FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

TEMPORAL ANALYSIS OF NARRATIVES: TIMELINES, TIMEML EVALUATION, AND DURATIONS

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Mustafa Ocal

2022

To: Dean John L. Volakis
College of Engineering and Computing

This dissertation, written by Mustafa Ocal, and entitled Temporal Analysis of Narratives: Timelines, TimeML Evaluation, and Durations, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Monique Ross

Giri Narasimhan

Leonardo Bobadilla

Selcuk Uluagac

Mark A. Finlayson, Major Professor

Date of Defense: September 19, 2022

The dissertation of Mustafa Ocal is approved.

Dean John L. Volakis
College of Engineering and Computing

Andrés G. Gil
Vice President for Research and Economic Development
and Dean of the University Graduate School

Florida International University, 2022

DEDICATION

I would like to dedicate this dissertation to my mother Gulnur Ocal, my father Ali Ocal, and my brother Osman Ocal, who gave everything they could to ensure I would have the opportunity of an education. Their love and support have allowed me to pursue my dreams.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Mark Finlayson, not only for his guidance and encouragement but also for believing in me. Without his enormous help and support, I wouldn't have reached these achievements. I would also like to thank my *minions* Adrian Perez, Antonela Radas, Jared Hummer, and Akul Singh for all of the help they provided. I had the honor of mentoring these brilliant undergraduate researchers without whom this work would not be possible. I am especially grateful to my two colleagues and friends, Dr. Deya Banisakher and Dr. Labiba Jahan who made my Ph.D. journey less stressful with their love and support. I am also thankful to my labmates Dr. Victor Yarlott, Dr. Mohammed Aldawsari, Dr. Anurag Acharya, and Dr. Mireya Jurado for their help and support. I'm grateful to Dr. Karine Megerdoomian for the helpful and valuable advice I have received throughout my doctorate. I would also like to thank the jTLEX capstone project team members Emmanuel Garcia, Luis Robaina, Ismael Clark, Franklin Bello Romero, Victoria Fernandez, Kevin Fontela, Adrian Silva, Ronald Pena, Felipe Arce, Raul Garcia, Leandro Estevez, Carlos Pimentel, Hector Borges, Ivan Parra Sanz, Tony Erazo, Sage Pages, and Gerardo Parra, for their contributions to the jTLEX Java library. Finally and most importantly, I would like to thank Amanda Pena for being on my side and supporting me to chase my dreams.

ABSTRACT OF THE DISSERTATION

TEMPORAL ANALYSIS OF NARRATIVES: TIMELINES, TIMEML EVALUATION, AND DURATIONS

by

Mustafa Ocal

Florida International University, 2022

Miami, Florida

Professor Mark A. Finlayson, Major Professor

Narratives contain a lot of temporal information. To capture the temporal information in texts, natural language processing researchers developed TimeML, the temporal markup language to annotate temporal information. Temporal graphs can be derived directly from TimeML annotations and can reveal partial ordering of events and times. However, for many purposes, a global order (timeline) is more useful.

The first component of my work focused on timeline extraction from TimeML annotations. Prior approaches have presented machine learning-based systems, which have certain limitations such as imperfect scores, ignoring subordinated relations, and being unable to handle all types of temporal relations. I addressed these issues and presented a constraint satisfaction problem-based solution that achieved state-of-the-art performance.

One way to generate TimeML annotation in texts is to perform manual annotation. However, manual annotations contain human-made errors. In the second component of my work, I built a system to detect errors in the gold-standard annotations and to help users fix them. I tested the system on the TimeBank corpus and provided corrections for the entire corpus.

Another way to generate TimeML annotations is to use automatic annotation systems. In the third component of my work, I developed a novel suite of methods to evaluate the performance of automatic annotators that measures the information loss during the

automatic annotation process. I presented eight metrics and evaluated four state-of-the-art automatic annotation tools.

In the last component, I successfully implemented a duration extraction system. This work resulted in a large dataset that contains hundreds of thousands of possible event durations. Combining this work with the timeline extraction system, I was able to extract the duration of entire narratives.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE PAGE

CHAPTER 1

INTRODUCTION

1.1 Motivation

Narratives are part of our lives. We can see narratives in many fields such as medicine, law, psychology, cognitive science, and artificial intelligence. Genette et al. (1982) defines narrative as "the representation of an event or of a sequence of events". Because narrative is a sequence of events, one important way to understand narratives is to interpret the events' order (i.e. timeline). Timelines are useful for many natural language understanding tasks such as question-answering, narrative summarization, and narrative representation. However, timelines are not explicit in texts and cannot be directly read off from texts. Instead, texts reveal partial orderings of events and times. Such information can be used to construct a temporal graph using a temporal representation language (e.g. TimeML). The main goal of this dissertation is to extract timelines from TimeML temporal graphs (Chapter 3).

One way to generate TimeML annotation on documents is annotating raw texts manually according to the TimeML annotation guide (Sauri et al., 2006). However, this process may include human-made errors. Gold-standard TimeML annotations have been used in Natural Language Processing (NLP) research, including event detection (Saurí et al., 2005; Färber and Rettinger, 2015), temporal expression recognition (Chang and Manning, 2013; Strötgen and Gertz, 2010), and temporal relation extraction (Mirroshandel and Ghassem-Sani, 2011; Mirza and Tonelli, 2016). If the manual annotation contains errors, it may affect all the work that uses it. The second main goal is to evaluate gold-standard TimeML annotations for errors and fix them (Chapter 4).

TimeML document annotation can also be achieved using state-of-the-art automatic TimeML annotators (Verhagen et al., 2005; Chambers et al., 2014; Bethard, 2013). Al-

though automatic TimeML annotators have reported good F1 scores for each individual subtask of TimeML annotation such as event detection, temporal expression recognition, and relation extraction, the overall performance combining these subtasks hasn't been tested. The third main goal is to evaluate automatic TimeML annotators and measure the information loss during the process (Chapter 5).

It is also crucial for narrative understanding is to estimate a narrative's duration, meaning the time that passes within the narrative. If I have an approximation of the lower bound and upper bound duration of each event in a narrative, combining these durations with the timeline of the narrative, I can estimate the duration of the narrative. The fourth and final goal is to generate a large dataset that consists of duration statistics of events (Chapter 6).

1.2 Problem Statement and Research Components

My research problem is to perform a temporal analysis of narratives, including four research components: timeline extraction, gold-standard TimeML evaluation, TimeML annotator evaluation, and duration extraction.

1.3 Component 1: Timeline Extraction

The first step of the temporal analysis of narratives is to extract their timeline. Timelines are not explicit in texts and cannot be extracted directly. Texts reveal partial information that can be used to construct a temporal graph using TimeML. There have been prior machine learning-based approaches to extracting timelines from TimeML annotations (Mani et al., 2006; Do et al., 2012; Kolomiyets et al., 2012). However, these approaches have limitations. For example, they do not handle all possible temporal relations, they ignore subordinated events, and they have imperfect scores. Addressing these problems, I built

TimeLine EXtraction (TLEX), a constraint satisfaction problem (CSP)-based solution to extract the *exact* timeline from TimeML annotations. TLEX takes a TimeML graph as input and first, it partitions the TimeML graph into temporally connected graphs. Second, it transforms the temporally connected graphs into point algebra (PA) graphs. Third, it solves the PA graph by assigning real numbers for each node on the PA graph using CSP solvers and sorts the real numbers to obtain a global order of events and times (i.e. a timeline). Fourth, TLEX detects inconsistency and helps annotators fix them in the case when a timeline cannot be extracted due to temporal inconsistency. Fifth and finally, TLEX detects temporal indeterminacy if the global order of events and times is indeterminant. I provided a formal proof of TLEX's correctness, and I also conducted an experimental evaluation by applying TLEX to 385 TimeML annotated texts from four corpora. I showed that TLEX achieved state-of-the-art performance (Chapter 3).

1.4 Component 2: Evaluation of Gold-Standard TimeML Annotations

The input of the TLEX algorithm is TimeML annotation. One way to generate TimeML annotation is to annotate raw texts manually according to the TimeML annotation guide (Sauri et al., 2006). The TimeBank corpus is a reference corpus for TimeML and contains rich gold-standard TimeML annotations. TimeBank has been used for many NLP research projects and if TimeBank contains human-made errors, it may affect all of the work that uses it. For the second step of my dissertation, I built a suite of methods to detect and correct redundant, missing, and incorrect annotations. First, I performed an automatic guideline checking where I checked whether the annotations follow the strict TimeML annotation guide rules. Second, my system automatically checked the consistency of annotations. Third, it checked disconnectivity in TimeML graphs extracted from

annotations. Fourth and finally, I performed a manual comparison with the output of state-of-the-art automatic annotators to identify missing annotations.

1.5 Component 3: Evaluation of Automatic TimeML Annotation Tools

Another way to generate TimeML annotation is using state-of-the-art automatic TimeML annotators (Verhagen et al., 2005; Bethard, 2013; Chambers et al., 2014). Although automatic TimeML annotation is challenging, there has been notable progress, with F1s of 0.8–0.9 for events and time detection subtasks, and F1s of 0.5–0.7 for relation extraction. Individually, these subtask results are reasonable, even good, but when combined to generate a full TimeML graph, is overall performance still acceptable? For the third step of my dissertation, I present a novel suite of eight metrics, combined with a new graph-transformation experimental design, for a holistic evaluation of TimeML graphs. I apply these metrics to four automatic TimeML annotation systems: TARSQI (Verhagen et al., 2005), ClearTK (Bethard, 2013),CAEVO (Chambers et al., 2014), and CATENA(Bethard, 2013).

1.6 Component 4: Duration Extraction

As I mentioned earlier, by combining event duration with the timeline of a narrative, I can estimate the duration of the entire narrative. Therefore, for the fourth and final part of my dissertation, I built a pipeline structure to extract possible duration for each event from large raw data to build a large dataset that contains event duration statistics. My pipeline has four steps: First, it performs preprocessing by cleaning the data and splitting sentences. Second, it detects temporal expression in each sentence. Third, using a rule-

based system, it mines temporal patterns (10 patterns). Finally, it performs temporal reasoning to obtain minimum, maximum, and most likely duration for each event.

1.7 Dissertation Contributions

My work presented in this dissertation has the following major contributions, resulting from the four research components.

Timeline Extraction

My contributions in this work are eight-fold. **First**, I introduced a new trunk-and-branch multi-timeline structure for organizing TimeML temporal information. This structure could be used in any domain that uses narrative summarization and representation. **Second**, I demonstrated (and released my code for) TLEX, an approach to extracting these timelines that uses all available information in the TimeML graph, and identifies indeterminate sections of the timelines. With this, multiple different temporal orderings can be represented in a single timeline, which helps for narrative visualization. **Third**, I demonstrated a new technique for identifying the specific links that contribute to temporal inconsistency, and used it to make corrections to four TimeML corpora that restore the consistency of the annotations. This technique can be used for corpus validation in the TimeML annotation task. **Fourth**, I provided a formal proof for the TLEX's correctness, including formal definitions of a number of objects and relationships of interest, in particular *timelines*, including *normal-form* and *shortest* subtypes; *tempomorphisms* and *tempomorphic equivalence*; *indeterminacy maps*; and *trunk-and-branch timelines*. These formal proofs and definitions connect the Quantitative Spatial Reasoning (QSR) work with the NLP work and help to explore how QSR techniques can be applied to the NLP tasks. **Fifth**, I experimentally evaluated TLEX using Simple Random Sampling (SRS)

on five features, and observed that TLEX achieved 98–100% accuracy with 95% confidence on all measures. I further identified several common classes of errors observed in the corpora under study. TLEX represents not just a significant improvement over the state of the art but also a theoretically exact solution to the problem of timeline extraction from non-metric temporal graphs. **Sixth**, I released a Java library of the TLEX algorithm (jTLEX) as an open-source library that is free for non-commercial use. Therefore, the TimeML community can use the TLEX algorithm on their TimeML annotations. **Seventh**, I presented an algorithm that transforms temporal graphs into full or abstracted Temporal Dependency Trees (TDT). This allows a direct comparison of TimeML and TDT formalisms because although there are several corpora with TimeML annotations, there are as yet no corpora annotated with both TimeML graphs and TDTs. This also allows the TimeML community to extract TDTs from their TimeML annotation without having any temporal dependency parsers. **Eighth**, by transforming TimeML graphs and TDTs into timelines, I showed that the TDTs are significantly more temporally indeterminant than TimeML graphs: anywhere from 24% to 109% more indeterminate depending on the corpus (average increase of 32%). This increase in indeterminacy is attributable to the omission of a mere 2.4% of temporal relations from TDTs. This suggests that in the NLP tasks where information on the global ordering of events and times is important, full TimeML representation is perhaps called for.

Gold-Standard TimeML Annotation Evaluation

My contributions in this work are three-fold. **First**, I presented a comprehensive evaluation of the TimeBank corpus comprising 10 different automatic or semi-automatic checks. This evaluation method can be used for correcting annotations for the TimeML community. **Second**, I showed that the TimeBank corpus has 1,630 incorrect or missing annotations. This suggests that the systems that use TimeBank to detect events, recognize times,

and extract relations could be potentially affected and their accuracy might be lower than reported. **Third**, I provided corrections for all TimeBank corpus incorrect or missing annotations and I released patch files as well as my code for use by other NLP researchers in the field.

TimeML Annotator Evaluation

My contributions in this work are three-fold. **First**, I presented eight metrics for evaluating the quality of temporal graphs—four graph-based and four timeline-based—and used these metrics to evaluate four mainstream, state-of-the-art temporal analysis systems. This helps NLP works that use these systems (or NLP researchers who consider using them) by showing the information loss during the annotation process. Also, these metrics can be used as an alternative accuracy measurement in the temporal information extraction tasks. **Second**, I showed that, beyond a certain point, small errors in the detection of events, times, or relations result in rapid degradation of the consistency of the final graph. This shows the NLP community how small errors in the subtasks can dramatically affect the overall performance. **Third**, I showed that current automatic TimeML annotators are far from optimal, and significant further improvement is needed. I released the code and data to enable the reproduction of the results for other NLP researchers.

Duration Extraction

I made two major contributions in the area of duration extraction. **First**, I presented a suite of methods to extract the duration of events from raw data. **Second**, I created a large dataset that contains event durations and I will release it as open source. This dataset provides an extensive source for commonsense knowledge extraction tasks.

1.8 Outline

This dissertation is organized as follows: **First**, I discuss the background information, datasets, and prior work that is related to this dissertation (§2). **Second**, in Chapter 3, I present my approach to extracting timelines from TimeML graphs, called TLEX. I provide mathematical proofs for TLEX (§3.3) as well as experimental evaluation (§3.4). I also discuss the Java library that I created for TLEX (§3.5) and the TLEX application to compare temporal data structures (§3.6). **Third**, I present a suite of methods to find errors in gold-standard TimeML annotations and fix them (§4). **Fourth**, I describe a novel suite of eight metrics, combined with a new graph-transformation experimental design, for the holistic evaluation of automatic TimeML annotators (§5). **Fifth**, I discuss a pipeline that extracts the duration of events from large raw data (§6). **Sixth and finally**, I end with a conclusion that revisits contributions and results for each research component and discuss the future directions of the work presented in this dissertation (§7).

CHAPTER 2

RELATED WORK

In this chapter, I discuss related work. First, I provide background information about temporal analyzing tasks such as temporal algebra (§2.1), TimeML and TimeML annotated corpora (§2.2), automatic TimeML annotators (§2.3), and temporal data structures (§2.4). Then, I review the prior work on timeline extraction (§2.5), TimeML evaluation (§2.6), and duration extraction (§2.7).

2.1 Temporal Algebra

One important part of understanding narratives is the interpretation of the order of the events. This interpretation first requires temporal algebra. Allen's interval algebra (Allen, 1983) is a calculus for temporal reasoning that defines possible relations between time intervals and the inferences sanctioned by combinations of relationships. Allen's algebra was one of the first attempts to computationally model temporal relationships between time intervals and is useful for describing the temporal relationships expressed in text. In Allen's approach, a time interval I comprises a start (I^-) and end time point (I^+), where the start point comes strictly before the end point ($I^- < I^+$). Intervals can be related by disjoint sets of 13 primitive temporal relations: *before*, *meets*, *overlaps*, *starts*, *during*, and *finishes*, their inverses, and *equal*. Allen presented a composition table for composing pairs of temporal relations, which can be used to infer the temporal relationship between intervals A and C given relationships between A and B and B and C. Using Allen's algebra, temporal graphs can be constructed from texts by representing events and times as intervals, and representing the relationships expressed in natural language using Allen's temporal relations. Allen's approach is referred to as a *qualitative* temporal algebra, because it does not represent exact times or durations.

Another qualitative framework is the point algebra (PA), which is a calculus for determining qualitative ordering constraints between time points (Barták et al., 2014, §2). PA defines only three possible primitive relations between a pair of time points: '$<$' if the first time point is before the second time point, '$=$' if the time points are identical, and '$>$' if the first time point is after the second time point. This information can be used to construct a PA graph, which is a graph where nodes are time points and edges are primitive temporal constraints. A temporal graph can be transformed into a PA graph with the correct mapping between Allen's temporal relations and temporal primitive constraints. This specific transformation is discussed in §3.2.2.

In a great deal of later work, the qualitative framework was generalized and extended to the quantitative frameworks in which time points and durations are given specific metric values. These frameworks are namely Simple Temporal Problems (STPs), Temporal Constraint Satisfaction Problems (TCSPs), Disjunctive Temporal Problems (DTPs), and Temporal Networks with Alternatives (TNAs). These types of temporal frameworks allow precise reasoning about the temporal distance between time points as represented in graphs. While quite useful for planning and scheduling problems, quantitative frameworks are less useful for natural language text (especially items like narratives and news), which usually do not contain a great deal of precise information about the duration of temporal intervals or relationships.

Theorists have proved a number of formal results concerning both qualitative and quantitative formalisms (reviewed in Barták et al., 2014). Of particular importance are those results related to checking the *consistency* of temporal graphs. Whether quantitative or qualitative, temporal graphs often need to be checked for consistency. To check the consistency of graphs constructed using his algebra, Allen developed an algorithm that uses the shortest path algorithm and the composition table to check path consistency (Allen, 1983). Like shortest path algorithms, this approach is a polynomial-time algo-

rithm; however, Vilain et al. (1990) showed that Allen's algorithm is not sufficient to ensure global consistency, and they showed by reduction to 3-SAT that complete checking of the consistency of a qualitative temporal graph is NP-complete.

It was also shown that Allen's interval algebra was just a specific example from a larger set of interval algebras, varying in the types of relations and the number of relations and their combinations allowed between intervals. Nebel and Bürckert (1995) introduced the ORD-HORN subclass, which is the subclass of all *pointisable* interval algebras: a *pointisable* interval algebra is an interval algebra whose consistency can always be shown by polynomial-time path consistency algorithms. A temporal graph that has only a single temporal relation on its edges is a member of the ORD-HORN subclass.

Many other theoretical results also pertain to *solving* a temporal graph (Barták et al., 2014; Derczynski and Gaizauskas, 2010), which means assigning specific time values to every time point in the graph (start and end points of all intervals). Such an assignment can be computed by treating this as a constraint problem, and is formally the same as extracting a timeline. Importantly, if a graph can be solved, it must be *consistent*; but showing that temporal graph is consistent does not necessarily produce an explicit solution.

2.2 TimeML & Annotated Corpora

Because of the utility of temporal frameworks for reasoning about time, and also the relevance of time to understanding natural language, researchers in NLP have sought to apply temporal algebras to text understanding. To do this requires annotation schemes that would allow a person or a machine to annotate a text for the time points, events, and temporal relations expressed.

With regard to time expressions themselves, which include expressions of when something happens, how often something occurs, or how long something takes, researchers developed a sequence of TIMEX annotation schemes (Setzer, 2001; Ferro et al., 2001; Pustejovsky et al., 2003a). This allows the annotation of expressions such as *at 3 p.m.* (when), *every 2 days* (how often), or *for 1 hour* (how long). Because events are also involved in temporal relations, these approaches were extended into schemes for capturing both times and events.

The Translingual Information Detection, Extraction, and Summarization scheme - TIDES - (Ferro et al., 2001) integrates TIMEX2 expressions as well as a scheme for annotating events. TIDES includes annotations for temporal expressions, events, and temporal relations. TIDES uses only six temporal relation types to represent the relationship between events, therefore it gives only a limited view of temporal information from texts.

Deficiencies in TIDES led to the development of TimeML (Sauri et al., 2006), another markup language for annotating temporal information, originally designed for news. TimeML added facilities for representing not just Allen's classic temporal relations but also added event coreference (i.e., the *identity* relation), as well as relations for expressing sub-event structure (aspectual relations) and relations of conditional, hypothetical, or counterfactual nature (subordinating relations).

TimeML has three different types of links: temporal (TLINK), aspectual (ALINK), and subordinating (SLINK). TLINKs comprise 14 different types of temporal relationships between events and times. Figure 2.1 graphically shows all types of TLINKs in TimeML. The following example represents one of the TLINKs, called IS_INCLUDED. This TLINK represents that the event (*went*) is included in (*on*) the time (*Monday*).

(1) Kai **went** to the school on **Monday**. (IS_INCLUDED)

ALINKs represent the relationship between an aspectual event and its argument event. There are five types of ALINKs (shown in Figure 2.1): INITIATES, REINITIATES, TERMINATES, CULMINATES, and CONTINUES. The following example shows the aspectual relationship between events, where the first event (*started*) INITIATES the second event (*study*).

(2) Mike **started** to **study**. (INITIATES)

SLINKs are used for contexts introducing possible (modal), counterfactual, or conditional relations between two events, spanning six types: MODAL, FACTIVE, COUNTER_FACTIVE, EVIDENTIAL, NEGATIVE_EVIDENTIAL, and CONDITIONAL. The MODAL relation introduces a relation to a possible event. In the following example, the event (*buy*) is a mere possibility and has not actually happened yet.

(3) Cindy **promised** him to **buy** some nachos. (MODAL)

The FACTIVE relation marks an entailment of an event's veracity. On the other hand, a COUNTER_FACTIVE relation marks a presupposition about the event's non-veracity. In other words, FACTIVE indicates a presupposition as to whether the event happened in the real world and COUNTER_FACTIVE indicates a presupposition as to whether the event did not happen in the real world. The following examples show the difference between these two SLINKs.

(4) Katy **forgot** that she **was** in Miami last year. (FACTIVE)

(5) Katy **forgot** to **buy** some chocolate. (COUNTER_FACTIVE)

EVIDENTIAL relations are introduced by reporting events asserting that the argument event happened. Similarly, NEGATIVE_EVIDENTIAL relations are introduced by reporting

events asserting that the argument event did not happen. The following examples show these two SLINK types.

(6) Colin **said** he **went** to the store. (EVIDENTIAL)

(7) Darius **denied** that he **has monkeypox**. (NEGATIVE_EVIDENTIAL)

Finally, the CONDITIONAL relationship identifies two events linked in a conditional manner, for example, with a signal such as "if".

(8) If Amanda **marries** him, she will **be happy**. (CONDITIONAL)

SLINKs do not necessarily entail a temporal relationship, but if they do, a TLINK representing that relationship should be added to the graph as well. Furthermore, three types of SLINKs explicitly indicate that the event is presumed to not have happened in the "real world" of the text. Therefore, ignoring subordinating relations or treating them as normal temporal relations gives an incorrect and impoverished view of the temporal structure of the text.

Out of the three types of TimeML links, only ALINKs and TLINKs represent temporal information in documents, while SLINKs indicate non-temporal relationships. SLINKs and ALINKs can only be present between two events; TLINKs, on the other hand, can relate two events, two times, or an event and a time.

There are two ways to generate TimeML annotations: using automatic TimeML annotators to generate them automatically (discussed in Section 2.3), and manually annotating TimeML in documents following the TimeML annotation guide (Sauri et al., 2006).

There are a number of manually annotated TimeML corpora. In this work, I used four manually annotated TimeML corpora: TimeBank 1.2, the N2 corpus, the ProppLearner corpus, and the NewsReader MEANTIME corpus, all in English. The number of texts,

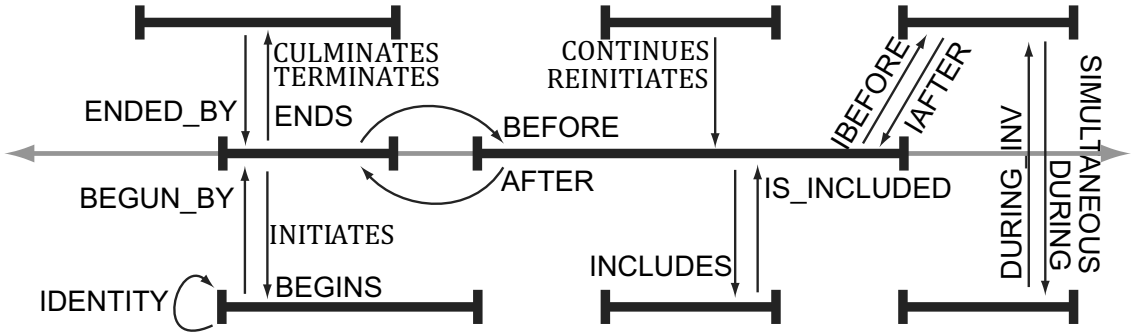

Figure 2.1: The 14 temporal and 5 aspectual TimeML link types. Temporal links can hold between two events, an event and a time, and two times. Aspectual links hold between an aspectual event and another event.

Corpus Name	**Texts**	**Words**	**Events**	**TIMEXs**	**Links**	**Text Types**
ProppLearner	15	18,862	3,438	142	2,778	Russian Hero tales
N2 Corpus	67	28,462	2,345	349	4,854	Religious Texts, Magazine
MEANTIME	120	13,981	2,096	525	1,717	Wikinews Articles
TimeBank 1.2	183	68,555	7,935	1,414	9,615	Newswire, Biography, etc.
Total	385	129,860	15,814	2,430	18,964	

Table 2.1: Summary of the corpora used in the experiments.

words, events, temporal expressions, and relations are listed in Table 2.1. TimeBank 1.2 is drawn from various American news sources such as ABC, CNN, and the Wall Street Journal (Pustejovsky et al., 2003b). The ProppLearner corpus was developed to enable the machine learning of Vladimir Propp's morphology of Russian hero tales and has 18 different layers of syntax and semantics annotated on it, including TimeML (Finlayson, 2017). Similarly, the N2 corpus is a collection of narratives relating to Islamic Extremism with 14 layers of annotation including TimeML (Finlayson et al., 2014). Finally, the NewsReader MEANTIME corpus is a semantically annotated corpus of English Wikinews articles (Minard et al., 2016). The corpus also includes TimeML annotations of articles' translations in Italian, Spanish, and Dutch), covering four news topics: "Airbus and Boeing," "Apple," "the Stock market," and "General Motors, Chrysler, and Ford."

2.3 Automatic TimeML Annotators

As I mentioned in §2.2, TimeML annotation can be achieved by manually annotating raw texts. However, this is extremely time consuming. To avoid that, NLP researchers have worked on automatic TimeML annotation. Automatic TimeML annotation from texts includes three subtasks: temporal expression (time) detection, event detection, and TimeML link extraction. There has been quite a lot of work addressing each of these subtasks individually (Verhagen et al., 2006; Seker and Diri, 2010; Lenzi et al., 2012, for example), but there are only a few systems that can provide integrated capabilities. Systems that tackle an individual subtask include the following:

Time Expressions HeidelTime recognizes temporal expressions using regular expressions as well as part-of-speech (POS) tags and handcrafted rules, achieving 0.86 F_1 (Strötgen and Gertz, 2010). SUTime, part of the Stanford CoreNLP pipeline, recognizes and normalizes times in documents using regex rules, POS tags, and named entity tags, and achieves 0.92 F_1 (Chang and Manning, 2013). SynTime is also a rule-based time recognition tool, additionally using token types to achieve 0.92 F_1 (Zhong et al., 2017). PTime generates patterns and selects them using the Extended Budgeted Maximum Coverage (EBMC) model, achieving 0.93 F_1 on tweets and 0.87 F_1 on a TimeML annotated corpus (Ding et al., 2019).

Events NavyTime detects events using POS tags, n-grams, lemmas, WordNet events, parse path, and typed dependencies with a MaxEnt Classifier, achieving 0.80 F_1 (Chambers, 2013). Sprugnoli and Tonelli (2019) detect events using a Conditional Random Field (CRF) classifier with lemma, POS tags, and text genre as features, achieving a 0.83 F_1 in historical texts. Multilingual Sequence Tagger (M-LiST) is a deep learning model that

uses word alignment, a feature encoder, and a sequence tagger and achieves 0.86 F_1 on event recognition (Goud et al., 2019).

Temporal Relations Galvan et al. (2018) presented a bidirectional Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) end-to-end neural model for medical domain texts to extract five types of TLINKs: BEFORE, BEGINS, INCLUDES, ENDS, and OVERLAP. The system uses event attributes as features and achieves 0.62 F_1. Ning et al. (2019) implemented a semi-supervised system combining a structured perceptron algorithm and constraint-driven learning (CoDL) to extract TLINKs from documents. The system achieves 0.67 F_1 on event to event relations on five TLINKs: BEFORE, AFTER, INCLUDES, IS_INCLUDED, and SIMULTANEOUS—notably, the system does not extract event to time or time to time relations.

Integrated Systems There are four systems that integrate multiple subtasks. TARSQI (Verhagen et al., 2005) takes raw texts as input and recognizes time expressions and normalizes their values using a component called GUTIME. TARSQI recognizes events using a component called EVITA, which is a domain-independent event tagger, and identifies SLINKs, ALINKs, and TLINKs between temporal entities using supervised classification models. TARSQI merges the links using its SPUTLINK component, which applies constraint propagation algorithms to obtain a consistent annotation. TARSQI generates all types of SLINKs and ALINKs. It also generates 9 types of TLINKs: BEFORE, AFTER, INCLUDES, IS_INCLUDED, SIMULTANEOUS, IDENTITY, BEGUN_BY, ENDS, and ENDED_BY.

CLEARTK (Bethard, 2013) ranked first for temporal relation identification on TempEval-2013, and comprises three modules. The first module identifies events in texts and determines their attributes using event text, stem, part-of-speech tags, n-grams, and other related features. The second module identifies time expressions as well as time type and

value using time-related features and the temporal type of each alphanumeric sub-token of time words. The third module predicts (only) TLINKs between each event and the Document Creation Time (DCT), between events and times in the same sentence, and between events in the same sentence. CLEARTK generates four types of TLINKs: BEFORE, AFTER, INCLUDES, and IS_INCLUDED.

CAEVO (Chambers et al., 2014) works similarly to TARSQI across four substeps: (1) time expression detection, (2) event detection, (3) temporal relation extraction, and (4) transitive inference over TLINKs. CAEVO uses the SUTIME to extract times, and NAVY-TIME to extract events. CAEVO then extracts (only) TLINKs between temporal entities in the same sentence and neighboring sentences using a supervised classifier. It also extracts TLINKs between the DCT and every temporal entity. Finally, it applies transitive rules on TLINKs to extract dense TimeML annotation. CAEVO produces five types of TLINKs: BEFORE, AFTER, INCLUDES, IS_INCLUDED, and SIMULTANEOUS.

Finally, CATENA (Mirza and Tonelli, 2016) is a state-of-the-art sieve-based system for temporal relation extraction that takes texts pre-annotated with times and events and generates TLINKs between all event-event, event-time, DCT-event, and DCT-time pairs using a Support Vector Machine (SVM) and a temporal reasoner. CATENA generates 10 types of TLINKs: BEFORE, AFTER, INCLUDES, IS_INCLUDED, SIMULTANEOUS, BEGINS, BEGUN_BY, ENDS, ENDED_BY, and DURING.

2.4 Temporal Data Structures

After a TimeML annotated file is parsed, the TimeML objects (such as events, time expressions, links, signals, etc.) need to be recorded in a data structure. For that purpose, NLP and QSR researchers developed temporal graphs and temporal trees.

Temporal Graph A temporal graph is a graph in which nodes are intervals (events and time expressions) and edges are temporal relations. Temporal graphs can be derived from texts that are annotated with temporal representation languages such as Allen's temporal algebra (Allen, 1983), TIDES (Ferro et al., 2003), or TimeML (Sauri et al., 2006). A TimeML graph is an example of a temporal graph and a visualization of a TimeML graph is shown in Figure 3.1 in Chapter 3. As can be seen in Figure 3.1, temporal graphs can be used for inference or determining partial orders of events and times.

Temporal Tree Temporal Trees were introduced due to deficiencies of temporal graph. For example, the generality of temporal graphs makes many operations on them computationally hard and they are less than ideal for visualization purposes, as they are difficult for people to read and understand. In response to these limitations, Cheng et al. (2007) introduced the idea of the TDT, a more computationally efficient representation where all events are arranged in a tree using only three temporal relation types: BEFORE, AFTER, and OVERLAPS. Both Kolomiyets et al. (2012) and Zhang and Xue (2018) further improved TDTs with more precise definitions and increased expressivity. TDTs have the advantage that they can be easily computed using adapted dependency parsers, and extracting timelines from TDTs is much easier than from temporal graphs.

Nevertheless, it is intuitive that TDTs should suffer from temporal information loss relative to temporal graphs. TDTs are restricted to a tree form, and so certain temporal relationships that can be expressed in a graph cannot be expressed in the tree (e.g., cycles). Furthermore, all TDT approaches restrict the types of temporal relationships, eliminating even more information. These omissions and restrictions should intuitively result in more indeterminacy in the global ordering of times and events. I investigate this problem in §3.6 and present a quantitative comparison between temporal graphs and temporal trees.

2.5 Timeline Extraction

There have been at least three efforts to provide timeline extraction for *quantitative* (i.e., metric) frameworks (Conte et al., 2014; Tran et al., 2015; Laban and Hearst, 2017). These approaches provide timeline extraction in the case of temporal networks with metric temporal information for all times, events, and relationships (i.e., numerical durations of nodes and edges in the temporal graph). Unfortunately, a solution for a quantitative framework does not immediately extend to a solution for qualitative cases such as Allen's algebra or TimeML. This is because qualitative frameworks explicitly allow the use of non-metric information in their temporal graphs; indeed, most of the temporal information in natural language is non-metric.

In the field of QSR, researchers have given significant attention to timeline extraction. van Beek (1994) presented an approach that transformed Allen's interval algebra temporal graphs into a point algebra network and solves them using backtracking techniques. The approach of Gereveni and Schubert (1995) built a "timegraph" given a set of point algebra relations and solved the timegraph using their own algorithm. Later, Wallgrün et al. (2006) presented the SparQ (Spatial Reasoning done Qualitatively) tool that comprises a set of modules to provide different services for qualitative spatial reasoning, which is a similar problem to qualitative temporal reasoning. SparQ transforms a quantitative description of a spatial configuration into a qualitative description, applies the operations in the calculi to spatial relations, and finally performs computations on constraint networks. Similarly, to solve binary qualitative constraint graphs, Gantner et al. (2008) built the Generic Qualitative Reasoner (GQR), which takes a temporal calculus description and one or more constraint graphs as input and solves them using path consistency and backtracking. To check the consistency of large qualitative spatial networks, Sioutis and Condotta (2014) presented Sarissa, which produces random scale-free-like qualitative spatial

networks using the Barabási-Albert (BA) model and uses a hash table-based adjacency list to efficiently represent and reason with them. Finally, Kreutzmann and Wolter (2014) showed how to use AND-OR linear programming (LP) and mixed integer linear programming (MILP) to solve qualitative graphs, a set of which includes graphs represented in Allen's algebra.

In contrast to QSR approaches, NLP researchers have explored machine learning methods for timeline extraction from TimeML annotated texts. Mani et al. (2006) presented a machine learning method to partially order events from a qualitative temporal graph generated using Allen's temporal relations. However, that method was demonstrated only with three relations—BEFORE, AFTER, and SIMULTANEOUS—and thus excludes large portions of TimeML and only achieves roughly 75% ordering accuracy. Do et al. (2012) used the same three relations to generate a full ordering, achieving a similar accuracy of 73%. In addition to the imperfect performance of these systems, in both cases, the methods only consider intervals, rather than start and end points, and so lose much detailed temporal information. Kolomiyets et al. (2012) presented two models for extracting timelines from qualitative temporal graphs, one based on shift-reduce parsing, and one based on graph parsing. Both approaches take a sequence of event words as input and produce a tree structure. Although they achieved 70% accuracy in event ordering, their work only deals with six temporal relations—BEFORE, AFTER, INCLUDES, IS_INCLUDED, IDENTITY, and OVERLAP—again, only a subset of the full possible set of temporal relations. Finally, Leeuwenberg and Moens (2020) combined event durations into their supervised machine learning-based system to extract timelines. However, the duration dataset that they used in their system has only a 44% inter-annotator agreement score and they only handle three temporal relations BEFORE, AFTER, and OVERLAP.

Summarizing these approaches for timeline extraction, it can be seen that on the one hand QSR approaches transform Allen's interval algebra graphs into constraint graphs and

solve them using constraint satisfaction techniques. Although these approaches provide a solution for solving qualitative temporal graphs, their methods cannot be applied directly to TimeML graphs because of the presence of subordinating relationships, and they also do not detect or represent indeterminacy, nor do they help with correcting inconsistencies. Machine learning-based approaches, on the other hand, have been limited by the number of relations considered, not least because of the noise inherent in a statistical solution, and provide inexact solutions.

2.6 Evaluation of TimeML Annotation

Automatic TimeML annotations contain errors due to imperfect scores on each TimeML object detection such as events, times, and relations. On the other hand, manual annotations comprise human-made errors that are introduced during the annotation process. Several previous works have provided different types of analysis for TimeBank annotations.

Boguraev and Ando (2006) evaluated the first version of the TimeBank corpus (TimeBank 1.1). They presented a quantitative analysis of the TimeBank corpus such as the distribution of relations, event classes, Timex types, and TimeML components. They showed that the annotation tool used to construct TimeBank caused a systematic shift by a single character. They also showed that for the same Timex signal, TimeBank 1.1 had different types of (or missing) Timex tags.

Similarly, Boguraev et al. (2007) presented a quantitative analysis not only for TimeBank 1.1 but also for TimeBank 1.2, which allows them to compare the two corpora. They selected a random document from the corpora and evaluated it manually to compare the number of errors between TimeBank 1.1 and TimeBank 1.2. Based on their results, the chosen document contained 96 errors (8 timex, 32 events, 43 links, and 13 signals) in

TimeBank 1.1 and 28 errors (1 timex, 10 events, and 17 links) in TimeBank 1.2, suggesting that TimeBank 1.2 was indeed an improvement over the prior version.

Caselli and Morante (2018) presented a detailed error analysis for automatic temporal processing systems that were submitted to TempEval-3. They manually evaluated 15% of the TimeBank corpus to check why automatic temporal processing systems failed to detect temporal relations in the corpus. The results showed that plenty of gold-standard temporal relations are either wrong or in dispute, with the resulting suggestion that annotators consider events' tense and aspect while annotating gold-standard temporal relations.

Inel and Aroyo (2019) compared the TimeBank corpus with other TimeML annotated corpora by manually evaluating events in each sentence. The results showed that TimeBank contains sentences that do not have any events, and there are a number of events that are not consistent with annotation guidelines. The results also showed that in some cases, the same phrases are tagged differently in different corpora. For example, "election day" was annotated as TIMEX3 in TimeBank while in other corpora "election" was labeled as an event. Finally, the comparison showed that the TimeBank corpus has only a single token for events while other corpora have multi-token events as well.

Ocal and Finlayson (2020) extracted timelines from the TimeBank corpus and presented a quantitative analysis of the timelines as well as an evaluation of the temporal indeterminacy of the timelines. They reported that the timelines extracted from TimeBank have an average of 9.3 time steps and 51.1 time points. Additionally, the timelines have a 67.9% indeterminacy score.

The Corpus Analysis and Validation for TimeML (CAVaT) tool (Derczynski and Gaizauskas, 2010) is the most similar work to that presented here. CAVaT is a sanity check system for TimeML annotated corpora that checks the temporal consistency of TLINKs, identifies disconnected subgraphs (TLINKs only), and detects self-loops. Additionally, CAVaT prints out the TLINK distribution and shows how many TLINKs are

triggered by temporal signals. CAVaT was run over the TimeBank corpus and detected 30 inconsistent texts and 26 self-loops, and showed that no text has a fully connected TLINK graph. In my work, I go further than CAVaT by checking the consistency of not only TLINKs, but of the entire TimeML graph. In contrast to CAVaT I check the disconnectivity of entire TimeML graphs (again, not just TLINKs), and moreover provide automatic corrections for them.

2.7 Duration

Because narrative is a sequence of events, inferring duration of events is critical for narrative understanding. For event duration prediction/extraction, two types of classifications are defined. Coarse-grained classification is to predict whether the event takes less than a day or more than a day, while fine-grained classification is to predict whether the event takes *seconds*, *minutes*, *hours*, *days*, *weeks*, *months*, or *years*.

For duration prediction, earlier NLP researchers have presented supervised learning-based solutions. Pan et al. (2006) presents a max. entropy system that uses tokens, lemmas, POS tags, and subject-object relations as features, achieving 73.5% accuracy on coarse-grained classification and 61.9% on fine-grained. Similarly, Gusev et al. (2011) also uses a max. entropy classifier but goes further by adding named entities, verbs, verb types, and verb dependencies as features, achieving 74.8% accuracy on course-grained classification and 66% on fine-grained. Later, Vempala et al. (2018) presents a neural networks-based system with a set of features that consists of event class, POS tags, named entities, and dependencies, achieving 83.2% accuracy on coarse-grained classification. Recent prior work presents rule-based systems instead of machine learning-based systems. Zhou et al. (2020) defines trigger words such as 'for', 'since', and 'on' for their rule-based system. Using a pre-defined list, semantic role labeling (SRL), their system

achieves 84.1% accuracy on coarse-grained classification. Finally, Yang et al. (2020) defines temporal patterns such as 'for', 'take', 'spend', 'last', 'lasting', 'duration', and 'period', and performs temporal pattern extraction, achieving 76.9% on coarse-grained classification and 76.2% on fine-grained classification.

These approaches have used two existing duration datasets: the McTaco dataset and the TimeBank duration dataset. The McTaco dataset is built by giving crowdsourcers a sentence and asking them how long did the event in the sentence take Zhou et al. (2019). Based on the answers, they defined a possible range for each event. However, it should be noted that the dataset contains only a few hundred event durations. The TimeBank duration dataset contains 58 news articles in which each event is annotated with lower-bound and upper-bound duration (Pan et al., 2011). However, the corpus has a 44% inter-annotator agreement score for fine-grained classification. because duration is relative and can be inferred in multiple ways I also note that the reason why prior approaches have high accuracy when the agreement score is low is that they use relaxed matching to evaluate their accuracy. For example, if the targeted event has [3 hours, 3 months] as lower and upper bound duration, the relaxed matching will accept the system's prediction as correct if the system's output is hours, days, weeks or months.

CHAPTER 3

EXTRACTING TIMELINES FROM TIMEML GRAPHS

3.1 Motivation

A timeline is a data structure that organizes events and times in a total ordering. Timelines are useful for a number of natural language understanding tasks, such as **question answering**, which can require understanding the overall temporal order of events to produce the correct answer (Saquete et al., 2004), **cross-document event coreference (CDEC)** and **cross-document alignment**, which can be improved by access to a total ordering of events (Navarro-Colorado and Saquete, 2016), and **summarization** and **visualization** (Liu et al., 2012), where timelines can enhance human understanding of the event and temporal structure of texts.

Unfortunately, timelines are rarely explicit in texts, and usually cannot be extracted directly. Instead, texts usually explicitly reveal only *partial* orderings of events and times. Such information can be used to construct a temporal graph by using a temporal representation language such as a temporal algebra (Allen, 1983) or TimeML (Sauri et al., 2006), either through automatic analyzers (Verhagen et al., 2005), manual annotation (Pustejovsky et al., 2003b), or some combination of the two.

If a temporal graph is encoded using a temporal algebra (Barták et al., 2014), then prior work in the field of *Qualitative Spatial Reasoning* (QSR) has shown how to extract an exact timeline (Kreutzmann and Wolter, 2014; van Beek, 1994; Gereveni and Schubert, 1995) that represents one possible total order solution. However, natural language involves temporal information that cannot be encoded in a strictly temporal algebra (i.e., expressions of possible, counterfactual, or conditional worlds—called *subordinated* events here), which is the reason schemes such as TimeML were developed. Also, natural

language is often ambiguous, so it is useful to know what portions of a timeline are *indeterminate*, that is, have multiple possible orderings consistent with the temporal graph.

There has been some prior work in the field of Natural Language Processing (NLP) on extracting timelines from TimeML graphs using machine learning, but these solutions fall short in that they do not deal with all possible relations and result in output that can contain ordering errors (see §2.5). These errors are a natural result of using statistical approaches to solve the problem.

In contrast, I merge work from QSR with TimeML to formulate timeline extraction as a *constraint satisfaction problem*, showing I can achieve a theoretically *exact* solution, unlike prior NLP approaches. Furthermore, I handle all possible temporal relations (including aspectual and subordination relations) and detect indeterminacy. Specifically, I present TLEX (TimeLine EXtraction), a method for extracting a set of timelines from a TimeML annotated text.

3.2 Approach

TLEX begins with a TimeML annotated file and it builds a TimeML graph from the file. TLEX uses the whole graph: events, time expressions, and all links, including temporal, aspectual, and subordinating. TLEX has five steps. **First** (§3.2.1), TLEX partitions the TimeML graph into subgraphs connected only with temporal and aspectual links; each of these subgraphs will correspond to an individual timeline (either a *main* timeline or a *subordinated* timeline). **Second** (§3.2.2), following prior work, TLEX transforms each subgraph into a *point algebra graph*, where the nodes represent time points and the edges represent the primitive temporal relations of *less than* ($<$) or *equals* ($=$). **Third** (§3.2.3), TLEX solves each separate point algebra graph, using constraint solving, to obtain a timeline. **Fourth** (§3.2.4), because some subgraphs are inconsistent and cannot be solved,

TLEX automatically finds the inconsistent subgraphs, which allows me to manually correct them. **Fifth** and finally (§3.2.5), TLEX identifies sections of the timelines that have an indeterminate order. The output of this five step process on a consistent TimeML graph is a set of timelines organized in a trunk-and-branch structure, with indeterminate sections marked, representing the full and exact temporal order of events and times in the text. In this section, I provide a detailed description of TLEX along with pseudocode for important steps and provide a time complexity analysis for each step individually as well as for the overall technique. I provide formal definitions of all relevant objects and a formal proof of the correctness of the technique in §3.3.

Below I consider each step in detail, using the following text as a running example:

(1) Rebecca's phone rang$_1$, and she answered$_2$ it. As soon as she picked up$_3$, John started$_4$ complaining$_5$. She was quickly bored$_6$, but realized$_7$ that if she hung up$_8$, John would be mad$_9$. So she continued$_{10}$ to listen$_{11}$.

In this example, each event is underlined and given a numerical subscript for reference. One notable thing about this example text is that events 8 and 9 did not happen in the "real world" of the story: because Rebecca did not hang up$_8$ the phone, John didn't get mad$_9$. These two events are related to the "real world" timeline by subordinating links and should be isolated onto their own "subordinated" timeline.

I informally define a "main" or "real world" timeline as a timeline that contains the events and times that actually happen in the "world" described in the text. Similarly I informally define "subordinated" timelines as containing events that did not occur in the world described in the text. As noted, in the example text, the events hang up$_8$ and would be mad$_9$ belong to a subordinated timeline. It is possible that there are multiple main timelines in a text; I treat this possibility formally in §3.3.3.

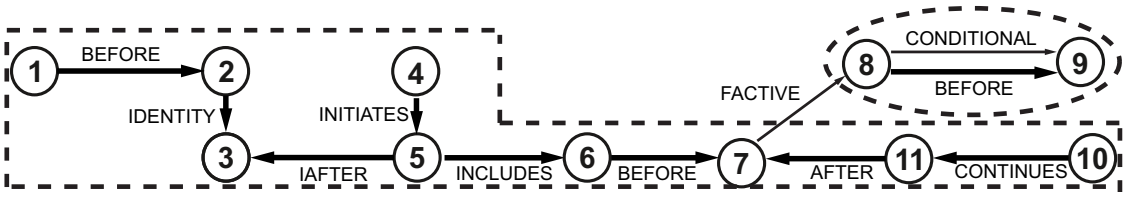

Figure 3.1: Visualization of the TimeML graph from the example. Numbers correspond to the events in the text, and arrows correspond to the temporal, aspectual, or subordinating links. The two temporally and aspectually connected subgraphs are separated by dashed lines, and links on the "real world" timeline are bolded.

3.2.1 Partitioning

First TLEX partitions the TimeML graph into subgraphs that are connected only with temporal and aspectual relations. The reason for that is temporal and aspectual relations represent temporal information about the text while subordinating relations do not. As noted previously, the edges in a TimeML graph represent temporal, aspectual, or subordinating TimeML links, whereas the nodes represent events and temporal expressions. A TimeML graph corresponding to Example (1) is shown in Figure 3.1. The 14 temporal and 5 aspectual TimeML link types are illustrated in Figure 2.1, and the presence of one of these links between two nodes implies that the nodes must lie on the same timeline.

Partitioning the TimeML graph into connected subgraphs is straightforward and can be achieved by walking the graph. TLEX can then further partition these connected subgraphs into subgraphs connected only with temporal and aspectual links. Importantly, the semantics of TimeML is such that a well-formed TimeML graph will have at most a single TLINK and a single ALINK relationship between two nodes. This is a consequence of the TimeML rule that says that between two event instances (or between an event instance and a time expression), there cannot be two or more TLINK or ALINK types. If there is both a TLINK relationship and an ALINK relationship, they must be temporally consistent Pustejovsky et al. (2003a,b). This means that TimeML graphs are atomic temporal

graphs, which makes them a member of the pointisable subclass of the interval algebra Ligozat (1998), and so they are solvable in polynomial time, a fact I make use of in §3.2.3.

This two-step partition results in a collection of temporally or aspectually connected subgraphs (each corresponding to a single timeline) that are only connected to other subgraphs through subordinating links (if at all). Figure 3.1 shows this partitioning with dashed lines. Because there are no temporal or aspectual links between events 7 and 8 and the rest of the graph (but the graph is otherwise connected), TLEX partitions the graph into two temporally connected subgraphs: the main, or "real world" subgraph (1-2-3-4-5-6-7-10-11) and the subordinated subgraph (8-9). TLEX will extract a separate timeline from each of these subgraphs.

Time Complexity Because partitioning only requires walking the graph, the time complexity of the first step is $O(m+n)$ where m is the number of nodes and n is the number of edges.

3.2.2 Transforming

Before applying constraint solving to derive a timeline from a temporally and aspectually connected TimeML subgraph, TLEX must first transform each subgraph into a *point algebra (PA) graph*. While a PA graph is formally defined in Definition 3.3.4, I explain it conceptually here. First, note that every node v in the TimeML graph represents a temporal interval (an event or time expression) with a start time point t_v^- and end time point t_v^+. Also, each TLINK and ALINK can be represented as a simple conjunction of the primitive temporal constraints *less than* ($<$) and *equals* (=), as shown in Table 3.1. The substitutions are straightforward and follow directly from the definitions of the TimeML link types. Given these facts, in this step TLEX replaces each node v with its start and end time points, and replaces each temporal or aspectual link with the corresponding primitive

temporal constraints, adding also the constraint $t_v^- < t_v^+$ for each interval. The PA graph for the running example is shown in Figure 3.2, and corresponds to the transformation of Figure 3.1 using the substitutions given in Table 3.1.

Time Complexity In this step, TLEX replaces each node with two nodes and each edge with two or three edges. These replacements therefore require at most $2m+3n$ operations, where m is the number of nodes and n is the number of edges. Therefore, the time complexity of the transformation step is $O(m+n)$.

A BEFORE B	$(A^+ < B^-)$
A AFTER B	$(B^+ < A^-)$
A IBEFORE B	$(A^+ = B^-)$
A IAFTER B	$(B^+ = A^-)$
A BEGINS B	$(A^- = B^-) \wedge (A^+ < B^+)$
A BEGUN_BY B	$(A^- = B^-) \wedge (B^+ < A^+)$
A ENDS B	$(B^- < A^-) \wedge (A^+ = B^+)$
A ENDED_BY B	$(A^- < B^-) \wedge (A^+ = B^+)$
A DURING B	$(B^- = A^-) \wedge (A^+ = B^+)$
A DURING_INV B	$(A^- = B^-) \wedge (B^+ = A^+)$
A INCLUDES B	$(A^- < B^-) \wedge (B^+ < A^+)$
A IS_INCLUDED B	$(B^- < A^-) \wedge (A^+ < B^+)$
A SIMULTANEOUS B	$(A^-$ = $B^-) \wedge (A^+$ = $B^+)$
A IDENTITY B	$(A^- = B^-) \wedge (A^+ = B^+)$
A INITIATES B	same as A BEGINS B
A CULMINATES B	same as A ENDS B
A TERMINATES B	same as A ENDS B
A CONTINUES B	same as A IS_INCLUDED B
A REINITIATES B	same as A IS_INCLUDED B

Table 3.1: Translation of the TimeML temporal and aspectual relations into primitive temporal relations between interval start and end points. For an interval I (an event or a time), the start point of the interval is denoted by I^- and the end point is denoted by I^+.

3.2.3 Solving

TLEX generates a timeline by solving the PA graph. A solution to a PA graph consists of an assignment of a single integer to each node, where the assignment is consistent with

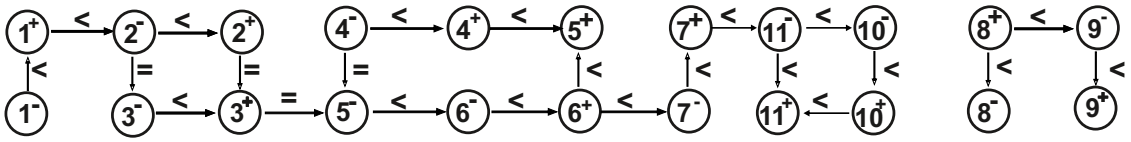

Figure 3.2: The two point algebra (PA) graphs corresponding to the temporally and aspectually connected subgraphs shown in Figure 3.1. These are produced by replacing each node I with its start and end time points I^- and I^+, and replacing each temporal or aspectual link with the set of primitive temporal relationships shown in Table 3.1.

all the constraints ($<$, =) in the graph. The order of the integers then represents the order of the time points on the timeline. There are numerous existing software packages that can solve constraint problems; for example, my reference implementation of TLEX generates solutions using the Java Constraint Programming solver (JaCoP; Kuchcinski and Szymanek, 2013), which is open source and offers a rich set of primitive, logical, conditional, and global constraints as well as configurable solution search methods. JaCoP uses a backtracking search algorithm that performs depth-first traversal of a search tree, where branches are alternative solutions that the algorithm may have to examine to find a solution, and constraints are used to eliminate subtrees with no solutions. Backtracking algorithms guarantee to find a solution if a solution exists. There are two main reasons that I choose JaCoP. (1) Unlike many other CSP solver tools, JaCoP is able to generate random solutions which I used in § 3.2.5. (2) JaCoP finds a solution faster than other CSP-solver tools (Benavides et al., 2005).

For this step, my implementation uses the JaCoP setting that finds the smallest solution, which starts at 1 and which represents each time point difference as a difference of 1. The timeline for the running example is shown in Figure 3.3. As before, there are two temporally and aspectually connected subgraphs, where the connection between them (the grey bar) represents the FACTIVE subordinating link. As can be seen, this is a trunk-and-branch multi-timeline structure, where the trunk is (in this case) the "main" timeline and the branch is the subordinated timeline.

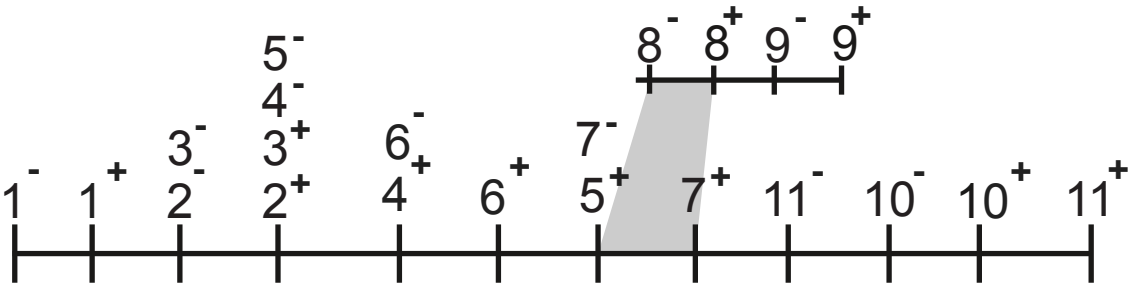

Figure 3.3: Visualization of the timeline extracted from the PA graphs shown in Figure 3.2. The two subgraphs are arranged into a main and subordinated timeline connected by a grey branch.

As I have noted, one major advantage of TLEX is the proper handling of subordinating relations. Prior approaches to timeline extraction ignored subordinating relations, and so were unable to represent such a trunk-and-branch timeline structure. If, for example, the subordinating relation were assumed to be some sort of temporal relationship, machine-learning-based techniques (e.g., Verhagen et al., 2005; Bethard, 2013; Mirza and Tonelli,2016) would likely insert events 8 and 9 between or nearby time points 7^- and 7^+. That would indicate events 8 and 9 happened after events 1, 2, and 3 but before events 10 and 11, while in the story world, these events did not occur at all.

Time Complexity Nebel and Bürckert (1995) showed that the time complexity of solving an atomic temporal graph is $O(n^3)$ where n is the number of edges in the graph. If TLEX extracts k atomic point algebra graphs from a TimeML graph, then the time complexity of the solving step is at most $O(kn^3)$.

3.2.4 Correcting

Step 3 of TLEX only completes if each temporally connected TimeML graph (and therefore each corresponding PA graph) is consistent. This follows from Barták et al. (2014, p. 20), who showed that there is a solution (i.e., assignment of real numbers to time points)

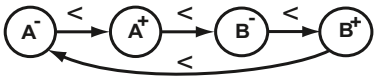

Figure 3.4: Example of an inconsistent cycle generated by A BEFORE B and B BEFORE A.

that satisfies the point algebra graph if and only if the graph is consistent. On the other hand, if the solver determines that there is no solution, I must first correct the inconsistencies before TLEX can solve the graph. Zaidi (1999, p. 5) showed that a PA graph is inconsistent if and only if the graph contains an inconsistent cycle of relations. There are three possible types of inconsistent cycles (including inconsistent self-loops; see Definition 3.3.27), with the result that in each case there is one time point in an overall *less than* relationship with itself[1]. An example of an inconsistent cycle is shown in Figure 3.4.

TLEX uses the novel Algorithm 1a to detect inconsistent cycles (including self-loops). The algorithm is split into four steps (a–d). Step (a) finds all self-loops; step (b) finds a single edge per inconsistent cycle; step (c) finds all other edges in each inconsistent cycle; and step (d) returns the completed set of cycles. The relations in these cycles can be traced back to their originating TimeML links to enable manual correction. A detailed explanation of the algorithm follows.

In step (a), the algorithm first detects all self-loops, which are single-edge inconsistent cycles where a node is less than itself ($v < v$). All detected self-loops are recorded in the set E_L.

In step (b), it finds one edge per inconsistent cycle. For all edges in the PA graph, line 9 adds each edge in turn to an initially empty set of edges E_N. After each edge is added, TLEX checks the consistency of the (partial) graph (V_P, E_N) (line 10). If the partial graph is inconsistent, then the last added edge is a part of an inconsistent cycle.

[1]Note that for a cycle to be inconsistent, its *less than* relations must all be in the same direction. For example, in Figure 3.2 there is a cycle of *less than* relations involving the 10^-, 10^+, 11^-, and 11^+ time points, however, their directions differ.

The algorithm therefore removes that edge from (V_P, E_N) to break the cycle (line 11), and records it in the breaking set E_B (line 12). After this loop completes I have a fully consistent (partial) graph (V_P, E_N), along with excluded non-self-loop edges (E_B) that are part of inconsistent cycles. Because the algorithm breaks each cycle as it finds it, every inconsistent non-self-loop cycle has exactly one edge in E_B.

Step (c) finds all remaining edges that participate in each inconsistent non-self-loop cycle. For each edge e_b in the breaking set E_B, the algorithm performs an exhaustive search for all cycles that include e_b via the function SEARCHFORCYCLES (line 17). For each cycle found, the algorithm tests it using the function ISCYCLECONSISTENT to see if it is a consistent cycle (line 19); if not, the edges in that cycle are recorded in the inconsistent set E_I (line 20). At the end of step (c) the set E_I contains all edges that participate in non-self-loop inconsistent cycles.

The SEARCHFORCYCLES function finds all cycles that include a particular edge e_b. The algorithm adds e_b and its nodes to an initially empty path E_Y (line 5–7), and it pushes E_Y to the stack Q (line 8). Then, in a while loop, it pops Q; if the popped path is a cycle (line 11), then it adds the cycle to X, the set of cycles found (line 12), and restarts the loop. If the popped path is not a cycle, the path is extended by one edge in all possible directions (lines 16–23), excepting already visited edges (lines 18–19), and those new paths are placed on the stack (line 24).

The ISCYCLECONSISTENT function checks whether the cycle in the PA graph is consistent by testing whether at least one $<$ edge in the cycle points oppositely to the other $<$ edges. The algorithm starts from a random $<$ edge e_0 (line 2), then walks forward along the cycle (lines 3–7), testing whether each $<$ edge points in the same direction as e_0. If a $<$ edge is found that points in the opposite direction, the cycle is necessarily consistent (lines 8–9). If all $<$ edges point in the same direction, then the cycle is inconsistent. It returns false (line 13).

Finally, in step (d) the algorithm finds the set of all maximal connected inconsistent subgraphs (line 22), which are the connected subgraphs made up solely of the inconsistent cycles or self-loops in graph (see Definition 3.3.28).

Algorithm 1a Detect all inconsistent cycles in a connected PA Graph

1: **procedure** FINDALLINCONSISTENTCYCLES(V_P, E_P)
Require: V_P ▷ all nodes in the connected PA graph
Require: E_P ▷ all edges in the connected PA graph

Step (a): Segregate self-loops
2: $E_L \leftarrow \emptyset$ ▷ set of edges that are inconsistent self-loops
3: **for all** $e \in E_P$ **do**
4: **if** ISINCONSISTENTSELFLOOP(e) **then**
5: $E_L.add(e)$

Step (b): Find one edge per inconsistent cycle
6: $E_N \leftarrow \emptyset$ ▷ empty graph used to test for consistency
7: $E_B \leftarrow \emptyset$ ▷ breaking set: set of edges that break cycles
8: **for all** $e \in E_P/E_L$ **do** ▷ examine all edges not known to be self-loops
9: $E_N.add(e)$ ▷ add each edge one-by-one
10: **if** $\neg$ISCONSISTENT(V_P, E_N) **then** ▷ test to see if that edge causes inconsistency
11: $E_N.remove(e)$ ▷ if so, set it aside to break the cycle
12: $E_B.add(e)$ ▷ save breaking edges in the breaking set

Step (c): Find all other edges in each inconsistent cycle
13: $E_I \leftarrow \emptyset$ ▷ inconsistent set: set of all edges in inconsistent cycles
14: $X \leftarrow \emptyset$ ▷ cycle set: set of all cycles containing a specific edge
15: $E_Y \leftarrow \emptyset$ ▷ a cycle, expressed as a set of edges
16: **for all** $e_b \in E_B$ **do** ▷ for each single edge in the breaking set
17: $X \leftarrow$ SEARCHFORCYCLES(e_b, E_P/E_L) ▷ perform exhaustive search for cycles with e_b
18: **for all** $E_Y \in X$ **do** ▷ find each inconsistent cycle
19: **if** $\neg$ISCYCLECONSISTENT(E_Y) **then** ▷ test each cycle to see if it is inconsistent
20: $E_I.addAll(E_Y.getEdges())$ ▷ if inconsistent, save the edges

Step (d): Return a set of maximal inconsistent connected PA subgraphs
21: $V_I \leftarrow \{v \mid (v = e.source \vee v = e.target)\ \forall e \in \{E_I \cup E_L\}\}$ ▷ nodes for inconsistent edges
22: **return** FINDCONNECTEDSUBGRAPHS($V_I, E_I \cup E_L$)

Once the maximal inconsistent subgraphs are found, these can be used to find and correct errors in the original TimeML annotation. In cases where the natural language

Algorithm 1b Search for all cycles starting with a particular edge

1: **procedure** SEARCHFORCYCLES(e_b, E_Q)
Require: e_b ▷ an edge involved in an inconsistent cycle
Require: E_Q ▷ all non-self-loop edges in the PA graph
2: $X \leftarrow \emptyset$ ▷ initialize the return set
3: $Q \leftarrow$ empty stack ▷ initialize the search queue
4: $E_Y \leftarrow$ empty list ▷ create the seed path
5: $E_Y.add(e_b.source)$
6: $E_Y.add(e_b)$
7: $E_Y.add(e_b.target)$
8: $Q.push(E_Y)$ ▷ seed the queue
9: **while** $\neg Q.isEmpty()$ **do** ▷ while the queue has more elements, continue searching
10: $E_Y \leftarrow Q.pop()$ ▷ get the topmost path
11: **if** ISCYCLE(E_Y) **then**
12: $X.add(E_Y)$ ▷ add it to the return set if it is a cycle
13: **continue**
14: $v_{prev} \leftarrow E_Y.getLast()$ ▷ Get the last node...
15: $e_{prev} \leftarrow E_Y.getSecondToLast()$ ▷ ... and edge in the path
16: $E_A \leftarrow$ GETADJACENTEDGES(v_{prev}, E_Q) ▷ get all edges adjacent to the last node
17: **for all** $e_a \in E_A / e_{prev}$ **do**
18: **if** $E_Y.contains(e_a)$ **then**
19: **continue** ▷ do not reuse any edge
20: $E_Z \leftarrow$ empty list ▷ for all other adjacent edges, initialize a new path
21: $E_Z.addAll(E_Y)$
22: $E_Z.add(e_a)$
23: $E_Z.add($GETOTHERENDPOINT(e_a, v_{prev})$)$
24: $Q.add(E_Z)$ ▷ add the newly extended path to the queue
25: **return** X ▷ return all cycles found

itself describes a consistent graph, these corrections will result in the graph being made consistent, and so Step 3 of TLEX can now continue.

Because every edge in the PA graph can be traced back to one and only one link in the TimeML graph, the TimeML links and nodes corresponding to the maximal inconsistent graphs can be provided directly to human annotators for correction. Although there has been some research in the field of qualitative spatial reasoning focused on automatically correcting inconsistencies in point algebra graphs in a minimum or approximate manner Iwata and Yoshida (2013); Condotta et al. (2016), these approaches cannot be applied to TimeML graphs. Inconsistencies in TimeML graphs usually come from incorrect or

Algorithm 1c Check if the cycle is consistent

procedure IsCycleConsistent(E_Y)
Require: E_Y ▷ a cycle represented as an alternative sequence of nodes and edges
  $e_0 \leftarrow$ PickRandomNonEqualsEdge(E_Y) ▷ pick a random $<$ starting edge
  $e_{prev} \leftarrow e_0$
  $v_{prev} \leftarrow e_0.target$
  do
    $e_{next} \leftarrow$ GetNextEdge(v_{prev}, E_Y) ▷ Get the next edge...
    $v_{next} \leftarrow$ GetNextNode(e_{next}, E_Y) ▷ ... and next node
    if IsLessThanEdge(e_{next}) $\wedge$ $e_{next}.target == v_{prev}$ **then**
      return true ▷ if any $<$ points opposite to e_0, the cycle is consistent
    $e_{prev} \leftarrow e_{next}$ ▷ move the focus to the next edge and node
    $v_{prev} \leftarrow v_{next}$
  while $e_{prev} \neq e_0$ ▷ continue until I return to the starting edge
  return false ▷ if all the $<$ edges point in the same direction, the cycle is inconsistent

incomplete annotations; more rarely, there may be fundamental inconsistencies in the meaning of the language. Therefore, errors need to be fixed by reference to the original text. It is possible that one could develop other knowledge- or inference-based artificial intelligence (AI) or NLP methods for automatically correcting these inconsistencies, but I will leave the exploration of this for future work. This means that any inconsistencies discovered by TLEX need to be manually corrected before proceeding. I discuss specific cases of inconsistencies for my selected corpora in Section 3.4.

Time Complexity Step (a) involves checking a property of each edge in the graph, which is a complexity of $O(n)$, where n is the number of edges in the graph. For step (b), Nebel and Bürckert (1995) showed that checking the consistency of a temporal graph expressed via a tractable subclass of Allen's algebra (i.e., the pointisable subclass, of which TimeML is a member) can be done via the path consistency algorithm with time complexity of $O(n^3)$. Therefore, the IsConsistent function is $O(n^3)$. Step (b) of the algorithm checks the consistency of n graphs because there are n edges (n^4 operations), and so its time complexity is $O(n^4)$. In step (c), the SearchForCycles function conducts an exhaustive depth-first search of paths in the graph starting at e_b, keeping

only those paths that are cycles, and such a search has a worst-case time complexity of $O(n + m)$, where m is the number of nodes in the graph. According to Ahrens (1897) there are at most $2^{n-m} - 1$ cycles in a connected graph. The function IsCycleConsistent is linear in the size of the cycle, as all this function must do is walk the cycle and keep track of the direction of the less than edges, and so has a time complexity of at most $O(n)$; it is run for every cycle found. So the overall time complexity of step (c) is $O((2^{n-m} - 1) * (n + m + n)) = O((2^{n-m} - 1) * (n + m))$. The time complexity of finding connected subgraphs is linear in the size graph, so step (d) is $O(n)$. Therefore, the worst-case time complexity of Algorithm 1a is

$$\begin{aligned} & O(n + n^4 + (2^{n-m} - 1) * (n + m) + n) \\ = & O(n^4 + (n2^{n-m} - n) + (m2^{n-m} - m)) \\ = & O(n^4 + n2^{n-m} + m2^{n-m}) \end{aligned} \tag{3.1}$$

where n is the number of edges in the graph, and m is the number of nodes.

3.2.5 Identifying Indeterminacy

In most cases, TimeML graphs lack enough information to uniquely specify the full ordering; a simple example is shown in Figure 3.5. For that TimeML graph, the uniquely determined orderings include the first and last sections of the timeline, namely $1^- < 1^+ < 2$ and $3 < 4^- < 4^+ < 5^- < 5^+$. On the other hand, the order of 2 and 3 is indeterminate. There are 11 possible orders, shown in Figure 3.6.

Many constraint solver implementations can produce all possible solutions for a constraint problem. TLEX assumes, in the absence of additional information, that the shortest normal form timeline (corresponding to the smallest solution, and formally defined in Definition 3.3.22) is the best. To determine which portions of the shortest normal form timeline are of indeterminate order, TLEX compares the shortest normal form timeline

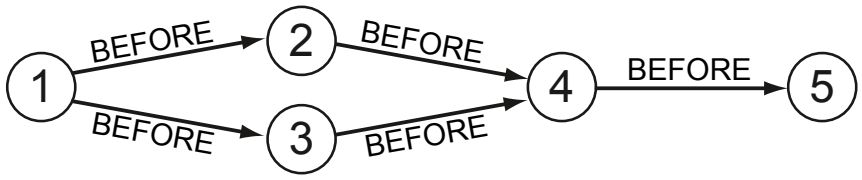

Figure 3.5: A TimeML graph with an indeterminacy. While the order of 4 and 5 is fixed, the relative order of 2 and 3 is indeterminate.

(1) $1^- < 1^+ < 2^- < 2^+ < 3^- < 3^+ < 4^- < 4^+ < 5^- < 5^+$
(2) $1^- < 1^+ < 3^- < 3^+ < 2^- < 2^+ < 4^- < 4^+ < 5^- < 5^+$
(3) $1^- < 1^+ < 2^- < 2^+ = 3^- < 3^+ < 4^- < 4^+ < 5^- < 5^+$
(4) $1^- < 1^+ < 3^- < 3^+ = 2^- < 2^+ < 4^- < 4^+ < 5^- < 5^+$
(5) $1^- < 1^+ < 2^- = 3^- < 2^+ < 3^+ < 4^- < 4^+ < 5^- < 5^+$
(6) $1^- < 1^+ < 2^- = 3^- < 3^+ < 2^+ < 4^- < 4^+ < 5^- < 5^+$
(7) $1^- < 1^+ < 3^- < 2^- < 2^+ = 3^+ < 4^- < 4^+ < 5^- < 5^+$
(8) $1^- < 1^+ < 2^- < 3^- < 2^+ = 3^+ < 4^- < 4^+ < 5^- < 5^+$
(9) $1^- < 1^+ < 2^- < 3^- < 3^+ < 2^+ < 4^- < 4^+ < 5^- < 5^+$
(10) $1^- < 1^+ < 3^- < 2^- < 2^+ < 3^+ < 4^- < 4^+ < 5^- < 5^+$
(11) $1^- < 1^+ < 2^- = 3^- < 2^+ = 3^+ < 4^- < 4^+ < 5^- < 5^+$

Figure 3.6: All possible solutions for the graph shown in Figure 3.5, with indeterminant orderings highlighted.

with alternative timeline solutions using Algorithm 2. This algorithm iterates through all adjacent time point pairs in the shortest timeline and checks to see that these two points are adjacent in some selection of other timelines. If they are not always adjacent, the order of that pair is indeterminate. With these results, I can visualize the indeterminate sections, as shown in Figure 3.7.

Time Complexity For a temporal graph with m nodes, there are at most $2m$ distinct time points in the largest timeline. For i time points there are $i!$ possible distinct orderings of time points (timelines). That means for a temporal graph with m nodes, there are at most $(2m)! + (2m-1)! + \ldots + 1 = \sum_{x=1}^{2m} x!$ timelines. Each timeline with p distinct time points has $p-1$ adjacent time point pairs, which means there are at most $2m-1$ pairs to inspect, corresponding to $O(m)$. This means that the worst case time complexity

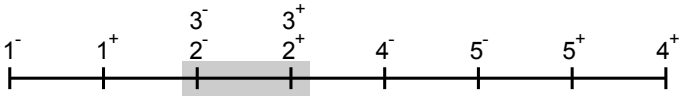

Figure 3.7: The shortest timeline extracted from the TimeML graph in Figure 3.5, with the indeterminate section highlighted.

of Algorithm 2 is $O(m \sum_{x=1}^{2m} x!)$. This is quite large. I can limit my exploration of alternative timelines to a constant number (lines 3–4 in Algorithm 2), producing a lower-bound estimate for time point pairs that are indeterminate. Limiting the computation in this way produces a time complexity of $O(m)$ for this step.

Algorithm 2 Identify Indeterminant Pairs

1: **procedure** FINDINDETERMINATESECTIONS(s, T, l)
Require: s ▷ shortest timeline
Require: T ▷ set of all other timelines
Require: l ▷ integer constant for limiting search; ≤ 0 means no limit
2: $D \leftarrow \emptyset$ ▷ indeterminacy map
3: **while** $l > 0 \wedge T.size > l$ **do** REMOVERANDOMELEMENT(T) ▷ reduce timeline set to size l
4: **for all** $p \leftarrow \langle s_i, s_{i+1} \rangle \in s$ **do** ▷ for each neighboring pair of times in the shortest timeline
5: $D(p) \leftarrow false$
6: **for all** $t \in T$ **do**
7: **if** $\neg$ISPAIRADJACENT(p, t) **then**
8: $D(p) \leftarrow true$ ▷ if the pair are not neighbors in one timeline, it is indeterminate
9: **return** D

Overall Time Complexity In sum, the time complexity of the overall TLEX algorithm is as follows. For m nodes and n edges in the TimeML graph with k temporally and aspectually connected subgraphs, the partitioning step has order $O(m + n)$, transforming has $O(m + n)$, solving $O(kn^3)$, correcting $O(n^4 + n2^{n-m} + m2^{n-m})$, and finding indeterminacies $O(m \sum_{x=1}^{2m} x!)$ in the exhaustive case or $O(m)$ in the limited case. Therefore in the exhaustive case TLEX has a time complexity of $O(m \sum_{x=1}^{2m} x!)$, and in the limited case $O(n^4 + n2^{n-m} + m2^{n-m})$.

3.3 Formal Proofs of TLEX's Correctness

TLEX is a procedure that takes a TimeML graph as an input and returns, for each temporally connected subgraph C_i of the TimeML graph, either (a) the timeline corresponding to C_i, with indeterminate sections marked, or (b) a subgraph $J_i \subset P_i$ containing all primitive temporal constraints in the PA graph P_i corresponding to C_i. In the case where all C_i are consistent, the timelines can be arranged into a trunk-and-branch structure where the timelines are connected to each other by subordinating TimeML relations. TLEX has 5 steps (partitioning, transforming, solving, correcting, and identifying indeterminacies), and I prove the correctness of each step in the theorems below. Note that I provide a comprehensive list of all notation and its meaning in Table A.

3.3.1 Proof of Step 1: Partitioning

I begin with the formal definition of the TimeML graph, which is the input to TLEX.

Definition 3.3.1 *TimeML Graph. A TimeML graph is a graph $T = (V_T, E_T)$, where V_T is a set containing temporal intervals—TimeML events and time expressions—and E_T is a set of TimeML links $l = (u, v, w)$, a tuple where $u, v \in V_T$ and $w \in \mathscr{L}$, where $\mathscr{L} = \mathscr{L}_T \cup \mathscr{L}_A \cup \mathscr{L}_S$ is a set containing all 25 link types in TimeML, including 14 temporal (*TLINK*, $\mathscr{L}_T$), 5 aspectual (*ALINK*, $\mathscr{L}_A$), and 6 subordinating (*SLINK*, $\mathscr{L}_S$) types.*

In Step 1, TLEX partitions the TimeML graph into temporally connected subgraphs, in other words, TimeML graphs that are connected only by TimeML links (TLINKs and ALINKs, which are the only links with temporal semantics). To do that, TLEX walks the graph, partitioning the graph into subgraphs by removing SLINKs. The formal definition of the temporally connected TimeML graph is as follows:

Definition 3.3.2 *Temporally Connected TimeML Graph. A temporally connected TimeML graph is a connected TimeML graph $C = (V_C, E_C)$ where for every TimeML link $l = (u, v, w) \in E_C$, $w \in \mathscr{L}_{TA}$, where $\mathscr{L}_{TA} = \mathscr{L}_T \cup \mathscr{L}_A$ is a set containing only the 19 link types with temporal semantics in TimeML, namely, the 14 temporal (*TLINK*, $\mathscr{L}_T$) and 5 aspectual (*ALINK*, $\mathscr{L}_A$) types.*

The theorem of Step 1 is as follows:

Theorem 3.3.3 *On the input of a TimeML graph T, Step 1 of* TLEX *will give as output (i) a set of temporally connected TimeML graphs $T_C = \{C_1, C_2, \ldots C_k\}$ that are subgraphs of the original TimeML graph, the union of which contain all the* TLINK*s,* ALINK*s, events, and times in the original graph; and (ii) T_S, the set of all* SLINK*s in the original graph.*

Proof. Part (i): Given a TimeML graph $T = (V_T, E_T)$, Step 1 of TLEX first creates an empty graph $C_i = (V_{C_i}, E_{C_i})$ and then conducts a breadth-first search of T, starting from a randomly selected node, following only TLINK or ALINK edges to other connected nodes. Each link traversed in this search is removed from E_T and placed in E_{C_i}, and each node traversed is removed from V_T and placed in V_{C_i}. C_i is therefore by construction a temporally connected TimeML graph. When the search completes, TLEX creates a new C_{i+1}, and starts a new search from a new randomly selected node, performing the same removals, until all nodes in V_T have been removed. As a result, TLEX constructs a set of temporally connected graphs $T_C = \{C_1, C_2, \ldots C_k\}$, the union of which contains all the events, times, and TLINK and ALINK edges present in the original T.

Part (ii): To assemble the list of subordinating links, TLEX starts with an empty set T_S and iterates over E_T, adding only SLINK edges to T_S. □

3.3.2 Proof of Step 2: Transforming

In Step 2, TLEX transforms each temporally connected TimeML subgraph into a point algebra (PA) graph that expresses the temporal semantics of the TimeML subgraph as primitive constraint relations (UzZaman and Allen, 2011). The formal definition of a point algebra graph is as follows:

Definition 3.3.4 *Point Algebra (PA) Graph. A point algebra (PA) graph is a graph* $P = (V_P, E_P)$ *where* V_P *is a set of time points* t_i*, and* E_P *is a set of temporal constraints* $c_i = (u, v, w)$*, a tuple where* $u, v \in V_P$ *and* $w \in \{<, =\}$.

The theorem of Step 2 is as follows:

Theorem 3.3.5 *On input of a vector of temporally connected TimeML graphs* $T_C = (C_1, C_2, \ldots C_k)$*, Step 2 of* TLEX *will give as output a vector of point algebra graphs* $T_P = (P_1, P_2, \ldots P_k)$ *in which each* P_i *is connected, and also corresponds to the primitive temporal semantics of* C_i.

Proof. Van Beek and Cohen (1990, p. 23) showed that the pointisable subset of Allen's interval algebra (IA) can be translated into a PA graph. The temporal subset of TimeML (including only TLINK and ALINK relations) is a pointisable subset of Allen's IA because it is atomic, meaning there is only one relation between any two nodes, and atomic interval algebras are a subset of pointisable algebras (Ligozat, 1998, p. 1). To perform the translation, for each temporally connected TimeML graph $C_i = (V_{C_i}, E_{C_i})$, TLEX constructs an empty PA graph $P_i = (V_{P_i}, E_{P_i})$. For every TimeML event and time expression $v \in V_{C_i}$, where t^- and t^+ are the start and end time points of v, respectively, TLEX adds t^- and t^+ to V_{P_i} and adds $(t^-, t^+, <)$ to E_{P_i}. Then, for every $l \in E_{C_i}$, TLEX adds the one or two primitive temporal relations corresponding to the primitive temporal semantics of that TLINK or ALINK as listed in Table 3.1. Therefore, at the end of the step I have constructed a vector of PA graphs $T_P = \{P_1, P_2, \ldots P_k\}$ in which each P_i corresponds to the

full primitive temporal semantics of each original temporally connected TimeML graph $C_i \in T_C$. $\square$

3.3.3 Proof of Step 3: Solving

In Step 3, TLEX extracts a timeline from each consistent PA graph P_i by assigning integers to each node on the graph using constraint solving. The case where a PA graph is inconsistent is covered in Step 4. Barták et al. (2014, p.15) defined a consistent PA graph as follows:

Definition 3.3.6 *Consistent PA Graph. Given a PA graph $P = (V_P, E_P)$ with $V_P = \{t_1, t_2, \ldots\}$, I say that a vector of real numbers $S = (r_1, r_2, \ldots)$ is a solution to a PA graph if and only if the values of $t_i = r_i$ satisfy all the constraints from E_P. I then say that P is a consistent PA graph if and only if a solution exists.*

Definition 3.3.7 *Consistent Temporally Connected TimeML Graph. A temporally connected TimeML graph C is consistent if and only if its PA graph P is consistent.*

A timeline associates sets of time points in a PA graph with the ordered set of solution values in the solution set. I construct the definition of a timeline as follows.

Definition 3.3.8 *Ordered Solution Set. Let $P = (V_P, E_P)$ be a PA graph, with $V_P = \{t_1, t_2, \ldots\}$, and let $S = (r_1, r_2, \ldots)$ be a vector of real numbers which is a solution of P. Let $O_S = (r_1, r_2, \ldots)$ be the set of unique real numbers that are contained in S, ordered according to the natural order of the reals $\mathbb{R}$. I say O_S is the ordered solution set of P and S.*

Definition 3.3.9 *Anchor Set. Let P be a PA graph with solution S. Let $A_i \subset V_P$ be a subset of time points that share a real number assignment $r_i \in S$. I say A_i is an anchor set of P and S.*

Definition 3.3.10 *Timeline. Let P be a PA graph with solution S and ordered solution set O_S. Let $A_P = \{A_1, A_2, \ldots\}$ be the set of all anchor sets for P and S. Let $L : O_S \rightarrow A_P$ be a function where each $s_i \in O_S$ is mapped to the A_i that contains the time points to which s_i is assigned. I call L a timeline for P.*

Lemma 3.3.11 *A timeline $L : O_S \rightarrow A_P$ is a bijective function.*

Proof. Let $P = (V_P, E_P)$ be a PA graph with solution S and ordered solution set O_S, and let $L : O_S \rightarrow A_P$ be a timeline corresponding to S. To show that L is bijective I must show that L is injective (every element in A_P is paired with one and only one element in O_S) and surjective (every element in A_P is paired with some element of O_S). To prove L is injective, suppose there are two $r_i, r_j \in O_S$ such that $L(r_i) = L(r_j) = A_i$. By the definition of a timeline and anchor set, every time point in A_i shares the same assignment $r_i \in O_S$, and time points cannot have multiple assignments, and so necessarily $r_i = r_j$. Thus L is injective. By construction, L is surjective, because every timepoint in P receives an assignment in S and so is included in some anchor set A_P, which is mapped by construction in L. Therefore, L is a bijective function. □

I deal next with the problem that there are an uncountably infinite number of timelines that all represent the same ordering of timepoints in a PA graph. For any given timeline $L : O_S \rightarrow A_P$ with $O_S = (r_1, r_2 \ldots)$, there exist an uncountably infinite number of other timelines whose ordered solution sets are order-isomorphic to O_S while maintaining the correct mapping of values in O_S to anchor sets in A_P. In cases where there is no direct metric temporal information in the original TimeML annotated text, for most practical purposes any one of these timelines can be substituted for another. Therefore it is useful to identify a "normal form" for timelines that simplifies the process of comparison. To establish this normal form I first define the following equivalence relation:

Definition 3.3.12 *Tempomorphic Equivalence. Let P be a consistent PA graph with distinct solutions S_1 and S_2 that share A_P, a set of anchor sets for P. Let $L_1 : O_{S_1} \to A_P$ and $L_2 : O_{S_2} \to A_P$ be timelines for P where there exists a bijective, strictly order-preserving function $f_p : O_{S_1} \to O_{S_2}$. I say that L_1 and L_2 are timeline isomorphic—or tempomorphic, or tempomorphically equivalent—if for every $r \in O_{S_1}$ and $f_p(r) \in O_{S_2}$, the equality $L_1(r) = L_2(f_p(r))$ holds. I express this relationship as $L_1 \overset{L}{\sim} L_2$.*

Lemma 3.3.13 *Tempomorphic equivalance $\overset{L}{\sim}$ is an equivalence relation.*

Proof. To show that $\overset{L}{\sim}$ is an equivalence relation it is sufficient to show it is reflexive, symmetric, and transitive.

- *Reflexivity*: Let $L : O_S \to A_P$ be a timeline. The identity function I is a bijective, order-preserving function $I : O_S \to O_S$. Therefore, for every $r \in O_S$, $I(r) \in O_S$ and $L(r) = L(I(r))$, and so L is tempomorphic with itself. Therefore $\overset{L}{\sim}$ is reflexive.
- *Symmetry*: Let L_1 and L_2 be timelines for the same PA graph P with $L_1 \overset{L}{\sim} L_2$. This means there is a bijective, order-preserving function f_p such that $L_1(r) = L_2(f_p(r))$. Because f_p is bijective, it must have an inverse f_p^{-1}, and $L_2(r) = L_1(f_p^{-1}(r))$, and so $L_2 \overset{L}{\sim} L_1$. Therefore $\overset{L}{\sim}$ is symmetric.
- *Transitivity*: Let L_1, L_2, and L_3 be timelines for the same PA graph P, and $L_1 \overset{L}{\sim} L_2$ and $L_2 \overset{L}{\sim} L_3$. This means there exists two bijective, order-preserving functions f_{p_a} and f_{p_b} such that $L_1(r) = L_2(f_{p_a}(r))$ and $L_2(r) = L_3(f_{p_b}(r))$. By substitution I also have $L_1(r) = L_2(f_{p_a}(r)) = L_3(f_{p_b}(f_{p_a}(r)))$. Because both f_{p_a} and f_{p_b} are bijective and order-preserving, the composition of $f_{p_b} \circ f_{p_a}$ is also bijective and order-preserving, and so $L_1 \overset{L}{\sim} L_3$. Therefore $\overset{L}{\sim}$ is transitive.

As it is shown above, $\overset{L}{\sim}$ is reflexive, symmetric, and transitive. Therefore $\overset{L}{\sim}$ is an equivalence relation. □

Definition 3.3.14 *Tempomorphic Equivalence Set. Let P be a consistent PA graph with a timeline L. Let $\mathbb{L}$ be a set of all timelines that are tempomorphic with L. I say then that $\mathbb{L}$ is **a** tempomorphic equivalence set for P, and **the** tempomorphic equivalence set for L.*

Definition 3.3.15 *Neighboring Assignment Pair. Let $L : O_S \rightarrow A_P$ be a timeline for a PA graph $P = (V_P, E_P)$. Let $t_i, t_j \in V_P$ be two time points in the PA graph, let r_i and r_j be the assignments of t_i and t_j in O_S, respectively, and let $A_i, A_j \in A_P$ be the anchor sets for t_i and t_j, namely, $t_i \in A_i$ and $t_j \in A_j$. Therefore $L(r_i) = A_i$ and $L(r_j) = A_j$. If there does not exist any $r \in O_S$ such that $r_i < r < r_j$, I call (r_i, r_j) a neighboring assignment pair of O_S, which I can also write (r_i, r_{i+1}).*

I am now able to define a unique object that captures a single possible ordering of timepoints from a solution to a PA graph.

Definition 3.3.16 *Normal Form Timeline. Let P be a consistent PA graph, and $\mathbb{L}$ be a tempomorphic equivalence set for P. Let $L_N \in \mathbb{L}$ be the timeline whose ordered solution set O_{S_N} contains only integers, whose first element is 1, and for which every neighboring assignment pair is of the form $(r_i, r_i + 1)$; that is, the difference between every r_i and r_{i+1} is exactly 1. Then I call L_N a normal form timeline for P and the normal form timeline for $\mathbb{L}$.*

Definition 3.3.17 *Normal Form Solution. Let L_N be a normal form timeline for P, and let S_N be the solution that underlies L_N and that consists of positive integers only, with each neighboring assignment pair differing by exactly 1. Then I call S_N a normal form solution.*

Lemma 3.3.18 *There exists a single unique normal form timeline L_N for each tempomorphic equivalence set $\mathbb{L}$.*

Proof. (a) Existence. By Definition 3.3.14 there exists at least one timeline $L_1 \in \mathbb{L}$ that has an ordered solution set $O_{S_1} = (r_1, r_2, \ldots, r_j)$. I can construct a second ordered solution set $O_{S_2} = (1, 2, \ldots, ..., j)$ as well as a new timeline $L_2 : O_{S_2} \to A_P$. By construction, O_{S_2} is order-isomorphic to O_{S_1}, and so replacing every r_i in S_1 with its corresponding integer index i is also a solution to the PA graph P for $\mathbb{L}$. This means that $L_2 \in \mathbb{L}$, and L_2 is by construction a normal form timeline.

(b) Uniqueness. Assume that $L_{N_1}, L_{N_2} \in \mathbb{L}$ are both normal form timelines for a tempomorphic equivalence set. Because L_{N_1} and L_{N_2} are tempomorphically equivalent, there exists a bijective order-preserving function f_p that relates their ordered solution sets O_{S_1} and O_{S_2}; this means $|O_{S_1}| = |O_{S_2}|$, namely, they are the same size. Further, because L_{N_1} and L_{N_2} are normal form timelines, the smallest element in O_{S_1} and O_{S_2} is 1, with all subsequent elements in each set being the integers greater than one up to the size of each set. Therefore $O_{S_1} = O_{S_2} = O_S$. Moreover, for each $r \in O_S$, let $L_1(r) = A_{r_1}$ and $L_2(r) = A_{r_2}$. Because f_p is bijective and order-preserving and $O_{S_1} = O_{S_2}$, f_p must be the identity function, and so $L_1(r) = L_2(f_p(r)) = L_2(r) = A_r$. Therefore, $L_1 = L_2$. □

Definition 3.3.19 *Timeline Length. Let $L : O_S \to A_P$ be a timeline. Then the number of elements in O_S (i.e., $|O_S|$) is called the length of the timeline L. Because a timeline is a bijective function, the length is also the same as the size of A_P. I will denote the length of L as $|L|$.*

Lemma 3.3.20 *Every timeline $L \in \mathbb{L}$ has the same timeline length.*

Proof. Let $L_1, L_2 \in \mathbb{L}$ be timelines where $L_1 : O_{S_1} \to A_P$ and $L_2 : O_{S_2} \to A_P$. By Definition 3.3.12 there exists a bijective function f_p that relates O_{S_1} and O_{S_2}. Therefore $|O_{S_1}| = |O_{S_2}|$ and so L_1 and L_2 have the same length. □

Definition 3.3.21 *Shortest Tempomorphic Equivalence Set. Let P be a consistent PA graph, and let $\hat{\mathbb{L}}_P$ be the set of all tempomorphic equivalence sets for P. Let $\mathbb{L}^0 \in \hat{\mathbb{L}}_P$*

be a tempomorphic equivalence set that contains a shortest timeline. I call $\mathbb{L}^0$ a shortest tempomorphic equivalent set for P.

Definition 3.3.22 *Shortest Normal Form Timeline. Let $\mathbb{L}^0$ be a shortest tempomorphic equivalent set for P. I will call the normal form timeline of $\mathbb{L}^0$ the shortest normal form timeline for P, or L^0.*

Because every timeline length is at least 1, I know that there exists at least one shortest tempomorphic equivalence set and at least one shortest normal form timeline. It remains to be shown, then, that the shortest normal form timeline is unique: namely, that for any PA graph there is a single solution that is both normal form and strictly smaller than any other solution.

Lemma 3.3.23 *Existence of Unique Smallest Positive Integer Solution. Let P be a PA graph, and let $\hat{S}$ be the set of all possible positive integer solutions to P. Then there exists a unique single smallest positive integer solution $S_0 = (s_{01}, s_{02}, \ldots s_{0j})$, such that for any other solution $S_i \in \hat{S}$, for each value $s_{ik} \in S_i$ of time point t_k in S_i, $s_{0k} \leq s_{ik}$.*

Proof. I shall proceed by contradiction. Assume $S_1 = (s_{11}, s_{12}, \ldots, s_{1j})$ and S_2 = $(s_{21}, s_{22}, \ldots, s_{2j})$ are distinct smallest positive integer solutions to P. Because they are distinct, they must differ at at least one assignment, namely $s_{1i} \neq s_{2i}$ for some i, which means that either $s_{1i} > s_{2i}$ or $s_{1i} < s_{2i}$. In either case one of the two solutions is not a smallest positive integer solution, in contradiction to my original assumption. □

Lemma 3.3.24 *Unique Smallest Positive Integer Solution is Normal Form. Let P be a PA graph, and let S_0 be the unique smallest positive integer solution. Then S_0 is a normal form solution.*

Proof. I shall proceed by contradiction. Assume S_0 is not normal form. That means there is a neighboring assignment pair that is not of the form $(r_i, r_i + 1)$, which means that

there is at least one integer r_j such that $r_i < r_j < r_{i+1}$, but $r_j \notin S_0$. I can construct a new assignment S_1 by subtracting 1 from all $r_i \in S_0$ where $r_i > r_j$. Because r_j was not in S_0, this new assignment still satisfies all the constraints in P, and so is a solution to P. But that means that S_0 was not a smallest positive integer solution, because for all indices that were assigned an integer $> r_j$ in S_0, they have a smaller positive integer assignment is S_1. Therefore S_0 must be a normal form solution. □

I have now established that there is a single, unique shortest normal form timeline L^0 for every consistent PA graph P. The theorem for Step 3 now follows:

Theorem 3.3.25 *On an input of a vector of consistent, connected PA graphs $T_P = \{P_1, P_2, \ldots P_k\}$, Step 3 of* TLEX *will output a vector $T_L^0 = (L_1^0, L_2^0, \ldots L_n^0)$ in which each L_i^0 is the shortest normal form timeline for P_i.*

Proof. By Definition 3.3.6, a PA graph is consistent if and only if it has a solution. Therefore, a solution to a consistent PA graph must exist. Pearl and Korf (1987) showed that solving a CSP can be formulated as depth-first search on a finite graph, which means that a procedure exists that will eventually find the single unique smallest solution S^0 containing only positive integers for that CSP Zhuk (2020, p. 7). For each PA graph $P_i \in T_P$, TLEX uses that procedure to find the smallest solution S_i^0, as well as the corresponding ordered solution set $O_{S_i^0} = (r_i \mid r_i \in S_i^0)$ that contains all the numbers in S_i^0 in their natural order. TLEX then creates an empty timeline L_i, and for each $s_i \in S_i$ creates an empty anchor set A_i and adds the ordered pair (s_i, A_i) to L_i. Then, for every pair $t_j = r_j$ in the assignment, TLEX adds t_j to the A_j corresponding to r_j. L_i is then, by construction, a timeline for P_i. Because each solution S_i^0 is the smallest positive integer solution, by Lemma 3.3.24 each S_i^0 is also a normal form solution, which means that each L_i is a normal form timeline. Therefore, I have shown that the timelines T_L^0 generated by TLEX in Step 3 are all the shortest normal form timelines for the PA graphs in T_P. □

3.3.4 Proof of Step 4: Correcting

The success of the third step requires that each PA graph be consistent. However, depending on how the TimeML graph was generated or the semantics of the underlying language, the TimeML graph (and so the corresponding PA graph) might not be consistent. To address this possibility TLEX uses Algorithm 1a in its fourth step to detect the specific sources of inconsistency in the PA graph so that it may be manually corrected (if possible). Zaidi (1999) defines inconsistency of a point algebra graph as follows:

Definition 3.3.26 *Inconsistency in a PA Graph. If a PA graph is inconsistent—that is, no solution exists according to Definition 3.3.6—then it must be the case that at least two different order relationships (drawn from the set* $\{<, =, >\}$*) must hold between the same two time points* t *and* s*. That is, both* tR_is *and* tR_js *must simultaneously be true, where* $R_i \neq R_j$*.*

In the fourth step, TLEX detects a set of connected sub-graphs of the PA graph that is responsible for one or more inconsistencies. These subgraphs, and their associated TimeML relations and text, can be presented to human annotators, which allows them to fix inconsistencies manually. The proof of the fourth step begins with the definition of an inconsistent cycle.

Definition 3.3.27 *Inconsistent Cycle. An inconsistent cycle is a cycle in a PA graph with one of the following forms van Beek (1994):*

1. $v = ... = w \neq v$
2. $v \leq ... \leq w \leq ... \leq v \neq w$*, where some or all of the* $\leq$ *could be* $=$ *or* $<$
3. $v < ... < w < ... < v$*, where all but one of the* $<$ *can be* $\leq$ *or* $=$ *(Note that this case also covers the case of self-loops,* $v < v$*.)*

Algorithm 1a produces a set of subgraphs, each of which is potentially a union of multiple overlapping inconsistent cycles. I call these maximal inconsistent connected PA subgraphs:

Definition 3.3.28 *Maximal Inconsistent Connected PA Subgraph. A maximal inconsistent subgraph $I = (V_I, E_I)$ is a connected subgraph of a PA graph P in which every edge $e \in E_I$ is part of an inconsistent cycle in P, and for every inconsistent cycle c_i of which e is a part, all the edges and nodes of c_i are contained in I.*

Building on these definitions, the proof of Step 4 (Algorithm 1a) is as follows.

Theorem 3.3.29 *On an input of an inconsistent PA graph $P = (V_P, E_P)$, Step 4 (Algorithm 1a) of* TLEX *will give as output a graph J containing the union of all maximal inconsistent connected PA subgraphs present in the graph.*

Proof. **Step (a):** This step identifies all self loops. Detecting a self loop is trivial: each edge can be directly inspected to see if its endpoints are equal, and whether the edge is a *less than* edge ($<$). Therefore at the end of step (a) E_L contains all self-loop edges.

Step (b): van Beek (1994) shows that a PA graph is inconsistent if and only if there is an inconsistent cycle in the graph. If an edge is removed from a cycle, the rest of the edges can be satisfied because there is no longer a cycle. Step (b) begins with an empty set of edges E_N, and adds all non-self-loop edges (E_P/E_L) to E_N one by one, testing consistency of the graph (V_P, E_N) after each addition. If, after the addition of each edge (V_P, E_N) is found inconsistent, it must be the case that the last added edge induced inconsistency; this edge is thus removed from E_N and added to E_B. In this way, at the end of step (b), (V_P, E_N) is a consistent graph, E_B contains one edge per inconsistent cycle, and the union of E_N and E_B contains all the edges in the original graph ($E_N \cup E_B = E_P$).

Step (c): The addition of any edge $e_b \in E_B$ to E_N will induce inconsistency in (V_P, E_N), completing any inconsistent cycles of which e_b is a member. Therefore each e_b is added back to E_N in turn, and the function SEARCHFORCYCLES performs an exhaustive Depth-First Search (DFS) starting from e_b, without repeating edges, discarding any paths that are not cycles that end on e_b, and adding found cycles to X. Therefore, X must contain all possible cycles containing e_b. For each cycle E_Y that is found, ISCYCLECONSISTENT walks the cycle, keeping track of whether or not *less than* relations all point in the same direction; if not, it is an inconsistent cycle, and it is added to E_I. Therefore, at the end of step (c), E_I contains all and only the edges involved in inconsistent cycles involving the edges in E_B.

Step (d): I can easily extract the nodes involved in all edges in $E_I \cup E_L$ to produce V_I. Because $E_I \cup E_L$ contains all edges involved in inconsistent cycles, the graph $J = (V_I, E_I \cup E_L)$ is equal to the union of all maximal inconsistent connected PA subgraphs in P. $\square$

3.3.5 Proof of Step 5: Identifying Indeterminacy

In the fifth step, TLEX detects indeterminate sections on the timeline using Algorithm 2. I begin the proof of this step by defining order-equivalence and order-uniqueness with respect to ordered solution sets.

Lemma 3.3.30 *The existence of a bijective, order-preserving function f_p between two ordered solution sets defines an equivalence relation, which I will call the order-equivalent equivalence relation $\overset{O}{\sim}$.*

Proof. To show that $\overset{O}{\sim}$ is an equivalence relation it is sufficient to show that it is reflexive, symmetric, and transitive.

- *Reflexivity*: Let O_S be a solution set for a PA graph P. The identity function I is a bijective, order-preserving function $I : O_S \to O_S$. Therefore, O_S is order-equivalent relative to itself.
- *Symmetry*: Let O_{S_1} and O_{S_2} be ordered solution sets for a PA graph P, and let $O_{S_1} \overset{O}{\sim} O_{S_2}$. This means there exists a bijective order-preserving function f_p between O_{S_1} and O_{S_2}. Because f_p is bijective, it must have an inverse f_p^{-1} that is also bijective and order-preserving. Therefore $O_{S_2} \overset{O}{\sim} O_{S_1}$.
- *Transitivity*: Let O_{S_1}, O_{S_2}, and O_{S_3} be ordered solution sets for a PA graph P, and let $O_{S_1} \overset{O}{\sim} O_{S_2}$ and $O_{S_2} \overset{O}{\sim} O_{S_3}$. This means there exists two bijective, order-preserving functions f_{p_a} and f_{p_b} such that $f_{p_a} : O_{S_1} \to O_{S_2}$ and $f_{p_b} : O_{S_2} \to O_{S_3}$. Because both functions are bijective and order-preserving, the composite function $f_{p_a}(f_{p_b})$ is also bijective and order-preserving, meaning $O_{S_1} \overset{O}{\sim} O_{S_3}$.

Therefore $\overset{O}{\sim}$ is an equivalence relation. □

Definition 3.3.31 *Order-Uniqueness. Let $\hat{O}$ be a set of ordered solution sets and let $O_i \in \hat{O}$. If O_i is not order equivalent to any other $O_j \in \hat{O}$ I say that O_i is order-unique with respect to $\hat{O}$.*

Algorithm 2 constructs an *indeterminacy* map that identifies which time points have indeterminate order.

Definition 3.3.32 *Canonical Neighboring Time Point Pair. Let $t_i, t_j \in V_P$ be a pair of time points for a PA graph $P = (V_P, E_P)$. Let $L_0 : O_S \to A_P$ be the shortest normal form timeline for P, and let $r_i, r_j \in O_S$ be the assignments for t_i and t_j, namely, $L_0(r_i) = A_i$ where $t_i \in A_i$, and $L_0(r_j) = A_j$ where $t_j \in A_j$. $p = (t_i, t_j)$ is a canonical neighboring time point pair for P if (r_i, r_j) is a neighboring assignment pair relative to L_0.*

Definition 3.3.33 *Indeterminacy Map. Let P be a PA graph. Let N_{2P} be the set of all canonical neighboring time point pairs $p_i = (t_i, t_j)$ for P. An indeterminacy map $D : N_{2P} \rightarrow \{true, false\}$ is a function where for every canonical neighboring time point pair $p_i \in N_{2P}$, p_i is mapped to true if there exists some timeline L for P such that the assignment pair (r_i, r_j) for p_i relative to L is not a neighboring assignment pair. Otherwise, p_i is mapped to false.*

The indeterminancy map captures whether or not two time points in the PA graph P are found next to each other in all possible timelines for P.

The theorem for the fifth step is as follows.

Theorem 3.3.34 *On an input of a vector of consistent, connected PA graphs $T_P = \{P_1, P_2, \ldots P_k\}$, Step 5 (Algorithm 2) of* TLEX *will output a vector $T_D = \{D_1, D_2, \ldots, D_k\}$ in which each D_i is the indeterminacy map for each P_i.*

Proof. Gent and Smith (1999) demonstrated a procedure for finding one solution of a CSP from each equivalence class of solutions defined by a generic symmetric relation g. By Lemma 3.3.30, the order-equivalent relation is a symmetric relation, and so the procedure of Gent and Smith can be used to find one order-equivalent solution from each order-equivalent equivalence class for the CSP. TLEX uses such a procedure to produce these solutions $S_{P_i} = \{S_{i1}, S_{i2}, \ldots\}$ for each P_i, with corresponding ordered solution sets $O_{P_i} = \{O_{S_{i1}}, O_{S_{i2}}, \ldots\}$ and timelines $L_{P_i} = \{L_{i1}, L_{i2}, \ldots\}$. By construction, each ordered solution set in $O_{S_{ij}} \in O_{P_i}$ is order-unique with respect to O_{P_i}, which means there does not exist a bijective, order-preserving function f_p that maps any $O_{S_{ij}} \in O_{P_i}$ to any other ordered solution set in O_{P_i}. This means no timeline in L_{P_i} is tempomorphically equivalent to any other timeline in L_{P_i}, which means every timeline $L_{ij} \in L_{P_i}$ is drawn from its own unique tempomorphic equivalent set $\mathbb{L}_{ij}$. Because every timeline from every

possible tempomorphic equivalent set is present in L_{P_i}, this means that the union of all the tempomorphic equivalence sets for each timeline in L_{P_i} is equal to $\hat{\mathbb{L}}_{P_i}$.

Recall that Theorem 3.3.25 showed that for a set of connected PA graphs $T_P = \{P_1, P_2, \ldots, P_n\}$ where each P_i is also a CSP, Step 3 of TLEX will output a vector of shortest normal form timelines $T_L = \{L_{0_1}, L_{0_2}, \ldots, L_{0_n}\}$. Algorithm 2 creates an empty indeterminacy map D_i for each timeline in T_L. Then, for every canonical neighboring time point pair p_i in each L_{0_i}, the algorithm checks whether that pair is a neighboring assignment pair in each timeline in L_{P_i}. If so, it adds $(p_i, false)$ to D_i, meaning the time point pair is determinate. Otherwise, it adds $(p_i, true)$, meaning the time point pair is indeterminate. At the end, TLEX returns a set of indeterminacy maps $T_D = \{D_1, D_2, \ldots, D_n\}$ for a set of connected consistent PA graphs $T_P = \{P_1, P_2, \ldots P_n\}$. $\square$

3.3.6 End-to-End Proof

I informally defined a "main" timeline as a timeline containing events and times that actually happen in the "world" described in the text. Because I do not currently have an automatic way of identifying main timelines, I will rely on information external to TLEX to identify the main timeline. For this, it is sufficient for a person to identify at least one event or time that occurs on every disjoint main timeline for the text.

Definition 3.3.35 *Trunk-and-Branch Timeline Let T be a consistent TimeML graph with temporally connected subgraphs $T_C = \{C_1, C_2, \ldots, C_n\}$ and corresponding timelines $T_L = \{L_1, L_2, \ldots, L_n\}$, and let $T_M \subset T_L$ be the set of main timelines. A trunk-and-branch timeline for T is a pair $B_T = (G_B, T_M)$ where $G_B = (T_L, E_S)$ is a graph where the edges E_S are subordinating TimeML links between timelines.*

Definition 3.3.36 *Normal-Form Trunk-and-Branch Timeline Let $B_T = (G_B, T_M)$ be a trunk-and-branch timeline for a consistent TimeML graph T with temporally connected*

subgraphs $T_C = \{C_1, C_2, \ldots, C_n\}$ and corresponding timelines $T_L = \{L_1, L_2, \ldots, L_n\}$. I say B_T is a normal-form trunk-and-branch timeline if every timeline in T_L is a normal-form timeline.

Definition 3.3.37 *Shortest Trunk-and-Branch Timeline Let $B_0 = (G_B, T_M)$ be a trunk-and-branch timeline for a consistent TimeML graph T with temporally connected subgraphs $T_C = \{C_1, C_2, \ldots, C_n\}$ and corresponding timelines $T_L = \{L_1, L_2, \ldots, L_n\}$. I say B_T is a shortest trunk-and-branch timeline if every timeline in T_L is a shortest timeline.*

My end-to-end theorem is as follows:

Theorem 3.3.38 *On an input of TimeML graph $T = (V_T, E_T)$, with temporally connected subgraphs $T_C = \{C_1, C_2, \ldots, C_n\}$, with a subset of events or times $V_M \subset V_T$ identified as being on a "main timeline,"* TLEX *produces one of the two following outputs: (a) If all $C_i \in T_C$ are consistent,* TLEX *outputs (i) the shortest normal-form trunk-and-branch timeline $B_0 = (G_0, T_M)$ for T, with $V_M \subset \bigcup_i V_{C_{M_i}}$, where each $C_{M_i} = \{V_{C_{M_i}}, E_{C_{M_i}}\} \in T_C$ is the temporally connected subgraph associated with a main timeline L_{M_i}; and (ii) a vector of indeterminacy maps $T_D = \{D_1, D_2, \ldots, D_n\}$ corresponding to each timeline in T_L. (b) If any $C_i \in T_C$ is inconsistent,* TLEX *outputs a vector $T_J = (J_1, J_2, \ldots, J_n)$ of unions of all maximal inconsistent PA subgraphs, where each J_i corresponds to the union of maximal inconsistent PA subgraphs in each C_i.*

Proof. Part (a): By Theorem 3.3.3, on input of a TimeML graph T, TLEX Step 1 outputs a set of temporally connected TimeML graphs T_C and a set of all subordinating links E_S. By Theorem 3.3.5, TLEX Step 2 transforms T_C into a vector T_P of point algebra graphs. By Theorem 3.3.25, if every $P_i \in T_P$ is consistent, and so every $T_i \in T_C$ is consistent, TLEX Step 3 outputs a vector of T_{L_0} of shortest, normal-form timelines for

T_P. I can identify which C_i has nodes in V_M and thus identify each $L_{M_i} \in T_{L_0}$. By Theorem 3.3.34, TLEX Step 5 outputs a vector T_D of indeterminacy maps for T_P. I can then construct $G_0 = (T_{L_0}, E_S)$ and $B_0 = (G_0, T_M)$.

Part (b): Steps 1 and 2 are same as for Part (a). However, in this case, there exists one or more $P_i \in T_P$ that are inconsistent. For each inconsistent P_i, by Theorem 3.3.29, TLEX Step 4 will produce J_i, a union of all maximal inconsistent subgraphs for P_i, and for each consistent P_i, $J_i = \emptyset$. These can be concatenated into a vector $J = (J_1, J_2, \ldots, J_n)$.

□

3.4 Experimental Evaluation

The prior section shows the formal correctness of TLEX; here I report an experimental evaluation and show that TLEX performs as expected on real data. I evaluated TLEX on the four corpora described previously in Table 2.1. As described in §3.2.4, I ran Algorithm 1a across all inconsistent texts and manually corrected those TimeML graphs with reference to the original text. To emphasize the importance of ALINKs, I performed inconsistency detection on TimeML graphs with and without ALINKs. Table 3.2 shows the number of inconsistent files in the corpora. Without ALINKs, TLEX determined that 30 texts in the TimeBank corpus. Importantly, these inconsistencies are exactly the same 30 as those found by Derczynski and Gaizauskas (2010). But more importantly, this number is increased significantly after the consideration of ALINKs. To correct these inconsistencies, I removed self-loops automatically and manually corrected larger cycles. All manual corrections were performed by reference to the original texts. I produced fully consistent TimeML graphs for all 385 texts. In some cases, I corrected inconsistencies by removing an incorrect link in the original TimeML graph; in other cases by changing an incorrectly labeled link.

Corpus	Total Files	Inconsistent Files (TLINKs only)	Inconsistent Files (TLINKs & ALINKs)
Timebank 1.2	183	30	65
N2	67	10	11
ProppLearner	15	9	11
MEANTIME	120	36	36*
Total	385	85	123

Table 3.2: Details of the inconsistencies of the four corpora. *The MEANTIME corpus does not have ALINKs.

	Length of Main Timeline			**# Subordinated Branches / Text**			**# of Indeterminate Sections / Text**		
Corpus	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
TimeBank 1.2	4	8.1	18	1	16.0	142	0	5.3	11
N2	6	24.7	72	1	19.3	106	0	10.2	21
ProppLearner	128	180.5	290	33	82.7	134	0	28.8	63
MEANTIME	4	7.3	13	0	2.0	7	0	3.8	7
Total	4	17.5	290	0	14.8	142	0	6.6	63

Table 3.3: Details of the timelines extracted from the four corpora.

After correction, I extracted all the timelines for all texts using TLEX Steps 1–3 and identified their indeterminate sections using Step 5. Table 3.3 shows statistics on the length of the shortest main timelines (i.e., the number of time steps in the shortest solution), the number of subordinated timelines, and the number of indeterminate sections.

3.4.1 Sampling Evaluation

As I do not have gold-standard annotated timelines against which to compare, I cannot directly compute recall, precision, or F_1 for the output of TLEX. Therefore I performed a sampling evaluation to check the extracted timelines, where I selected aspects of timelines at random to check against the original texts. I used *Simple Random Sampling* (SRS; Saunders et al., 2009, p. 222) wherein every member of a population has an equally likely

chance of being selected. In SRS, I check the correctness of a specific feature of a set of n members randomly selected from a population with size N to obtain an estimate of the correctness of that feature over all the data. The formula for calculating sample size for a finite population is the following:

$$n = \frac{n_0}{1 + \frac{n_0 - 1}{N}} \quad \text{where} \quad n_0 = \frac{Z^2}{4c^2}$$

Where c is the confidence interval, Z is the Z-score for the confidence level, and N is the population size. For all experiments, I used a c of 0.02 (2%) and a confidence level of 0.95 (95%), which corresponds to a Z-score of 1.96.

I evaluated the extracted timelines via simple random sampling, checking five features: (1) the ordering of time points with respect to each other, (2) the number of main timelines, (3) the placement of events on main versus subordinated timelines, (4) the attachment point of subordinated timelines, and (5) whether indeterminate sections are truly indeterminate with respect to the underlying TimeML graph.

Time Point Ordering

Time point ordering was the first feature I evaluated. I randomly picked neighboring pairs of time points from the TLEX-generated timelines and compared them to the TimeML graph to ensure that the timeline and the graph were consistent. For example, if I examined a pair of neighboring points labeled A^+ and B^-, I examined the paths in the TimeML graph between A and B to determine whether this ordering was correct. Using SRS I selected 2,264 pairs of time points out of 39,534 possible neighboring assignment pairs, sampled from each corpus in proportion to the total number of time points in each. I found each pair to be correctly ordered with respect to the original text. Because the confidence interval is 2%, this results in an estimated accuracy of 98–100% with 95% confidence. I can compare this result with three pieces of prior work. Mani et al. 2006,

as described above, used only three types of temporal relations, achieving an ordering accuracy of roughly 75% for events, while Kolomiyets et al. 2012 worked with only six temporal relations, achieving an ordering accuracy of 70%. In contrast, TLEX achieves a perfect ordering accuracy, modulo sampling error, and correctness of the TimeML graph, using all 25 relation types.

Number of Main Timelines

The number of main timelines was the second feature I evaluated. Identification of the main timelines requires the identification of at least one event or time on each main timeline, which is a laborious process. I approximated this identification, therefore, by identifying all timelines that did not have incoming subordination links from another timeline.

With this identification in hand, I assessed whether multiple main timelines actually corresponded to temporally disjoint and otherwise unrelatable graphs. In texts with multiple main timelines, I define the *breaking pair* between two timelines as the pair of events, times, or an event and time, one from each timeline that is closest in the text measured by the number of intervening words. Because the population size (1,241) was not substantially larger than the sample size (818), I manually checked all breaking pairs in the extracted timelines, and whether they actually indicated disjoint timelines. I observed that all of the breaking pairs corresponded with true disjoint breaks between timelines in the texts.

In some cases, I observed that sometimes a text had multiple timelines that were disjoint, but should have been collected together into a single main timeline because they were temporally relatable. This is because the annotation of the text is missing a relationship between one group of events or times and another group of events or times, but in both cases, those events occur in the "real world" of the text. In other words, the TimeML graph was disconnected in a way that was inconsistent with the actual semantics of the

text, meaning the TimeML annotation was incorrect. I observed this situation for 181 texts, which is another dimension along which the TimeML annotations for these corpora might be improved.

Time Point Placement

The third feature I evaluated was whether time points were correctly placed on main versus subordinated timelines, using my approximation for main timelines as indicated in the last subsection. I checked to see whether a time point placed on a subordinate timeline did not occur in the "real world" described in the text. A total of 11,474 time points were on subordinated timelines, and I sampled 1,986 time points across the corpora, again in proportion to their number in each corpus. Again, all samples were correct, giving an estimated accuracy of 98–100% with 95% confidence.

Subordinated Attachment Points

The fourth feature I checked was to confirm that subordinated timelines were connected to the correct time points on identified main timelines. There are 5,701 subordinated timelines in my data, and each one was connected to a main timeline with at least one subordinating link. I checked 1,690 connections, observing that every connection was correctly placed, giving an estimated accuracy of 98–100% with 95% confidence.

Indeterminate Sections

The fifth and final feature I checked was whether indeterminate sections were truly indeterminate with respect to the TimeML graph. There are 2,541 indeterminate sections, comprising 11,688 pairs of time points. I randomly selected 1,992 time point pairs from the indeterminate sections. In all cases the pairs were truly indeterminate, meaning an estimated accuracy of 98–100% with a 95% confidence.

3.5 jTLEX: A Java library for TimeLine EXtraction

TLEX has been used for several NLP tasks such as corpus validation, evaluating temporal dependency parsers, and narrative representation (Ocal et al., 2022a,b; Ocal and Finlayson, 2020). To enable better access to this approach for the community, we have implemented jTLEX, an open-source Java implementation of TLEX for other researchers in the field to use. [2]

TLEX provides several types of functionality. In its canonical usage, jTLEX takes a TimeML annotated file as input, then (1) parses the annotations into TimeML objects, (2) builds a TimeML graph, (3) partitions the TimeML graph into temporally connected graphs to separate real-life events and subordinated events, (4) transforms the temporally connected graphs into point algebra (PA) graphs, and (5) solves the PA graphs to extract a timeline. If a timeline cannot be extracted, meaning the graph is temporally inconsistent, (6) it detects the minimum inconsistent subgraph and returns it to the annotator to fix it. Finally, if the order of events and times are indeterminant (multiple possible ordering), (7) it calculates the temporal indeterminacy. These steps correspond to steps of the TLEX algorithm as described earlier.

We (the jTLEX capstone team and I) have tested jTLEX on the entire TimeBank corpus Pustejovsky et al. (2003b), which is a reference corpus for TimeML that contains 183 TimeML annotated news articles. For each file, jTLEX took no longer than 1 second to extract all timelines on a currently available, standard consumer laptop (3.0 GHz Intel Core i7-1185G7 with 32GB of RAM).

[2] As mentioned in the Acknowledgments, the jTLEX capstone team contributed to the jTLEX implementation.

3.5.1 Library Overview

User Input

jTLEX processes and allows the manipulation of all the information available in a TimeML annotation. jTLEX can read in preexisting TimeML annotations from a `.tml` file, accept TimeML annotations directly via a JavaScript Object Notation (JSON)-style TimeML encoding, or accept the raw text of TimeML annotations as a Java String object. TimeML annotations can be generated via manual annotation by following the TimeML annotation guide (Sauri et al., 2006) or by using state-of-the-art TimeML annotators such as TARSQI (Verhagen et al., 2005), ClearTK (Bethard, 2013), CAEVO (Chambers et al., 2014), or CATENA (Mirza and Tonelli, 2016). Note that there are limitations when using automatic TimeML annotators such as information loss and temporal inconsistency (Ocal et al., 2022a). Fortunately, jTLEX can detect inconsistencies and help users to fix them as explained in Section 3.5.1.

TimeML Parser

TimeML is a standard generalized markup language (SGML)-based annotation scheme to annotate temporal information in texts. TimeML defines tags for events, temporal expressions, temporal signals, event instances, and links between events and times, namely TLINK, ALINK, and SLINK. Each tag has attributes that contain information about a TimeML object. For example, the "`polarity`" attribute of <MAKEINSTANCE> indicates whether an event is negated and it contains either `POS` or `NEG` values.

jTLEX provides a TimeML parser that can parse TimeML annotations into a set of TimeML Java objects, including events, times, and links. Additionally, it can strip the TimeML tags and return the raw text. The TimeML annotation guide identifies optional and non-optional attributes for each TimeML tag. If the input to jTLEX is missing a non-

optional attribute, the parser returns an error message to the user about which attribute is missing for which object. Therefore, jTLEX's TimeML parser can be used to check the compliance of annotations to the TimeML standard.

Graph Constructor

A TimeML graph is a graph in which nodes are events and times, and edges are TimeML links (as shown in Figure 3.1). After jTLEX parses a TimeML annotation into the TimeML objects (events, times, links, etc.), it builds a TimeML graph. Any information about the graph can be then programmatically queried, such as the set of links, the set of nodes, a link by ID, a node by ID, a list of incoming or outgoing links, and more.

jTLEX allows users to directly modify the TimeML graph if they wish. Users can add or remove links or nodes to the graph, and can also build their own custom graph by creating an empty graph and adding events, times, and links. The graph implementation has a method that returns a JSON output of the graph. This allows users to take advantage of existing graph visualization systems such as React Flow (So, 2018) to inspect the TimeML annotations.

Partitioner

TLEX Step 1 partitions the graph into temporally connected graphs. jTLEX implements this with its partitioner class. The partitioner has two steps: jTLEX walks the graph to partition the TimeML graph into connected subgraphs. Then it further partitions these connected subgraphs into subgraphs connected only with temporal and aspectual links. We name the subgraph(s) that contain "real world" events *main* subgraph(s), and subgraphs that connect to main subgraphs via subordination links *subordinated* subgraphs.

Transformer

As prescribed by the TLEX algorithm, jTLEX transforms each temporally connected subgraph into a PA graph. This transformation is necessary because the timeline is generated by solving the temporal constraint satisfaction problem (TCSP) represented by the PA graph, as discussed in the next section.

Solver

After the transformer transforms each temporally connected subgraph into a PA graph, jTLEX assigns integers to each time point in the graph using Java Constraint Programming (JaCoP) library (Kuchcinski and Szymanek, 2013). The library then obtains a timeline after sorting the assigned integers. jTLEX by default produces the smallest solution, which starts at 1 and which represents each time point difference as a difference of 1. Users can also use jTLEX to produce a random solution.

When run over all the PA graphs, jTLEX produces an *exact* trunk-and-branch *timeline* where the trunk is the main timeline corresponding to the main subgraph and branches are subordinated timelines corresponding to the subordinated subgraphs. Therefore, the main timeline consists of the global order of "real world" events and times, while subordinated branches consist of subordinated events. Users can retrieve the length of the timeline, the first and last time points, the main timeline, subordinated branches, the number of subordinated branches, the number of time points, and the list of attachment time points where subordinated branches are connected to the main timeline. Users can also retrieve the JSON output of the timeline, and therefore they can visualize the timeline using a third-party graph visualizer application.

Inconsistency Detector

As described in the TLEX work (§3.2.4), the solver can only extract a timeline of the TimeML annotation if the annotation is temporally consistent. If the integer assignment is not possible, then it means the TimeML graph has temporal inconsistency (Barták et al., 2014). jTLEX provides an inconsistency detector to detect inconsistent cycles in the TimeML graph.

The inconsistency detection algorithm detects self-loops, where the start and end node of an edge is the same such as A-*AFTER*$\rightarrow A$. This can be removed automatically as users request. For other non-self-loop cycles, it detects the minimum inconsistent subgraph in the TimeML graph. Further, in the case when two or more inconsistent subgraphs overlap (have a shared edge), it can distinguish the subgraphs and return each inconsistent cycle to the users. A jTLEX output of an inconsistent cycle is shown in Section 3.5.2.

Users can use jTLEX to check whether their annotation is temporally consistent. If the annotation is not consistent, jTLEX returns the links that cause inconsistency, therefore, users can fix the inconsistent annotation. This could be used for corpus validation after NLP researchers create a corpus.

Indeterminacy Calculator

jTLEX uses TLEX's Step 5 to measure temporal indeterminacy in a timeline. Using jTLEX, users can retrieve the indeterminant sections of a timeline. jTLEX also provides a standard way of scoring the amount of indeterminacy present in a particular timeline.

3.5.2 Use Cases

Here I illustrate an approach for one of the TimeML annotations of the TimeBank corpus, called `wsj_0006.tml` ((1) in Chapter 5). This file and the rest of the corpus can be

```
Link: {ID = 1, LinkTag = TLINK, Syntax = "", Temporal Relation =
    BEFORE, Origin = null
    Signal: {Id = sid12, String = "by"}
    Related to event - Timex: {tID = t10, Type = DATE, Value =
       1989-12-31 , Mod = null, Temporal Function = true,
       AnchorID = t9, Begin Point = t0, End Point = t0, Quantity
        = null, Frequency = null}
    Event Instance - Event Instance:
    {ID = eiid80, Tense = PRESENT, Aspect = NONE, Part of Speech
        = VERB, Polarity = POS, Modality = "null", Cardinality =
        "null", Signal = null
        EVENT: eid = e7, class = ASPECTUAL, stem = complete}
    }
```

Listing 3.1: jTLEX parser output for printing the information about the first link of the graph.

obtained from the LDC website[3].

Users can read the file and create the TimeML graph as follows:

```
File tmlFile = new File(fName);

ITimeMLGraph graph = GraphReader.TimeMLGraph(tmlFile);
```

Here, `fName` is the path to the file. Users can retrieve any information about the graph such as links (all or one by ID), nodes (all or one by ID), incoming links, outgoing links, JSON output, number of nodes, number of links, number of link types, etc. Using the following code, users can retrieve the information of the first link:

```
System.out.print(graph.getLinkById(1));
```

The output will be as shown in Listing 3.1. As can be seen, jTLEX provides all the available information in the TimeML annotation about the link and its components using the TimeML parser.

After the TimeML graph is created, users can create a `TLEX` object to perform the timeline extraction including partitioning, transforming, solving, inconsistency detection

[3] `https://catalog.ldc.upenn.edu/LDC2006T08`

```
Main Timeline: {
    eiid75- = 1
    eiid75+ = 2
    eiid74- = 3
    eiid74+ = 4
    eiid73- = 5
    eiid76- = 5
    eiid73+ = 6
    eiid76+ = 6
    t9- = 7
    t9+ = 8
}
Attachment Points: {eiid77->eiid78, eiid77->eiid80, eiid76->
   eiid77, eiid78->eiid79}
Subordinated Timelines: {
[eiid81- = 1, eiid80- = 2, eiid81+ = 3, eiid80+ = 3, t10- = 4,
   t10+ = 5],
[eiid79- = 1, eiid79+ = 2],
[eiid78- = 1, eiid78+ = 2],
[eiid77- = 1, eiid77+ = 2]}
```

Listing 3.2: jTLEX timeline output for the `wsj_0006.tml` file.

(if the graph is inconsistent), and temporal indeterminacy. Creating the TLEX object is as follows:

```
TLEX tlex = new TLEX(graph);
```

Users can retrieve the *exact* trunk-and-branch timeline structure using the following:

```
tlex.getTimeline();
```

The output will be as shown in Listing 3.2. As can be seen, jTLEX returns the *main* timeline, subordinated timelines, and the attachment points for each subordinated timeline.

Because the graph of `wsj_0006.tml` is consistent, jTLEX's inconsistency detection method returns an empty set. We illustrate the inconsistency detection algorithm using a temporally inconsistent file from the TimeBank corpus, called `wsj_1011.tml`.

```
[Graph Type: Main Graph
Nodes Count = 2
Links count = 2
TLinkType: 2
ALinkType: 0
SLinkType: 0
Nodes:
eiid2048, t57
Links: (From -> To)
(t57 BEFORE eiid2048)
(eiid2048 BEFORE t57)
]
```

Listing 3.3: jTLEX inconsistent subgraph output for the `wsj_1011.tml` file.

After running the method for graph construction and creating the TLEX object, users can simply call the method `tlex.getInconsistentSubGraphs()` and retrieve the inconsistent cycle. For this file, jTLEX returns the output shown in Listing 3.3. As can be seen from the output, jTLEX returns the inconsistent subgraph along with the information about the subgraph.

3.6 TLEX Application: Evaluating Information Loss in Temporal Dependency Trees

As I mentioned in §2.4, Temporal Dependency Trees (TDTs) have emerged as an alternative to full temporal graphs for representing the temporal structure of texts, with a key advantage being that TDTs can be straightforwardly computed using adapted dependency parsers. Relative to temporal graphs, the tree form of TDTs naturally omits some fraction of temporal relationships, which intuitively should decrease the amount of temporal information available, potentially increasing temporal indeterminacy of the global ordering. Using TLEX, I demonstrated an approach that measures the temporal link omission during the TDT construction and compares the timelines extracted from both TDTs and

TimeML graphs based on the global orders of events and times, inconsistency, and indeterminacy.

In order to evaluate what is lost when moving from TimeML temporal graphs to TDTs, I needed texts that have both TimeML and TDT annotations. This data could be generated in several ways. First, I could automatically generate TimeML and TDT annotations for texts using TimeML and TDT parsers. This is problematic because of a great deal of noise introduced by even state-of-the-art TimeML and TDT parsers, which then obscures which deficiencies are a result of the TDT representation as opposed to parser performance. Second, I could use a dataset that has gold-standard annotations of TimeML and TDTs; unfortunately, these do not yet seem to exist. There are numerous corpora with gold-standard TimeML annotations, but I was unable to find even a single manually annotated, publicly available TDT corpus that also had TimeML annotations. One corpus, the Temporal Dependency Tree Corpus (Zhang and Xue, 2018), has automatically generated TDT annotations, but no TimeML. The TDT Corpus is an automatically annotated collection of Chinese news reports and fairy tale stories. The TDTs in that corpus were generated using an adapted dependency parser (Robaldo et al., 2011), but the texts do not have corresponding TimeML graphs. I use this corpus only to compare the raw indeterminacy of automatically computed TDTs to my TimeML-derived TDTs. The third option, which I use here, is to generate TDTs programmatically from gold-standard TimeML annotated texts. This approach guarantees that I will have both gold-standard TimeML annotations as well as the best possible TDT for every annotation, therefore, they can be directly compared.

3.6.1 Generating TDTs

Because I'm automatically generating TDTs from TimeML graphs, and not generating TDTs using a TDT parser, there is a question as to whether the TDTs are faithful to the original TDT scheme. I first provide a brief description of how TDT parsers work, so that it can be seen that my generation algorithm (Algorithm 3) intuitively follows the automatic TDT parsing and thus produces a "best possible" TDT.

A TDT parser, as described by (Zhang and Xue, 2018), starts by initializing a TDT with a ROOT node that represents the document creation time (DCT), and which has four children (the *meta-nodes*): PAST_REF, PRESENT_REF, FUTURE_REF, and ATEMPORAL. The parser then proceeds through the text in reading order. Every time the TDT parser detects a new event or time expression, it attempts to find a relationship between that event or time and an existing node in the TDT in a breadth-first manner. If a relationship (link) is found to an existing node (by running a link classifier, e.g., an SVM), the new event or time is added as a child to that node, with the appropriate link type, and removed from further consideration. If no relationship is found, it is set aside and checked for a relationship with each new node that is later added to the tree. If, at the end of the text, there are still nodes that have no relationship to any other nodes in the tree, they are added as children of the ATEMPORAL meta-node.

To illustrate the TDT structure, I introduce here a snippet from the TimeBank 1.2 text *APW19980213.1320.tml* (Pustejovsky et al., 2003b), which I will use as an example throughout the remainder of this chapter. In this snippet, the underlined text refers to a marked TimeML event or a temporal expression, and is labeled with its ID (e.g., e_1 or t_{44}) from the annotation. To enhance understandability, I only show events and times related to the FUTURE_REF meta-node, which corresponds to one top-level branch of the resulting TDT.

DCT: 1998-02-13$_{t_{41}}$

Qantas will almost double$_{e_1}$ *its flights between Australia and India by August*$_{t_{43}}$ *in the search for new markets untouched by the crippling Asian financial crisis. This move*$_{e_{28}}$ *comes barely a month*$_{t_{44}}$ *after Qantas suspended*$_{e_5}$ *a number of services*$_{e_{29}}$ *between Australia, Indonesia, Thailand and Malaysia in the wake of the Asian economic crisis*$_{e_{30}}$*. The airline has also cut*$_{e_7}$ *all flights*$_{e_{31}}$ *to South Korea.*

The temporal graph for this snippet is shown in Figure 3.8(a). I can generate a TDT from this temporal graph by following Zhang and Xue's definition and procedure. During the process, I generate two types of TDTs: first, TDTs that use the full set of TimeML temporal and aspectual links (*full* TDTs). Then I apply the abstraction mappings shown in Table 3.4 to produce TDTs that use only Zhang and Xue's four types (*abstract* TDTs).

The algorithm for generating full TDTs from TimeML graphs is shown in Algorithm 3. It follows the TDT parsing procedure almost exactly, but rather than using a classifier to determine whether there is a relationship between a new node and the TDT, it queries the TimeML graph. I begin by initializing a FIFO queue with all events and times in the TimeML graph such that they will be returned in text order (line 2). I next initialize an empty TDT with a ROOT node and add all four meta-nodes to the tree as children of the root (lines 3–7). I then pop an event or a time from the queue (I call this event or time n; lines 8–20), first looking through the tree in a breadth-first manner for an event or time to which n is linked (line 11). The presence of a link is determined by querying the TimeML graph. Note that in a consistent TimeML graph no two nodes will be connected by more than one temporal or aspectual link. If n is not found to link to an existing node (line 12), then n is added to the unlinked set for future processing (line 13). If n is found to link to an existing node, it is added as a child to that node (line 15) and all unlinked nodes are checked for relationships to the new node (lines 16–20). Any events

TimeML Link Types	Zhang and Xue (2018) Abstracted Link Types
A BEFORE B A IBEFORE B B TERMINATES A	A BEFORE B
A AFTER B A IAFTER B	A AFTER B
A BEGINS B A BEGUN_BY B A ENDS B A ENDED_BY B A SIMULTANEOUS B A IDENTITY B B INITIATES A B CULMINATES A	A OVERLAPS$_{tdt}$ B
A INCLUDES B A DURING_INV B B CONTINUES A	A INCLUDES B
A IS_INCLUDED B A DURING B B REINITIATES A	B INCLUDES A

Table 3.4: Mapping of 19 TimeML temporal and aspectual link types into 4 abstract temporal link types used by Zhang and Xue (2018).

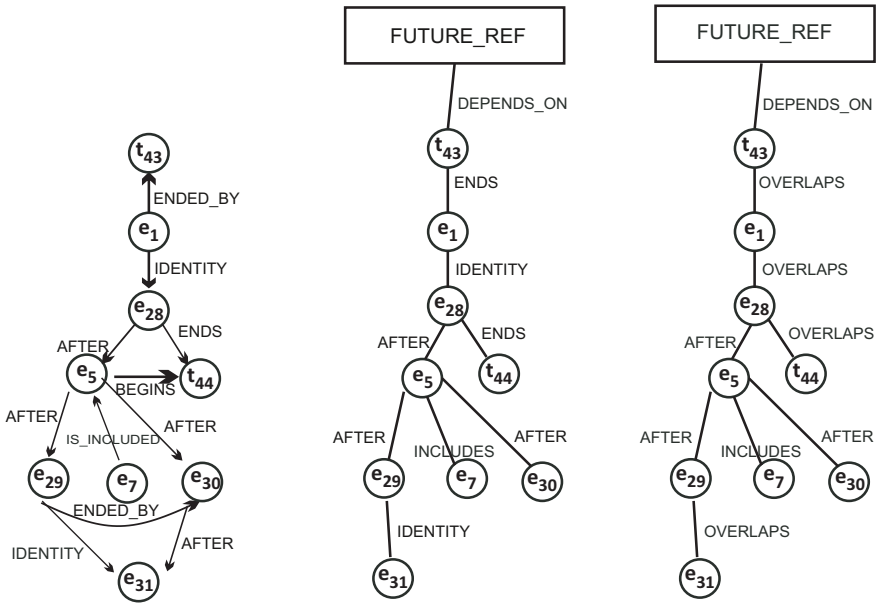

Figure 3.8: Temporal structures for the example snippet, including (a) the full TimeML temporal graph, (b) a temporal dependency tree that uses all link types (full TDT), and (c) a temporal dependency tree that uses only the abstract link types (abstract TDT).

or times that remain unrelated to any other nodes at end of the text are added as children to the ATEMPORAL meta-node (line 21).

When this procedure is applied to the example temporal graph in Figure 3.8(a), it produces the full TDT shown in Figure 3.8(b). As can be seen, 3 of the original 11 links in the TimeML graph are omitted in the TDT. One of my main questions is how much temporal information is lost in this process, which I measure directly in Section 3.6.3.

During the transformation, I iterate events and times in text order. I also experimented with other orders to determine whether the order of iteration mattered. While different specific links were omitted for different orders (e.g., reverse text order, or random), it resulted in the same average loss of temporal information. The main reason for this is that to transform a graph into a tree, the algorithm must ultimately remove one edge from every cycle. After generating full TDTs, I generated abstract TDTs by applying

Algorithm 3 Generating a Full TDT

```
 1: procedure GENERATEFULLTDT(G)
Require: G                                              ▷ TimeML graph
Require: N                                              ▷ FIFO queue
Require: U                                  ▷ set of as-yet unlinked events and times
 2:     N.pushAll(G.V)                      ▷ Add all events and times in text order
 3:     T ← ROOT                            ▷ initialize TDT with the ROOT node
 4:     T.ROOT.addChild(PAST_REF)           ▷ add meta-nodes
 5:     T.ROOT.addChild(PRESENT_REF)
 6:     T.ROOT.addChild(FUTURE_REF)
 7:     T.ROOT.addChild(ATEMPORAL)
 8:     while !N.isEmpty() do
 9:         n ← N.pop()
10:         for all v ∈ T_v do                  ▷ iterate over tree breadth-first
11:             l ← GETLINK(G, n, v)            ▷ identify link, if any
12:             if l = ∅ then
13:                 U.add(n)
14:             else
15:                 v.addChildWithLink(n, l)
16:                 for all u ∈ U do            ▷ check all as-yet unlinked
17:                     l ← GETLINK(G, n, u)
18:                     if l ≠ ∅ then
19:                         n.addChildWithLink(u, l)
20:                         U.remove(u)
21:     ATEMPORAL.addChildren(U)
22:     return T
```

the mappings shown in Table 3.4. Figure 3.8(c) shows the abstract TDT for the example snippet.

3.6.2 Applying TLEX

I can precisely compare the information lost in moving from TimeML graphs to TDTs by converting both the TimeML graphs and TDTs into a uniform representation, namely, timelines. To extract a timeline, I first translate the TimeML or TDT into a temporal constraint graph using primitive temporal relations and then solve the graph.

If the TimeML graphs were inconsistent this meant there was an error in the manual annotation; I corrected these graphs by hand using the original text as a reference. There were 11, 11, and 65 inconsistent texts in the ProppLearner, N2, and TimeBank corpora respectively. Because the TDT Corpus is in Chinese, which I do not speak, I was unable to correct these annotations, so I discarded inconsistent TDT Corpus annotations. Out of 235 TDT Corpus texts, 25 were inconsistent and were discarded.

Once I have integer assignments to time points for the TimeML graph or TDT, I can sort these integers to obtain the corresponding timeline. Because the same integer might be assigned to different time points, the length of timelines can be measured in two ways: the number of *time points*, which is directly proportional to the number of events and times, or the number of *time steps*, which is the number of integers in the solution to the temporal constraint problem. For instance, the timelines extracted from the TimeML graph, full TDT, and abstract TDT for my example snippet are shown in Figure 3.9. The timeline for the TimeML graph has 10 time steps and 18 time points. In this example, the TimeML and TDT timelines both have the same number of time points, but the number of time steps in the TDT case is smaller on account of discarded temporal information. In the general case, because TimeML graphs encode subordinating relations that are completely

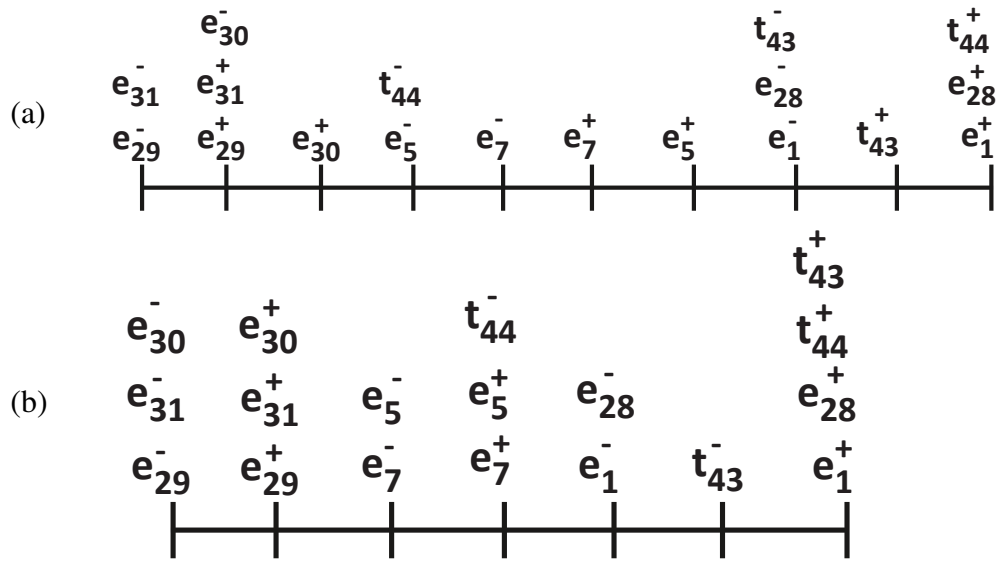

Figure 3.9: Timelines corresponding to the (a) TimeML graph and (b) full and abstract TDT.

disregarded by TDTs, certain events and times might be removed from the TDT timeline altogether, and so the number of time points in TDT timelines can be smaller than in the equivalent TimeML timeline.

Finally, I compare the information loss between timelines by computing the *indeterminacy* of time point orderings relative to the original temporal graph or tree. Temporal graphs or trees often do not have enough information to identify a unique timeline. I call sections of the timeline that have multiple possible solutions *indeterminant*. Similarly, time points or time steps involved in these sections are also called indeterminant.

3.6.3 Information Loss Results

Omitted Temporal Relations

Using Algorithm 3 I transformed the TimeML graphs from the 265 texts (including corrected texts) in the TimeBank, N2, and ProppLearner corpora into full TDTs. The overall

Corpus	# TimeML Links	# Links Omitted	% Omitted
ProppLearner	3,900	24	0.6%
N2 Corpus	3,273	98	3.0%
TimeBank	5,058	166	3.3%
Total or Average	12,231	288	2.4%

Table 3.5: Counts of temporal relations present in the TimeML graphs and omitted in the TDTs.

counts of TimeML relations and omitted links is shown in Table 3.5, and on average 2.4% of temporal relations are omitted. The two reasons for these omissions are (1) tree nodes may only have one parent, and (2) the TDT representation ignores subordinating links. This observation emphasizes that in the general case TDTs cannot represent all of the temporal information in a text.

Increase in Indeterminacy

After extracting full TDTs from the corpora I generated abstract TDTs as described at the end of Section 3.6.1, and extracted timelines from the TimeML graphs, full TDTs, and abstract TDTs. Table 3.6 shows various characteristics of the timelines so extracted, including their average length in terms of both time steps and time points (first and second groups of columns), the total number of time points (third group), and average percentage decrease of TDT timeline lengths relative to TimeML timelines in terms of time steps (last group). In the last group of columns, I see that overall timeline lengths in full and abstract TDTs decrease by anywhere from 3.4% to 14.7% on average.

I applied Algorithm 2 to these timelines to identify indeterminate sections and time points; Table 3.7 shows the results. I can compare the relative indeterminacy of timelines by computing the percentage of time steps that are assigned an indeterminate time point (last group of columns). Transformation of TimeML graphs into full TDTs increases the

Corpus	Avg. Len. of Main Timeline in Time Steps			Avg. Len. of Main Timeline in Time Points			Total # of Time Points / Corpus			Avg. % ↓ in Timeline Len. in Time Steps	
	TimeML Graphs	Full TDTs	Abs. TDTs	TimeML Graphs	Full TDTs	Abs. TDTs	TimeML Graphs	Full TDTs	Abs. TDTs	Full TDTs	Abs. TDTs
TDT Corpus	-	-	67.2	-	-	72.6	-	-	17,081	-	-
ProppLearner	123.5	108.7	108.7	238.7	230.6	230.6	3,580	3,460	3,460	11.9%	14.7%
N2 Corpus	17.7	17.1	17.1	40.2	35.6	35.6	2,694	2,390	2,390	3.4%	6.7%
TimeBank	9.3	8.5	8.5	51.1	48.2	48.2	9,349	8,821	8,821	8.6%	10.8%
						Total	15,623	14,671	14,671		

Table 3.6: Characteristics of the timelines extracted from the corrected corpora. The TDT Corpus is included for comparison only and includes only abstract TDTs, with inconsistent TDTs (25 texts) excluded.

temporal indeterminacy by 76%, 16%, and 22% (average 22%) for the ProppLearner, N2, and TimeBank corpora, respectively. These percentages are computed by dividing the numbers in the second-to-last column of Table 3.7 by those in the third-to-last column. Similarly, transformation of TimeML graphs into abstract TDTs increased indeterminacy by 109%, 51%, and 25% (average 32%). Overall, 11,437 out of 14,671 (78%) time points are indeterminate for abstract TDT timelines and 10,023 out of 14,671 (70%) are indeterminate for full TDT timelines, compared with 8,769 out of 15,623 (56%) for TimeML timelines. Thus, even full TDTs increase temporal indeterminacy significantly compared to TimeML graphs. In contrast to time points, on average 52.2% of time steps in TimeML timelines are indeterminate, compared with 67.2% and 78.1% of time steps in full and abstract TDT timelines. This increase in indeterminacy is potentially important to downstream NLP stages; for example, for a question answering (QA) system that is addressing temporal or causal questions, the text may provide enough information to produce a single answer, but a TDT representation may not include all of that information, making it impossible for the QA system to answer unambiguously.

Corpus	Total # Ind. Time Points / Corpus			Total # of Ind. Sections / Corpus			Average # of Ind. Sections / Text			Avg. % of Time Steps in Timelines that are Ind.		
	TimeML Graphs	Full TDTs	Abs. TDTs	TimeML Graphs	Full TDTs	Abs. TDTs	TimeML Graphs	Full TDTs	Abs. TDTs	TimeML Graphs	Full TDTs	Abs. TDTs
TDT Corpus	-	-	11,444	-	-	10,081	-	-	42.9	-	-	63.7%
ProppLearner	1,066	1,892	2,159	432	669	910	36.8	59.5	67.8	29.8%	54.7%	62.4%
N2 Corpus	1,355	1,568	1,816	683	770	944	8.9	11.2	13.0	50.3%	65.6%	76.0%
TimeBank	6,348	6,563	7,462	970	1,079	1,976	6.3	6.3	7.2	67.9%	74.4%	84.6%
Total	8,769	10,023	11,437	2,085	2,518	3,830		**Weighted Avg.**		**61.3%**	**71.1%**	**81.2%**

Table 3.7: Indeterminacy in timelines extracted from TimeML graphs vs. TDTs. The TDT Corpus is included for comparison only and includes only Abstract TDTs, with inconsistent TDTs (25 texts) excluded. Sections are defined as unbroken sequences of indeterminate time points or steps. The weighted average was computed by weighting with time points.

3.7 Discussion

In this chapter, I have presented both a formal proof of the correctness of TLEX and an experimental evaluation of various features of interest. My investigation has emphasized that manually annotated TimeML texts often contain errors that result in temporal inconsistencies; automatic methods for generating TimeML annotations are currently even more noisy and error-prone. Because of this, a logical next step is to develop methods for automatically suggesting corrections to maximal inconsistent PA subgraphs identified by Step 4. One can imagine both rule-based and supervised machine learning-based approaches to this problem.

I have also presented "jTLEX" which is a programming library that provides a Java implementation of the TLEX algorithm, along with utilities for programmatic manipulation of TimeML graphs. I have released the software as open source with a free license for non-commercial use.

Using TLEX, I have presented an evaluation method to measure information loss in TDTs. My study showed that TDTs result in up to a 109% increase in temporal indeterminacy over their corresponding temporal graphs for the three corpora I examine. On

average, the increase in indeterminacy is 32%, and I show that this increase is a result of the TDT representation eliminating on average only 2.4% of total temporal relations. This result suggests that small differences can have big effects on temporal graphs, and the use of TDTs must be balanced against their deficiencies, with tasks requiring an accurate global temporal ordering potentially calling for use of the full temporal graph.

For future work, I would like my system to automatically suggest additional links to increase graph connectivity, especially in the case of multiple main timelines. In my observations of the corpora under study, one of the most common annotator errors is that a main timeline is accidentally split into two disconnected parts. I imagine that this could be relatively easily detected with high precision, and new connecting links automatically suggested; I provide a method for this in §4.3.2.

Finally, the performance of TLEX on automatically generated TimeML graphs is of interest, and TLEX can be used in this case to measure the quality of the timelines so extracted. This will give a good indication of how useful automatic TimeML extraction ultimately is in practice. I investigate this in Chapter 5.

CHAPTER 4

EVALUATION OF GOLD-STANDARD TIMEML ANNOTATIONS

4.1 Motivation

The input of the TLEX algorithm is TimeML annotations. There are two ways to produce TimeML annotations from raw texts: (1) Annotating raw texts manually or (2) using state-of-the-art automatic TimeML annotators to generate annotations (which I discuss in the next chapter). In this chapter, I discuss the first option, which is annotating raw text manually by following the TimeML annotation guide (Sauri et al., 2006).

The TimeBank corpus was released as a reference corpus for TimeML and provides extensive manually annotated data for training and testing temporal analysis systems that produce TimeML (Pustejovsky et al., 2003b). The TimeBank corpus has been used in much Natural Language Processing (NLP) research, including works on event detection (UzZaman and Allen, 2010; Färber and Rettinger, 2015; Bansal et al., 2018; Veyseh et al., 2021), temporal expression recognition (Kolomiyets and Moens, 2009; Zhong et al., 2017; Chen et al., 2019), and temporal link extraction (Mani et al., 2006; Mirroshandel and Ghassem-Sani, 2011; Kadir et al., 2016; Ning et al., 2018). If there are errors in TimeBank, they could potentially affect the accuracy of all works that use it.

In my prior work, I have used TimeBank as a starting point for developing methods for detecting temporal inconsistency, measuring temporal indeterminacy, and automatically correcting and enriching temporal graphs. In the course of that work, I have identified a number of previously unrecognized issues in the TimeBank corpus as I discussed in Chapter 3, including numerous violations of TimeML annotation guide rules, incorrectly disconnected temporal graphs, as well as inconsistent, redundant, missing, or otherwise incorrect annotations.

In this chapter, I describe these issues and present a suite of methods for detecting errors in the gold-standard annotations and correcting them.

4.2 Methodology

For this experiment, I used the TimeBank corpus (Pustejovsky et al., 2003b). My system contains methods for detecting the errors in the gold-standard annotations and correcting them. First, I implemented a system to check whether TimeBank follows the TimeML annotation guidelines, which can be formulated as strict, syntactic, no-exceptions rules. Second, for each annotated file I checked overall temporal consistency, which revealed additional problems not described in prior work. This includes checking for redundant or inconsistent self-loops in the graphs. Third, I analyzed the connectivity of the TimeML graphs, and this analysis showed that numerous TimeML graphs were improperly disconnected. Finally, I used an automatic TimeML parser (CAEVO) to parse raw TimeBank files and manually compared those automatic annotations with the gold-standard annotations, thereby identifying a number of places where the TimeBank corpus misses events, times, and relations.

4.2.1 Strict TimeML Annotation Rules Checker

The first category of corrections I examine are rules from the TimeML annotation guide that are strictly syntactic with no exceptions. These rules can be checked for compliance automatically, without reference to the semantics of the text. I identified 5 rules this in the annotation guide (Sauri et al., 2006) that satisfy this requirement.

Evidential Link Rule

As described in Section 2.2, EVIDENTIAL links are a type of SLINK, typically introduced by reporting and perception events such as *see*, *say*, *tell*, etc.:

(1) He **said** he **played** basketball.
(said −EVIDENTIAL→ played)

Similarly, NEG_EVIDENTIAL links are typically introduced by reporting and perception events but with negative polarity:

(2) She **denied** that she **went** to Miami.
(denied −NEG_EVIDENTIAL→ went)

According to the TimeML annotation guideline (Sauri et al., 2006, p. 53), perception events will **always** introduce SLINKs of type EVIDENTIAL or NEG_EVIDENTIAL. This rule can be checked automatically.

Conditional Link Rule

As discussed in Section 2.2, a CONDITIONAL link is a type of SLINK that holds between two events and is usually introduced by a signal such as *if... then*:

(3) If she **gets** the medicine, she'll **feel** better.
(gets −CONDITIONAL→ feel)

The TimeML annotation guideline (Sauri et al., 2006, p. 54) specifies that the conditional conjunctions (if-clauses) will **always** introduce CONDITIONAL SLINK. Based on the rule, I can automatically detect whether each if-clause is associated with a CONDITIONAL link.

Causative Event Rule

The TimeML annotation guideline (Sauri et al., 2006, p. 7), specifies that any construction of the structure "*subject* event + *causative* event + *object* event," **must** introduce an IDENTITY TLINK between the subject event and the causative event. For example,

(4) The **rains** **caused** the **flooding**.
(rains $-$IDENTITY$\rightarrow$ caused)

For this rule, I implemented a system that identifies the "*subject* event + *causative* event + *object* event" where the causative event has a head word that is any inflected form of *cause*, then checks whether an IDENTITY TLINK is defined between the *subject* event and the *causative* event.

ALINK Replacement Rule

As described in Section 2.2, an ALINK occurs between an aspectual event and its argument event. The TimeML annotation guidelines (Sauri et al., 2006, p. 58) specify that the `eventInstanceID` attribute of an ALINK must contain the ID of the aspectual event while `relatedToEventInstance` attribute indicates the ID of the argument event. For example,

(5) He **finished**$_1$ **watching**$_2$ The Office.
<ALINK eventInstanceID="1" relatedToEventInstance="2" relType="TERMINATES"/>

For ALINK replacement rules, I implemented a system that goes through every ALINK and checks the `eventInstanceId` to ensure compliance.

ALINK-SLINK Incompatibility Rule

An ALINK occurs between an aspectual event and its argument event while an SLINK occurs between a subordinated event and its related event (§2.2). A TLINK can be introduced along with an ALINK. Similarly, a TLINK can also be introduced with an SLINK. However, by definition an ALINK and an SLINK cannot hold between the same pair of events. For example, in Example (1), *said* and *played* are related by an EVIDENTIAL subordination relationship. They also have an AFTER temporal relationship that indicates *said* happened after *played*. This is an instance where TLINK and SLINK can be present simultaneously. Similarly in Example (5), between *finished* and *watching* there is both a TERMINATES aspectual link and an ENDS temporal link. For this rule, I implemented a system that checks every pair of nodes in the TimeML graph for compliance.

4.2.2 Graph Rules Checker

As discussed previously, a TimeML graph is a graph where the nodes are events and temporal expressions, and edges are TimeML links. TimeML graphs are useful for visualizing TimeML annotations. I evaluated the graphs in TimeBank for temporal inconsistency, graph disconnectivity, and unnecessary self-loops.

Temporal Inconsistency

Although developers of the TimeBank corpus claimed that the corpus is fully consistent, Derczynski and Gaizauskas (2010) found 30 inconsistent annotations. However, they considered only TLINKs when computing consistency. Here I consider both TLINKs and ALINKs when computing overall temporal consistency because ALINKs also indicate temporal relationships.

For this evaluation method, I used TLEX, the method described in Chapter 3. This part of TLEX is a system for producing timelines from TimeML graphs; as one of its substeps, it implements a method for checking graph consistency. TLEX proceeds as follows. First, it partitions the TimeML graph into a set of temporally connected subgraphs (subgraphs of the full graph whose nodes are connected to each other via TLINKs and ALINKs). Second, TLEX transforms temporally connected subgraphs into point algebra (PA) graphs. Then finally, it assigns real numbers for each node in the PA graph. Barták et al. (2014) showed that there is a solution (i.e., assignment of real numbers to time points) that satisfies the PA graph if and only if the graph is consistent. If the real number assignment is not possible, then the graph is inconsistent Barták et al. (2014). Therefore, TLEX tells us whether the graph is consistent.

Graph Disconnectivity

A TimeML graph extracted from a single annotated file might comprise multiple disconnected subgraphs. This is because the annotators sometimes forget to annotate TimeML links between events and temporal expressions. Therefore multiple disconnected subgraphs in a single annotated file suggest the possibility of missing links, which can be manually checked. Disconnectivity can be easily detected by walking the graph. In Figure 4.1, I show the TimeML graph of the TimeBank file `wsj_0991.tml`. When the graphs are compared with the text, one can identify one missing link (t23$-$BEFORE$\rightarrow$t19) that connects the subgraph in the upper left to the subgraph on the right. The subgraph in the lower left is correctly disconnected from the other two subgraphs.

Redundant Self-Loops

A self-loop is a link from a node to itself and they are always incorrect in the TimeML scheme. There are two types of self-loops: inconsistent self-loops and redundant self-

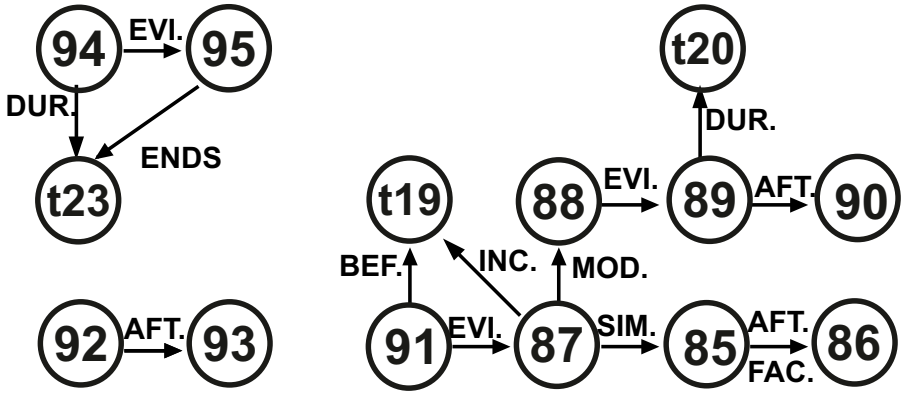

Figure 4.1: Visualization of a disconnected graph. The numbers indicate the event instance IDs or time IDs (start with *t*). The file contains three disconnected graphs, therefore, potentially two links are missing.

loops. Any ALINK or TLINK (except SIMULTANEOUS or IDENTITY) from a node to itself is an inconsistent self-loop and will be detected when evaluating the graph for inconsistency as described in Section 4.2.2. On the other hand, a redundant self-loop is a SIMULTANEOUS or IDENTITY TLINK self-loop. While they do not cause inconsistency, they also do not provide any useful information. Therefore, they are redundant.

4.2.3 Missing Annotations Detector

My final check involved comparing the gold-standard TimeBank annotations with automatically generated annotations in an attempt to find missing annotations. As described in Section 2.3, CAEVO is an automatic TimeML parser that can take raw text as an input and generate TimeML annotations. Because it has the best score on detecting events and temporal expressions, I used CAEVO to generate annotations for comparison with the TimeBank corpus.

I processed raw TimeBank texts with CAEVO, using it to detect events and temporal expressions. I then manually compared the output with the gold-standard annotations,

judging whether events and times detected by CAEVO, but not in TimeBank, were correct. Because CAEVO is not perfect, not all events and times detected by CAEVO are correct; indeed, approximately half of the events detected by CAEVO but not in TimeBank were generic events, which should be excluded according to the TimeML annotation guide (Sauri et al., 2006, p. 7). In the following example, *driving* should not be tagged as an event because it's a generic event, however, CAEVO will tag it as an event.

(6) **Driving** under influence is illegal.

For temporal expressions, CAEVO cannot correctly filter some proper names, and so labels them incorrectly. In the following example, SEC represents the US Security and Exchange Commission. However, CAEVO detects SEC as a second.

(7) Through his lawyers , Mr. Antar has denied allegations in **the SEC** suit.

For the manual comparison process, we[1] did a double annotation. First, we compared the events and times separately, which allowed us to calculate the inter-annotator agreement score as 0.67. The inter-annotator agreement score is relatively low because of disagreements over whether particular events were generic, which is often a subtle judgement. Then, we conferred on the annotations, examining disagreements one by one and comparing them to the text. When we discovered a missing event or temporal expression to be added, we also added any missing links to connect the event or temporal expression to the rest of the graph. We didn't compare the CAEVO-annotated links with the TimeBank links. This is because (a) CAEVO has only a 0.51 F1 score on extracting TLINKs, (b) it can only generate 5 types of TLINKs and does not generate ALINKs or SLINKs, and (c) it uses transitive closure to generate links, which creates noisy graphs with extra links.

[1] Jared Hummer helped me with the manual comparison process.

4.3 Results & Corrections

4.3.1 Results of Evaluation for TimeML Annotation Rules

I built a system that checks the strict TimeML annotation rules defined in Section 4.2.1. Table 4.1 lists the results for each of the five rules. As can be seen in the table, there are 109 instances that violate the TimeML annotation guidelines rules.

Evidential Link Rule My system detected 48 perception events in the corpus. Only 18 of them are involved in an EVIDENTIAL link. For the rest of the perception events, 15 of them have different types of SLINKs and the other 15 of them have no SLINKs at all. I also noted that, out of 18 perception events with EVIDENTIAL links, two have negative polarity, meaning, they should introduce NEG_EVIDENTIAL instead of EVIDENTIAL. An example from `ea980120.1830.0071.tml` that violates the evidential link rule is shown below. The perception event is underlined and in bold.

(8) Cubans want to know what we're going to tell Americans, in many cases, what their relatives in the United States are going to **hear**. (No EVIDENTIAL SLINK)

For the missing annotations where the evidential link rule applies, I define the necessary EVIDENTIAL or NEG_EVIDENTIAL links based on polarity with the new linkID. And if an SLINK was incorrect, I corrected it with a proper SLINK. For the example, I define an EVIDENTIAL link between *hear* and *tell*.

Conditional Link Rule My system detected 64 conditional conjunctions and 45 of them have CONDITIONAL links. The rest of the 19 conditional statements have no SLINKs at all. An example that violates this rule from `wsj_0586.tml` is as follows. I added any missing CONDITIONAL links.

(9) If you take away the outside influences, the market itself looks very cheap. (No CONDITIONAL SLINK)

Causative Event Rule My system detected 14 "*subject* event + *causative* event + *object* event" cases and only four of them correctly introduced IDENTITY TLINK between the subject event and the causative event. One of the 10 cases that violate the rule (from `ea980120.1830.0071.tml`) is shown below. For each of the 10 cases, I added the missing IDENTITY link between the subject event and the causative event. For the example, I define IDENTITY TLINK between *concern* and *caused*.

(10) Well, this is the eve of the Pope's **visit** to one of the last bastions of Communism anywhere in the world, and it is already **causing** enormous **expectations**. (No IDENTITY TLINK)

ALINK **Replacement Rule** Out of 265 ALINKs, my system took the first eventID and traced it back to the annotations to see if the event is an aspectual event. The results showed that 46 of them are not aspectual events, meaning 46 ALINKs violate the rule. In the following example from `WSJ900813-0157.tml`, annotators put the ALINK in the reverse order.

(11) Iraq's Saddam Hussein, his options for **ending** the Persian Gulf **crisis** growing increasingly unpleasant. (crisis −TERMINATES-> ending)

ALINK-SLINK **Incompatibility Rule** I checked whether there are any cases that violate the rule. My system found four violations where there was a TLINK, SLINK, and ALINK between the same two nodes. All four cases involved the verbs *launched*, *offer*, *suit*, or *bid*, in a BEGINS, FACTIVE, or INITIATES relationship. For the four examples, I removed the SLINKs to follow the rule.

(12) Acquisition has **launched** a **suit** in a Delaware court.

(13) Dow Jones **launched** the **offer** on Sept. 26.

(14) A unit of DPC Acquisition Partners **launched** a $10-a-share tender **offer** for the shares.

(15) Before the **bid** was **launched**, he sought approval to boost his Paribas stake above 10%.

Rules	**Total Instances**	**# of Violations**
EVIDENTIAL **Link**	48	30
CONDITIONAL **Link**	64	19
Causative Event	14	10
ALINK **Replacement**	265	46
ALINK-SLINK **Incompatibility**	-	4
Total		109

Table 4.1: Results of Evaluation for TimeML Annotation Rules. The number of instances that match the rule are given in **Total Instances**, while the number of those matches that violate the rule are given in **# of Violations**.

4.3.2 Results of Evaluation for Graph Rules

I checked the TimeBank graphs against the graph rules that are defined in Section 4.2.2, and the results are summarized in Table 4.2.

Temporal Inconsistency I showed that 65 texts have inconsistent graphs, roughly 1/3 of the TimeBank corpus. As can be seen in Table 4.2, 30 inconsistent files were caused by TLINKs, which matches the results reported by (Derczynski and Gaizauskas, 2010). However, a slightly greater number of inconsistencies were caused by ALINKs (35 texts), which were not investigated previously. In total, I detected 110 inconsistent subgraphs

across 65 texts. Additionally, I showed that 15 of those inconsistencies were caused by inconsistent self-loops, which can be easily fixed by simple removal of the self-loops. Other inconsistencies required a more careful comparison of the annotations with the text. An example of an inconsistency from `wsj_1011.tml` is as follows.

(16) [**DCT:10/26/1989**$_{t57}$] The latest **results**$_{ei2048}$ include a $2.6 million one-time payment from a "foreign entity."

t57 −BEFORE−> ei2048

ei2048 −BEFORE−> t57

Graph Disconnectivity My system showed that only 35 texts have a single fully connected graph. The remainder of the files contained anywhere from 2 to 34 disconnected subgraphs, suggesting the same number of potentially missing links, for an overall total of up to 739 missing links. Upon manual inspection, I found only 625 missing links, with the remaining 108 subgraphs being correctly disconnected from the other graphs in the file because of the lack of temporal information in the text. Note that Derczynski and Gaizauskas (2010) reported that all files were disconnected, however, they did not consider ALINKs and SLINKs in their system. I also found 65 singleton nodes (all temporal expressions) that had no incoming or outgoing links, all of which should have been connected to some other node in the file.

To resolve some disconnections, I implemented a system that automatically suggests links between temporal expressions in disconnected subgraphs. To do that, my system selects a temporal expression from the main graph (timex-1) and from the subgraph (timex-2), then, it normalizes their value. Finally, it creates a TLINK based on their value such as timex-1 −BEFORE→ timex-2. This allows me in many cases to achieve connectivity automatically and correctly.

Rules	# of Files	# of Violations
Inconsistency (TLINKs)	30	38
Inconsistency (all)	65	110
Disconnectivity	148	625
Redundant Self-loops	9	10
Total		783

Table 4.2: Results of checking the Graph Rules. The number of files that contained a violation of the rule is given in **# of Files**, while the number individual violations across all files is given in **# of Violations**.

Redundant Self-loops My system detected 10 redundant self-loops. Redundant self-loops can be removed automatically.

4.3.3 Results of Evaluation for Missing Annotations

I processed the raw TimeBank texts with CAEVO, which detected 947 events not present in the gold-standard TimeBank annotations. Then, I performed the double annotation process (described in Section 4.2.3) and I identified 317 missing events in the TimeBank corpus. The reason for the significant difference between the number of detected events and those determined to be correct is that (a) CAEVO is not perfect (0.81 F_1 score in event detection) and (b) CAEVO is especially weak in the detection of generic events, which are explicitly excluded by the TimeML scheme.

For temporal expressions, CAEVO detected 93 different temporal expressions not present in TimeBank. After the manual annotation process, I determined 52 of these were correct. The main reason for 41 false positive temporal expressions is that CAEVO detected some proper names as temporal expressions such as "the **SEC**," "USA **Today**," etc.

Each missing event or temporal expression also potentially implied missing links to connect the node to the main graph. I added the missing links based on my understanding

of the text. For all corrections, I strictly followed the TimeML annotation guidelines rules. The total number of each type of correction is shown in Table 4.3.

Because TimeBank 1.2 is distributed under license by the Linguistic Data Consortium (LDC), I do not directly provide the corrected corpus. Instead, I provide patch files that can be applied to the original TimeBank 1.2 using the standard Linux or Unix `diff` command.

Rules	**Events**	**Times**	TLINK	ALINK	SLINK	**Total**
TimeML Rules	-	-	10	46	53	109
Graph Rules	-	-	722	61	-	783
Missed Annotations	317	52	369	-	-	738
Total Corrections	317	52	1,101	107	53	1,630
Total # Objects in TimeBank 1.2	7,935	1,414	6,418	265	2,932	18,964
% Total Corrected	3.9%	3.7%	17.2%	40.8%	1.9%	8.6%
Total # Objects in Corrected TimeBank	8,252	1,466	7,351	271	2,982	20,322

Table 4.3: Summary of corrections to the TimeBank 1.2 corpus, split into categories.

4.4 Discussion

In this chapter, I presented a semi-automatic evaluation of the TimeBank 1.2 corpus. In particular, I show that TimeBank has 109 instances that violate the strict TimeML annotation guidelines rules. I also show that roughly 1/3 of the TimeBank annotated files result in an inconsistent cycle in the extracted TimeML graphs, which includes 15 inconsistent self-loops. Additionally, I detected on average 8.1 disconnected graphs per file, which suggested potentially up to 4 missing temporal relations per file. I also discovered 10 redundant self-loops (a loop that doesn't cause inconsistency but also doesn't provide additional information, and is not signaled in the text). After I manually compared

the gold-standard annotations and CAEVO-based annotations, I identified that the gold-standard annotations had 317 missing events, 52 missing temporal expressions, and 369 additional missing TimeML links.

The detected errors in the TimeBank corpus are the result of either incorrect or missing annotations. As I have shown through extensive experimentation elsewhere, errors such as those described above can have dramatic effects on the quality of the final graphs and downstream tasks (Ocal et al., 2022a). We[2] corrected the errors by adding or changing the necessary event, temporal expression, or TimeML link. The correction process was a double annotation process. First, we took the CAEVO-generated patch file and annotated it separately. Then, together we went through each annotation one by one. In total, I corrected 317 events, 52 temporal expressions, and 1,265 TimeML links, which correspond to 4%, 4%, and 13% of the totals. I released both my analysis code and diff files that allow other researchers in the field to apply my corrections to their own copies of TimeBank.

I showed that TimeBank has 1,630 incorrect or missing annotations, for which I provide corrections. These misannotations are mainly caused by manual annotation mistakes. There are many other gold-standard TimeML corpora that might suffer from the manual annotation process. The complexity of the schema suggests that any gold-standard TimeML annotations should, as a best practice, use such an evaluation approach to ensure the highest quality annotations.

[2]Jared Hummer helped me with the correction process.

CHAPTER 5

EVALUATION OF AUTOMATIC TIMEML ANNOTATION TOOLS

5.1 Motivation

In Chapter 4, I discussed the human-made errors during the manual TimeML annotation process. The other way to produce TimeML annotations is using state-of-the-art automatic TimeML annotators to generate annotations.

Automatic TimeML annotation is a challenging task, not only because it comprises three subtasks (time expression detection, event detection, and TimeML relation extraction), but also because temporal information is often implicit in the commonsense meaning of the language. There are several systems that can automatically generate TimeML graphs by combining the output of components that each focus on one of the individual subtasks. When evaluated in isolation, the components perform well on event detection (0.80–0.85 F_1 score) and time expression detection (0.82–0.93 F_1 score), while automatic relation extraction trails both in performance (0.50–0.63 F_1) and coverage (4–11 out of 25 relation types extracted). Despite these respectable individual performances, the question remains of how the performance of the individual components relates to overall performance, and how that affects a logical next task, namely, timeline extraction. Limited coverage of relation types is especially problematic because while a reported F_1 for relation extraction might be respectable, even good, all extant systems only extract a subset of relation types, and how these omissions affect later tasks has not been evaluated. Furthermore, running each component in isolation and then combining the results does not attend to the question of consistency, which is critical to the utility of the final TimeML graphs.

In this chapter, I present a new suite of methods for evaluating automatically generated annotation based TimeML graphs (automatic graphs) and, consequently, measuring

the *holistic* performance of existing TimeML annotation suites. My evaluation methods include four graph-based metrics (relation distribution, number of closure links, edit-distance from the gold-standard graph, and the overall consistency of the graph) and four timeline-based metrics (timeline length, missing times and events, subordination structure, and temporal indeterminacy). I also combine five of these metrics with a novel graph-transformation experimental design that allows me to investigate how the metrics vary as TimeML graphs degrade.

5.2 Metrics

I present eight different metrics: four graph-based metrics, where I assess (1) relation distribution, (2) closure links, (3) edit-distance from the gold standard, and (4) graph consistency; and four timeline-based metrics, where I assess (5) timeline length, (6) missing time points, (7) subordination structure, and (8) temporal indeterminacy.

To illustrate some metrics, I will use the text shown in Example (1), which is a snippet of the TimeML-annotated text from the TimeBank corpus. The TimeML graph corresponding to the snippet is shown in Figure 5.1, where nodes of the graph are either events or times, and the edges are TimeML relations (DCT = *Document Creation Time*).

(1) [**DCT:11/02/89**$_1$]: Pacific First Financial Corp. **said**$_2$ shareholders **approved**$_3$ its **acquisition**$_4$ by Royal Trustco Ltd. of Toronto for $27 a share, or $212 million. The thrift holding company **said**$_5$ it **expects**$_6$ to **obtain**$_7$ regulatory **approval**$_8$ and **complete**$_9$ the **transaction**$_{10}$ by **year-end**$_{11}$. [from `WSJ.0006.tml`]

Running the TLEX algorithm, I can obtain the timeline that corresponds to the TimeML graph and is shown in Figure 5.2.

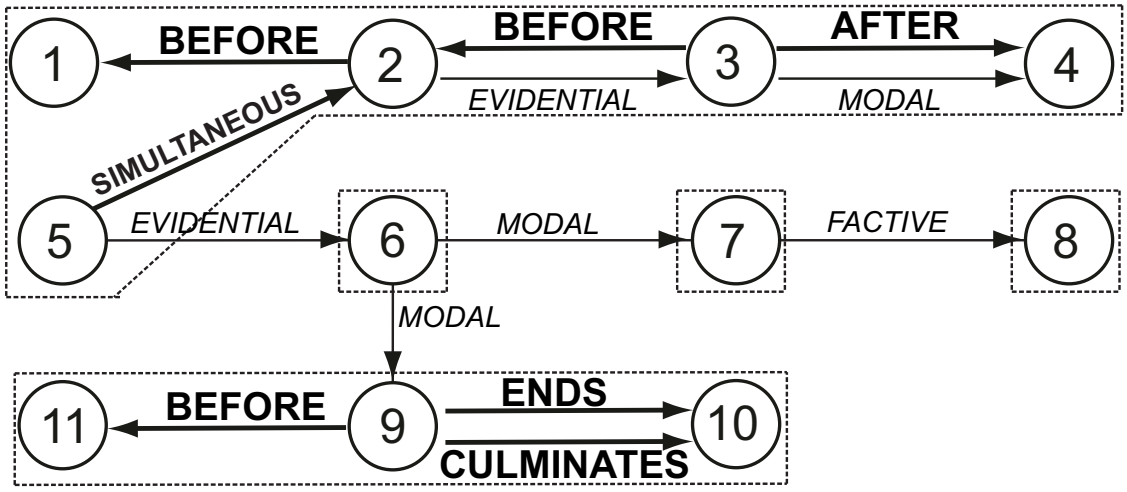

Figure 5.1: Visualization of the gold-standard TimeBank TimeML graph for Example (1). TLINKs and ALINKs are in bold, and SLINKs are in italic. Numbers correspond to events and times numbered in the example. The five temporally and aspectually connected subgraphs are separated by dashed lines.

5.2.1 Metric 1: Relation Distribution

This metric computes the difference between the number of TLINK, ALINK, and SLINK relations in the automatic graph and the gold-standard graph, expressed as a fraction of the number of links in the gold-standard graph. A positive fraction means there are more relations in the target (automatic graph); negative means fewer. This metric can be broken into 28 sub-metrics: 3 measuring TLINK, ALINK, and SLINK categories overall, and 25 metrics each corresponding to each individual relation type (BEFORE, AFTER, etc.).

5.2.2 Metric 2: Closure Links

This metric is the number of closure links that can be generated from a graph. In temporal algebra, transitive closure is a relationship between three time points x, y, and z where the temporal link from x to z is inferable from the temporal links x to y and y to z. For example, if x is AFTER y and y is AFTER z, I can infer x is AFTER z. Unlike

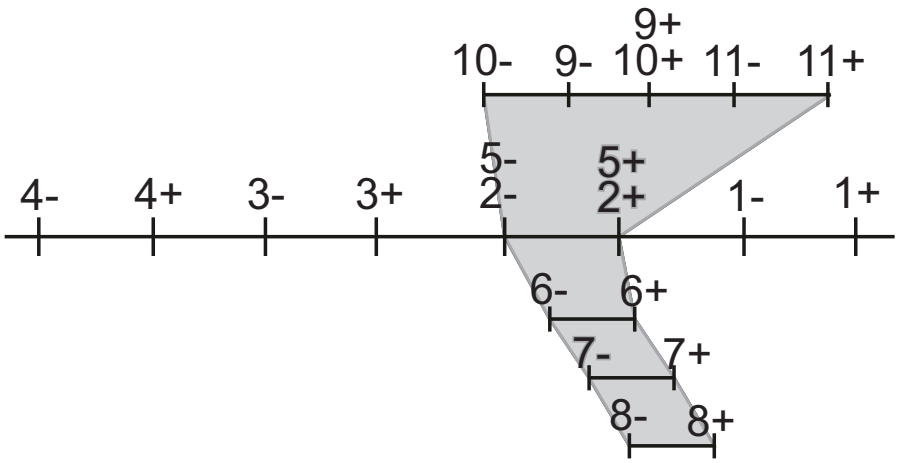

Figure 5.2: Visualization of the timeline of the gold-standard TimeML graph. Grey areas indicate subordinated timelines.

gold-standard annotations, automatic TimeML annotators use transitive closure to produce more TLINKs. If the existing two links are incorrect, transitive closure can produce another incorrect link. While the existing links are correct, transitive closure generates a link that does not provide any additional information to global order, overall consistency, or temporal indeterminacy. I exclude links generated by transitive closure when comparing the automatically generated graphs with the gold-standard graphs because the gold-standard graphs do not have transitive closure links.

To compute this metric, I followed Chambers et al. (2014)'s transitive closure rules and implemented a system that counts the extra links (which also allows me to eliminate them during other comparisons). Figure 5.4 illustrates the case where there are two extra links that are generated by transitive closure: $2-\text{AFTER}\rightarrow 4$ (because 2 is AFTER 3 and 3 is AFTER 4) and $5-\text{BEFORE}\rightarrow 7$ (because 5 is IS_INCLUDED 6 and 6 is BEFORE 7).

5.2.3 Metric 3: Edit Distance

The third metric captures the edit distance of the target from the gold-standard graph, represented as a fraction: the number of graph edits required to transform a target graph into the gold-standard graph divided by the number of nodes and edges in the gold standard. This metric will always be positive. To compute the raw edit distance, I compute the maximum common subgraph between the target and the gold-standard graphs (Bunke and Shearer, 1998). Next, I count both (a) the number of relations and nodes present in the target, but not in the gold standard, and (b) the number of relations and nodes present in the gold standard, but not in the target. The edit distance is the sum of these.

To illustrate this, compare the example gold-standard graph (Figure 5.1) with the automatic graph computed using CAEVO (Figure 5.4). The maximum common subgraph between these two graphs comprises Nodes 2–10 and $\text{AFTER}_{3\rightarrow4}$. There are seven relations (all other relations) in the target that are not present in the gold standard, however, two of them are extra links and they are eliminated by the previous process, therefore, five remain. There are two nodes and thirteen relations that are present in the gold standard but not in the target. This means that the edit distance is 20.

5.2.4 Metric 4: Graph Consistency

Barták et al. (2014) showed that a timeline can be extracted if and only if the temporal graph is inconsistent. If the timeline cannot be extracted, that means the graph is consistent. Zaidi (1999) showed that temporal consistency is caused by a cycle of relations that indicates no assignment is possible for the graph.

Using the algorithm provided in §3.2.4, I calculated the number of each automatic TimeML annotator's files that were inconsistent as well as the total number of inconsistent cycles of relations that automatic annotations have.

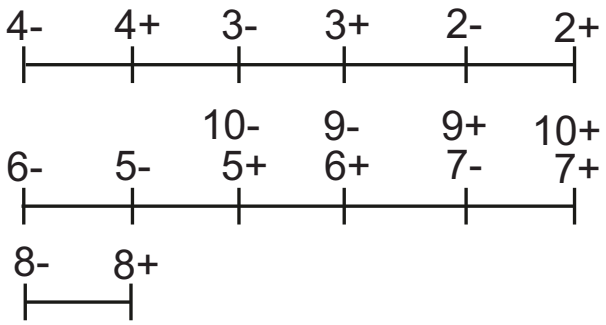

Figure 5.3: Visualization of the timeline of the CAEVO annotated TimeML graph. There are three different timelines for three temporally connected subgraphs.

5.2.5 Metric 5: Timeline length

This metric computes the difference in length between the longest timeline of a target TimeML graph and that of the gold standard. For example, the longest timeline of the gold-standard graph of the example text (Figure 5.2) has a length of eight while the longest timeline of the CAEVO annotated graph (Figure 5.3) has six, which means the target timeline is two units shorter than the gold-standard timeline.

5.2.6 Metric 6: Missing Timepoints

This metric computes the difference between the number of unique timepoints across all timelines of a target graph and that of the gold standard. For example, according to the timeline of the gold-standard graph of the example text (Figure 5.2), the timeline of the CAEVO annotated graph (Figure 5.3) does not contain the time points of $1-$, $1+$, $11-$, and $11+$. And according to Metric 6, the timeline of the CAEVO annotated graph misses four time points.

5.2.7 Metric 7: Subordination Structure

This metric computes the number of subordinated timelines, which can be compared with the number of subordinated timelines in the gold standard.

5.2.8 Metric 8: Temporal Indeterminacy

As described earlier, I provide an algorithm for computing temporal indeterminacy given all possible solutions to a temporal graph (§3.2.5). This relies on the ability of most constraint solvers to produce both the smallest solution as well as all possible solutions for a constraint graph. The algorithm determines whether the order of a pair of adjacent timepoints is the same in all possible timelines as in the smallest timeline. If they are not always adjacent or in the same order, this pair is marked as indeterminate time points, allowing visualization of indeterminate sections of the timeline as shown in Figure 3.7 for the temporal graph shown in Figure 3.5.

5.3 Experimental Results

5.3.1 Generating TimeML Graphs

The TimeBank corpus (Pustejovsky et al., 2003b) has 183 texts in three formats: raw text, pre-annotated texts, and gold-standard annotated texts. Gold-standard annotated texts are files where each TimeML component (events, times, links, etc.) is labeled with TimeML tags while pre-annotated texts have only TimeML tags for events and times. Later work showed that 30 texts had errors and provided corrections (Derczynski and Gaizauskas, 2010); I used the corrected versions.

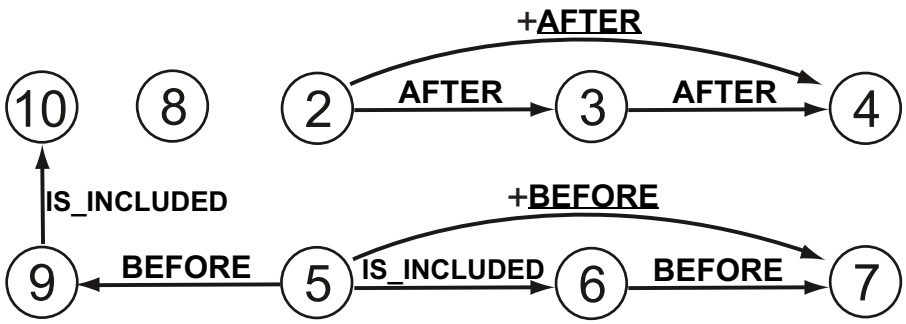

Figure 5.4: The TimeML graph generated by CAEVO when run on Example (1). Numbers correspond to events and times numbered in the example. Underlined TLINKs indicate closure links.

To generate baseline TimeML graphs for evaluation, I used four annotators. I applied CAEVO, CLEARTK, and TARSQI to the raw TimeBank texts to generate TimeML-annotated texts. Only TARSQI has the capability of producing SLINKs and ALINKs between events. Because CATENA only generates TLINKs (it does not detect events and times) I gave CATENA pre-annotated texts as input. Similar to CAEVO and CLEARTK, CATENA also can't generate SLINKs or ALINKs between events. I provided an illustration of the TimeML graph that is extracted from the CAEVO annotation of our example text (Example (1)) in Figure 5.4.

Comparing Figures 5.1 and 5.4, several important differences are evident. First, the gold-standard graph is a single connected graph while the generated graph comprises three separate graphs. Second, the generated graph contains no SLINKs or ALINKs because CAEVO, like nearly all automatic TimeML annotators, does not generate those link types. Third, the generated graph has more TLINKs than the gold-standard graph, because automatic annotators use transitive closure to generate links consistent with automatically detected links. For example, in Figure 5.4, event 2 is BEFORE event 3, which is BEFORE event 4; consequently, CAEVO adds a BEFORE link between events 2 and 4.

5.3.2 Results of Graph-Based Metrics

First, I analyzed how many types of TLINK automatic TimeML graphs have and how many total TLINKs they contain (shown in Table 5.1). Second, I calculated the closure links and set them aside for comparison. Then, as it's described in Section 5.2.3, I calculated the average distance between automatic graphs and gold-standard graphs. The results are shown in Table 5.2.

Metric 1: Relation Distribution The distribution of TLINK types in the graphs is shown in Table 5.1. I note that automatic graphs have many more BEFORE and AFTER links than the gold-standard graphs. While only 35.9% of gold-standard links are BEFORE and AFTER, this number goes up to 76.9% in CAEVO graphs, 78.4% in TARSQI graphs, 69.8% in CATENA graphs, and 76.8% in CLEARTK graphs. This means the automatic annotators tend to generate BEFORE and AFTER TLINKs more than any other TLINK type, meaning they miss a large portion of TLINK types.

I note that TimeML graphs of automatic annotations don't have every TLINK type. The state-of-the-art TLINK detection system, CATENA, produces 10 types of TLINKs while CLEARTK only produces four types of TLINKs. This means they're missing a significant amount of temporal relations. I also note that although automatic systems produce fewer types of TLINKs, they produce more total TLINKs than gold-standard annotation. While the gold-standard graphs have only 6,418 TLINKs, CLEARTK and CAEVO graphs have more than 9,000 TLINKs. One reason is that is automatic systems use transitive closure when they produce TLINKs. As I mentioned earlier, while automatic TimeML annotators use transitive closure as a feature in their systems, gold-standard annotations do not necessarily include closure links. This results in automatic annotations having more TLINKs than gold-standard annotations. However, even without closure, automatic annotators generate many more TLINKs than gold-standard annotations. For example, CATENA an-

Annotations	BEFORE	AFTER	I_BEFORE	I_AFTER	SIMULTANEOUS	IDENTITY	INCLUDES	IS_INCLUDED	BEGINS	BEGUN_BY	ENDS	ENDED_BY	DURING	DURING_INV
GS	**21.9%**	**14.0%**	**0.5%**	**0.6%**	**10.5%**	**11.6%**	**9.1%**	**21.1%**	**1%**	**1.1%**	**1.2%**	**2.8%**	**4.7%**	**0.02%**
CAEVO	43.8%	33.1%	-	-	1.9%	-	2.3%	18.8%	-	-	-	-	-	-
TARSQI	47.6%	30.8%	-	-	0.8%	9.2%	3%	8.4%	-	0.1%	-	0.03%	-	-
CATENA	44.5%	25.3%	-	-	6.6%	-	3.9%	18.6%	0.2%	0.2%	0.2%	0.3%	0.2%	-
CLEARTK	53.8%	23%	-	-	-	-	2.8%	20.4%	-	-	-	-	-	-

Table 5.1: Metric 1: Relation Distribution (TLINKs only). GS means gold standard

Annotations	# of TLINK types	Metric 1: Total TLINKs	Total ALINKs	Total SLINKs	Metric 2: Closure Links	Metric 3: Avg. Edit Distance to GS Graphs	Metric 4: # of Inconsistent Main Graphs
Gold-Standard	**14**	**6,418**	**265**	**2,932**	**-**	**-**	**-**
CAEVO	5	9,062	-	-	876	125.26	72
TARSQI	8	6,366	93	1,325	268	77.54	9
CATENA	10	16,875	-	-	1,085	138.47	159
CLEARTK	4	9,015	-	-	67	94.49	34

Table 5.2: Metrics 1–4: Summary Counts, Closure Links, Edit Distances, and Graph Consistency.

notated graphs have two times more TLINKs than gold-standard graphs do. This suggests automatic annotators generate many incorrect TLINKs.

Metric 2: Closure Links I calculated the extra links implied by temporal closure. As shown in Table 5.2, CAEVO generated 876 extra links, 9.7% of its total links. In the meantime, TARSQI generated 268 extra links (4.2%), CATENA generated 1085 (6.4%), and CLEARTK generated only 67 (0.7%).

Metric 3: Edit Distance I calculated the edit distance between gold-standard graphs and automatic graphs. Table 5.2 shows that on average, each automatic graph requires more than 77 steps to generate gold-standard graphs. Although CATENA is the state-of-the-art for TLINK identification, CATENA graphs are the most distant graphs among all automatic graphs. This is because CATENA produces a large number of TLINKs, most of which are incorrect.

Metric 4: Graph Consistency In some cases, I cannot extract timelines from target graphs because they are temporally inconsistent. Inconsistency in the original gold-standard TimeBank annotations is the result of annotators' mistakes, and for the timeline metrics later I use corrected TimeBank gold-standard files where each inconsistency is manually fixed by Ocal and Finlayson (2020). For this metric, I measured how many inconsistent graphs exist for the 183 annotated texts. As mentioned previously, there were 65 inconsistent annotated files in the original gold-standard TimeBank corpus; I use the corrected versions for comparison in my experiments. Although CATENA has the highest F_1 score for TLINK detection, CATENA annotations have 159 inconsistent annotated files out of 183. TARSQI annotations introduce less inconsistency than the gold-standard annotations.

5.3.3 Results of Timeline-Based Metrics

Metric 5: Timeline Length Some automatic annotators compute shorter timelines than gold-standard timelines while others output longer timelines. I report Root Mean Square Error (RMSE) measures for timeline length over the corpus for each system, and it can be clearly seen that RMSE scores for automatic annotations are high. Timelines of CATENA annotations are very different from those other annotations. The main reason some automatic annotation timelines are longer than gold-standard annotation timelines is that some automatic annotations produce a high number of BEFORE and AFTER relations (See §5.3.2).

Metric 6: Missing Timepoints My results show that although automatically generated timelines have a similar number of time points to gold-standard timelines, the RMSE is high. This is because state-of-the-art systems detect incorrect events and times and miss a

Annotations 183 files	Metric 5: Avg. Main. Timeline Len. (RMSE)	Metric 6: Avg. Time Pts / Timeline (RMSE)	Metric 7: Avg. Sub. Branches (RMSE)	Metric 8: % Indeterminacy
Gold-Standard	**8.61**	**130.79**	**16.02**	**67.9%**
CAEVO	8.32 (3.10)	138.41 (10.40)	0 (N/A)	92.2%
TARSQI	6.49 (3.85)	100.39 (37.24)	7.28 (17.09)	80.7%
CATENA	12.85 (7.56)	-*	0 (N/A)	66.7%
CLEARTK	9.62 (3.04)	94.55 (14.96)	0 (N/A)	91%

Table 5.3: Metrics 5–8: Timeline Length, Missing Timepoints, Subordination Structure, and Temporal Indeterminacy (* excluded because it uses the gold-standard events and times).

large number of correct ones. Because CATENA uses pre-annotated texts, I didn't include it in the table.

Metric 7: Subordination Structure SLINKs are critical for timeline extraction: without them I will include events that may not happen in the real world in the main timeline. Among state-of-the-art TimeML annotators, only TARSQI detects SLINKs between events. Based on TARSQI's RMSE score for the number of subordinated branches, my result shows that TARSQI misses a significant number of SLINKs.

Metric 8: Temporal Indeterminacy I measured a 67.9% overall indeterminacy score on gold-standard annotations. The reason the indeterminacy score is already high for gold-standard annotation is that natural language texts usually don't specify the order of some events. However, as can be seen in Table 5.3, the automatic annotators increased indeterminacy significantly. CAEVO annotations have a 92.2% indeterminacy score, while CLEARTK annotations have 91% indeterminacy.

5.3.4 Effect of TimeML Graph Degradation

I also investigated how errors in the detection of events, times, and relations affect the quality of resultant timelines. To do this I took advantage of the ability to transform one

graph into another by incrementally removing or adding nodes and links (the total number of steps between a gold-standard graph and an automatically generated graph is captured by the edit distance metric). To measure this, I began with the gold-standard graph and transformed it into the automatically generated graph step-by-step in a random order, at each step either (i) removing a link or node that is not present in the automatic graph, or (ii) adding a link or node that is not present in the gold-standard graph. After each editing step, I ran Metrics 4–8 on the new graph, which allowed me to chart the change in those metrics as the graphs degrade. I can average these measures over multiple random sequences of edits.

I started with the 183 gold-standard graphs and transformed them into the 734 automatically generated graphs (734 = 183*4, one generated graph for each of the four systems). I show the results for Metric 4 (Inconsistency) and Metric 8 (Indeterminacy) in Figure 5.5 and 5.6, respectively. As can be seen in Figure 5.5, for CAEVO, CATENA, and CLEARTK, the number of inconsistent graphs increases non-linearly: the number of inconsistencies increases relatively slowly and linearly until a breakpoint is reached (roughly 35–55% changed), after which the number of inconsistent graphs increases rapidly. TARSQI is an outlier to this pattern, I suspect because TARSQI imposes consistency checks in its pipeline; ultimately TARSQI only generates 9 inconsistent graphs.

In contrast, it can be seen that the increase in indeterminacy follows a smoother, more linear pattern for CAEVO, TARSQI, and CLEARTK. CATENA is the outlier here, because CATENA generates an extremely large number of links (see Table 5.2, CATENA's Metric 1, which is 16,875, more than twice of those the other systems), and so when gradually adding in these links indeterminacy temporarily surges, before falling back to the ultimate level found in the automatically generated graphs.

Note that although this graph transformation technique does work for Metrics 5, 6, and 7, I do not show those results for several reasons. First, Metric 5 (average main time-

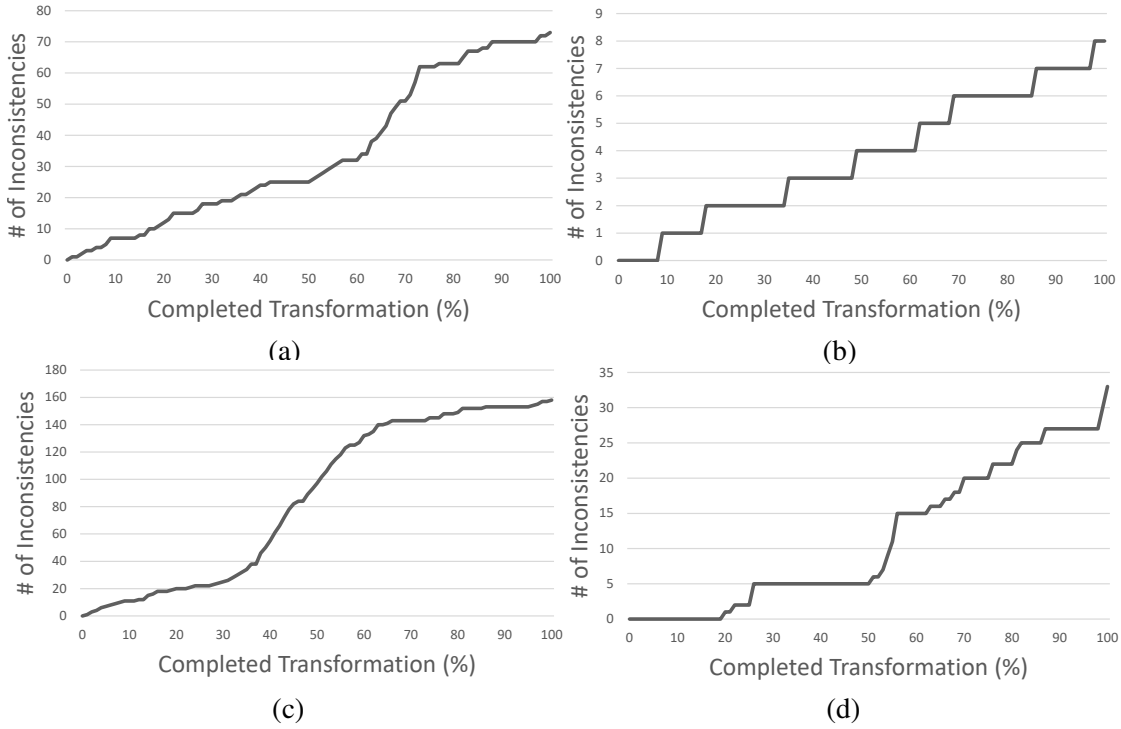

Figure 5.5: Increase in inconsistency during gradual stepwise transformation of gold-standard graphs (left side of each chart) to automatically generated graphs (right side). Charts represent the number of inconsistent graphs for (a) CAEVO, (b) TARSQI, (c) CATENA, and (d) CLEARTK. Each X-axis indicates the percentage of completed transformation, and each Y-axis indicates the raw number of graphs that are inconsistent. The scale for each Y-axis is different because the total number of inconsistent graphs generated by each system is different.

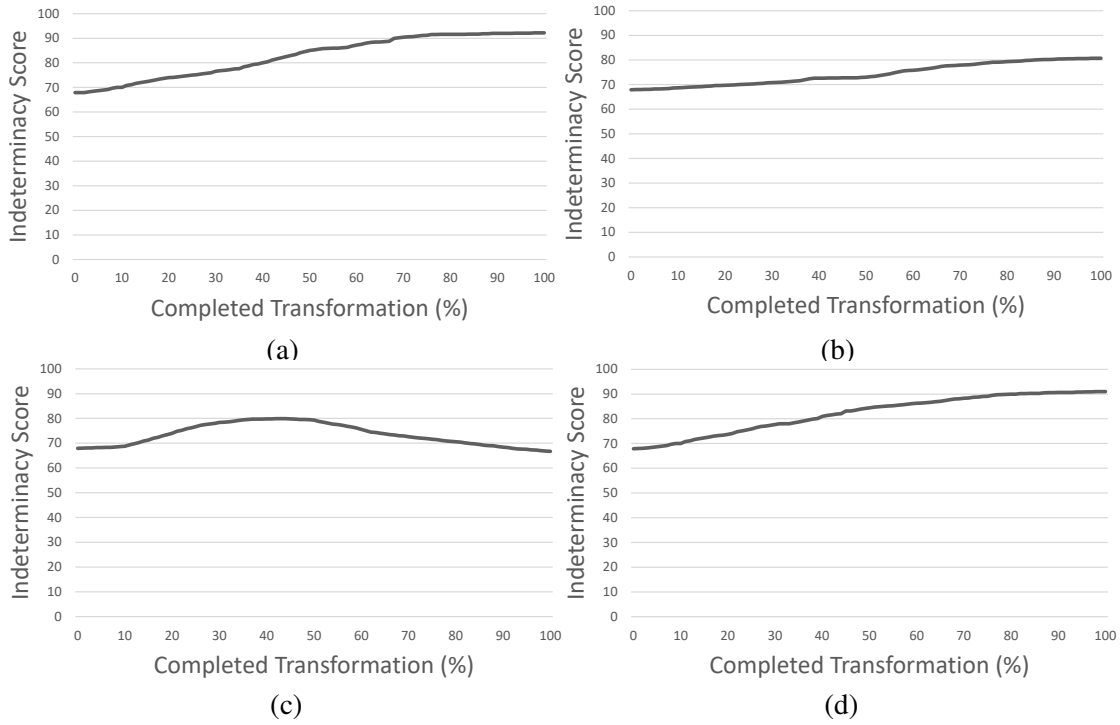

Figure 5.6: Change in average indeterminacy score during gradual, stepwise transformation of gold-standard graphs (left side of each chart) to automatically generated graphs (right side). Charts represent the average indeterminacy across all graphs for (a) CAEVO, (b) TARSQI, (c) CATENA, and (d) CLEARTK. Each X-axis indicates the percentage of completed transformation, and each Y-axis indicates the average indeterminacy score. Each Y-axis has the same scale.

line length) and Metric 6 (average number of time points) do not reveal any interesting patterns: the graphs are mainly flat with some noise. This means that when the graphs are transformed, the average main timeline length and the average number of time points stay roughly constant. For Metric 7, only TARSQI generates subordinating links, and I could discern no pattern: its graph is extremely noisy, as the number of subordinated branches should almost directly correlate with the number of subordinating links, and so addition or removal of those links will immediately change that metric.

5.4 Discussion

In this chapter, I presented a suite of evaluation metrics for automatic TimeML annotators and I evaluated four mainstream state-of-the-art automatic TimeML annotators. The results showed that automatic TimeML annotators generate many inconsistent annotations, which does not necessarily correlate with the raw performance of the annotator: for example, even though CATENA has the best F1 performance for TLINK extraction (Mirza and Tonelli, 2016), it generates 159 inconsistent files, many more than TARSQI, which has worse TLINK detection performance but only 9 generates inconsistent files.

Unlike the gold-standard annotations, automatic annotations have much more BEFORE and AFTER links (70–80% of total TLINKs). This is mainly because the automatic TimeML annotators generate only a limited number of TLINK types, therefore, many links are mislabeled.

Three of the four of the automatic TimeML annotators do not generate SLINKs, and so they don't distinguish real-life events from possible, conditional, or modal subordinated events in their timelines. The system that does generate SLINKs, TARSQI, misses more than half of the ALINKs and SLINKs.

As expected, mislabeled events, times, and links resulted in graphs that are very different from gold-standard graphs (78–139 edits distant). I investigated how various metrics will change during gradual transformation from the gold standard to the automatically generated. This showed that, for inconsistencies, the relationship is not linear; in particular, beyond a certain point inconsistencies rapidly increase.

CHAPTER 6

DURATION EXTRACTION

6.1 Motivation

A narrative is a sequence of events and for narrative understanding, it's critical to infer event durations. Because the timeline of a narrative gives the global order of events and times, combining the timeline with the event durations will give the duration of the entire narrative.

There are two datasets for event duration. The McTaco dataset contains only a few hundred of event durations (Zhou et al., 2019). The TimeBank duration dataset contains 58 news articles with a 44% inter-annotator agreement score for the fine-grained task of duration prediction. Both datasets have only a limited number of event durations and have low agreement scores for a wide range of event durations. This suggests that a large dataset that contains more accurate event durations is needed.

6.2 Methodology

I present a pipeline to extract event durations from large data. The pipeline consists of four steps. First, I clean the data and split the texts into sentences. Second, using SynTime, I detect temporal/time expressions (TIMEX) in each sentence. Third, using my own rule-based system, I perform temporal pattern mining and categorize possible event durations into 10 categories. Finally, I apply duration reasoning to extract the minimum and maximum duration of each event along with the most observed duration. In the end, I build a large dataset that contains all detected possible durations for each event, as well as statistical information about the event durations.

6.2.1 Data Preprocessing

For this experiment, I used two datasets: The motif dataset and the intelligent Web (iWeb) corpus. The motif dataset contains 8,059 texts about folktales from Irish, Puerto Rican, and Jewish cultures (Yarlott et al., 2022). On the other hand the iWeb corpus is web-based corpus and contains nearly 14 billion words from 22 million web pages (Davies and Kim, 2019). Because the iWeb corpus is already too large, in my experiment, I used the first 14 million sentences of the corpus.

First, for both datasets, I removed the irrelevant content such as headlines, usernames, XML tags, phone numbers, and website names. Next, for each text, I split the sentences using Spacy. Spacy is a python library to perform NLP tasks such as tokenization, sentence splitting, name entity recognition, dependency parsing, and much more (Vasiliev, 2020). Finally, I removed the duplicate sentences.

6.2.2 Temporal Expression Recognition

Now that I have millions of sentences, the next goal is to extract sentences that contain the possible duration of an event. Because durations are temporal expressions, using SynTime, I performed temporal expression recognition. SynTime is a light-rule-based temporal expression detector. It identifies the time tokens from raw text, then looks for modifiers and numerals next to time tokens to form time segments, and finally merges the time segments into time expressions (Zhong et al., 2017).

An example of SynTime output for one of the sentences in the dataset ("The three hotels are all within about 6 to 10 minutes peaceful walk to Consulate.") is shown below.

(1) The three hotels are all within about <TIMEX3 tid="t1" type="DURATION" value="PT6M">6</TIMEX3> to <TIMEX3 tid="t2" type="DURATION"

value=”PT10M”>10 minutes</TIMEX3> peaceful walk to the Consulate.

Here, *PT6M* and *PT10M* are normalized TIMEX values, indicating 6 minutes and 10 minutes respectively.

After the temporal expression recognition, I eliminated the sentences that did not have <TIMEX> tags because they did not have any duration candidate.

6.2.3 Temporal Pattern Mining

After the temporal expression recognition step, I obtained sentences with temporal expressions. However, this doesn't mean every remaining sentence was useful for duration inference. The following example contains a temporal expression (5 pm), however, it does not imply anything about the duration of the event (playing basketball).

(2) He played basketball after 5 pm.

On the other hand, we cannot apply the same duration extraction logic for each temporal expression. In the below examples, there are two sentences that require two different methods for duration extraction.

(3) Each quarter of the basketball game takes 12 minutes.

(4) He's been working since Monday.

In example (3), the event's duration is the exact value of the TIMEX. However, in example (4), the event's duration is at least that from the TIMEX value to the document creation date. As can be seen, each duration candidate requires different types of duration extraction methods. Therefore, I defined 10 temporal patterns to categorize duration candidates, shown below with an example for each temporal pattern.

1. **for + TIMEX**

   He took a walk for 45 minutes.

2. **between + TIMEX + and + TIMEX**

   The power went out between 4 pm and 6 pm.

3. **TIMEX1 + “to” + TIMEX2**

   He played soccer from 1 pm to 2.30 pm.

4. **last + TIMEX**

   The dinner lasted 40 minutes.

5. **take + TIMEX**

   Each class in middle school in Turkey takes 40 minutes.

6. **on/in/at + TIMEX**

   The picnic is on Wednesday.

   The movie comes out in October.

   The robbery happened at twilight.

7. **daily**

   He works out daily.

8. **each/every + TIMEX**

   He plays video games every day.

   He ate croissants each morning when he was in Paris.

9. **per/once/twice/... times + TIMEX**

   She watches a movie once a day.

   Muslims pray five times a day.

10. **since + TIMEX**

    She’s been doing the puzzle since 2 pm.

Corpus	For	Between	.. to ..	Last	Take	on/ in/at	daily	each/ every	per/ times	since	Total Duration
Motif	3,126	239	825	28	243	18,523	351	745	426	962	**25,468**
iWeb	72,943	4,229	8,228	860	5,439	268,247	10,038	17,739	13,923	18,121	**419,767**
Combined	**76,069**	**4,468**	**9,053**	**888**	**5,682**	**286,770**	**10,389**	**18,484**	**14,349**	**19,083**	**445,235**

Table 6.1: Results of temporal pattern mining for categories 1–10. The combined row is the statistics for the duration dataset that I created.

Using regular expressions (RegEx), I performed temporal pattern mining on the sentences with TIMEX. I classified duration candidates into 10 categories. The quantitative result of this temporal pattern mining is shown in Table 6.1.

6.2.4 Duration Reasoning

As explained in the previous subsection, each duration candidate may require a different duration extraction method. I have defined 10 duration categories. Categories 1 to 5 can return the exact duration of the mentioned event. For instance, the example sentence for category 5 indicates that the duration of a middle school class in Turkey is 40 minutes.

Categories 6 to 9 return the upper-bound duration. If there's an event that has no duration candidate in categories 1 to 5, I can check the duration candidates for that event in categories 6 to 9. Assume *picnic* falls under this situation and we have a duration candidate in category 6: "The picnic is on Wednesday." This duration candidate indicates that the *picnic* takes less than a day. Finally, the last category (10) returns the lower-bound duration of the mentioned event.

My duration reasoning starts as follows. First, I extract the exact duration of events from the candidates in categories 1–5. For 1, 4, and 5 the event duration will be the *normalized TIMEX value*. For 2 and 3, the event duration will be *TIMEX2 value minus TIMEX1 value*. For example, for "He played soccer from 1 pm to 2.30 pm." the duration of *playing soccer* will be *2.30 - 1* = 1.5 hours (PT1.5H).

Then, with the exact duration candidates, I calculate the minimum duration and maximum duration. And again using the normalized TIMEX value, finally, I extract the most frequent exact duration of an event. Let's illustrate this for the event *walking*. In the duration dataset that I created, for *walking*, there are 54 event durations from 37 sentences that fall under categories 1 to 5. These 37 sentences can be seen in the Appendix. Figure 6.1 is the bar graph representation of these 54 exact event durations. My duration reasoning system would indicate that *walking* takes between 3 minutes and 3 days, and most likely it takes 30 minutes because 30 minutes is the most frequent value.

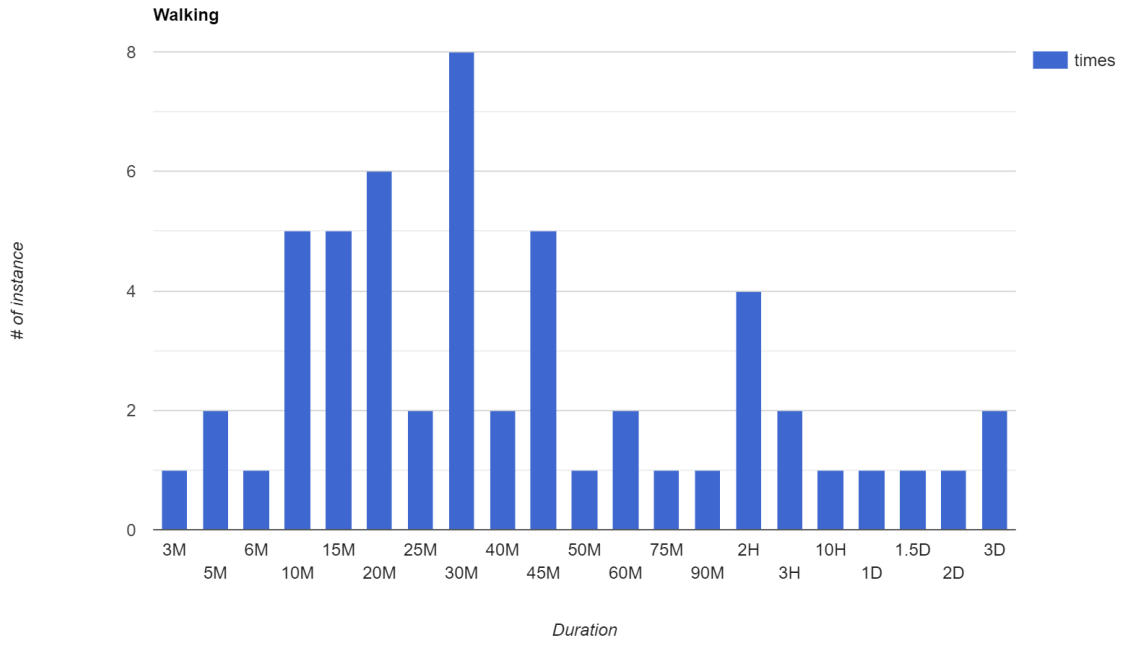

Figure 6.1: The bar graph of exact duration candidates for the event *walking*. The X-axis is the exact duration (M, minutes; H, hours; D, days) and the Y-axis is the number of instances of the duration.

If there's no event duration in categories 1–5, my duration reasoning system checks for categories 6–10 to extract lower-bound and upper-bound durations. Categories 6 and 8, the upper bound duration will be *the normalized TIMEX value*. For 7, the upper bound duration will be *P24H* because the TIMEX is always *daily*. For 9, the upper bound du-

ration will be *the normalized TIMEX value divided by repeating number*. For example, once a day: P24H / 1 = P24H, twice a day: P24H / 2 = P12H, three times a day: P24H / 3 = P8H, etc. On the other hand, category 10 gives the lower-bound duration and it's *DCT minus the normalized TIMEX value*. For example, "The construction has been going on since Monday (DCT=Friday)". Here the lower-bound duration for *construction* is Friday - Monday = P4D.

6.3 Combining Durations with Timelines

By combining event durations with the timeline, one can estimate the duration of the narrative. In §6.2.3, I present a dataset that contains over 400,000 possible event durations, and in §6.2.4, I present a duration reasoning to extract duration statistics for each event including minimum, maximum, and most likely duration. Combining these two with the timeline of a narrative, I can obtain a possible minimum, maximum, and most likely duration of the narrative. To illustrate this on a natural language text, I will use the text in Example(5), which is a snippet of the TimeML-annotated text from the ProppLearner corpus. In this text, each event is underlined.

(5) The fox ran$_{e1}$ home. "Bukhtan Bukhtanovich, have you any clothes? Put them on$_{e2}$." He dressed$_{e3}$ and, accompanied$_{e4}$ by the fox, went$_{e5}$ to the tsar. They walked$_{e6}$ along the market place and had to cross on$_{e7}$ a board over a muddy ditch. The fox gave Bukhtan a push$_{e8}$ and he fell$_{e9}$ into the mud. The fox ran$_{e10}$ to him. "What is the matter with you, Bukhtan Bukhtanovich?" Saying$_{e11}$ this, the fox smeared$_{e12}$ him with mud all over. "Wait$_{e13}$ here, Bukhtan Bukhtanovich, I shall run$_{e14}$ to the tsar." The fox came$_{e15}$ to the tsar and said$_{e16}$: "Tsar, I was walking$_{e17}$ with Bukhtan Bukhtanovich on a board over a ditch - it was a wretched little

board; we were not careful enough and somehow fell$_{e18}$ into the mud. Bukhtan Bukhtanovich is all dirty and unfit to come$_{e19}$ to town; have you some clothes you could lend$_{e20}$ him?" "Here, take these." The fox took the clothes and came$_{e21}$ to Bukhtan Bukhtanovich. "Here, change$_{e22}$ your clothes, Bukhtan, and let us go$_{e23}$."

For each event in this text, I extracted minimum, maximum, and most frequent duration from the duration dataset, and they are shown below. Note that 'MF' means most frequent in the dataset.

- **Running**: [15mins–30mins] (MF:20mins)
- **Dressing/changing/putting clothes on**: [few mins–morning] (MF:morning)
- **Being accompanied**: [5secs—few days] (MF:15mins)
- **Going**: [2secs–39years] (MF:15mins)
- **Walking**: [3mins–3days] (MF:30mins)
- **Crossing on**: [55secs–1day] (MF:1min)
- **Pushing**: [1sec–1day] (MF:1min)
- **Falling**: [1sec–1year] (MF:few seconds)
- **Smearing**: [few seconds–1M] (MF:few seconds)
- **Saying**: [3secs–24H] (MF:seconds)
- **Waiting**: [20secs–25years] (MF:40mins)
- **Coming**: [5secs–1year] (MF:1min)
- **Lending**: [2days–10months] (MF:few days)

```
Main Timeline: {
    e1- = 1
    e1+ = 2
    e3- = 3
    e3+ = 4
    e4- = 5
    e5- = 5
    e6- = 5
    e17- = 5
    e17+ = 6
    e6+ = 6
    e7- = 7
    e7+- = 8
    e5+ = 8
    e4+ = 8
    e8- = 9
    e8+ = 10
    e9- = 11
    e18- = 11
    e9+ = 12
    e18+ = 12
    e10- = 13
    e10+ = 14
    e11- = 15
    e11+ = 16
    e12- = 17
    e12+ = 18
    e15- = 19
    e15+ = 20
    e16- = 21
    e16+ = 22
    e21- = 23
    e21+ = 24
}
Attachment Points: {e1->e2, e12->e13, e16->e20, e21->e22}
Subordinated Timelines: {
[e19- = 1, e19+ = 2, e20- = 3, e20+ = 4],
[e13- = 1, e14- = 2, e14+ = 3, e13+ = 4],
[e22- = 1, e22+ = 2, e23- = 3, e23+ = 4],
[e2- = 1, e2+ = 2]
```

Listing 6.1: jTLEX timeline output for the example text.

The timeline of this text is shown in Listing 6.1. Here I used the jTLEX's output (described in §3.5).

By combining all of the most likely durations together, we could say that this narrative most likely takes *1 hour* (PT20M + PT1M + PT15M + PT1M + PT5S + PT5S + PT20M + PT5S + PT5S + PT1M + PT3S + PT1M).

6.4 Discussion & Future Work

I presented a suite of methods to extract event durations from large raw data. I also created a duration dataset that contains over 400,000 event durations.

Prior work performed fine-grained classification and they classified event durations into seconds, minutes, hours, days, weeks, months, and years. Even though this is a simple classification, the inter-annotator agreement score is very low because time is relative and so is duration. For example, the duration of a wedding in Europe is a couple of hours. However, the same event takes a few days in South Asia.

I also note that prior work's inter-annotator agreement score is between only two people. A third annotator can decrease the inter-annotator score dramatically even though the score is already low. This suggests that there may not be a duration that is acceptable for everyone, even for sports. For example, in Europe, each quarter of a basketball game takes 10 minutes; meanwhile, in the US, each quarter of an NBA game takes 12 minutes, and in the NCAA, there are no quarter (two halves that are 20 minutes each).

Another reason for a poor agreement score is duration depends on the context. While *walking* takes minutes, *walking with God* can take years. And as can be seen in this example, the subject or the object of the event can completely change the event duration. In future work, I would like to address this problem by including the subject and object

of the events in temporal reasoning in order to have more reasonable event durations for each context.

CHAPTER 7

CONCLUSION

As I discussed in Chapter 1, my research problem is analyzing narratives temporally and this temporal analysis consists of four main tasks: Timeline extraction, gold-standard corpus evaluation, automatic TimeML annotator evaluation, and duration extraction.

In Chapter 2, I explained the existing work related to temporal analysis of narratives, including definitions, prior approaches, and the datasets that I use for my experiments.

In Chapter 3, adapting prior work on solving point algebra problems to the task of extracting timelines from TimeML annotated texts, I demonstrated an exact, end-to-end solution, which I call TLEX (TimeLine EXtraction). TLEX transforms TimeML annotations into a collection of timelines arranged in a trunk-and-branch structure. As has been done in prior work, TLEX checks the consistency of the temporal graph and solves it; however, it adds two novel functionalities. First, it identifies the specific relations involved in an inconsistency (which can then be manually corrected) and, second, TLEX performs a novel identification of sections of the timelines that have indeterminate order, information critical for downstream tasks such as aligning events from different timelines. I provided formal proof of TLEX's correctness, and I also conducted an experimental evaluation by applying TLEX to 385 TimeML annotated texts from four corpora. I showed that 123 of the texts are inconsistent, 181 of them have more than one "real world" or main timeline, and there are 2,541 indeterminate sections across all four corpora. A sampling evaluation showed that TLEX is 98–100% accurate with 95% confidence along five dimensions: the ordering of time points, the number of main timelines, the placement of time points on main versus subordinate timelines, the attachment point of branch timelines, and the location of the indeterminate sections. I provided a reference implementation of TLEX, the extracted timelines for all texts, and the manual corrections to the temporal graphs of the inconsistent texts.

Using TLEX, I compared TDTs with temporal graphs and showed that generating TDTs results in up to a 109% increase in temporal indeterminacy over their corresponding temporal graphs for the three corpora I examined. On average, the increase in indeterminacy is 32%, and I showed that this increase is a result of the TDT representation eliminating on average only 2.4% of total temporal relations. This result suggests that small differences can have big effects in temporal graphs, and the use of TDTs must be balanced against their deficiencies, with tasks requiring an accurate global temporal ordering potentially calling for use of the full temporal graph.

Finally, I presented jTLEX, an open-source Java library to extract exact timelines from TimeML annotated texts. jTLEX provides many useful methods for the TimeML community such as TimeML parsing, graph extraction, timeline extraction, inconsistency detection, and temporal indeterminacy calculation. jTLEX can be used on any TimeML annotations in any domain of natural language. I released jTLEX as an open-source library that is free for non-commercial use.

In Chapter 4, I described methods for detecting and correcting errors in the TimeBank corpus: (a) automatic guideline checking (109 violations); (b) automatic inconsistency checking (65 inconsistent files); (c) automatic disconnectivity checking (625 incorrect breakpoints); and (d) manual comparison with the output of state-of-the-art automatic annotators to identify missing annotations (317 events, 52 temporal expressions). I provided the code as well as a set of patch files that can be applied to the TimeBank corpus to produce a corrected version for use by other researchers in the field.

In Chapter 5, I presented a novel suite of eight metrics, combined with a new graph-transformation experimental design, for the holistic evaluation of TimeML graphs. I applied these metrics to four automatic TimeML annotation systems (CAEVO, TARSQI, CATENA, and ClearTK). I showed that on average 1/3 of the TimeML graphs produced using these systems are inconsistent, and there is on average 1/5 more temporal indeter-

minacy than the gold standard. I also showed that the automatically generated graphs are on average 109 edits from the gold standard, which is 1/3 toward complete replacement. Finally, I showed that the relationship between individual subtask performance and graph quality is non-linear: small errors in TimeML subtasks result in rapid degradation of final graph quality. These results suggest current automatic TimeML annotators are far from optimal and significant further improvement would be useful.

In Chapter 6, I presented a pipeline system to extract the durations of events and build a large event duration dataset. I applied my system to the iWeb corpus and the Motif corpus and extracted over 400,000 event durations. With these events with durations, I created the duration dataset and I will release it open source.

7.1 Future Work

In this dissertation, I presented a temporal analysis system for narratives including timeline extraction, gold-standard TimeML evaluation, automatic TimeML annotator evaluation, and duration extraction. The detailed analysis of results revealed several major limitations that are useful for the future directions.

Timeline Extraction

The results of the TLEX algorithm suggest that many TimeML annotations contain temporal inconsistency. Algorithm 1a in §3.2.4 finds the inconsistent subgraphs in the PA graph, which helps annotators to fix them manually. For future work, I would like to develop a method that removes or changes the links to fix the temporal inconsistency automatically. This could be done in several ways. First, many inconsistencies introduced by ALINKs showed that annotators accidentally assign the aspectual event as the second node of the ALINK. A rule-based system can make sure the aspectual event is the first node of the ALINK and fix the broken ALINKs. Second, 40 out of 110 detected inconsis-

tencies were caused by two different temporal relations between two events in the same sentence. A supervised learning-based system can infer the correct relation between these two events and remove the incorrect relation. Third, the results showed many inconsistencies were in a cycle form, comprising three or more links. A hybrid system (supervised learning and rule-based) can detect the problematic link and correct it to break the cycle.

Another limitation of the TLEX algorithm is that it only detects if the two intervals are indeterminate. However, the results showed that the temporal indeterminacy between two intervals can be weak (only two possible orders), strong (two to seven possible orders), or fully indeterminate (all possible orders). Weak indeterminacy can be fixed easily with a rule-based system. However, with the current algorithm (algorithm 2 in §3.2.5), all timelines need to be extracted in order to detect weak indeterminate intervals. This is computationally expensive considering there are $\sum_{x=1}^{2m} x!$ possible timelines. Therefore, I would like to implement an algorithm to detect the temporal indeterminacy on the TimeML graph itself, using temporal closure rules, instead of extracting all possible timelines from the graph.

Gold-Standard TimeML Evaluation

I implemented a suite of methods to evaluate the TimeBank corpus and showed the corpus has 1,630 incorrect or missing annotations due to manual annotation mistakes. Considering there are many other manually annotated corpora that might suffer from these mistakes, I would like to build an automatic sanity check algorithm that works for any TimeML annotated corpora. As described in Chapter 4, the automatic sanity check algorithm will also check TimeML rules, inconsistency, redundancy, and disconnectivity but go further by checking the markup language format (TEI, XML, and SGML), grammars, typos, irrelevant tags (such as CoreNLP tags), and non-optional/obligatory attributes.

TimeML Annotator Evaluation

I designed a suite of metrics to evaluate four state-of-the-art automatic TimeML annotators and the results showed that these systems generate incorrect links and inconsistent annotations and do not produce many types of TimeML links. Therefore, I would like to build an automatic TimeML annotator addressing these problems. In addition to the supervised learning-based part, first, I would like to integrate TLEX's inconsistency correction algorithm into my system to obtain fully consistent annotations. Second, I would like to use the disconnectivity detector (presented in §4.2.2) to have fully connected graphs. Third, using TimeML rules checker in §4.2.1, my system will produce annotations that follow the TimeML rules. Fourth, my system will annotate SLINKs automatically using a rule-based system that uses event classes. Finally, my system will use the pre-defined trigger words such as *in, on, at, during, for, while*, and *since* to extract temporal relations.

Duration Extraction

I presented an event duration extraction system. The results showed that the duration range for an event can be from seconds to years. The reason is that the arguments of the event such as subjects and objects can increase or decrease the duration significantly. Therefore, I would like to use the entity that performs the event as a feature for my event duration extraction system. Similarly, the entity affected by the event will be considered during the event extraction process. For this, I could take advantage of semantic role labeling.

BIBLIOGRAPHY

W Ahrens. Ueber das gleichungssystem einer kirchhoff'schen galvanischen stromverzweigung. *Mathematische Annalen*, 49(2):311–324, 1897.

James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983. ISSN 0001-0782. doi: 10.1145/182.358434. URL http://doi.acm.org/10.1145/182.358434.

Rakshita Bansal, Monika Rani, Harish Kumar, and Sakshi Kaushal. Temporal event detection using supervised machine learning based algorithm. In *International Conference on Innovations in Bio-Inspired Computing and Applications*, pages 257–268. Springer, 2018.

R. Barták, R.A. Morris, and K.B. Venable. *An Introduction to Constraint-Based Temporal Reasoning*. Morgan & Claypool Publishers, 2014.

David Benavides, Sergio Segura, Pablo Trinidad, and Antonio Ruiz-Cortés. Using java csp solvers in the automated analyses of feature models. In *International Summer School on Generative and Transformational Techniques in Software Engineering*, pages 399–408. Springer, 2005.

Steven Bethard. Cleartk-timeml: A minimalist approach to tempeval 2013. In *Second joint conference on lexical and computational semantics (* SEM), volume 2: proceedings of the seventh international workshop on semantic evaluation (SemEval 2013)*, pages 10–14, 2013.

Branimir Boguraev and Rie Kubota Ando. Analysis of TimeBank as a resource for TimeML parsing. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy, May 2006. European Lan-

guage Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2006/pdf/346_pdf.pdf.

Branimir Boguraev, James Pustejovsky, Rie Ando, and Marc Verhagen. Timebank evolution as a community resource for timeml parsing. *Language Resources and Evaluation*, 41(1):91–115, 2007.

Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters*, 19(3-4):255–259, 1998.

Tommaso Caselli and Roser Morante. Systems' agreements and disagreements in temporal processing: An extensive error analysis of the tempeval-3 task. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

Nathanael Chambers. Navytime: Event and time ordering from raw text. Technical report, NAVAL ACADEMY ANNAPOLIS MD, 2013.

Nathanael Chambers, Taylor Cassidy, Bill McDowell, and Steven Bethard. Dense event ordering with a multi-pass architecture. *Transactions of the Association for Computational Linguistics*, 2:273–284, 2014. doi: 10.1162/tacl_a_00182. URL https://doi.org/10.1162/tacl_a_00182.

Angel Chang and Christopher D Manning. Sutime: Evaluation in tempeval-3. In *Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 78–82, 2013.

Sanxing Chen, Guoxin Wang, and Börje Karlsson. Exploring word representations on time expression recognition. Technical report, Technical report, Microsoft Research Asia, 2019.

Yuchang Cheng, Masayuki Asahara, and Yuji Matsumoto. NAIST.Japan: Temporal relation identification using dependency parsed tree. In *Proceedings of the 4th International Workshop on Semantic Evaluations (SemEval '07)*, pages 245–248, Prague, Czech Republic, 2007. URL `http://dl.acm.org/citation.cfm?id=1621474.1621526`.

Jean-François Condotta, Issam Nouaouri, and Michael Sioutis. A sat approach for maximizing satisfiability in qualitative spatial and temporal constraint networks. In *Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'16, page 432–442. AAAI Press, 2016.

Carlo Andrea Conte, Raphael Troncy, and Mor Naaman. Extracting resources that help tell events' stories. In *Proceedings of the International Workshop on Social Multimedia and Storytelling (SoMuS 2014)*, 2014. Glasgow, UK.

Mark Davies and Jong-Bok Kim. The advantages and challenges of" big data": Insights from the 14 billion word iweb corpus. *Linguistic Research*, 36(1):1–34, 2019.

Leon Derczynski and Robert J. Gaizauskas. Analysing temporally annotated corpora with CAVaT. In *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*, 2010. Valetta, Malta.

Wentao Ding, Guanji Gao, Linfeng Shi, and Yuzhong Qu. A pattern-based approach to recognizing time expressions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6335–6342, 2019.

Quang Xuan Do, Wei Lu, and Dan Roth. Joint inference for event timeline construction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL'12)*, pages 677–687, 2012.

Michael Färber and Achim Rettinger. Toward real event detection. In *derive@ ESWC*, pages 24–34, 2015.

L. Ferro, L. Gerber, I. Mani, B. Sundheim, and G. Wilson. Tides temporal annotation guidelines, ver. 1.0.2, 2001. http://www.timeml.org/terqas/readings/MTRAnnotationGuide_v1_02.pdf.

Lisa Ferro, Laurie Gerber, Inderjeet Mani, Beth Sundheim, and George Wilson. Tides: 2003 standard for the annotation of temporal expressions. Technical report, MITRE CORP MCLEAN VA, 2003.

Mark Finlayson, Jeffry Halverson, and Steven Corman. The n2 corpus: A semantically annotated collection of islamist extremist stories. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 896–902, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2014/pdf/48_Paper.pdf.

Mark A. Finlayson. Propplearner: Deeply annotating a corpus of russian folktales to enable the machine learning of a russian formalist theory. *Digital Scholarship in the Humanities*, 32(2):284–300, 2017. doi: 10.1093/llc/fqv067. URL +http://dx.doi.org/10.1093/llc/fqv067.

Diana Galvan, Naoaki Okazaki, Koji Matsuda, and Kentaro Inui. Investigating the challenges of temporal relation extraction from clinical text. In *Proceedings of the Ninth International Workshop on Health Text Mining and Information Analysis*, pages 55–64, 2018.

Zeno Gantner, Matthias Westphal, and Stefan Wolfl. Gqr - a fast reasoner for binary qualitative constraint calculi, 2008.

Gérard Genette, Alan Sheridan, and Marie-Rose Logan. *Figures of literary discourse*. Columbia University Press New York, 1982.

Ian P Gent and Barbara Smith. *Symmetry breaking during search in constraint programming*. Citeseer, 1999.

Alfonso Gereveni and Lenhart Schubert. Efficient algorithms for qualitative reasoning about time. *Artif. Intell.*, 74(2):207–248, April 1995. ISSN 0004-3702. doi: 10.1016/0004-3702(94)00016-T. URL https://doi.org/10.1016/0004-3702(94)00016-T.

Jaipal Goud, Pranav Goel, Allen J Antony, and Manish Shrivastava. Leveraging multilingual resources for open-domain event detection. In *Proceedings 15th Joint ACL-ISO Workshop on Interoperable Semantic Annotation*, pages 76–82, 2019.

Andrey Gusev, Nathanael Chambers, Divye Raj Khilnani, Pranav Khaitan, Steven Bethard, and Dan Jurafsky. Using query patterns to learn the duration of events. In *Proceedings of the Ninth International Conference on Computational Semantics (IWCS 2011)*, 2011.

Oana Inel and Lora Aroyo. Validation methodology for expert-annotated datasets: Event annotation case study. In *2nd Conference on Language, Data and Knowledge (LDK 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

Yoichi Iwata and Yuichi Yoshida. Exact and Approximation Algorithms for the Maximum Constraint Satisfaction Problem over the Point Algebra. In Natacha Portier and Thomas Wilke, editors, *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 127–138, Dagstuhl, Germany, 2013. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-939897-50-7. doi: 10.4230/LIPIcs.

STACS.2013.127. URL http://drops.dagstuhl.de/opus/volltexte/2013/3928.

Mahfujul Kadir, Sadman Sobhan, and Md Zahidul Islam. Temporal relation extraction using apriori algorithm. In *2016 5th International Conference on Informatics, Electronics and Vision (ICIEV)*, pages 915–920. IEEE, 2016.

Oleksandr Kolomiyets and Marie-Francine Moens. Comparing two approaches for the recognition of temporal expressions. In *Annual Conference on Artificial Intelligence*, pages 225–232. Springer, 2009.

Oleksandr Kolomiyets, Steven Bethard, and Marie-Francine Moens. Extracting narrative timelines as temporal dependency structures. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL'12)*, pages 88–97, 2012.

Arne Kreutzmann and Diedrich Wolter. Qualitative spatial and temporal reasoning with and/or linear programming. In *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, ECAI'14, pages 495–500, Amsterdam, The Netherlands, The Netherlands, 2014. IOS Press. ISBN 978-1-61499-418-3. doi: 10.3233/978-1-61499-419-0-495. URL https://doi.org/10.3233/978-1-61499-419-0-495.

Krzysztof Kuchcinski and Radoslaw Szymanek. JaCoP: Java constraint programming solver, 2013. http://jacop.cs.lth.se/.

Philippe Laban and Marti Hearst. newsLens: building and visualizing long-ranging news stories. In *Proceedings of the Events and Stories in the News Workshop*, pages 1–9, 2017. Vancouver, Canada.

A. Leeuwenberg and M. F. Moens. Towards extracting absolute event timelines from english clinical reports. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:2710–2719, 2020. doi: 10.1109/TASLP.2020.3027201.

Valentina Bartalesi Lenzi, Giovanni Moretti, and Rachele Sprugnoli. Cat: the celct annotation tool. In *LREC*, pages 333–338. Citeseer, 2012.

Gérard Ligozat. "corner" relations in allen's algebra. *Constraints*, 3(2):165–177, 1998.

Shixia Liu, Michelle X. Zhou, Shimei Pan, Yangqiu Song, Weihong Qian, Weijia Cai, and Xiaoxiao Lian. Tiara: Interactive, topic-based visual text summarization and analysis. *ACM Trans. Intell. Syst. Technol.*, 3(2):25:1–25:28, February 2012. ISSN 2157-6904. doi: 10.1145/2089094.2089101. URL http://doi.acm.org/10.1145/2089094.2089101.

Inderjeet Mani, Marc Verhagen, Ben Wellner, Chong Min Lee, and James Pustejovsky. Machine learning of temporal relations. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics (ICCL-ACL'06)*, pages 753–760, 2006. Sydney, Australia.

Anne-Lyse Minard, Manuela Speranza, Ruben Urizar, Begoña Altuna, Marieke van Erp, Anneleen Schoen, and Chantal van Son. MEANTIME, the NewsReader multilingual event and time corpus. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 4417–4422, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA). URL https://www.aclweb.org/anthology/L16-1699.

Seyed Abolghasem Mirroshandel and Gholamreza Ghassem-Sani. Temporal relation extraction using expectation maximization. In *Proceedings of the International Conference Recent Advances in Natural Language Processing 2011*, pages 218–225, 2011.

Paramita Mirza and Sara Tonelli. Catena: Causal and temporal relation extraction from natural language texts. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 64–75, 2016.

Borja Navarro-Colorado and Estela Saquete. Cross-document event ordering through temporal, lexical and distributional knowledge. *Know.-Based Syst.*, 110(C):244–254, October 2016. ISSN 0950-7051. doi: 10.1016/j.knosys.2016.07.032. URL https://doi.org/10.1016/j.knosys.2016.07.032.

Bernhard Nebel and Hans-Jürgen Bürckert. Reasoning about temporal relations: A maximal tractable subclass of allen's interval algebra. *J. ACM*, 42(1):43–66, January 1995. ISSN 0004-5411. doi: 10.1145/200836.200848. URL https://doi.org/10.1145/200836.200848.

Qiang Ning, Zhongzhi Yu, Chuchu Fan, and Dan Roth. Exploiting partially annotated data for temporal relation extraction. *arXiv preprint arXiv:1804.08420*, 2018.

Qiang Ning, Zhili Feng, and Dan Roth. A structured learning approach to temporal relation extraction. *CoRR*, abs/1906.04943, 2019. URL http://arxiv.org/abs/1906.04943.

Mustafa Ocal and Mark Finlayson. Evaluating information loss in temporal dependency trees. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 2148–2156, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL https://www.aclweb.org/anthology/2020.lrec-1.263.

Mustafa Ocal, Adrian Perez, Antonela Radas, and Mark Finlayson. Holistic evaluation of automatic timeml annotators. In *Proceedings of The 13th Language Resources and Evaluation Conference*, Marseille, France, June 2022a. European Language Resources Association.

Mustafa Ocal, Antonela Radas, Jared Hummer, Karine Megerdoomian, and Mark Finlayson. A comprehensive evaluation and correction of the timebank corpus. In *Proceedings of the Language Resources and Evaluation Conference*, pages 2919–2927, Marseille, France, June 2022b. European Language Resources Association. URL https://aclanthology.org/2022.lrec-1.313.

Feng Pan, Rutu Mulkar-Mehta, and Jerry R Hobbs. Learning event durations from event descriptions. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 393–400, 2006.

Feng Pan, Rutu Mulkar-Mehta, and Jerry R Hobbs. Annotating and learning event durations in text. *Computational Linguistics*, 37(4):727–752, 2011.

Judea Pearl and Richard E Korf. Search techniques. *Annual Review of Computer Science*, 2(1):451–467, 1987.

James Pustejovsky, José Castaño, Robert Ingria, Roser Saurí, Robert Gaizauskas, Andrea Setzer, and Graham Katz. TimeML: robust specification of event and temporal expressions in text. In *Fifth International Workshop on Computational Semantics (IWCS-5)*, pages 1–11, 2003a. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.161.8972&rep=rep1&type=pdf.

James Pustejovsky, Patrick Hanks, Roser Saurí, Andrew See, Rob Gaizauskas, Andrea Setzer, Dragomir Radev, Beth Sundheim, David Day, Lisa Ferro, and Marcia Lazo. The timebank corpus. *Proceedings of Corpus Linguistics*, 01 2003b.

Livio Robaldo, Tommaso Caselli, Irene Russo, and Matteo Grella. From italian text to TimeML document via dependency parsing. In *Proceedings of the 12th Computational Linguistics and Intelligent Text Processing (CICLing 2011)*, pages 177–187, Tokyo, Japan, 2011. doi: 10.1007/978-3-642-19437-5_14.

E. Saquete, P. Martínez-Barco, R. Muñoz, and J. L. Vicedo. Splitting complex temporal questions for question answering systems. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics. doi: 10.3115/1218955.1219027. URL `https://doi.org/10.3115/1218955.1219027`.

M.N.K. Saunders, Philip Lewis, and Adrian Thornhill. *Research Methods for Business Students, Fifth Edition*. Prentice Hall, New York, 2009.

Roser Saurí, Robert Knippen, Marc Verhagen, and James Pustejovsky. Evita: a robust event recognizer for qa systems. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 700–707, 2005.

Roser Sauri, Jessica Littman, Robert Gaizauskas, Andrea Setzer, and James Pustejovsky. TimeML annotation guidelines, version 1.2.1, 2006. `https://catalog.ldc.upenn.edu/docs/LDC2006T08/timeml_annguide_1.2.1.pdf`.

Sadi Evren Seker and Banu Diri. Timeml and turkish temporal logic. In *IC-AI*, volume 10, pages 881–887, 2010.

Andrea Setzer. *Temporal Information in Newswire Articles: an Annotation Scheme and Corpus Study*. PhD thesis, University of Sheffield, 2001.

Michael Sioutis and Jean-François Condotta. Tackling large qualitative spatial networks of scale-free-like structure. In *Proceedings of the 8th Hellenic Conference on AI (SETN 2014)*, volume 8445, pages 178–191, Ioannina, Greece, 05 2014. doi: 10.1007/978-3-319-07064-3_15.

Preston So. React. In *Decoupled Drupal in Practice*, pages 313–334. Springer, 2018.

Rachele Sprugnoli and Sara Tonelli. Novel event detection and classification for historical texts. *Computational Linguistics*, 45(2):229–265, June 2019. doi: 10.1162/coli_a_00347. URL https://aclanthology.org/J19-2002.

Jannik Strötgen and Michael Gertz. Heideltime: High quality rule-based extraction and normalization of temporal expressions. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 321–324, 2010.

Giang Binh Tran, Mohammad Alrifai, and Eelco Herder. Timeline summarization from relevant headlines. In *Proceedings of the 37th European Conference on Information Retreival (ECIR 2015)*, pages 245–256, 2015. Vienna, Austria.

Naushad UzZaman and James Allen. Temporal evaluation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 351–356, 2011.

Naushad UzZaman and James F Allen. Extracting events and temporal expressions from text. In *2010 IEEE Fourth International Conference on Semantic Computing*, pages 1–8. IEEE, 2010.

Peter van Beek. *Reasoning about Qualitative Temporal Information*. MIT Press, Cambridge, MA, USA, 1994. ISBN 0262560755.

Peter Van Beek and Robin Cohen. Exact and approximate reasoning about temporal relations 1. *Computational intelligence*, 6(3):132–144, 1990.

Yuli Vasiliev. *Natural Language Processing with Python and SpaCy: A Practical Introduction*. No Starch Press, 2020.

Alakananda Vempala, Eduardo Blanco, and Alexis Palmer. Determining event durations: Models and error analysis. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 164–168, 2018.

Marc Verhagen, Inderjeet Mani, Roser Saurí, Jessica Littman, Robert Knippen, Seok Bae Jang, Anna Rumshisky, John Phillips, and James Pustejovsky. Automating temporal annotation with tarsqi. In *ACL*, pages 81–84, 2005. URL `http://aclweb.org/anthology/P/P05/P05-3021.pdf`.

Marc Verhagen, Robert Knippen, Inderjeet Mani, and James Pustejovsky. Annotation of temporal relations with tango. In *LREC*, pages 2249–2252, 2006.

Amir Pouran Ben Veyseh, Minh Van Nguyen, Bonan Min, and Thien Huu Nguyen. Augmenting open-domain event detection with synthetic data from gpt-2. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 644–660. Springer, 2021.

Marc Vilain, Henry Kautz, and Peter van Beek. Constraint propagation algorithms for temporal reasoning: A revised report. In Daniel S. Weld and Johan de Kleer, editors, *Readings in Qualitative Reasoning About Physical Systems*, pages 373–381. Morgan Kaufmann, San Francisco, CA, 1990.

Jan Oliver Wallgrün, Lutz Frommberger, Diedrich Wolter, Frank Dylla, and Christian Freksa. Qualitative spatial representation and reasoning in the sparq-toolbox. In *Proceedings of the International Conference Spatial Cognition*, pages 39–58, Bremen, Germany, 09 2006. doi: 10.1007/978-3-540-75666-8_3.

Zonglin Yang, Xinya Du, Alexander Rush, and Claire Cardie. Improving event duration prediction via time-aware pre-training. *arXiv preprint arXiv:2011.02610*, 2020.

W Victor H Yarlott, Armando Ochoa, Anurag Acharya, Laurel Bobrow, Diego Castro Estrada, Diana Gomez, Joan Zheng, David McDonald, Chris Miller, and Mark A Finlayson. Finding trolls under bridges: Preliminary work on a motif detector. *arXiv preprint arXiv:2204.06085*, 2022.

A. K. Zaidi. On temporal logic programming using petri nets. *Trans. Sys. Man Cyber. Part A*, 29(3):245–254, May 1999. ISSN 1083-4427. doi: 10.1109/3468.759269. URL `https://doi.org/10.1109/3468.759269`.

Yuchen Zhang and Nianwen Xue. Neural ranking models for temporal dependency structure parsing. *CoRR*, abs/1809.00370, 2018. URL `http://arxiv.org/abs/1809.00370`.

Xiaoshi Zhong, Aixin Sun, and Erik Cambria. Time expression analysis and recognition using syntactic token types and general heuristic rules. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 420–429, 2017.

Ben Zhou, Daniel Khashabi, Qiang Ning, and Dan Roth. " going on a vacation" takes longer than" going for a walk": A study of temporal commonsense understanding. *arXiv preprint arXiv:1909.03065*, 2019.

Ben Zhou, Qiang Ning, Daniel Khashabi, and Dan Roth. Temporal common sense acquisition with minimal supervision. *arXiv preprint arXiv:2005.04304*, 2020.

Dmitriy Zhuk. A proof of the csp dichotomy conjecture. *Journal of the ACM (JACM)*, 67 (5):1–78, 2020.

APPENDIX A

APPENDIX

Duration Sentences for "Walking"

The following list contains the sentences for the event *walking* in the duration dataset discussed in §6.2.4.

1. It took about 30 minutes of walking along the road with our thumbs up before a van pulled over and offered us a ride most of the way.
2. I walked for 45 minutes, and mynetdiary records 250 calories burned.
3. I walked in the 90-degree heat for 30 minutes and realized I had no idea where I was going.
4. Back in his home, the Viscount walked for several minutes up and down his room with long quick strides."
5. He walked on for a quarter of an hour or twenty minutes, so stupefied that he no longer thought of anything.
6. We walked up and down the river for about 3 hours.
7. I had walked up a creek bed for about 30 minutes or so in the Currumbin Valley.
8. The grade ones walked for 45 minutes on Tuesday, May 30th.
9. Once there, Shackleton, Frank Worsley and Tom Crean walked almost non-stop for 36 hours without equipment across the ice-capped, mountainous island to reach the whaling station at Stromness and help.
10. Walked through water for 20 minutes with water almost up to the edge of my boot, and only got a few drops in.

11. Back then you walked 10-15 minutes back to your house and ate lunch for 15-20 minutes and then you went back.
12. My dog is walked twice a day for 40 mins each time, work it out, good value for money I walked for another hour and got back to my hovel.
13. Bath has free guided walking tours lasting two hours
14. The walk lasts about 90 minutes and then you will repair to an atmospheric pub for more magic and story-telling, and maybe a little discussion.
15. It is about 50 km long, and lasts about 3 days walking and 2 days by horse.
16. The walks last 75-minutes and cost $10 per person.
17. After running, this could be as simple as slowing down then walking for three to five minutes, sometimes lengthening your stride or lifting your knees very high.
18. As soon as you're feeling up to it, I recommend walking daily for 30 to 60 minutes.
19. A brisk walk or bike ride for 40 to 50 minutes is usually enough.
20. The area is filled with mature leafy trees and is within a 15 to 20 minute walk to the city centre, if you are going to Stephens Green.
21. The location of the Croke Park Hotel is not city centre, however it is within walking distance, for those who like to walk - between 20 to 30 minutes or so.
22. A walk up Arthur's seat can take, at a good pace, two to three hours in total.
23. There is a circular walk around the woodland which takes approximatley 20 to 25 minutes to complete, ensure you're wearing wellies or walking boots though, as the woodland surface can become very muddy in places, especially after rain in the winter!
24. The complex is about a 10 to 15 minute walk from the project offices and there is a bus for the volunteers to use for transportation as they wish.

25. ”Exercise could be anything like walking around campus for 10 to 15 minutes,” Andrew Nation, sophomore in athletic training, said.
26. The three hotels are all within about 6 to 10 minutes peaceful walk to the Consulate.
27. Driving to the location can take 25-45 minutes and the walk to the ice cave can take from 5 to 60 minutes.
28. Located in the heart of Kensington with great transport links, including Ladbroke Grove, Latimer Road and Westbourne Park all within 5 to 10 minutes walk.
29. They walked and ran for 20 to 30 minutes until Lai finally declared he was going home.
30. To get your digestion moving again, fit in some aerobic exercise, such as 30 to 45 minutes of brisk walking or another activity that raises your heart rate.
31. For humans, that would probably translate into walking 30 to 45 minutes a day five.
32. Apparently, there’s this night walk between 10PM until 8AM.
33. At a relaxed pace, walking the entire span of the waterfront could take a few hours.
34. It takes about 3 days walking or 2 days by horse to get there from the nearest place
35. Those were the days when collectors would take Friday afternoons off to learn all they could by walking around museums and galleries
36. The trail officially launches later this month and walking the whole thing will usually take two days.
37. Once you’ve taken a walking tour, which would take about two hours if you were really ambling, youve really seen it.

BEFORE	TimeML link types are in small caps
i, j	general indices
k	number of temporally and aspectually connected subgraphs in a TimeML graph
m, n	number of nodes and edges in a graph, respectively
$c = (u, v, w)$	a tuple representing a temporal constraint
$l = (u, v, w)$	a tuple representing a link in a TimeML graph
$w \in \mathscr{L}$	type of a TimeML link
$\mathscr{L}$;	set of TimeML link types
$\mathscr{L}_T, \mathscr{L}_A, \mathscr{L}_S \subset \mathscr{L}$	set of TimeML temporal, aspectual, and subordinating link types
$\mathscr{L}_{TA} = \mathscr{L}_T \cup \mathscr{L}_A$	set of TimeML temporal and aspectual link types
xRy	a relation between two objects x and y
r	a real number
s	an integer
t, t^-, t^+	a time point, and beginning and ending time points of an interval
$G = (V, E)$	a graph is made up of a set of nodes (V) and a set of edges (E)
$v \in V, e \in E$	a node or edge in a graph, respectively
$e.source, e.target$	the source and target node of an edge, respectively
$T = (V_T, E_T)$	a TimeML graph
$C = (V_C, E_C)$	temporally connected TimeML graph
$P = (V_P, E_P)$	a Point Algebra graph
$Y = (V_Y, E_Y)$	an inconsistent cycle in a PA graph
$I = (V_I, E_I)$	a maximal inconsistent PA subgraph
$J = \bigcup_i I_i = (V_J, E_J)$	union of maximal inconsistent PA subgraphs
$T_C = (C_1, C_2, \ldots C_i)$	vector of temporally connected TimeML graphs
$T_P = (P_1, P_2, \ldots P_i)$	vector of PA graphs
$T_J = (J_1, J_2, \ldots J_i)$	vector of unions of maximal inconsistent PA subgraphs
$T_L = (L_1, L_2, \ldots L_i)$	vector of timelines
$T_M = \{L_{M_1}, L_{M_2}, \ldots L_{M_i})$	set of main timelines
$T_D = (D_1, D_2, \ldots, D_i)$	vector of indeterminacy maps
$E_S = (D_1, D_2, \ldots, D_i)$	set of subordination links
$S = (r_1, r_2, \ldots r_i)$	a vector of real numbers, solution to a PA graph
$O_S = (r_1, r_2, \ldots r_i)$	an ordered set of real numbers (an *ordered solution set*)
$(r_j, r_{j+1}) \subset O_S$	neighboring pair in an ordered set of real numbers

$A = \{t_1, t_2, \ldots t_i\}$	set of time points anchored to same real number assignment (an *anchor set*)
$A_P = \{A_1, A_2, \ldots A_i\}$	set of anchor sets for a PA graph P
$L : O_S \rightarrow A_P$	a timeline
$f_p : O_{S_i} \rightarrow O_{S_j}$	bijective, strictly monotonically increasing, order-preserving function
$\overset{O}{\sim}$	order-equivalent equivalence relation over ordered solution sets
$\overset{L}{\sim}$	tempomorphic equivalence relation over timelines
$\mathbb{L}$	tempomorphic equivalence set
$\mathbb{L}^0$	shortest tempomorphic equivalence set
$\hat{\mathbb{L}}_P$	set of all tempomorphic equivalence sets for a PA graph P
L_N	normal form timeline
$\|L\|$ or $\|O_S\|$ or $\|A_P\|$	length of a timeline
L^0	shortest normal form timeline
L_M	main timeline
$p_j = (t_j, t_{j+1}) \subset V_P$	neighboring time point pair from a timeline for a PA graph $P = (V_P, E_P)$
N_{2P}	set of all neighboring time point pairs for a timeline for a PA graph P
$D : N_{2P} \rightarrow \{true, false\}$	indeterminacy map
$G_B = (T_L, E_S)$	a timeline graph where timelines are related by subordinating edges in E_S
$B_T = (G_B, T_M)$,	trunk-and-branch timeline for a TimeML graph T
B^0	shortest normal-form trunk-and-branch timeline

Table A: List of Symbols

VITA

MUSTAFA OCAL

September 18, 1995	Born; Isparta, Turkey
2017	B.S., Computer Engineering Gazi University Ankara, Turkey
2021	M.S., Computer Science Florida International University Miami, Florida, USA

PUBLICATIONS & PATENTS

Ocal, M., Radas, A., Hummer, J., Megerdoomian, K., & Finlayson, M. A. (2022). *Holistic Evaluation of Automatic TimeML Annotators*. In the 13th Edition of Language Resources and Evaluation Conference (LREC-2022). Marseille, France.

Ocal, M., Perez, A., Radas, A., & Finlayson, M. A. (2022). *A Comprehensive Evaluation and Correction of the TimeBank Corpus*. In the 13th Edition of Language Resources and Evaluation Conference (LREC-2022). Marseille, France.

Ocal. M. & Finlayson, M. A. *Systems and Methods for Evaluating Temporal Dependency Trees*, United States Patent No. 11,170,303, issued on November 9, 2021.

Finlayson, M. A., Cremisini, A., & Ocal, M., (2021) *Extracting and Aligning Timelines*, In the Computational Analysis of Storylines: Making Sense of Events. Cambridge University Press.

Ocal, M. & Finlayson, M. A. (2020). *Evaluating Information Loss in Temporal Dependency Trees*. In the 12th Edition of Language Resources and Evaluation Conference (LREC-2020). Virtual Conference.