Analysing Temporally Annotated Corpora with CAVaT

Leon Derczynski, Robert Gaizauskas

University of Sheffield
211 Portobello, S1 4DP, UK
L.Derczynski@dcs.shef.ac.uk, R.Gaizauskas@dcs.shef.ac.uk

Abstract

We present CAVaT, a tool that performs Corpus Analysis and Validation for TimeML. CAVaT is an open source, modular checking utility for statistical analysis of features specific to temporally-annotated natural language corpora. It provides reporting, highlights salient links between a variety of general and time-specific linguistic features, and also validates a temporal annotation to ensure that it is logically consistent and sufficiently annotated. Uniquely, CAVaT provides analysis specific to TimeML-annotated temporal information. TimeML is a standard for annotating temporal information in natural language text. In this paper, we present the reporting part of CAVaT, and then its error-checking ability, including the workings of several novel TimeML document verification methods. This is followed by the execution of some example tasks using the tool to show relations between times, events, signals and links. We also demonstrate inconsistencies in a TimeML corpus (TimeBank) that have been detected with CAVaT.

1. Introduction

In essence, TimeML mandates the mark up of expressions referring to *times*, expressions denoting *events* and expressions *signalling* temporal relations between times and events or events and events; it also allows *links* to be added between entities, which are labelled with the temporal relation holding between them.

Existing TimeML tools can be divided into two categories: those which produce or alter mark-up, for example to assist annotation, and those that perform analysis. Only a few tools have as yet been developed for TimeML, mostly focusing on the annotation task, such as TTK (Verhagen and Pustejovsky, 2008), which does not support analysis. From the second category, in the absence of other software, the TimeML-using community is restricted to generic XML analysis tools, such as Xaira (Burnard and Dodd, 2003) or LT-XML[1], as well as similar format-specific tools (TEI). These generic corpus tools are powerful applications, but require substantial effort to apply to TimeML data.

We have constructed CAVaT (Corpus Analysis and Validation for TimeML) to process collections of temporally annotated documents. CAVaT's functionality is divided into two main parts; an integrated browsing and report generation system, and a modular extensible error checking and corpus validation framework.

In this paper, we first describe the technical aspects of the tool. We then present the reporting part of CAVaT, and then its error-checking ability, followed by the execution of some example tasks using the tool. We present an overview of the tool's operation and capabilities in Section 2. This includes details of the corpus loading and folding process (Section 2.1), report generation, and also a detailed explanation of the advanced validation modules that are included with CAVaT (Section 2.3). A brief syntax summary is presented in Section 3; the full guide is on the CAVaT website[2]. Next, in Section 4, we present sample queries and output. In Section 5, we show inconsistencies and observations in a TimeML corpus (TimeBank) that have been detected with CAVaT. Finally, Section 6 summarises the tool and discusses future work.

2. Overview of the tool

CAVaT is an open source tool, constructed from a set of Python modules and a database. It uses NLTK[3] and MySQL[4]. The interface is a text-based interactive prompt, and all operations are performed with text commands. Command syntax strives to be simple, flexible and close to natural language. After loading and pre-processing a TimeML corpus, one can analyse it using built-in reporting functions, and perform data validation with one of many checking components.

2.1. Preprocessing

CAVaT can work on any TimeML-annotated corpus that is stored as a collection of uncompressed files in a single directory, by importing it to a set of database tables. The corpus is initially processed by an XML parser (using Python's `minidom` and `expat` implementations), which retrieves document level data as well as all temporally annotated information, and places it into a MySQL database. Temporally annotated data includes all TimeML tags and their attributes, as well any enclosed tokens for EVENT, SIGNAL and TIMEX3 tags.

In TimeML, events are represented with the EVENT tag, and temporal expressions with the TIMEX3 tag. These intervals are the elements which CAVaT and the rest of this paper assume as temporal primitives, unless otherwise stated. Temporal relations between intervals are described with the TLINK tag, and temporal signals with SIGNAL. See Figure 1 for an example.

Automatically classifying the type of temporal relation between intervals is currently a difficult problem in temporal processing of text (Mani et al., 2006; Lapata and Lascarides, 2006; Hepple et al., 2007). The task is often made simpler by reducing the number of temporal link classes. TimeML includes BEFORE and AFTER relations, though

[1] From http://www.ltg.ed.ac.uk/software/ltxml
[2] Available at http://code.google.com/p/cavat/
[3] See http://www.nltk.org/
[4] See http://www.mysql.com/

Figure 1: Example text and TimeML annotation.

Text: "On Thursday, he took the plane to Copenhagen"

TimeML:

```
<SIGNAL sid="s1">On</SIGNAL>
<TIMEX3 tid="t1" type="DURATION">Thursday</TIMEX3>,
he <EVENT eid="e1" class="I_ACTION">took</EVENT>
the plane to Copenhagen.

<MAKEINSTANCE eiid="ei1" eventID="e1" pos="VERB"
 tense="PAST" polarity="pos" />
<TLINK eventInstanceID="e1" relatedToTime="t1"
 signalID="s1" relType="DURING">
```

Table 1: Mappings between TimeML relations that can be applied in order to reduce the size of the relation set; when applying the transformation in the table, TLINK argument order is swapped.

Original relation type	Folds to relation
AFTER	BEFORE
IS_INCLUDED	INCLUDES
IAFTER	IBEFORE
BEGUN_BY	BEGINS
ENDED_BY	ENDS
DURING_INV	SIMULTANEOUS
DURING	SIMULTANEOUS
SIMULTANEOUS	SIMULTANEOUS

one may simply reverse the arguments of a BEFORE relation to turn it into an AFTER one — so, *June 2008 was before August 2009* is equivalent to *August 2009 was after June 2008*. It is thus possible to convert all links of one of these types to the other. We call this technique **folding**. Given a set of mappings, the 13 TimeML relations can be reduced.

CAVaT offers three folds:

- **CAVaT fold** – Collapses all inverse relations, such as mapping INCLUDED_BY to INCLUDES (see Table 1).
- **SputLink fold** – The mapping introduced by Marc Verhagen, included in TTK (Verhagen, 2005).
- **Compact fold** – Reduces TimeML's link relation set to 3 classes, using mappings defined in Setzer et al. (2005).

The first two are lossless, in that no temporal information is removed by the folding process. The third is lossy. It is possible to perform a lossy fold by, for example, reducing the TimeML BEGUN_BY relationship to one of INCLUDES.

2.2. Querying

The reporting part of CAVaT makes analysis of TimeML corpora simpler and easier than working directly with a set of XML documents, allowing flexible queries, and catering for inquiries specific to temporally-annotated data.

The development of CAVaT has been driven by investigations of TimeML corpora. Many of the operations performed against a corpora had common elements, often centred around the retrieval of class distributions or token frequencies. A tool for TimeML corpus research could encompass all the required operations, while providing access to a larger range of reports.

CAVaT uses a report generation system where one can view any number of pre-defined features that match conditions of the user's choosing. Queries can produce reports at varying levels of granularity – one may choose to examine data at sentence, document or corpus level. Reports can output counts, distributions, lists or text extracts. Example queries are listed in Section 4. Data such as part-of-speech information, tense, aspect, and event recurrence are captured by attributes described by TimeML, and any data like this (annotated by tags and their attributes) can be queried. In addition, properties specific to temporal data but not directly present in mark-up are implemented, including:

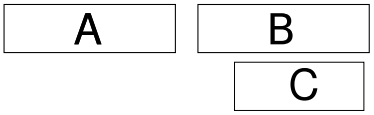
- **Event / event instance abstraction** In some cases, one piece of text may refer to two separate events (an example is given later in Section 5.1.). To permit annotation of this, TimeML's EVENT tag is placed around the text, and then event instances are specified using one or more MAKEINSTANCE tags. Data relating to a piece of event text, such as part of speech, polarity and modality, are described in the MAKEINSTANCE tag. However, we would often like to see the part of speech data for an event; indeed, when discussing temporal entities, the term "event" is often used in place of "event instance". Thus, CAVaT implicitly translates between these two related tags when requested; for example, when one asks to see event modality or cardinality.
- **Signalled links** TLINKs may indicate a textual signal that suggests the type of relationship between their arguments. For example, in *Lydia ate dinner before leaving the house*, the word *before* acts as a signal, ordering two events. As signals are explicit indicators of temporal association, and correctly typing a temporal link is difficult, it is useful to be able to quickly identify which links employ a signal.
- **Signal text and TLINKs** As SIGNAL text referenced from a TLINK may be thought of as that TLINK's signal text, CAVaT permits queries that specify signal text as an attribute of a TLINK.
- **Text position and lemma** Although not part of the TimeML annotation schema, CAVaT logs text position (by sentence number and word number), and maintains lemmas of text found within tags.

One may view a particular TLINK's location in the original document, showing the link's arguments and their relation type. This helps understand the context of a single TLINK. For example, one may often see many links to a single document date, or discover that most links have arguments within the same paragraph – something not immediately obvious to humans while browsing the TLINK markup, and unclear with generic corpus tools.

2.3. Checking

Temporal annotation is a complex task, and as a result, a relatively small amount of text has been annotated to date. The largest TimeML corpus is TimeBank (Pustejovsky et al., 2003), with less than 200 documents, and around 65000

Figure 2: With time flowing from left to right, this represents A BEFORE B and B INCLUDES C. It is not possible for C and A to be the same interval.



tokens. Because of the complexity of temporal annotation, errors can arise beyond those that may be detected using an XML DTD. CAVaT is both a reporting and validation tool, and seeks to automatically detect high-level and complex errors that are rarely immediately obvious. Part of the motivation behind this part of the tool is similar to that of writing unit tests that highlight bugs in an application: to improve quality by automatically detecting previously seen errors. In this section we detail some checks that CAVaT can perform on a TimeML corpus.

Error checks are defined as Python modules, so that one may describe a detection method for an error case and share it with other researchers without modifying CAVaT's core code. The modules inherit from the `cavatModule` class; documentation is in the source code, and one may view a list of available modules with the command `check list`.

2.3.1. Inconsistent closure

It is possible to create an inconsistent configuration of temporal links. For example, we may have A BEFORE B and B INCLUDES A; this is clearly not possible, as INCLUDES stipulates that the start point of A occur after the start point of B (see Figure 2). While this example is fairly clear, it may not be at all clear to human annotators that a partial temporal link annotation could imply an inconsistent configuration.

Automatically checking the consistency of a temporal network is hard. TimeML's relations are based on those of Allen (1983), and it is difficult to guarantee the consistency of networks formed using the latter set of relations (Vilain et al., 1989; Tsang, 1987). We re-state the problem in a more simple fashion, as follows. Intervals are represented by pairs of endpoints, and we define intervals and the TimeML relations between them in terms of relations between these points. Our model uses only simultaneous ($=$) and before ($<$) relations.

The consistency checker works in a similar way to the closure algorithm in Setzer et al. (2005). It maintains an agenda and database. Assertions are taken from the agenda and used to infer further assertions when combined with assertions in the database. We initially process intervals in the document (taken from TLINK arguments) – for each interval I we add $I_{start} < I_{end}$ to the database. We then generate initial data for the agenda based on TLINKs in the document and a mapping for each TLINK to one or more assertions, listed in Table 2.

The only inference rules needed with our minimal set of relations are:

- If $x = y$ then $y = x$
- If $x = y$ and $y = z$ then $x = z$
- If $x < y$ and $y < z$ then $x < z$

Table 2: Mapping from TimeML relation types to a simple point-based temporal algebra. The TimeML relation is of the form a RELATION b. Where multiple relations are given, all hold. Similar to the table listed in (Verhagen, 2005).

TimeML relation type	Relation added to agenda
BEFORE	$a_{end} < b_{start}$
AFTER	$b_{end} < a_{start}$
IAFTER	$b_{end} = a_{start}$
IBEFORE	$a_{end} = b_{start}$
INCLUDES	$a_{start} < b_{start}, b_{end} < a_{end}$
IS_INCLUDED	$b_{start} < a_{start}, a_{end} < b_{end}$
BEGINS	$a_{start} = b_{start}, a_{end} < b_{end}$
BEGUN_BY	$a_{start} = b_{start}, b_{end} < a_{end}$
ENDS	$a_{end} = b_{end}, b_{start} < a_{start}$
ENDED_BY	$b_{end} = a_{end}, a_{start} < b_{start}$
SIMULTANEOUS	$a_{start} = b_{start}, a_{end} = b_{end}$
IDENTITY	$a_{start} = b_{start}, b_{end} = a_{end}$
DURING	$a_{start} = b_{start}, a_{end} = b_{end}$
DURING_INV	$a_{start} = b_{start}, a_{end} = b_{end}$

We can take items from the agenda. For each such item, we compare it against the database, and deduce new relations using the above rules. If a newly generated relation conflicts with anything in the agenda or database, then the document is inconsistent. Otherwise, we will move the item from the agenda to the database, and add newly generated relations to the agenda. If we can clear the agenda, then the document is consistent; otherwise, it is not. Whether we add new relations to the top or bottom of the agenda (achieving depth- or breadth-first search, respectively) is irrelevant to the success of the algorithm, though computational performance differences have not been measured.

Our baselines are the results of the `tlink_loop` test (Section 2.3.3.) and also the results of closure success according to SputLink (Verhagen, 2005). This algorithm detected all known inconsistencies in TimeBank, and found one more; full details are later in Section 5.2. A test TimeML corpus is included with CAVaT for verifying that the consistency checker works, which alternative implementations may use for validation.

Below is sample output from a consistency check:

```
cavat> check consistent in 3
# Temporal graph consistency checker v1 loaded
# Checking wsj_0927.tml (id 3)
! Inconsistent closure - could not assert
 (ei2415_2 < ei2414_1)
```

2.3.2. Disconnected sub-graph detection

After inferring a temporal closure (Verhagen, 2005) of a document, one is usually left with a single interconnected temporal graph, where nodes are TIMEX3s or EVENTs and edges represent TLINKs. However, disconnected groups of links may exist post-closure. This should be brought to the attention of the user; it often suggests that annotating a small number of additional links can greatly increase the amount of data inferable though closure, and that an annotation is incomplete.

CAVaT's sub-graph identification module, `split_graph`, works by processing TLINKs from a document sequentially. We maintain a list of sets that will hold intercon-

nected intervals, beginning with an empty list. For each TLINK, we check to see if either of its arguments (which are both intervals) can be found in any set in our list. If one argument can but the other cannot, the new interval is added to the same set as the found interval. If they are both found in the same set, no action is taken. If they are found in different sets, those two sets are merged. If neither TLINK argument can be found anywhere, a new set holding both intervals is created. This process is repeated until all TLINKs have been processed, at which point each set in the list represents an independent sub-graph of connected intervals.
The module will then report statistics about the graph(s) found in the specified document. These include:

- Count of sub-graphs, intervals and TLINKs;
- The number of "isolated" sub-graphs – that is, those described by only one temporal link – and the proportion of intervals/links used to describe all these isolated sub-graphs;
- Mean and maximum sub-graph size, and the proportion of the document's intervals that are in the largest sub-graph;
- The entropy of sub-graph sizes, which acts as a "fracturedness" measure, showing how far the document is from having one single totally connected temporal graph including all TLINKs;
- The distribution of sub-graph sizes.

Even though sub-graphs are populated by processing the two intervals of a TLINK at the same time, it is possible to have a sub-graph containing just one node, in the case of a TLINK loop (Section 2.3.3.). Note that a document containing intervals but no temporal links between them is marked as "un-fractured", as this check ignores any items not referenced at least once by a temporal link.
Here is sample output from an attempt to identify disconnected sub-graphs:

```
cavat> check split_graph in 3
# Split graph detection v1 loaded
# Checking wsj_0927.tml (id 3)
Subgraphs found: 13 - composed of 69 nodes and linked
 by 65 TLINKS.
Isolated subgraphs, that contain just one TLINK: 5
 (making up 38.5% of all subgraphs / consuming 14.5%
 of all nodes / described by 7.7% of all TLINKs);
Mean graph size 5.3 nodes; largest subgraph (size 35)
 has 50.7% of all nodes.
Entropy of subgraph sizes:  0.448277644573
    2 nodes: ( 5) .....
    3 nodes: ( 4) ....
    4 nodes: ( 3) ...
   35 nodes: ( 1) .
```

2.3.3. Superfluous TLINKs

Some TLINKs in TimeML corpora have been specified that associate an event with itself. For example:

```
<TLINK lid="l67" relType="IDENTITY"
eventInstanceID="ei1241" relatedToEventInstance="ei1241"
/>
```

In this case, the only information conveyed is that ei1241 is identical to itself, making this a redundant TLINK. CAVaT includes a check that will identify TLINKs where both arguments reference the same event instance or event. Although such TLINK **loops** might be detected by consistency checking, those which specify a SIMULTANEOUS or IDENTITY relation will not.
Below is sample output, showing some superfluous TLINKs:

```
cavat> check tlink_loop in 165 159 143
# TLINK loop checker v1 loaded
# Checking ABC19980304.1830.1636.tml (id 165)
TLINK ID l23 may be a loop (eventID match), type
 INCLUDES, event ei286 / ei288 - check document
 manually
# Checking wsj_1013.tml (id 159)
TLINK ID l107 loops directly (instanceID match), type
 IDENTITY, event ei2495 / ei2495
# Checking wsj_0586.tml (id 143)
TLINK ID l192 loops directly (instanceID match), type
 BEFORE, event ei1404 / ei1404
```

2.3.4. Orphaned object details

There is not yet a definition for TimeML annotation completeness, that states a minimal satisfactory level of annotation for a document. In the absence of such a definition, it is not a mistake to annotate entities without attaching them to anything else in the document. However, we believe that wherever possible, every interval should be connected to at least one other interval, and that the annotation of entities that do not contribute or relate to any other annotated information is superfluous. For example, if one chooses to mark text as a temporal signal, a related link or event instance should reference the signal. In this example, if the signal conveys no temporal information, it should not be annotated.
To this end, CAVaT includes a module that is aware of five cases which describe objects attached to nothing else, and reports such **orphan** objects. Firstly, any TIMEX3 that is not related by any link is deemed to be independent. Also, any event instance (from MAKEINSTANCE) that is not referenced by a link is also orphaned. Next, an EVENT that is never instantiated is unattached, as instantiation is required by current TimeML syntax before EVENTs can be linked to anything else. Instances that come from nonexistent or mislabelled EVENTs are also orphans. Finally, SIGNALs that are not referenced by any link or event instance (as in our example above) are included in the list of orphaned objects.
Here is the sample output from a check for orphans:

```
cavat> check orphans in wsj_0927.tml
# Orphaned tag detection v1 loaded
TIMEX3 t104 not in any link
TIMEX3 t131 not in any link
```

2.4. Limitations

CAVaT is currently limited in the number of objects (based on TimeML tags) that it can store for a single corpus. Objects are stored in MySQL tables, and these are limited by the operating system's maximum file size limit. The maximum number of corpora that CAVaT can stored is restricted to the operating system limit of files in a single directory.

3. Syntax

Here we briefly introduce CAVaT's basic top-level commands, and some of their more useful features. A full specification of CAVaT's syntax is available at `http://code.google.com/p/cavat`.

3.1. Corpus manipulation

Commands for manipulating TimeML corpora within CAVaT begin with `corpus`. One may view a list of available corpora with `corpus list`, and use a name from the resulting list to select a corpus for querying or checking with the `corpus use` command. It is also possible to view any notes attached to the currently selected corpus by using `corpus info`. Before one can `use` a corpus, though, a directory of TimeML files must be imported into CAVaT, using `corpus import`. One may also opt to fold the corpus on import (see Section 2.1.); a note will be attached to the database if this has been done.

3.2. Querying

The `show` command generates reports from the current corpus. Reports focus on one tag type, and give information about its attributes. One can view all values for a tag with "list" reports, or the distribution of values with "distribution" reports, or simply see how many instances of that tag use a particular field with "state" reports.
The general format for report generation is:

```
show <report type> of <tag> <field> [as <format>]
```

From the above example, `<tag>` corresponds to a TimeML tag, and is one of `event`, `instance`, `timex3`, `signal`, `tlink`, `slink` or `alink`. As well as the attributes available for each tag, the following extra values for `<field>` are available:

- For TLINKs, `signaltext` refers to the text enclosed by the start and end tags of an associated signal;
- For EVENTs, one may reference all the attributes of a MAKEINSTANCE tag too;
- In TLINKs, SLINKs and ALINKs the arguments are referred to as `arg1` and `arg2`, so that the CAVaT user does not have to worry about the implicit indication of interval type present in attribute names.

Reports are available in multiple formats. These can be specified by adding `as <format>` to the end of a show query.

- `screen` - The default choice, screen gives text formatted for display in a fixed-width font.
- `csv` - Output as comma separated values.
- `tex` - TeX table format, including caption.

The TeX output of an example report, showing the distribution of TLINK relTypes in TimeBank v1.2, can be generated with `show distribution of tlink reltype as tex` and is shown in Table 3.
One may also specify a subset of a corpus to be used for reporting, using a simple `where` clause. For example, one may ask:

```
cavat> show state of tlink signalid where reltype is
after
```

to see how many TLINKs of type AFTER use a signal; or, one may ask:

```
cavat> show distribution of tlink reltype where signalid
is not filled
```

to find out which relTypes are most likely in TLINKs that do not specify a signal. As part of CAVaT's goal to be easy to use and close to natural language, there are multiple valid syntaxes for filled/unfilled attributes.

Table 3: Distribution of Tlink reltype

Tlink reltype	Frequency	Proportion
BEFORE	1408	21.9%
IS_INCLUDED	1357	21.1%
AFTER	897	14.0%
IDENTITY	743	11.6%
SIMULTANEOUS	671	10.5%
INCLUDES	582	9.07%
DURING	302	4.71%
ENDED_BY	177	2.76%
ENDS	76	1.18%
BEGUN_BY	70	1.09%
BEGINS	61	0.95%
IAFTER	39	0.608%
IBEFORE	34	0.53%
DURING_INV	1	0.0156%
Total	6418	

3.3. Browsing

The ability to examine annotated entities in a TimeML corpus is required as part of investigative research. To enable this, CAVaT includes the `browse` command.
Browsing allows the user to select a document (with `browse doc`, followed by a document ID or filename), and then view any tag within that document. Associated data is also shown; for example, if one browses an EVENT tag, any related MAKEINSTANCE tags will also be listed. One may view the tag in three formats – `screen`, the default; `csv`, as two rows of comma-separated values (the first with attribute names as column headings); or `timeml`, giving valid TimeML for the requested object. The syntax for these is the same as that for `show` commands; simply append `as <output type>` to the browse command.
The document selected for browsing is also used as the default document for checks, which are detailed in Section 2.3.

4. Example tasks

Below are some examples of using CAVaT to address real research problems. All are based on TimeBank v1.2.

4.1. Show all temporal links that employ a signal

As part of research toward better automatic TLINK annotation, we wanted to know what proportion of TLINKs in a corpus had been annotated as employing a signal.

```
cavat> show state of tlink signalid
  Count  State of Tlink signalid
 ========================================
    718  signalid filled   (11.2%)
   5700  signalid unfilled (88.8%)
```

The `state` keyword here treats signalID as having two states – filled or unfilled. The TLINK's `signalid` field will either be empty/absent or contain a reference to a signal annotated in text; for this task, we do not care which specific signal is being referenced.

Table 4: Distribution of Event part-of-speech

Event pos	Frequency	Proportion
VERB	5122	64.5%
NOUN	2225	28.0%
OTHER	299	3.77%
ADJECTIVE	266	3.35%
PREPOSITION	28	0.353%
Total	7940	

Table 5: Distribution of Tlink signal text when Reltype is "before"

Signal text	Frequency	Proportion
before	24	31.6%
Previously	10	13.2%
by	7	9.21%
already	6	7.89%
Earlier	6	7.89%
until	5	6.58%
then	4	5.26%
followed by	2	2.63%
prior to	2	2.63%
Other signals, frequency 1	10	13.2%
Total	76	

4.2. Dealing with ambiguous "part of speech" values

Many instances of events in TimeBank assert `pos="other"`. This is a problem when, e.g., using WordNet to lemmatise event strings. The distribution in Table 4 can be created with the command:

```
cavat> show distribution of event pos
```

After this, we would like to view event text that is classified as `other`, using the following query:

```
cavat> show list of event text where pos is other
#10.86
#39.8 million
#54.8 million
$1
$1.05
(truncated)
```

The result suggests that there are at least some numeric values for these event tokens, as well as the more typical verbs. This led to the substitution of all currency and numeric event strings with representative tokens, as a feature for a CRF classifier, yielding a performance increase in TLINK classification (in unpublished results).

4.3. Which signals does the BEFORE relation use?

Sometimes, particular relation types are strongly suggested by related signals. To determine the signal texts used with BEFORE TLINKs, one may query:

```
cavat> show distribution of tlink signaltext where
reltype is before
```

From the results in Table 5, we can see that the token "before" suggests a BEFORE relation, but that the majority of annotated BEFORE relations do not employ this signal (from Table 3, there are a total of 1408 such relations, only 24 of which use the signal). This indicates that building a relation classifier that relies solely on such signals will not be useful.

4.4. Superfluous TLINK checking

One may want to find instances where a link has been made between an entity and itself. We have an error checking module for this, `tlink_loop`:

```
cavat> check tlink_loop in WSJ910225-0066.tml
TLINK ID l383 matches, type IS_INCLUDED, event ei1482
TLINK ID l376 matches, type AFTER, event ei1454
TLINK ID l345 matches, type AFTER, event ei1356
```

One can explicitly query `in all` to search the entire corpus for similar mis-annotations.

5. Validation of a sample corpus

As we can now load and process any TimeML corpus, and have a set of advanced validation tests, it is logical to test existing TimeML annotated corpora and examine them. In this section, we present the results of running CAVaT's check modules on TimeBank v1.2. This corpus is not new and has been amended and improved by the community (Boguraev et al., 2007), so may contain many fewer errors than freshly annotated documents.

5.1. Checking for loops

We used the `tlink_loop` module (Section 2.3.3.) on the corpus. This identifies TLINKs where both arguments are the same event or event instance.

Of TimeBank's 183 documents, 19 have at least one TLINK containing such a loop, and there are 26 loops in total. Of these loops, 10 are on TLINKs of type SIMULTANEOUS or IDENTITY. Such TLINKs will not make a graph inconsistent, but are certainly redundant. The remaining 16 loops of other types will cause an inconsistency. All but one of the loops found are temporal links where both arguments reference the same event instance; only one references two different instances of the same event (TLINK L23, in document ABC19980304.1830.1636.tml). The TimeML in question is as follows:

```
But they still have <EVENT eid="e28"
class="I_ACTION">catching</EVENT> up to do two hundred
and thirty four Americans have <EVENT eid="e30"
class="OCCURRENCE">flown</EVENT> in space, only twenty
six of them women.
<MAKEINSTANCE eventID="e30" eiid="ei286" tense="PRESENT"
aspect="PERFECTIVE" polarity="POS" cardinality="234"
pos="VERB"/>
<MAKEINSTANCE eventID="e30" eiid="ei288" tense="PRESENT"
aspect="PERFECTIVE" polarity="POS" cardinality="26"
pos="VERB"/>

<TLINK lid="l23" relType="INCLUDES"

eventInstanceID="ei286" relatedToEventInstance="ei288"/>
```

In this case, the annotation suggests that during the flying in space of 234 Americans, 26 women flew, which is a correct interpretation of the text. CAVaT recommends the manual examination of `eventID` loops upon their detection. All the other tags reported by this check indicate redundant or incorrect annotations.

5.2. Checking for consistent graphs

Since the consistency checker uses a novel method (see Section 2.3.1), we verified its output by comparing it with that of SputLink and CAVaT's loop detection, and finding explanations for every inconsistency. A small test corpus of TimeML documents is also included with CAVaT for assuring the accuracy of this tool.

SputLink would not report an inconsistency with a TLINK loop that was not of type SIMULTANEOUS or IDENTITY; many of the inconsistent documents were found faulty by both SputLink and CAVaT. Some documents had an erroneous initial TLINK configuration; most faults were subtler than this, and their discovery required a closure attempt.

5.3. Checking for split graphs

The `split_graph` module checks for documents whose temporal graphs contain sets disconnected TLINKs. No single document in TimeBank has a fully-connected temporal graph, with a path traceable between every interval. The "best-connected" document (least fractured) is `wsj_0144.tml`, which has 34 intervals split into only two subgraphs; one containing 32 intervals, the other two.
The most fractured document is `wsj_1033.tml`, which is split into 12 sub-graphs having a mean graph size of only 2.7 intervals (a single TLINK connected to no other creates a graph of size 2). Despite having 32 intervals in total to connect, the largest sub-graphs in this document include only 4 intervals.

5.4. Replication

The results above can be simply replicated by downloading CAVaT v1, gathering a copy of TimeBank v1.2[5], importing the `data/timeml/` subdirectory of TimeBank, and running `check` *`test`* `in all` in CAVaT, where *`test`* is the name of the desired test module.

6. Conclusion and future work

We have described CAVaT, a language-independent tool which adds a layer of abstraction between TimeML markup and human researchers, making data easier to analyse, and patterns easier to spot. It also helps identify trouble spots in annotations.
TimeML corpora are only available at this time in Romanian (Forăscu et al., 2007) and English; this makes multilingual testing of the tool difficult. However, the markup is not language-specific, and results are likely to be equally useful across many languages; this may be shown using test corpora released for TempEval 2[6], which will include English, Italian, Spanish, Chinese, Korean and French.

Future work CAVaT may be able to provide repair suggestions. These may include fixes for inconsistent graphs, as well as suggestions for missing fields based on lexical resources, third-party tools or heuristics. The modular error checks allow creation of an open database of TimeML validations, to help improve the integrity of all TimeML corpora. Check modules that match the output of rule-based high confidence tools such as S2T (Verhagen and Pustejovsky, 2008) can be added.
The consistency checker is "a TimeML closure engine that uses the precise relations behind the scenes" (Verhagen, 2005). Therefore, it may be used to empirically discover how often incorrect links are introduced in closure, when compared with the existing leading closure tool, SputLink.

[5]LDC catalogue number LDC2006T08

[6]http://www.timeml.org/tempeval2/

Acknowledgements The first author would like to acknowledge the UK Engineering and Physical Science Research Council for support in the form of a doctoral studentship, and Marc Verhagen of Brandeis University for useful comments on the temporal closure process.

7. References

J.F. Allen. 1983. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11).

B. Boguraev, J. Pustejovsky, R. Ando, and M. Verhagen. 2007. Timebank evolution as a community resource for timeml parsing. *Language Resources and Evaluation*, 41(1):91–115.

L. Burnard and T. Dodd. 2003. Xara: an XML aware tool for corpus searching. In *Proceedings of Corpus Linguistics*, pages 142–4.

C. Forăscu, R. Ion, and D. Tufiş. 2007. Semi-automatic Annotation of the Romanian TimeBank 1.2. In *Proceedings of the RANLP Workshop on Computer-aided language processing*, pages 978–954.

ISOTimeML Working Group. 2008. 24617-1: 2008. *Language resources management–Semantic annotation framework (SemAF)–Part1: Time and events*. ISO/TC 37/SC 4/WG 2.

M. Hepple, A. Setzer, and R. Gaizauskas. 2007. USFD: preliminary exploration of features and classifiers for the TempEval-2007 tasks. In *Proceedings of the 4th International Workshop on Semantic Evaluations*, pages 438–441. Association for Computational Linguistics.

M. Lapata and A. Lascarides. 2006. Learning sentence-internal temporal relations. *Journal of Artificial Intelligence Research*, 27(1):85–117.

I. Mani, M. Verhagen, B. Wellner, C.M. Lee, and J. Pustejovsky. 2006. Machine learning of temporal relations. In *Proceedings of the 44th annual meeting of the Association for Computational Linguistics*, page 760.

J. Pustejovsky, P. Hanks, R. Sauri, A. See, R. Gaizauskas, A. Setzer, D. Radev, B. Sundheim, D. Day, L. Ferro, et al. 2003. The TimeBank corpus. In *Corpus Linguistics*, pages 647–656.

A. Setzer, R. Gaizauskas, and M. Hepple. 2005. The role of inference in the temporal annotation and analysis of text. *Language Resources and Evaluation*, 39(2):243–265.

E.P.K. Tsang. 1987. The consistent labeling problem in temporal reasoning. In *Proc. AAAI Conference, Seattle*, pages 251–255.

M. Verhagen and J. Pustejovsky. 2008. Temporal processing with the TARSQI toolkit. *Proceedings of CoLing: Posters and Demonstrations*, pages 189–192.

M. Verhagen. 2005. Temporal closure in an annotation environment. *Language Resources and Evaluation*, 39(2):211–241.

M. Vilain, H. Kautz, and P. Van Beek. 1989. Constraint propagation algorithms for temporal reasoning: A revised report. *Readings in qualitative reasoning about physical systems*, 373:381.