corpus of documents whatsoever, rather than applying them to web documents whose accuracy is questionable.

sources. We will also outline algorithms to extract entity-attribute-value triples from relational and XML sources.

NOTE 3.1. The developer of an application requiring stories about a certain domain needs to specify the functions $f_{A,s}$ in the source access table. Such a function must return timevalues when A is a time-varying attribute. This can be quite difficult. For instance, determining when Pentheus was killed from a text document is a nontrivial task. Had we allowed timevalues to be more general (e.g. to say Pentheus was killed after some other event, or to say Pentheus was killed within 5 years of yet another event), then the functions $f_{A,s}$ would need to infer this even more complex information from textual sources. This incredibly challenging problem is beyond the scope of this paper.

DEFINITION 3.2 (valid instance). *Suppose $(\mathcal{E}, \mathcal{A})$ is a story schema, SAT is a source access table, and \mathcal{I} is an instance. \mathcal{I} is said to be valid w.r.t. SAT iff for every entity $e \in \mathcal{E}$ and every attribute $A \in \mathcal{A}$, if $\mathcal{I}(e, A)$ is defined, then there is a triple of the form $(A, s, f_{A,s})$ in SAT such that $f_{A,s}(e) = \mathcal{I}(e, A)$.*

Intuitively, the above definition says that an instance is valid w.r.t. some source access table if every fact (i.e. every assignment of value to an attribute for an entity) is supported by at least one source. *Note that different sources may disagree on the value of a given attribute for a given entity.* For instance, one source may say Pentheus' mother is Agave, while another may say it is Hera. A valid instance reflects one way of resolving such a conflict. We now define the concept of a *full instance* that collects together the set of all values for attribute A of entity e from all sources.

DEFINITION 3.3 (full instance). *Suppose $(\mathcal{E}, \mathcal{A})$ is a story schema and SAT is a source access table. Suppose \mathcal{I} is an instance w.r.t. $(\mathcal{E}, \mathcal{A}')$ where the attributes in \mathcal{A}' are the same as the attributes in \mathcal{A} with one difference - if an attribute $A \in \mathcal{A}$ has $\text{dom}(A) = 2^{2^S}$, then the corresponding attribute $A' \in \mathcal{A}'$ has $\text{dom}(A') = \text{dom}(A)$. Otherwise, $\text{dom}(A') = 2^{\text{dom}(A)}$, i.e. the powerset of the original domain. \mathcal{I} is said to be the full instance w.r.t. $(\mathcal{E}, \mathcal{A})$ and SAT iff for all entities $e \in \mathcal{E}$ and attributes $A \in \mathcal{A}$,*

$$\mathcal{I}(e, A) = \begin{cases} \bigcup_{(A,s,f_{A,s}) \in \text{SAT}} f_{A,s}(e) & \text{if } \text{dom}(A) = 2^{2^S} \\ \{f_{A,s}(e) \mid \forall s (A, s, f_{A,s}) \in \text{SAT}\} & \text{otherwise} \end{cases}.$$

Intuitively, the above definition says an instance is full when it accumulates all the facts reported by various sources, independently of

whether these facts are conflicting or not. We will describe how conflicts may be resolved later on in this paper (Definition 3.6).

3.2. STORIES

In this section, we define how to generalize information contained in full instances, resolve conflicts in this information, and create stories out of instances. A generalized story schema is a story schema, together with an equivalence relation on attribute domains and a generalization function.

DEFINITION 3.4 (generalized story schema).

A generalized story schema is a quadruple $(\mathcal{E}, \mathcal{A}, \sim, \mathcal{G})$ where $(\mathcal{E}, \mathcal{A})$ is a story schema, \sim is a mapping which associates an equivalence relation on $\text{dom}(A)$ with each attribute $A \in \mathcal{A}$ and \mathcal{G} is a mapping which assigns, to each attribute $A \in \mathcal{A}$, a generalization function Γ_A for attribute A .

Intuitively, a generalized story schema consists of a regular story schema, a function that associates an equivalence relation with each attribute domain and a function that associates a generalization function with each attribute domain. An equivalence relation on the domain $\text{dom}(A)$ of attribute A specifies when certain values in the domain are considered equivalent. For example, we may consider string values “king” and “monarch” to be equivalent in $\text{dom}(\text{occupation})$. For a time varying attribute, we may consider (“king”, L , U) and (“monarch”, L' , U') to be equivalent independently of whether $L = L' \wedge U = U'$ is true or not. Likewise, in the example of Pope Paul III, the equivalence relationship may say that the triplet (“Bishop of ...”, $-$, $-$) is always equivalent to other triplets of the form (“Bishop of ...”, $-$, $-$) independently of whether the bishops involved governed different places. Our system uses WordNet (Miller, 1995) to infer equivalence relationships between terms.

The definition of a closed instance below takes a full instance associated with a source access table and closes it up so that generalization information can be included.

DEFINITION 3.5 (closed instance). Suppose $(\mathcal{E}, \mathcal{A}, \sim, \mathcal{G})$ is a generalized story schema and \mathcal{I} is the full instance w.r.t. $(\mathcal{E}, \mathcal{A})$. The closed instance w.r.t. a source access table SAT and generalized story schema $(\mathcal{E}, \mathcal{A}, \sim, \mathcal{G})$ is defined as $\mathcal{I}'(e, A) = \mathcal{I}(e, A) \cup \{\Gamma_A(X') \mid X' \text{ is a } \sim_A\text{-equivalence class of } \mathcal{I}(e, A)\}$.

Intuitively, here is how we find the closed instance associated with a given source access table and a given generalized story schema. For each entity e and each attribute A of the entity:

1. We first compute the set $\mathcal{I}(e, A)$ where \mathcal{I} is the full instance associated with our source access table.
2. We then split $\mathcal{I}(e, A)$ into equivalence classes using the equivalence relation $\sim (A)$ on $\text{dom}(A)$. Suppose the equivalence classes thus generated are X_1, \dots, X_n .
3. For each equivalence class X_i , we compute $\Gamma_A(X_i)$ - this is the generalization of the equivalence class X_i using the generalization function Γ_A associated with attribute A . Suppose $\Gamma_A(X_i) = v_i$.
4. We insert the tuple (e, A, v_i) into the full instance.

This process is repeated for all entities e and all attributes A . After all tuples of the form shown above inserted into the full instance, it becomes the closed instance.

A story cannot be defined based on a full instance alone. In the real world, the “full story” about any single person or event is likely to be very complex and involve a large amount of unimportant minutiae. For example, consider the story of Pope Paul III. Depending on what items about Pope Paul III are considered important, we may choose to merely say that he served as a bishop from 1538 to 1556 and ignore the details. However, the full instance associated with Pope Paul III may not explicitly say this - rather it might state (as in our example) that he was a bishop of this place for some time, that place for another time period, and so on. Generalization is needed for this.

Note that so far we have not tried to resolve possible conflicts between attribute values obtained from different sources. However, such conflicts need to be resolved before we can create a story. In other words, suppose \mathcal{I}' is a closed instance. Whenever $\mathcal{I}'(e, A)$ is of cardinality two or more, some mechanism is required to get rid of all but one member in $\mathcal{I}'(e, A)$.

DEFINITION 3.6 (conflict management policy).

Given an attribute A such that $\text{dom}(A) = 2^{2^S}$, the conflict management policy χ_A is a mapping from $\text{dom}(A)$ to $\text{dom}(A)$ such that $\chi(X) \subseteq X$. For any other attribute A , χ_A is a mapping from $2^{\text{dom}(A)}$ to $\text{dom}(A)$ such that $\chi(X) \in X$.

There is an extensive literature (Jamil and Lakshmanan, 1995) on conflict resolution whose results can be directly plugged in as conflict management policies — three of these are shown below.

1. **Temporal conflict resolution.** Suppose different data sources provide different values v_1, \dots, v_n for $\mathcal{I}(e, A)$. Suppose value v_i was

inserted into the data source at time t_i . In this case, we pick the value v_i such that $t_i = \max\{t_1, \dots, t_n\}$. If multiple such i 's exist, one is selected randomly.

2. **Source based conflict resolution.** The developer of a story may assign a *credibility* c_i to each source s_i that provides a value v_i for attribute A of entity e . This strategy picks value v_i such that $c_i = \max\{c_1, \dots, c_n\}$. If multiple such i 's exist, one is selected randomly.
3. **Voting based conflict resolution.** Each value v_i returned by at least one data source has a *vote*, $\text{vote}(v_i)$. $\text{vote}(v_i)$ is the number of sources that return value v_i . In this case, this conflict resolution strategy returns the value with the highest vote. If multiple v_i 's have the same highest vote, one is picked randomly and returned.

These are just three example strategies. It is easy to pick hybrids of these strategies as well. For example, we could first find the values for $\mathcal{I}(e, A)$ with the highest votes and then choose the one which is most recent (temporal). A deconflicted instance is one from which conflicts have been removed.

DEFINITION 3.7 (deconflicted instance). *Suppose $(\mathcal{E}, \mathcal{A}, \sim, \mathcal{G})$ is a generalized story schema and \mathcal{I}' is the closed instance w.r.t. $(\mathcal{E}, \mathcal{A})$. The deconflicted instance w.r.t. a source access table SAT , generalized story schema $(\mathcal{E}, \mathcal{A}, \sim, \mathcal{G})$, and conflict management policy χ is the instance \mathcal{I}^\sharp such that for all entities $e \in \mathcal{E}$ and all attributes $A \in \mathcal{A}$ if $\mathcal{I}^\sharp(e, A) \neq \perp$ then $\mathcal{I}^\sharp(e, A) = \chi(\mathcal{I}'(e, A))$.*

Note that finding any arbitrary strong or deconflicted instance is not enough. The reason is a technical one. The instance \mathcal{I}_{null} which is undefined for all $\mathcal{I}_{null}(e, A)$ has no conflicts - however it is not very useful as it has no information in it.

DEFINITION 3.8 (story). *Suppose \mathcal{I} is a closed instance w.r.t. a generalized story schema $(\mathcal{E}, \mathcal{A}, \sim, \mathcal{G})$ and a source access table SAT , and $e \in \mathcal{E}$ is an entity. Then a story $\sigma(e, \mathcal{I})$ of size k , is a sequence of attribute-value pairs $\langle A_1, v_1 \rangle, \dots, \langle A_k, v_k \rangle$ such that for all $1 \leq i \leq k$, $A_i \in \mathcal{A}$ and $v_i = \mathcal{I}(e_i, A_i)$.*

A deconflicted story w.r.t. a given conflict management policy is a sequence of attribute-value pairs $\langle A_1, v_1 \rangle, \dots, \langle A_k, v_k \rangle$ such that for all $1 \leq i \leq k$, $A_i \in \mathcal{A}$ and $v_i = \mathcal{I}^\sharp(e_i, A_i)$ where \mathcal{I}^\sharp is the deconflicted instance w.r.t. χ .

Note that the above definition of a story only lists the essential facts in a story. Our STORY system presents these facts in English (a Spanish

version also exists) to the user in one of two ways: if the fact was derived from a text document, then the sentence from which the fact was extracted is output in English. If the fact was derived from a relational or XML source, then a template is used to output the fact. We have approximately 400 templates currently in our system.

NOTE 3.2. *Throughout the rest of this paper, we will use the word “story” to refer to both ordinary and deconflicted stories.*

3.3. OPTIMAL STORIES

There are good stories and bad stories even when they are about the same topic and even derived from the same instance. So, what makes a story good?

First of all, the facts included in the story have to be *relevant* to the user. For example, the fact that Pentheus’ mother was Agave is probably more important than the length of Pentheus’ big toe. Thus, the first fact is better be told to the user in the beginning of the story while the second fact can be included at the end or omitted altogether.

Secondly, the story has to be continuous by delivering facts in the order expected by the user. If the facts occur over some period of time, it is logical to tell them in the order of occurrence. But even for such basic facts as the place of birth or parents’ names, there is a certain customary order of delivery (i.e. “X has been born in P from Y and Z”). Violating this order will make story less comprehensible.

Finally, we would not want to repeat same or similar facts again and again in the same story. Redundancy is *not* a virtue when it comes to storytelling.

To help us choose stories that are “better” than others from the universe of possibilities, let us define the story evaluation function:

DEFINITION 3.9 (story evaluation function).

Suppose \mathcal{S} is the set of all possible stories about some entity e w.r.t. the same schema and source access table. The story evaluation function $eval(s)$ takes a story s and returns a real value in the $[0, 1]$ range that measures how good s is, with higher values corresponding to better stories.

The reader will note immediately that there are many ways of defining the evaluation function, not limited to the three general criteria outlined above. Our story creation algorithms can work with *any* evaluation function. For example, (Fayzullin, 2005) develop algorithms to compute the value of a video summary based on priorities of frames in

the summary, continuity of the summary, and lack of repetition. Similar criteria can be used to create such an objective function.

PROBLEM 1 (optimal story computation). *Given a closed instance \mathcal{I} , a positive integer k , and an entity $e \in \mathcal{E}$ as input, find a story $\sigma(e, \mathcal{I})$ of size $\leq k$ that maximizes the value of a given evaluation function. In this case, $\sigma(e, \mathcal{I})$ is called an optimal story.*

THEOREM 3.1. *Given all the parameters listed in the above problem statement, and given a story S , determining if S is optimal is an NP-complete problem.*

Membership in NP will be proved by the **OptStory** algorithm presented in Section 5. NP-hardness is proved by a straightforward reduction of the knapsack problem to that of stories.²

4. Attribute Extraction

Section 3 has formally defined a *full instance* and how it can be obtained using the SAT table, but avoided discussion of the actual ways to extract attribute values from heterogeneous data sources. We show how to do this in detail for text sources, and briefly describe how to do this with XML and relational sources.

4.1. ATTRIBUTE EXTRACTION FROM TEXT SOURCES

The Text Attribute Extraction (TAE) algorithm to extract attributes from text sources shown below takes as input, a domain name (e.g. “Greek Mythology”) and a set of sources (these can, for example, be obtained by a straight Google search rather than enumerating them explicitly).

```

Algorithm TAE(Domain, Sources)
  Domain is the domain of knowledge
  Sources is a set of document sources
begin
   $\mathcal{D} := \text{GetDocuments}(\text{Domain}, \text{DataSources})$ 
   $\mathcal{E} := \emptyset$ 
   $\mathcal{T} := \emptyset$ 
  for each document  $D \in \mathcal{D}$  do
     $T := \text{Tokenize}(D)$ 
     $\mathcal{E} := \mathcal{E} \cup \text{RecognizeNamedEntities}(T)$ 

```

² Intuitively, given an instance of the knapsack problem involving a knapsack of capacity C and objects o_1, \dots, o_k of weights w_1, \dots, w_k and profits p_1, \dots, p_k respectively, we can consider each o_i to be a fact with the same weights and profits.

```

    T := DisambiguatePartOfSpeech(T)
    T := RecognizeCompoundForms(T)
    T := ResolvePronouns(T)
    T := T ∪ {T}
end for
for each document T ∈ T do
  for each rule R ∈ SemRules
    for each S ∈ MatchSequences(T, R.Tail) do
      (e, a, v) := ExtractPattern(S, R.Head)
      if exists E ∈ E such that E.Name = e then
        InsertIntoDatabase(e, a, v)
      end if
    end for
  end for
end for
end

```

The TAE algorithm assumes the existence of various subroutines. We describe each of these below.

1. *GetDocuments*: This function retrieves all domain-specific documents from the specified data sources. We implemented *GetDocuments* using the *keyword spices* approach (Oyama, 2004).
2. *Tokenize*: Each relevant document $D \in \mathcal{D}$ is then tokenized, i.e. fragmented into units corresponding to single words or punctuation marks, and each token is tagged with its corresponding part of speech. Thus, a token can be defined as a pair $(Word, PartOfSpeech)$. We use WordNet (Miller, 1995) for part of speech tagging.
3. *RecognizeNamedEntities*: A *named entities recognition* algorithm (Callan and Mitamura, 2002) is applied in order to identify and classify named entities (people, organizations, places, etc.) that appear in \mathcal{D} . This allows us to find entities of interest inside the domain (e.g. all the Greek Mythology characters, all the people and organizations involved in nuclear research activities, etc.) and extract data about these entities in advance. A *named entity* can be defined as a tuple $(Name, Class)$, where *Class* can be (i) person's name (PN), (ii) geographic location (LN), (iii) organization (ON), (iv) date/time (DT), (v) unclassified (NC), (vi) not an entity (NaE). In our implementation, we developed our own named entities recognition algorithm — this is described in Section 4.1.1. The set \mathcal{E} consists of all recognized named entities (repetitions are removed).
4. *DisambiguatePartsofSpeech*: A *part of speech disambiguation* algorithm is applied to resolve situations in which a word has been tagged with more than a single part of speech. We apply simple heuristic rules to resolve this issue, but more sophisticated algorithms have been proposed in the literature (Rosso et.al., 2003).

5. *RecognizeCompoundForms*: Next, we use WordNet to *identify compound forms*, such as verbs followed by prepositions or adverbs (eg. “take off”, “get out”), and merge their tokens together.
6. *ResolvePronouns*: Finally, we *resolve pronouns* by discovering which entities previously named in a document the pronouns refer to. Many sophisticated algorithms (Oliverira and Wazlawick, 1998) have been proposed for this task.

At this stage of the TAE algorithm, we get unambiguous versions of the source documents.

We now extract data by applying a set *SemRules* of semantic rules that allow us to deduce (*Entity, Attribute, Value*) triples from sentences. Semantic rules have the form $Head \leftarrow Tail$, where *Tail* is a condition to be evaluated on a sequence of tokens from the tokenized document. If a sequence satisfies this condition then the *Head* part of the rule tells us how to extract one or more triples from this sequence. An example rule is given below.

$$EAV\$0\$1\$2 \leftarrow ent.\#\$0\&poss.\ case\#'\ s\&noun\#\$1\&verb\#be\&ent.\#\$2$$

This rule says that in a sequence made up of an entity name, a possessive case, a noun, a verb “be”, another entity name, the (*Entity, Attribute, Value*) triple is built of the first entity name, the noun and the second entity name. For each triple we also try to determine the *time interval* where this triple is valid (if this information is present in the text) and the *pedigree* (source and last update time). This information is needed to manage conflicting data. Our system has several hundred such semantic rules in it.

EXAMPLE 4.1. Consider a piece of text $D = \text{“West Palm Beach’s mayor is T. J. Smith”}$. After tokenization, the named entity recognition (see also example 4.2) and the other pre-processing steps, we obtain $T = \langle (\text{“West Palm Beach”}, \text{entity:LN}), (\text{“’s”}, \text{possessive case}), (\text{“mayor”}, \text{noun}), (\text{“is”}, \text{verb}), (\text{“T. J. Smith”}, \text{entity:PN}) \rangle$. The LN annotation tells us that West Palm Beach is a place and the PN annotation tells us T.J. Smith is a person. The tokenized text matches the tail of the rule given above, so the following (e,a,v) triple is extracted according the head of the rule:

$$(e, a, v) = (\text{WestPalmBeach}, \text{mayor}, \text{T.J.Smith})$$

4.1.1. Named Entities Recognition

We have developed a named entities recognition algorithm which recognizes and classifies *named entities* in a document. A significant amount

of work has been done on this topic. Some authors propose knowledge-based approaches (Callan and Mitamura, 2002) while others favor the use of statistical models such as Hidden Markov Models (GuoDong and Jian, 2003).

In this paper, we propose the Tokenized-HMM algorithm (tHMM) which uses two phases: in the first phase we use HMMs to identify and classify all tokens that are part of a named entity. In the second phase, named entities are classified based on the classification of their tokens. Tokens associated with a single named entity may have different classifications – we resolve any conflicting classifications using three alternative approaches that we present in this paper.

Simply stated, an HMM has finite set of states, each of which has an associated probability distribution. Transitions among the states are governed by a set of probabilities called transition probabilities. Each state generates an output based on the associated probability distribution. The observer can only see the outcome, not the state.

In our case the set of possible states coincides with the set of possible named entities classes mentioned earlier - , i.e. the set $\{PN, LN, ON, DT, NC, NaE\}$. As the document is read, the HMM receives tokens from the document which may cause state changes to occur. Our algorithm considers not only the current token, but also the features of the previous and the next tokens: this has greatly improved the accuracy of recognition, since it takes into account some contextual information. Let us assume the following notation:

- N is the number of states;
- $V = \{v_1, \dots, v_M\}$ is a finite set of M observation symbols;
- S_t is the state of the system at time t ;
- $\pi = \{\pi_1, \dots, \pi_N\}$ is the initial state vector with $\pi_i = P(i_1 = i) \forall i \in \{1, \dots, N\}$; in other words π_i is the probability that the system is in the state i at time 1, i.e. when the first token is observed;
- $A = \{a_{ij}\}$ is the the state transition probabilities matrix, with $a_{ij} = P(i_{t+1} = j | i_t = i)$;
- $B = \{b_j(k)\}$ is the probability distribution of observation symbols, with $b_j(k) = P(v_k \text{ in } t | i_t = j)$;
- O_t is the observed symbol at time t , that in our case consists of the observable features of the t-th token and, eventually, of the surrounding tokens.

A Hidden Markov Model is a triple $\lambda(A, B, \pi)$. We wish to identify the most likely state sequence corresponding to the observed features:

PROBLEM 2. Given an HMM $\lambda(A, B, \pi)$ how do we choose a state sequence $S = S_1, S_2, \dots, S_T$ so that the joint probability $P(O, S|\lambda)$ of S and the observation sequence $O = O_1, O_2, \dots, O_T$ is maximized w.r.t. the model.

```

Algorithm tHMM( $T$ )
   $T$  is a tokenized document
begin
   $\mathcal{E} := \emptyset$ 
   $O := ExtractFeatures(T)$ 
   $S := Viterbi(O, \lambda(A, B, \pi))$ 
  for each  $j, m$  such that
     $S[j - 1] = \text{NaE} \wedge S[j, \dots, j + m] \neq \text{NaE} \wedge S[j + m + 1] = \text{NaE}$ 
  do
     $Name := Merge(T[j].Word, \dots, T[j + m].Word)$ 
     $Class := Select(S[j], \dots, S[j + m])$ 
     $\mathcal{E} := \mathcal{E} \cup \{(Name, Class)\}$ 
     $T := Replace(T, j, j + m, (Name, Class))$ 
  end for
return  $\mathcal{E}$ 
end

```

Phase I: Above, you can see the tHMM algorithm for recognizing named entities. This algorithm takes as input a tokenized document T and builds the sequence of observation symbols using a function that extracts features from the tokens. The current implementation of the extraction function returns an array of 18 boolean features which include, for example `CityNameSuffix` (e.g. “Vaccaville” has a city name suffix). We have also considered two more solutions in which the features of the previous token (bigram configuration) and of both previous and next tokens (trigram configuration) are taken into account. The Viterbi algorithm (Neuhoff, 1975) is a well-known approach to solving Problem 2. It finds the state sequence S that maximizes the joint probability of the observation sequence O and state sequence given the model. In order to learn the parameter A , B and π of the model, the system has been trained on text documents randomly selected from the *Brown Corpus* (Francis, 1979).

Phase II: Each sequence of tokens whose corresponding state is not `NaE` (i.e. not an entity) undergoes further processing to be identified. The entity’s name is clearly the concatenation of the words in each token, while its class is determined based on the classes of the components’ tokens. If tokens have been assigned to different classes (let C be the set of these classes) we need to select one of them. To this aim we have proposed three solutions.

- *Probabilistic classification*, selects the class in C that is the most likely to produce the observation sequence corresponding to the sequence of tokens;

	R	$P_{probabilistic}$	P_{voting}	$P_{evolving}$
monogram	55,80 %	63,30 %	63,30 %	63,30 %
bigram	65,75 %	68,57 %	68,48 %	68,86 %
trigram	71,88 %	66,36 %	66,45 %	66,36 %

Figure 1. Recall and precision performance of the Named Entities Recognition Algorithm

- *Voting classification* selects the class that has been assigned to most of the tokens;
- *Evolving classification* selects the class assigned to the last token, assuming that the precision of the recognition increases over time.

Though we cannot describe the entire HMM used in our approach, we present an example of how it works.

EXAMPLE 4.2. Consider a document D with a single sentence “West Palm Beach’s mayor is T. J. Smith”. After the first step of the algorithm we obtain the following classification
 $\langle (“West”, ON), (“Palm”, ON), (“Beach”, LN), (“’s”, NaE), (“mayor”, NaE), (“is”, NaE), (“T.”, PN), (“J.”, PN), (“Smith”, PN) \rangle$

In the second phase “T. J. Smith” is correctly classified as a person name using all the three approaches, and “West Palm Beach” is correctly classified as a location name with both the probabilistic and evolving approach, while is wrongly classified as an organization name with the voting approach.

Figure 1 shows the results in terms of recall (the percentage of identified named entities) and precision (the percentage of correctly classified entities) that we have obtained for each configuration and for each class selection strategy. The results are comparable with the ones described in the literature. Since we are mainly interested in recall, the trigram configuration seems to be the most promising.

4.2. ATTRIBUTE EXTRACTION FROM RELATIONAL AND XML SOURCES

First, let us consider a relational table $T = \{c_1, \dots, c_m, \dots, c_n\}$ where c_1, \dots, c_n are columns and c_1, \dots, c_m are also keys. Then for each two columns $c_{1 \leq i \leq m}, c_{1 \leq j \leq n}$ we add the following entry to the SAT table:

$$\langle c_j, T : c_i, f_{c_j, T : c_i}(e) = \pi_{c_j} \sigma_{c_i=e} T \rangle.$$

In other words, given a table T as the source, we obtain an attribute c_j for an entity e by looking for all table rows that can be referred by e .

It is also possible to extract attribute values from XML sources. Consider an XML node

$$N = \langle name, value, \{c_1, \dots, c_n\} \rangle$$

where c_1, \dots, c_n are children nodes. Assuming that N is a root node in an XML document, and nodes may act both as entities and the attributes, one can write the following algorithm to return a given attribute of an entity:

```

Algorithm GetXMLAttr( $N, e, A$ )
   $N$  is the root XML node
   $e$  is the entity
   $A$  is the attribute
begin
   $Result := \emptyset$ 
  if  $N.value = e$  or  $N.name = e$  then
    for each child  $c$  of  $N$  such that  $c.name = A$  do
       $Result := Result \cup \{c.value\}$ 
    end for
  else
    for each child  $c$  of  $N$  do
       $Result := Result \cup GetXMLAttr(c, e, A)$ 
    end for
  end if
  return  $Result$ 
end

```

The *GetXMLAttr*() recursively finds all occurrences of an entity in the XML tree, collects all values of the requested attribute, and returns the collected set of values. Notice that the algorithm tries to match e to both *node value* and *node name*. We can now enumerate all node names and values occurring in the XML tree as A_1, \dots, A_m , and for each A_i , add a SAT table entry $\langle A_i, N, GetXMLAttr(N, e, A_i) \rangle$.

5. Story Computation

We start this section by presenting an algorithm to find optimal stories. We then present three heuristic algorithms that build upon our past work in video summarization algorithms in (Fayzullin, 2005) (which did not have to deal with issues such as full instances, deconflicted instances, generalizations, etc.) do not necessarily find an optimal story, but create “good enough” stories in a reasonable time.

Given an entity e , the *OptStory* algorithm finds an *optimal* story of length k by maximizing the value of the evaluation function:

```

Algorithm OptStory(e, SAT, k)
  e is an entity
  SAT is a source access table
  k is the requested story size
begin
   $\mathcal{I} := \text{DeconfI}(\text{ClosedI}(e, \text{FullI}(e, \text{SAT})))$ 
  return RecStory( $\emptyset, \mathcal{I}, k$ )
end

```

The *OptStory* algorithm first picks all data about the entity *e* available in \mathcal{I} . It then calls the recursive *RecStory* algorithm that enumerates over all possible stories of *k* or fewer attributes that can be derived from the given data and returns the best story with respect to the evaluation function *eval*():

```

Algorithm RecStory(Story, Data, k)
  Story is the story so far
  Data is the set of attribute-value pairs to assign
  k is the remaining story size
begin
   $\langle \text{BestS}, \text{BestW} \rangle := \langle \text{Story}, \text{eval}(\text{Story}) \rangle$ 
  if k > 0 then
    for each  $\langle A, v \rangle \in \text{Data}$  do
      S := Story with  $\langle A, v \rangle$  attached to the tail
       $\langle S, W \rangle := \text{RecStory}(S, \text{Data} \setminus \{\langle A, v \rangle\}, k - 1)$ 
      if W > BestW then  $\langle \text{BestS}, \text{BestW} \rangle := \langle S, W \rangle$ 
    end for
  end if
  return  $\langle \text{BestS}, \text{BestW} \rangle$ 
end

```

5.1. RESTRICTED OPTIMAL STORY ALGORITHM

Given *n* attributes, the *RecStory* algorithm will have to sort through $\sum_{0 \leq i \leq k} \frac{n!}{(n-i)!}$ stories. Even if we restrict the algorithm to the *k*-length stories, it will still have to consider $\frac{n!}{(n-k)!}$ stories. To make story creation more manageable, let us consider the following algorithm:

```

Algorithm RecStory+(Story, Data, k, b)
  Story is the story so far
  Data is the set of attribute-value pairs to assign
  k is the remaining story size
  b is the branching factor
begin
   $\langle \text{BestS}, \text{BestW} \rangle := \langle \text{Story}, \text{eval}(\text{Story}) \rangle$ 
  Q is a priority queue
  if k > 0 then
    for each  $\langle A, v \rangle \in \text{Data}$  do
      S := Story with  $\langle A, v \rangle$  attached to the tail
      Q.add(S, eval(S))
      if length(Q) > b then Q.delete(tail(Q))
    end for
  end if
end

```

```

for each  $SS \in Q$  do
   $\langle S, W \rangle := RecStory(SS, Data \setminus SS, k - 1)$ 
  if  $W > BestW$  then  $\langle BestS, BestW \rangle := \langle S, W \rangle$ 
end for
end if
return  $\langle BestS, BestW \rangle$ 
end

```

The $RecStory^+$ algorithm essentially limits the search at each step to the b best stories w.r.t. the evaluation function. Given n attributes, this algorithm only considers $1 + \sum_{0 \leq i < k} (b^i \cdot (n - i))$ stories. We use $OptStory^+$ to denote the algorithm that calls $RecStory^+$.

5.2. GENETIC PROGRAMMING APPROACH

In this section, we present a story creation algorithm $GenStory$ based on genetic programming (Langdon and Poli, 2002). $GenStory$ creates suboptimal stories, but does so quickly.

```

Procedure  $GenStory(e, SAT, k, N, \delta)$ 
   $e$  is an entity
   $SAT$  is a source access table
   $k$  is the requested story size
   $N$  is the desired number of iterations
   $\delta$  is the desired fitness threshold
begin
   $Data := DeconfI(ClosedI(e, FullI(e, SAT)))$ 
   $R := \lceil \frac{card(Data)}{k} \rceil$ 
   $Q := R$  random solutions of  $k$  attributes from  $Data$ 
  for  $j \in [1, N]$ 
    for  $i \in [1, R]$ 
       $S :=$  solution randomly chosen from  $Q$ 
      Choose random  $\langle A, v \rangle \in Data$  and  $\langle A', v' \rangle \in S$ 
      Replace  $\langle A', v' \rangle$  in  $S$  with  $\langle A, v \rangle$ 
       $Q := Q \cup \{S\}$ 
       $Q := Q \setminus \{S'\}$  where  $\forall S \in Q eval(S) \geq eval(S')$ 
      if  $max_{S_1, S_2 \in Q} |eval(S_1) - eval(S_2)| \leq \delta$  then
        Return best solution from  $Q$ 
      end if
    end for
  end for
  Return best solution from  $Q$ 
end

```

The $GenStory$ algorithm starts by creating a population Q of $\lceil \frac{card(Data)}{k} \rceil$ random stories. It will then repeatedly choose a random story S from this population and replace a random attribute in this story with a different attribute not occurring in S . The resulting story is added to Q , and then the story with the lowest $eval()$ value is deleted from Q . The $GenStory$ algorithm will terminate when all story candidates in the population Q have approximately the same worth (w.r.t. the value of δ) or when the maximal number of iterations N is reached.

5.3. DYNAMIC PROGRAMMING APPROACH

In this section, we present a story creation algorithm *DynStory* based on the dynamic programming approach. This algorithm also yields stories that are suboptimal, yet does it in less time than the the *OptStory* algorithm.

```

Procedure DynStory(e, SAT, k)
  e is an entity
  SAT is a source access table
  k is the requested story size
begin
  Data := DeconfI(ClosedI(e, FullI(e, SAT)))
  S := random solution of k attributes from Data
  Data := Data \ S
  while Data ≠ ∅
    subs := false
    r := 1
    while r < k and subs = false
      S' := S with  $\langle A_r, v_r \rangle$  replaced with first(Data)
      if eval(S) < eval(S') then
        S := S'
        Add  $\langle A_r, v_r \rangle$  to the tail of Data
        subs := true
      else
        r := r + 1
      end if
    end while
    remove first(Data) from Data
  end while
  return S
end

```

The *DynStory* algorithm starts by creating a random solution *S* and proceeds by trying to replace each attribute in *S* with the first attributes from the list of candidates *Data*. As soon as a better solution is found, it takes the place of *S*. The algorithm terminates when the list of candidates is exhausted. The time complexity of *DynStory* is linear w.r.t. the number of attributes in the instance *Data*.

6. Story Rendition

Our system renders the set of facts constituting an optimal story in one of two ways: (1) when extracting facts, we try to store sentences from which these facts have been extracted. When narrating a story, we use these sentences whenever available; (2) one can use templates to construct sentences from facts. In our system, a template exists for each distinct attribute name that the system fills out with the entity name and the attribute value(s).

7. Experiments

We evaluated both the quality of stories produced by our algorithms and the time taken to construct them. A team of 61 students has been subdivided into two groups: (i) 10 experts and (ii) 51 non-experts. They were asked to review stories about the following mythological Greek characters: Pentheus, Cadmus, Apollo, Agave, Semele, and Dionysus. Before the experiment, the expert group was asked to study some documentation about story subjects, while non-experts were not given any a priori knowledge about the subjects. Each reviewer was asked to evaluate the story value (in terms of included facts) as well as the quality of the narrative prose on a scale from 1 (worst) to 5 (best). Reviewers were presented with stories of different lengths, computed with five different algorithms, varying continuity, priority and repetition parameters. In addition, we rendered stories in two ways: (1) using sentences from textual sources whenever available and (2) using templates only. A total of 200 stories was created for each story subject, making the evaluation process highly time-consuming and limiting the number of story subjects (lest the reviewers become bored and inaccurate).

7.1. STORY QUALITY: NON-EXPERT REVIEWERS

We were interested in determining which algorithm and which rendering strategy performs best.

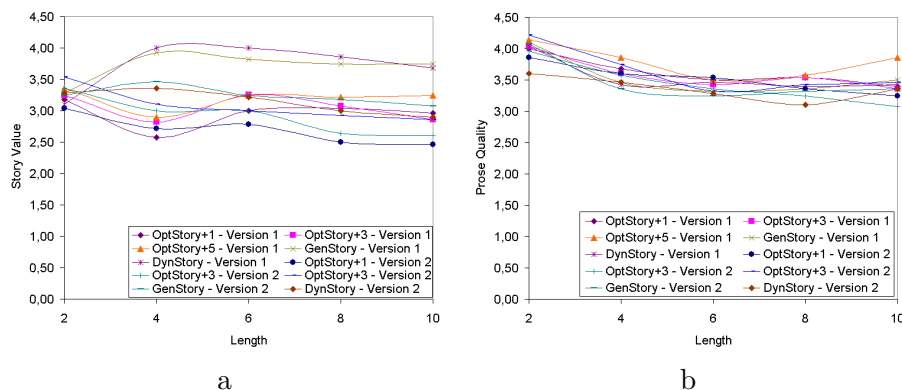


Figure 2. (a) Story Value and (b) Prose Quality vs. Story Length.

Figure 7.1.a shows the value of the story as judged by human subjects as a function of story length for five different algorithms. Figure 7.1.b shows the quality of the prose in the story as perceived by the reviewers. Version 1 refers to the case where sentences from the sources are used, while version 2 refers to the case where templates are used.

Which algorithm produces the best story value?

- DynStory algorithm yields the highest story value, while algorithm GenStory is the second best.
- Both DynStory(1) and GenStory(1) algorithms are consistently better than algorithms DynStory(2) and GenStory(2). This is due to the fact that including original sentences into the story adds more information than rendering the same facts through templates.
- The prose quality (as assessed by human readers) is almost the same for all algorithms and decreases slightly as the story length increases. This is to be expected, given that all algorithms use the same mechanism to render actual stories.

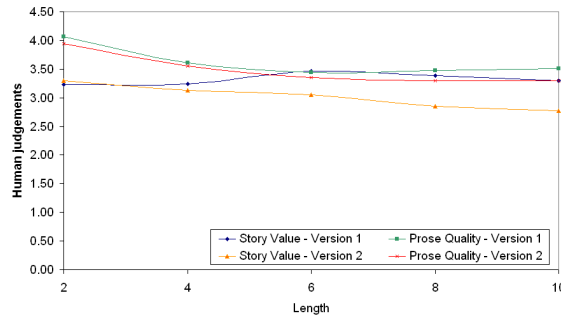


Figure 3. Story Value and Prose Quality.

How does using source sentences compare with using templates? Let us now consider the graph in Figure 3. The curves have been obtained by averaging, for each rendering technique, the results of all five algorithms. The graph indicates that using source sentences significantly improves the story value as perceived by a human, but only slightly improves the prose quality over the template rendering method. The y-axis in this graph shows the quality of the story as judged by human subjects on a 1 to 5 scale.

7.2. EXPERT REVIEWERS

We now present the results of experiments with expert reviewers and point out the difference from the non-expert group’s results.

Which algorithm produces the best story value? Figures 7.2.a and 7.2.b were obtained in the same way as Figures 7.1.a and 7.1.b, but for the expert group of reviewers. These figures allow to make the following observations:

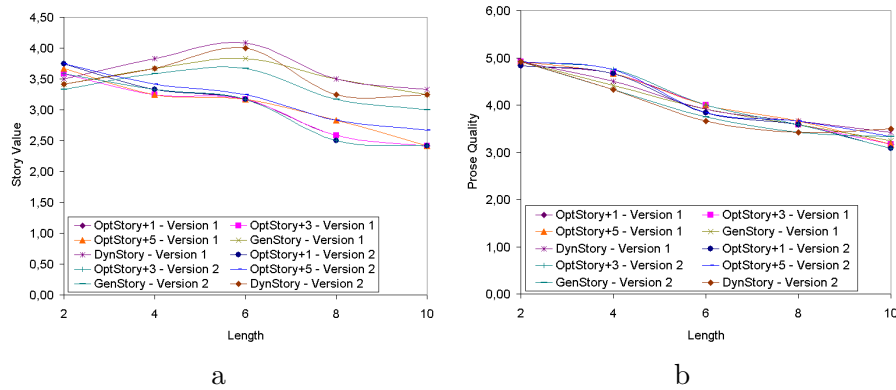


Figure 4. (a) Story Value and (b) Prose Quality vs. Story Length.

- DynStory and GenStory algorithms ensure the story value, same as in the case of non-experts. However, in this case their source-based and template-based versions are much closer than in the previous case. This may be due to the fact that expert reviewers recognize that the rendered facts are the same in both versions.
- The prose quality is still the same for all algorithms, but it rapidly decreases with rising story length.

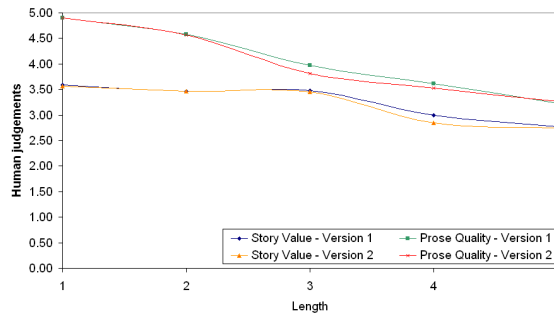


Figure 5. Story Value and Prose Quality.

How does using source sentences compare with using templates? Figure 5 shows the quality of a story as judged by human subjects on a 1 to 5 scale. It shows that for expert reviewers, using source sentences does not improve the story value – prose quality is slightly higher for templated-rendered stories.

How do experts compare with non-experts? Let us now compare results from expert and non-expert reviewers, as shown in Figure 6 (the y -axis shows human subject judgements of the stories produced).

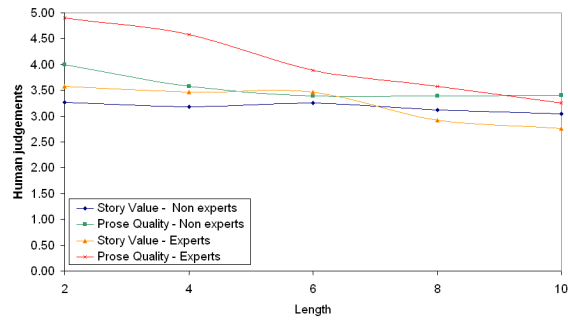


Figure 6. Experts vs. Non-Experts Comparison.

- Experts rate short stories with higher value than non-experts.
- As stories become longer, experts rate their value lower, while ratings from non-experts remain almost the same.
- For all but very long stories, experts rate prose quality higher than non-experts. This may be due to the fact that they are more aware of the machine-generated story nature.

We used the t-test to analyze the statistical validity of our experiments. We first considered the non-experts and compared their judgements about both the value of stories and the quality of the prose. We obtained *tValues* between 0.4 to 0.5. Those numbers indicate no statistically significant difference in the sample means. We then applied the same analysis to the judgements of expert reviewers, obtaining similar results, and thus concluding that human judgments can be considered significant within groups of similar people. We then compared expert and non-expert judgements, obtaining *tValues* greater than 1. The difference in the means became significant in some particular cases, such as judgements about the prose for low values of the story length (*tValues* greater than 3), thus confirming the interpretation of Figure 6.

Besides the qualitative evaluation we also evaluated the algorithm execution times. In particular we compared the time needed to create a story when the necessary data are already in the database and the time for creating the story when data has to be extracted real-time from the data-sources. Figure 7 reports the results of this comparison and clearly shows that the data extraction time is two orders of magnitude greater than the story creation time (tens of milliseconds vs seconds).

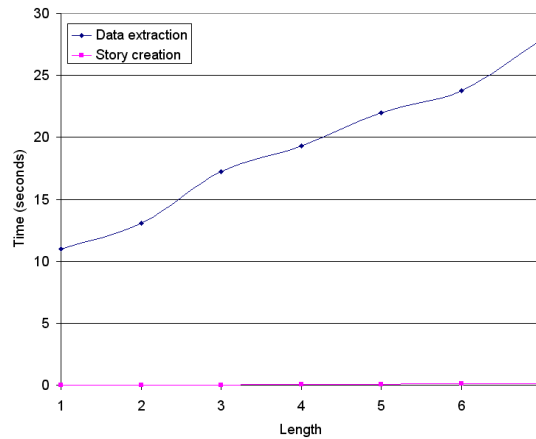


Figure 7. Comparison between execution times

8. Related Work

There is a rich body of work on creating stories in the non-database literature. Many of them focus on specific audiences (e.g. children). For example, the TEATRIX system (Machado et.al., 2000) allows children to collaboratively create stories. The SAGE system (Bers et.al., 1998) allowed young heart patients to describe their own stories, as well as to create characters. Theune et. al. (Theune et.al., 2003) develop agents that allow each player in the story to have some autonomy so that the final outcome of the story is not a static script. All these papers have a very different goal than ours because they either focus on either having humans create a story, or provide some kind of action specification for the agents involved in the story and allow the story to develop having a non-deterministic outcome.

9. Conclusions

Our work focuses on *extracting stories* from multiple heterogeneous data sources, and creating a *size-limited* set of facts that have *maximal priority and relevance* to the user. These facts (the triples in this paper) may then be presented to the user in an appropriate way. Many of the GUIs and linguistic storytelling methods developed for interactive storytelling are wonderful devices that can be adapted to graphically and interactively present the facts we create - these are complementary efforts that do not attempt to figure what goes into the “story”.

We plan to move forward in several new directions. The first goal is to quantitatively define what constitutes a good story. We studied numerous definitions in literature - most of them are behavioral definitions that describe the quality of a story in terms of its impact on its readers (e.g. a story is good if I can't stop reading it and when I do stop reading it, I can't stop thinking about it). Coming up with quantitative models of such statements is a formidable challenge. Our second goal is to further improve our algorithms to extract entity attribute value triples from text sources. A third goal is to assess how best to weight the conditions of continuity, non-repetition and priority of facts within a given story.

Acknowledgements. Work supported in part by ARO grant DAAD190310202, ARL grants DAAD190320026 and DAAL0197K0135, NSF grants IIS0329851 and 0205489 and UC Berkeley contract number SA451832441 (subcontract from DARPA's REAL program).

References

- R. Agrawal, R. Bayardo, and R. Srikant. Athena: Mining-based interactive management of text databases. *Proc. Intl. Conf. on Extending Database Technology*, pps 365-379.
- M. Bers, E. Ackermann, J. Cassell, B. Donegan, J. Gonzalez-Heydrich, D. DeMaso, C. Strohecker, S. Lualdi, D. Bromley and J. Karlin. Interactive Storytelling Environments: Coping with Cardiac Illness at Boston's Children's Hospital. *Proc. CHI-1998*, pp. 603-610.
- J. Callan and T. Mitamura. Knowledge-Based Extraction of Named Entities. *Proc. 4th Intl. Conf. on Information and Knowledge Management*, 2002.
- M. Fayzullin, V.S. Subrahmanian, A. Picariello, and M.L. Sapino. The CPR Model for Summarizing Video. *accepted for publication in Multimedia Tools and Applications journal*.
- W.N. Francis. Brown Corpus Manual. <http://helmer.aksis.uib.no/icame/brown/bcm.html>, 1979.
- Z. GuoDong and S. Jian. Integrating Various Features in Hidden Markov Model Using Constraint Relaxation Algorithm for Recognition of Named Entities Without Gazetteers. *Proc. Int. Conf. on Natural Language Processing and Knowledge Engineering, October 2003*, pp. 465-470.
- H. M. Jamil and L. V. S. Lakshmanan. A Declarative Semantics for Behavioral Inheritance and Conflict Resolution. *Proc. International Logic Programming Symposium*, pps 130-144.
- W.B. Langdon, R. Poli. Foundations of Genetic Algorithms. *Springer*.
- I. Machado, R. Prada, and A. Paiva. Bringing Drama into a Virtual Stage. *Proc. CVE-2000*.
- G. A. Miller. WordNet: a Lexical Database for English. *Communications of the ACM*, Vol. 38(11), November 1995, pp. 39-41.
- D. Neuhoff. The Viterbi Algorithm as an Aid in Text Recognition. *IEEE Transactions on Information Theory*, Vol. 21(2), 1975, pp. 222-226.
- I.L. de Oliverira and R.S. Wazlawick. A Modular Connectionist Parser for Resolution of Pronominal Anaphoric References in Multiple Sentences. *Proc. Int. Joint Conf. on Neural Networks, IEEE World Congress on Computational Intelligence, May 1998*, Vol. 2, pp. 1194-1199.
- S. Oyama, T. Kokubo and T. Ishida. Domain-Specific Web Search with Keyword Spices. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 13(1), January 2004, pp. 17-27.
- P. Rosso, F. Masulli and D. Buscaldi. Word Sense Disambiguation Combining Conceptual Distance, Frequency and Gloss. *Proc. Int. Conf. on Natural Language Processing and Knowledge Engineering, October 2003*, pp. 120-125.
- M. Theune, S. Faas, A. Nijholt and D. Heylen. The Virtual Storyteller: Story Creation by Intelligent Agents. *Proc. of Technologies for Interactive Digital Storytelling and Entertainment Conference, 2003*, pp. 204-215.