Dynamic Relationship and Event Discovery*

Anish Das Sarma†, Alpa Jain†, Cong Yu‡
†Yahoo! Labs, Sunnyvale, CA; ‡Google Research, New York, NY
{anishdas,alpa}@yahoo-inc.com, congyu@google.com

ABSTRACT

This paper studies the problem of *dynamic relationship and event discovery*. A large body of previous work on *relation extraction* focuses on discovering predefined and static relationships between entities. In contrast, we aim to identify temporally defined (e.g., co-bursting) relationships that are not predefined by an existing schema, and we identify the underlying *time constrained* events that lead to these relationships. The key challenges in identifying such events include discovering and verifying dynamic connections among entities, and consolidating binary dynamic connections into events consisting of a set of entities that are connected at a given time period. We formalize this problem and introduce an efficient end-to-end pipeline as a solution. In particular, we introduce two formal notions, *global temporal constraint cluster* and *local temporal constraint cluster*, for detecting dynamic events. We further design efficient algorithms for discovering such events from a large graph of dynamic relationships. Finally, detailed experiments on real data show the effectiveness of our proposed solution.

Categories and Subject Descriptors

H.0 Information Systems [**General**]

General Terms

Algorithms, Experimentation

Keywords

event discovery, dynamic events, entity relationships

1. INTRODUCTION

Relation extraction is the task of identifying relations between entities[1] from unstructured or semi-structured data sources [1, 10]. Traditionally, studies on relation extraction have been largely focused on identifying a predefined set of relations or identifying statistical associations between instances of predefined attributes [1, 4, 10]. Examples of such relationships include *CEOs managing companies* and *actors acting in movies*.

However, such approaches are not applicable for discovering relationships that can not be predefined. For example, majority of the relationships associated with news events are often hard to describe, let alone define, as the following example shows.

EXAMPLE 1.1. *On March 1, 2009, the US Coast Guard reported that a 21-foot fishing boat was missing off the Gulf Coast near Clearwater Pass, Florida. The boat was reportedly carrying four passengers, including Corey Smith, a former North Carolina State football player; Marquis Cooper, a member of the Oakland Raiders; as well as Nick Schuyler and Will Bleakley, former University of South Florida football players. The search and rescue effort started on March 1 and ended a day later when Schuyler was discovered clinging to the overturned boat and the others gone missing. The incident stayed in the national news media for many days because of the victims' connection with the NFL.* □

Clearly, the four players are deeply connected due to their involvement in this boating incident. However, the conventional approaches fail to detect such connections between people because such relationships are not of a predefined type; moreover, the relationship exists primarily due to the boating incident. Our goal is to identify such *dynamic* relationships that are formed due to such real-world events.

Detecting such dynamic relationships and associated events poses multiple interesting technical challenges. First, as discussed before, these relationships *do not conform to any pre-existing schema* and therefore can not be discovered by leveraging language patterns as previous works on static relationship extraction often do. Second, the underlying events often have a *flexible timeline* that is hard to know *a priori*. For example, the boating incident event lasted for a bit over a week, while the scandal involving the US golfer, Tiger Woods, lasted over several months. Third, the entities display a great deal of *flexibility in their participation* in the underlying events, mainly reflected in: (1) the number of participants (some events can involve tens of entities while others are between two entities), and (2) the degree of involvement of participants (some entities participate fully in the event the whole time while others may drop out earlier or join later and only participate in an indirect way). For example, Nick Schuyler, being the sole survivor, is involved in the incident longer and to a greater degree than the others. Finally, similar to traditional relation extraction, noise is inherent in many steps of the discovery process, including entity instance identification, relationship discovery, and event detection.

*Work performed while CY was at Yahoo! Labs, New York, NY

[1] The term 'entity' is used broadly to refer to any real world concepts such as a person, a company, or a movie.

WSDM'11, February 9–12, 2011, Hong Kong, China.
Copyright 2011 ACM 978-1-4503-0493-1/11/02 ...$10.00.

This paper presents the DROP system, which addresses the aforementioned challenges. Briefly, DROP employs the following basic steps for relationship and event discovery. Given a time period and a set of entities, we leverage existing burst detection techniques [6] for discovering pairs of entities that are buzzing around the same time period. These pairs serve as candidate relationships. After verifying the validity of the pairwise relationships, we design event consolidation algorithms to group pairwise relationships into a set of final events, often consisting of more than two entities. The event consolidation process takes into account temporal constraints between participating entities. Finally, DROP enriches each detected event with auxiliary information such as entity involvement score, event confidence, description, and popularity. We formally study each step in the DROP pipeline in the subsequent sections. In summary, we make the following main contributions:

- We introduce and formalize the problem of dynamic relationship and event discovery (Section 2).
- We present techniques for an efficient pipeline for discovering and verifying dynamic relationships between entities (Section 3).
- We introduce two temporal-constraint-based formalisms for defining the notion of dynamic events based on consolidating dynamic relationships, which are able to capture the diverse nature of dynamic events. We further provide efficient algorithms for identifying these events and enriching them with additional attributes. (Section 4)
- We evaluate our DROP system on real world data and demonstrate the effectiveness of the system. (Sections 5 and 6).

Finally, we describe related works and conclude in Sections 7 and 8, respectively.

2. PROBLEM STATEMENT AND OVERVIEW OF OUR APPROACH

We start by introducing some basic definitions and the overall problem statement (Section 2.1). We then describe a data structure, *entity dynamic relation graph*, which is core to our solution (Section 2.2). Finally, we provide an overview of our system and techniques (Section 2.3), which are built on in the rest of the paper.

2.1 Definitions and Problem Statement

As mentioned in Section 1, our goal is to discover events representing *dynamic relationships* among entities. In this work, instead of starting from scratch, we assume that a list of known entities and their variant names are given[2]. Specifically, an **entity** is an instance of a real world concept. Each entity may be represented by a set of textual phrases $\{n_1, n_2, ..., n_k\}$, each referring to a *variant* of the entity. For instance, the entity 'IBM' is an instance of company and may be represented as {'IBM', 'IBM Corp.', 'IBM Corporation'}.

Entities can appear in various data sources, such as search engine query logs, news and blog articles, and short status updates (e.g., twitter feeds). We consider a **data source** to be a sequence of time-stamped documents, where each document contains textual information. Some examples of documents are a single user query in the query logs or a tweet in a twitter feed.

DEFINITION 2.1. **[Entity Appearance]** *Given an entity* $e = \{n_1, n_2, ..., n_k\}$ *and a document* d *in data source* D*, we say* e appears *in* d *if* $\exists n_i \in e, d$ *contains* n_i*. We use* D_e *to denote the sequence of documents in* D *that* e *appears in. Furthermore, we use* D_e^t *to denote the set of documents in* D_e *within time window* t*.* □

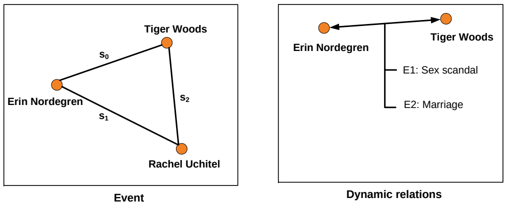

Figure 1: Sample event and dynamic relation

We note here that detecting the appearances of an entity within a document is by itself a challenging problem attracting many research efforts [5]. In this paper, we assume the set of variant names for the entity is complete and focus on our core problem of dynamic relationship and event discovery. The *dynamic relationship* between two entities is defined by their appearances in the given data sources. Formally, we have:

DEFINITION 2.2. **[Dynamic Relationship]** *Given a set of data sources* $\mathcal{D}$*, and two entities* e_1 *and* e_2*, we say* e_1 *and* e_2 *are* dynamically related*, denoted as a 4-tuple* $r = \langle e_1, e_2, t, s\rangle$*, where* t *is a time window and* s *is a score indicating the strength of the connection, if* $\exists D \in \mathcal{D}, D_{e_1}$ *and* D_{e_2} *are* temporally connected *during time window* t*.* □

While the notion of *temporal connection* will be formalized in Section 3, we provide an intuitive description here. When the data source is the query log, two entities are more likely to be dynamically related if they are bursting (i.e., appearing in more than usual number of documents) around the same time. When the data source is the news archive, two entities are more likely to be dynamically related if they appear close together in a large number of documents in the given time period. We emphasize again that relationships discovered this way are fundamentally different from static, predefined, relationships that can be extracted based on language patterns. The latter relationships are generally persistent and time-independent, such as a spousal relationship. The former type of relationships, however, are time-sensitive and often a product of underlying real world events, as formally defined next.

DEFINITION 2.3. **[Event]** *A* dynamic event v *is represented as a 3-tuple* $\langle E, R, t\rangle$*, where* E *is a set of entities,* $R \subseteq E \times E$ *is a set of dynamic relationships,* t *is a continuous time window. Furthermore,* $\forall e \in E, \exists e' \in E$ *s.t.* $(e, e') \in R$ *or* $(e', e) \in R$*.* □

We illustrate the above concepts using another example:

EXAMPLE 2.1. *Consider the real-world event involving 'Tiger Woods' that started on 'November 27th, 2009.' Prior to this event, typical associations for 'Tiger Woods' were 'Phil Mickelson' or some other athletes. The break of the scandal quickly altered what information users are searching on him and the media coverage of him. Indeed, we can notice that entities such as 'Erin Nordegren' and 'Rachel Uchitel' are found to be bursting together with 'Tiger Woods' and appear in the same new articles. Based on those cues, we can identify the dynamic relationships between those entities and uncover the underlying dynamic events.* □

Figure 1 pictorially depicts the event from the above example. Note that 'Tiger Woods' and 'Erin Nordegren' are statically related through the spouse relationship (a static relationship) as well, but the example relationship occurs as a result of the sex scandal. Finally, we define two scores typically associated with a dynamic event.

[2] Such a list can be obtained using many existing entity recognition techniques [5] and publicly available extraction tools such as `http://opencalais.com`.

DEFINITION 2.4. **[Entity Involvement]** *The entity involvement of an entity e for an event v, denoted as $I(e, v)$, is a score measuring the level of participation that e has in v.* □

An intuitive way to compute involvement score is to use the strength of the relationship between entity e and every other participating entity. Section 4.3 rigorously provides a method for entity involvement computation.

DEFINITION 2.5. **[Event Confidence]** *The event confidence of an event v, denoted as $C(v)$, is a score that measures the closeness of the entities involved in v, and therefore captures the overall confidence in the actual occurrence of v.* □

Informally, event confidence is a measure of the likelihood of the set of all participating entities being dynamically connected during the particular time interval. Section 4.3 formalizes this intuition to provide a concrete method to compute the event confidence.

We now formalize the Dynamic Relationship and Event Discovery Problem.

PROBLEM 2.1 (PROBLEM DEFINITION). *Given a set $\mathcal{S}$ of entities, a set $\mathcal{D}$ of data sources, and a time period $\mathcal{T}$, discover the set of dynamic events V, such that each $v \in V$ is a dynamic event $(\langle E, R, T \rangle)$, where $v.E \subseteq S$, $v.R \subseteq (v.E \times v.E)$ is a set of dynamic relationships based on $\mathcal{D}$, and $v.T$ is a time interval within $\mathcal{T}$. Furthermore, for each event $v \in V$, compute $C(v)$, and $I(e, v)$ for each entity $e \in v.E$.*

At the core of this problem are the challenge of discovering and determining the dynamic relationships among the entities based on the given data sources and the challenge of consolidating those relationships into semantically meaningful events. The main contributions of our work is the design and implementation of a principled, end-to-end, pipeline for addressing these challenges. We start by describing the basic data structures used in our approach in the next section, followed by presenting our techniques in detail.

2.2 Entity Dynamic Relation (EDR) Graph

DEFINITION 2.6. **[Entity Dynamic Relation Graph]** *An* entity dynamic relation (EDR) graph *$G = (V, E)$ is an undirected graph consisting of a set V of nodes representing the entities and a set E of edges representing the dynamic relationships among the entities.* □

Intuitively, the EDR graph is a collection of dynamic relationships discovered from the data sources, and from which the set of dynamic events can then be extracted. As a result, the overall problem can be decomposed into two main sub-problems: *dynamic relationships detection*, which constructs the EDR graph based on the given data sources, and *dynamic event identification*, which extracts subgraphs from the EDR graph to form dynamic events. We provide a brief overview of both steps in the next section.

2.3 Overview of Our Approach

Figure 2 illustrates an architecture diagram of our approach, which shows the two main phases. During the *dynamic relationship detection* phase, we leverage multiple data sources to discover and corroborate pairs of entities that are temporally connected. We design *peak detection techniques* and *relationship corroboration techniques* for this phase. The EDR graph can then be formed using the collection of dynamic relationships. Based on this EDR graph, the second phase, *dynamic event identification* then extracts the dynamic events. More specifically, we introduce formal definitions of dynamic events based on the EDR graph and design *event identification algorithms* to efficiently extract the dynamic events with temporal constraints.

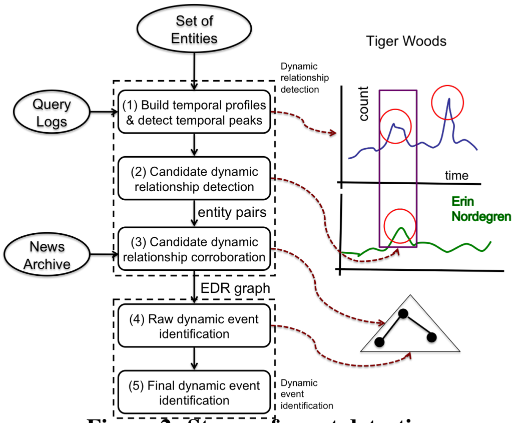
Set of Entities
Query Logs
News Archive
(1) Build temporal profiles & detect temporal peaks
(2) Candidate dynamic relationship detection
entity pairs
(3) Candidate dynamic relationship corroboration
EDR graph
(4) Raw dynamic event identification
(5) Final dynamic event identification
Dynamic relationship detection
Dynamic event identification
Tiger Woods
count
time
Erin Nordegren

Figure 2: Stages of event detection

3. DYNAMIC RELATIONSHIP DETECTION

In this section we elaborate on each step of the dynamic relationship detection phase described in Figure 2: *temporal profile generation and peak detection* (Section 3.1), *candidate dynamic relationship detection* (Section 3.2), and *dynamic relationship corroboration* (Section 3.3).

3.1 Profile and Peak Generation

For each given entity, we leverage the available data sources and count the number of documents in which the entity appears during each time window to construct the temporal profile. We then design *peak detection techniques* to locate time periods when the entity is bursting (i.e., appears in more documents than usual). Formally, we have:

DEFINITION 3.1. **[Temporal Raw Profile]** *Given a data source D and a time range T consisting of equal-sized non-overlapping time windows $(t_1, t_2, \ldots, t_k)$, the* temporal raw profile *of an entity e is $T_e = \langle c_{t_1}, c_{t_2}, \ldots, c_{t_k} \rangle$, where each $c_{t_i} = \Sigma_{d_i} I(e, d_i)$, where the indicator function $I(e, d_i)$ evaluates to 1 if $(d_i \in D) \wedge (d_i.timestamp \in t_i) \wedge (e \in d_i)$. I.e., c_{t_i} measures the number of documents mentioning e within the time window t_i.* □

This raw profile can be derived from many different sources such as the query log (where each d is a search query) or twitter feeds (where each d is a tweet). On the other hand, we choose not to use new articles as the source for raw profile generation because queries and/or tweets contain much more entity occurrences than news articles do, and they reflect the interests of the general Web users.

While many previous studies [6] use time series analysis to discover entities with similar temporal profiles, we are only interested in entities that are connected within certain time windows because some underlying events have likely happened. Therefore, we have:

DEFINITION 3.2. **[Temporal Peaking Profile]** *Given a temporal raw profile of an entity e, $T_e = \langle c_{t_1}, c_{t_2}, \ldots, c_{t_k} \rangle$, the* temporal peaking profile *of e is $P_e = \langle p_{t_1}, p_{t_2}, \ldots, p_{t_k} \rangle$, where p_{t_i} is 1 if e is peaking during time window t_i and 0 otherwise.* □

We adopt two main peak detection strategies: **Rapid Rising** and **Rise and Fall**, extended from previous techniques [8]. For a given time window t within a raw profile T_e, both strategies compare the count, c_t, with the counts of surrounding windows in T_e. In Rapid Rising, the leading k windows are compared, and p_t is set to true iff c_t is more than l standard deviations away from the mean count of those leading windows. In Rise and Fall, both leading and trailing k windows are compared, and p_t is set to true iff c_t is more than l standard deviations away from the mean count of both leading and trailing windows. Intuitively, the Rapid Rising strategy captures majority of the dynamic events, including those news stories that stay in the headline for quite some time. The Rise and Fall strategy, on the other hand, focuses on identifying the short-spiking stories such as a wedding or divorce story. Finally, all entities without a single peak are removed from further consideration.

3.2 Candidate Dynamic Relationship Detection

The peaks in the entity temporal profiles provide initial indicators of possible connection between entities. Specifically, given the peaking profiles of two entities, the candidate dynamic relationship detection step checks whether the two entities are potentially connected. We adopt a simple strategy called **Co-Peaking**, where a candidate dynamic relationship is generated if a consecutive sequence of peaking time windows are shared by two entities. Formally, we have:

DEFINITION 3.3. **[Co-Peaking Dynamic Relationship]** *Given two entities e_1 and e_2, a* co-peaking dynamic relationship *exists between them, denoted as $r_{cp} = \langle e_1, e_2, t, s\rangle$, where t is a time window, iff $P_{e_1}.p_t = 1 \wedge P_{e_2}.p_t = 1$, and s is a connection strength between e_1 and e_2.* □

We make a few comments about co-peaking relationships. First, although the connection strength s is part of the definition of dynamic relationships, it is not computed using the temporal profiles; we shall get to the computation of s shortly. Second, two entities can have multiple dynamic relationships for multiple time windows at this stage. These relationships may be consolidated before event identification based on overall strength of all pairwise connections. Third, we note that Co-Peaking is one among multiple possible approaches (albeit a simple yet very effective one), that can be used to detect candidate dynamic relationships.

3.3 Dynamic Relationship Corroboration

The candidate relationships detected by the Co-Peaking strategy can connect two entities. However, peaking at the same time does not always mean that the entities are indeed related: they may bursting simply coincidentally. We further adopt a **Co-Occurrence** based strategy, which leverage the news archives, to corroborate and score the strength of the the connection. We may have used other sources, such as query logs again, but in general news articles provide richer information (namely, text) that is useful in corroboration. Formally, we have:

DEFINITION 3.4. **[Co-Occurrence Based Dynamic Relationship Corroboration]** *Given a data source D and a dynamic relationship between two entities, $r_{cp} = \langle e_1, e_2, t, s\rangle$, where connection strength s has yet to be computed, the* co-occurrence based corroboration *computes s as follows:*

1. *Let $f(e_1)$ measure the number of documents in which e_1 appears, and whose time stamps are within t, denoted as $|D^t_{e_1}|$;*
2. *Similarly, let $f(e_2) = |D^t_{e_2}|$;*
3. *let $f(e_1, e_2) = |D^t_{e_1 \wedge e_2}|$, i.e., the number of documents e_1 and e_2 both appear in;*
4. $s = \frac{f(e_1,e_2)}{f(e_1)\cdot f(e_2)}$ □

Intuitively, we derive entity co-occurrence statistics from textual data sources based on the intuition that entities involved in the same event will be mentioned together in the same document such as a news article. The specific formula we adopt here is the point-wise mutual information (PMI) score [14] between e_1 and e_2. For the purpose of corroboration using news archive, we generate fixed-sized text segments for each article and treat a segment as a document. After corroboration, we combine the remaining dynamic relationships to form the EDR graph. The EDR graph is then consolidated to from the Pairwise Temporal Graph (PTG), from which the events are identified. We describe this process next.

4. EVENT CONSOLIDATION

In this section, we describe how to derive holistic dynamic events from the dynamic relationships we have generated, a process we call *event consolidation*. Specifically, our goal is to combine multiple binary relationships and the entities involved in those relationships, whenever they are *temporally consistent*, into a single n-way event that represents a real world event. The main challenges are: (1) formalize the notion of temporally consistent, i.e., determine when multiple pairwise relationships can be consolidated, (2) event identification: efficiently identify such temporally consistent dynamic events from a potentially large relationship graph, (3) event enrichment: enrich the event with estimations of how likely the event is a true event (event confidence score), how likely the entity truly belongs to the event (entity participation score) and appropriate keyword descriptions for the event.

We introduce the basic notion of *Pairwise Temporal Graph* in Section 4.1. In Section 4.2, we motivate and introduce the two formal definitions of dynamic events based on entity clusters, *global temporal constraint cluster* and *local temporal constraint cluster*, in Sections 4.2.1 and 4.2.2, respectively. Finally, we briefly discuss techniques for enriching each event with aforementioned information in Section 4.3.

4.1 Event Consolidation Basics

We first introduce the notion of Pairwise Temporal Graph, which combines Definitions 2.6 and 3.4, and is the raw input to the event consolidation process.

DEFINITION 4.1. **[Pairwise Temporal Graph]** *A pairwise temporal graph (PTG) $G = (V, E, W)$ is an extended EDR graph consisting of a set V of vertices, a set $E \subseteq V \times V$ of edges, and a function $W{:}E \rightarrow 2^N$ that associates each edge with a set of time windows; N denotes the finite set of time windows of which the data is present.* □

Intuitively, each edge represents one or more corroborated dynamic relationships and is annotated with the set of time windows for which a corroborated dynamic relationship is observed between the two entities. We use the notation $W(e) \subseteq N$ to denote the set of time windows[3] associated with $e \in E$. It is easy to see that the PTG graph can be constructed by creating an edge between two entities if they share a dynamic relationship in at least one week, to create the basic EDR graph; and then annotate each edge with the set of weeks that a dynamic relationship exists for the two involved entities.

We consider a dynamic event as a cluster of entities in the PTG graph, along with the edges induced by those entities. However,

[3] Note that, while our techniques apply to any granularity of event consolidation, our specific data and experiments are based on weekly time windows. Henceforth, we use weeks to represent time windows.

how to define the formation of the cluster is not an easy task. To understand the difficulties involved, let's take another look at the event from Example 2.1.

EXAMPLE 4.1. *The 'Tiger Woods Scandal' received world wide media attention for several weeks. From techniques presented in Section 3, we are able to obtain pairwise dynamic relationships between Tiger Woods (TW) and his wife Erin Nordegren (EN), and TW and his mistress Rachel Uchitel (RU). The relationship between EN and RU, however, were not detected with sufficient confidence as they are not co-occurring often in news articles. Furthermore, RU was not event part of the story until a few weeks later as the story unfolded and her identity was revealed. Despite all those uncertainties, we would still like to cluster all three into the same event. Such issues can become even more prominent for events involving a much larger number of entities such as 'Wimbledon 2009', where we expect to find relationships between multiple participating individuals, but obviously not for every pair of players.* □

One extreme approach for event consolidation is to find *maximal cliques* within the graph, with the additional constraint that all edges share the same week. This approach can lead to almost certain events, but it will also likely only discover partial events, as the example illustrates. Another extreme approach is to simply find *connected component* within the graph with no additional temporal constraints. This approach, however, will likely lead to multiple semantically separate events being merged together and therefore produce *very large* events that says little about the real world events. We call the latter the `tarjan` approach, named after the famous connected component discovery algorithm [13]. We adopt the middle ground and aim to detect event boundaries via carefully defined temporal constraints, which will be discussed in details in Sections 4.2.1 and 4.2.2.

4.2 Event Identification

This section formally describes techniques for aggregating pairwise relationships into holistic dynamic events. Note that temporal constraints are the main factor that prevents us from directly applying simple connected component detection algorithms. We describe two formal holistic event definitions: one based on connected components satisfying a global temporal constraint, and the second based on local temporal constraints on adjacent edges. The main objective of these notions of connected components is to be more "flexible" in terms of missing temporal connections allowed for pairs of entities in a given event. For instance, in an event with three participating entities, it is possible that one pair of entities had a pairwise relationship due to co-peaking and co-occurrence in a specific week t_w, while a second pair displayed a relationship in week t_{w+1}. In the following, we will formalize such flexibilities and introduce efficient algorithms for computing the event clusters.

4.2.1 Global Temporal Constraint

The global definition of events captures the intuition that any event may span multiple weeks, up to a specific number K weeks, and therefore tolerates clusters of nodes and edges as long as there edges with weeks that are not separated by more than K weeks.

DEFINITION 4.2. **[GTC Cluster]** *Given a PTG* $G = (V, E, W)$, *and a maximum week different parameter* $K \in \mathbb{N}$ *we say that* $C \subseteq V$ *is* global-temporally constrained cluster *(GTC cluster) if and only if all the following hold:*

1. **Connected:** $\exists E_C \subseteq E$ *and* $\exists w_{min}, w_{max}$, $0 \leq w_{max} - w_{min} \leq K$ *such that:*
   - *(a) The graph* $G_C = (C, E_C)$ *is connected.*
   - *(b)* $\forall e \in E_C : \exists w : w_{min} \leq w \leq w_{max}$ *and* $w \in W(e)$.
2. **Maximal:** $\nexists C' \supset C$ *satisfying (1) above.*

□

```
Require: Graph $G(V, E, W)$, max week difference parameter $K$,
    and min week number for each cluster $w^*$.
Set $V^* = \emptyset$ and $E^* = \emptyset$
for each $e \in E$ do
    if $(W(e) \cup [w^*, w^* + K]) \neq \emptyset$ then
        $E^* = E^* \cup \{e\}$
        $V^* = V^* \cup \{v_1, v_2\}$, where $e = \{v_1, v_2\}$
    end if
end for
Find set $\mathcal{C}$ of all connected components in $G^* = (V^*, E^*)$
Set $\mathcal{C}_{ret} = \emptyset$.
for each cluster $c \in \mathcal{C}$ do
    if exists edge in $\mathcal{C}$ containing week $w^*$ then
        $\mathcal{C}_{ret} = \mathcal{C}_{ret} \cup \{c\}$
    end if
end for
Return $\mathcal{C}_{ret}$
```

Algorithm 1: Algorithm to find all GTC clusters with minimum week number on the cluster being a fixed w^*.

For the special case k is 0, every single edge must contain the same week, which is equivalent to finding connected components on a week-by-week basis. Note that for the purpose of the discussion in this section, we do not consider specific related keywords corresponding to each edge; filtering of connected components based constraints in terms of overlap in keywords follows the same argument as the filtering in terms of overlapping weeks.

We address the following problem in this section:

> Given a PTG $G = (V, E, W)$ and parameter K, find *all* GTC clusters of G w.r.t K.

To solve the above problem, we exploit the straight forward observation that for any two edges e_1 and e_2 connecting the cluster, we cannot have that $\forall w_1 \in W(e_1), w_2 \in W(e_2)|w_1 - w_2| > K$. We can therefore proceed to find each GTC cluster based on the minimum week number on each cluster. Algorithm 1 gives an algorithm for finding all GTC clusters with w_{min} from Definition 4.2 being an input w^*: We first restrict the graph to only contain relevant edges, and then find connected components with the restricted graph. Finally, we output the set of clusters that have at least one edge containing the input minimum week number w^*.

A simple sequential algorithm to find all GTC clusters is to run Algorithm 1 for all $w^* \in [W_{min}, W_{max}]$, where $[W_{min}, W_{max}]$ gives the entire week interval for the data-set. However, we note that the finding GTC clusters for each w^* can easily parallelized in a distributed environment: In each computing node, we simply send the set of edges relevant for the specific w^*, and all computing nodes can independently find GTC clusters. Therefore, in a distributed environment, the overall complexity of finding all GTC clusters is equal to that of finding a single GTC cluster. Further, with a simple index we can find all edges relevant for a single GTC cluster, so the complexity of finding GTC clusters for a given w^* is equivalent to that of finding connected components in the smaller graph.

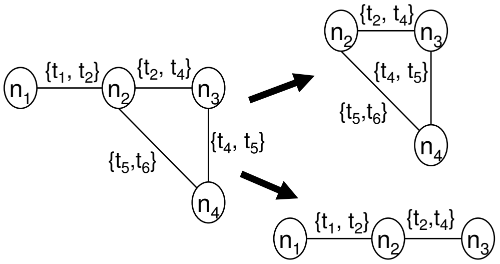

Figure 3: LTC Clusters.

4.2.2 *Local Temporal Constraint*

Contrary to the GTC, which caps the maximal span (i.e., number of weeks) of the events, the *local temporal constraint* (LTC) focuses more on the temporal continuity among the dynamic relationships. The intuition is the following: within a given event, if there is an entity participant that connects with two different other entities at two different time periods, then the three entities are more likely to be truly connected as a result of the event if the two time periods overlaps. Furthermore, an event spanning a period of multiple weeks is more likely to be a true event if each consecutive weeks within the period are supported by some dynamic relationships. These intuitions are formally defined as:

DEFINITION 4.3. **[LTC Cluster]** *Given a PTG* $G = (V, E, W)$*, we say that* $C \subseteq V$ *is a* local-temporally constrained cluster *(LTC cluster) if and only if all the following hold:*

1. *The graph* $G_C = (C, E_C)$ *is connected, and* $\exists T = (C, E_T)$*, a spanning tree of* G_C*, such that:*
2. **Sharing:** $\forall v, v_1, v_2 \in C$, if $e_1 = (v_1, v) \in E_T \wedge e_2 = (v, v_2) \in E_T$, then $W(e_1) \cap W(e_2) \neq \emptyset$
3. **Continuity:** $\exists w_{min}, w_{max}, w_{max} \geq w_{min}$, such that*:*
   (a) $\forall w_1, w_2 \in [w_{min}, w_{max}], |w_1 - w_2| \leq 1$, $\exists e \in E_T$, $w_1 \in W(e) \wedge w_2 \in W(e)$
   (b) $\forall e \in E_T, \exists w \in [w_{min}, w_{max}], w \in W(e)$
4. **Maximal:** $\nexists C' \supset C$ *satisfying (1) above.*

□

This LTC definition enforces that any two dynamic relationships involving the same entity must share a common week. Furthermore, it enforces that those dynamic relationships must together form a continuous time period and that each relationship must be covered by this time period as well. Figure 3 illustrates two LTC clusters (on the right) that can be detected from the input graph on the right. For the top cluster, the time period is $\{t_4, t_5, t_6\}$, and for the bottom cluster, $\{t_1, t_2\}$.

Algorithm 2 describes the pseudo code for discovering all LTC clusters based on the intuition that each LTC cluster is a merge of individual weekly clusters. The algorithm starts with two sets of connected component clusters formed purely based on edges in the first and second week, respectively. At each iteration, all clusters formed based on edges in 1 through (i-1)-th weeks are put together into $C_{w_{i-1}}$. The algorithm then merges a cluster from this group into a cluster formed based on edge in the next (i-th) week if the two clusters share a common edge. This merging should happen because this common edge ensures that the (i-1)-th and i-th weeks are continuously represented. For any cluster in $C_{w_{i-1}}$, if it can not be merged with any cluster in the i-th week, it can be safely returned. The existence of a common edge between two clusters can be checked in time linear to the number of edges in both clusters as long as the edges are sorted according to the nodes they connect.

```
Require: Graph $G(V, E, W)$.
Set $\mathcal{C}_{ret} = \emptyset$.
Let $\mathcal{C}_{w_i}$ be the set of connected components (clusters) generated based on edges containing the $i$-th week ($w_i$).
$i = 2$
{$M$ is the overall maximum week number in the system.}
while $i \leq M$ do
    for each cluster $c_j \in \mathcal{C}_{w_{i-1}}$ do
        merged = False
        for each cluster $c_k \in \mathcal{C}_{w_i}$ do
            if $c_j$ and $c_k$ share a common edge then
                $c_k = c_k \cup c_j$ {merge the two clusters}
                merged = True
            end if
        end for
        if not merged then
            $\mathcal{C}_{ret} = \mathcal{C}_{ret} \cup c_j$
        end if
    end for
end while
$\mathcal{C}_{ret} = \mathcal{C}_{ret} \cup \mathcal{C}_{w_M}$
Return $\mathcal{C}_{ret}$
```

Algorithm 2: Algorithm to find all LTC clusters.

There are a total of $|\mathcal{T}|$ iterations, where $\mathcal{T}$ is the full time period that we are considering, and each iteration costs $O(nM^2)$ where M is the maximum number of clusters for each week and n is the maximum number of edges for the clusters.

4.3 Event Enrichment

Finally, we consider enriching the events with *entity involvement score* and *event confidence score*, as well as descriptive information about the events, including *event description* and *event popularity*. In this work, we provide simple techniques for coming up with those enrichments. As we shall see in our experiments, such simple heuristics worked very well in our practical setting. Therefore, we provide brief descriptions of our approach for each type of enrichment, leaving more sophisticated techniques (if necessary) for future work.

Entity Involvement Score: For each event we discover, we provide individual scores for each entity as a measure of their overall participation in the event. These scores are determined using the number and strength of pairwise relationships within the event cluster, for each entity. Given an event cluster $v = (E, R, T)$ representing a cluster with entities E, and weighted edges R between every pair of entities (with absent edges being equivalent to edges being present with weight 0), the involvement score of an entity $e \in v.E$ is given by:

$$I(e, v) = \frac{\sum_{e' \in v.E, e' \neq e} s(e, e')}{|e.R| - 1}$$

where $s(e, e')$ gives the weight of the edge between e and e', obtained from Section 3 based on the number of relevant news articles and number of queries that mention the entities. One useful side effect of the involvement score is to show which entities are the "primary" participants in a event and which ones are the "secondary" participants.

Event Confidence: We provide a confidence score associated with the fact that we've correctly extracted an event consisting of the participating entities. To determine the confidence, we effectively want to know that the connected component we extracted was indeed meant to be connected. Therefore, we use an estimate of the probability of connectedness as our event confidence, with edge weights being the probabilities of individual edges being present. Given an event cluster $v = (E, R, T)$ as above, the confidence of v is computed as:

$$C(v) = \sum_{S \in 2^R} (Ind(v|_S) \times \Pi_{r \in S} s(r) \times \Pi_{r \notin S}(1 - s(r)))$$

where r is an edge in the event cluster, $s(r)$ is the weight of the edge, and $Ind(v|_S)$ is an indicator variable that is equal to 1 if the restriction $v|_S$ to the edges in S is connected, and 0 otherwise. The formula above adds the probabilities of every combination of inclusion/omission of edges that results in V being connected. Note that as written, the formula needs to enumerate all possible subsets to compute $C(v)$, but in practice this isn't necessary; as we employ multiple optimizations, details of which are omitted. For instance, we know that any combination of edges that is a superset of a spanning tree of C has to be connected; therefore, we don't need to enumerate all supersets of spanning trees. Likewise, we don't need to enumerate subsets of disconnected set of edges.

Event Description: We further aim to obtain a set of relevant keywords that describe an event. To accomplish this, we use the bag of words surrounding the mentions of the event entities in the documents obtained from in dynamic relationship corroboration (Section 3). We prune away noisy keywords and rank the remaining ones by standard TF-IDF and PMI measures.

Event Popularity: Finally, we are interested in an (approximate) measure of event popularity, which allows us to globally rank events, perhaps restricted to a specific (set of) entities. Note that event confidence refers to the confidence our system has of having correctly produced an event, while, event popularity refers to ranking events based on external importance assuming the event is correct. For this study, we use the total query volume generated by this event as the primary feature. We use query logs to determine the number of times keywords in the event description as well as queries for each participating entity appear within the time period of the event. We then aggregate these counts weighted by the relevance of each keyword and involvement score of each entity to obtain the total event popularity.

5. EXPERIMENTAL SETTINGS

We now present experimental evaluation of our DROP system. Although the techniques presented in this paper are applicable to a wide variety of entity types, our experiments primarily focus on people as entities. We begin by describing our data collection methods and evaluation metrics. Later in Section 6, we discuss our experimental results.

Query logs: We collected a random sample of 100 million fully anonymized queries sent to a major web search engine[4] in 2009 and first three months of 2010, along with their frequency. We used the first three months of 2009 as a development set and report the results for the remainder of the query logs.

Text corpus: To compute the co-occurrence statistics for a pair of entities (see Section 3.3), we used a collection of news articles from 2009 and 2010.

Seed entities: To bootstrap DROP with a dictionary of entities, we used Wikipedia[5]. Specifically, we examined the page category for each entity and identified the subset of entities involving people. This resulted in a total of 30827 entities.

User studies: All user studies and manual annotation tasks described in this Section were performed by a group of three evaluators experienced with assessing the relevance and quality of queries.

Techniques to compare: We build two techniques based on the `GTC` and `LTC` algorithms in Section 4. We build a third algorithm as our baseline, denoted as `TRJ`. As the name suggests, `TRJ` uses the simple time-agnostic Tarjan's Algorithm [13] for event clustering. Note that all three approaches share the temporal profiling, peak detection, and co-occurrence techniques presented in Section 3. The main difference in these approaches is the generation of holistic events from the entity dynamic relationship graph.

Evaluation approach: We ran two forms of quantitative evaluation: (a) *entity-based* and (b) *list-based.* (Later in Section 6.6, we discuss a qualitative analysis of the performance of DROP.) The entity-based evaluation focuses on evaluating the performance of DROP for a given set of entities, whereas, the list-based evaluation examines performance for the entire output generated by various DROP techniques. For our entity-based evaluation, we draw a random sample of 30 entities from our entity list and derive precision and recall values using a gold-set constructed as described below. Since we may have multiple values for precision and recall for each gold event, we note the lower-bound as well as the upper-bound on these values. For our list-based evaluation, we draw a random sample of 35 events from the results produced by each technique. For each event, we compute the precision and recall using a gold-set constructed as described below.

Gold-set generation: To the best of our knowledge, there is no existing Web resources for a comprehensive list of the dynamic events (e.g., new events) that we are focusing on. Manually examining all the entities in our dictionary is prohibitively expensive. For our entity-based evaluation, we build a "gold-set" consisting of perfect real-world events that occurred for each entity in our sample in the follow way. Given an entity e, annotators were requested to examine various web-based sources (e.g., Wikipedia, official home pages, news search, and web search, in that order) to identify all events involving e, and for each discovered event, record (a) the time period for the event, and (b) all other entities involved in the event. This resulted in a gold-set of events consisting of records $\langle \{e_1, e_2, \cdots, e_n\}, T \rangle$ indicating that an event involving entities $e_1, \cdots, e_n$ occurred at time period T.

For list-based evaluation, the gold-set is generated as follows. For each event in the sample, the annotators are requested to look at all entities in the event and identify a real-world event that best explains the co-occurrence of these entities for the specified time period. To create a gold event, annotators identified the subset of entities within the discovered event that actually participated in the real-world event. As our final gold-set, we use the union of the gold events generated for all three techniques, which consists of a total of 86 events.

Evaluation metrics: To quantitatively measure the effectiveness of the various techniques, we adapt standard information retrieval evaluation metrics, namely, *precision* and *recall*. Precision and recall for both our evaluations are computed as follows: Consider an event $V_g = (E_g, T)$ in the gold-set where E_g is the set of entities involved in the event that occurred at time period T. For each gold event, we identify DROP events that (a) occurred at T, and (b) involve at least one of the entities in E_g. Specifically, for V_g, we identify the set of DROP events $\{V_{d_i} = (E_{d_i}, T), i \in \mathbb{N}\}$ such that $E_{d_i} \cap E_g \neq \emptyset$. Note that, we would ideally like DROP to identify only one event per gold event. For each pair of gold event $V_g = E_g, T$ and identified event $V_{d_i} = E_{d_i}, T$, we compute the precision and recall as:

[4] `http://search.yahoo.com/`

[5] `http://www.wikipedia.org/`

$$\text{Precision(V)} = \frac{|E_{d_i} \cap E_g|}{|E_{d_i}|} \quad (1)$$

$$\text{Recall(V)} = \frac{|E_{d_i} \cap E_g|}{|E_g|} \quad (2)$$

If there are multiple DROP events matching a given gold event, the final precision and recall measure for the gold event is computed as the average across all those events.

6. EXPERIMENTAL RESULTS

We now report the results of our experiments. We begin by comparing basic statistics pertaining each consolidation algorithm and then present our two-fold experiments, namely, list-based and entity-based evaluation.

6.1 Basic Statistics

Table 1 presents basic statistical information about each event generation process. We notice that TRJ generates far fewer events (an order of magnitude fewer) than those generated by LTC and GTC: This is because TRJ is time-agnostic, and therefore, brings together entities that share any relationship, irrespective of when they co-peaked and co-occurred. In particular, TRJ results in extremely large events[6] that combine many small events from different time periods. As a consequence, the average size of the TRJ events is significantly larger than the size of actual events (typically fewer than 10 entities), as observed using GTC and LTC.

Method	#Events	Average size
GTC	1288	7.22
LTC	1380	4.88
TRJ	120	24.42

Table 1: Basic statistical information for event generation approaches for our entire dataset.

6.2 List-Based Evaluation

Table 2 reports the precision and recall for events obtained using GTC, LTC, and TRJ. The precision and recall metrics in the table are as defined in Section 5. Time-constrained algorithms GTC and LTC outperform TRJ in terms of both precision and recall: both GTC and LTC exhibit ∼21% higher precision than that for TRJ, and 24% and 47% higher recall, respectively, than that for TRJ.

Method	precision	recall
LTC	0.468 ± 0.17	0.438 ± 0.16
GTC	0.466 ± 0.15	0.519 ± 0.16
TRJ	0.387 ± 0.16	0.354 ± 0.15

Table 2: Average precision and recall for various methods for our list-based evaluation.

We further observe that our experimental setup favors TRJ and, in practice, the recall value for TRJ can be further lower than reported. As discussed in Section 5, the recall values were generated using a manually generated gold-set. If a “true” gold-set consisting of all perfect real-world events[7] were used, TRJ would result in a lower precision since it generates relatively large events and also result in lower recall since it generates far fewer events. This can be further seen from Table 1 where the number of events generated by GTC and LTC are an order of magnitude more than those generated by TRJ.

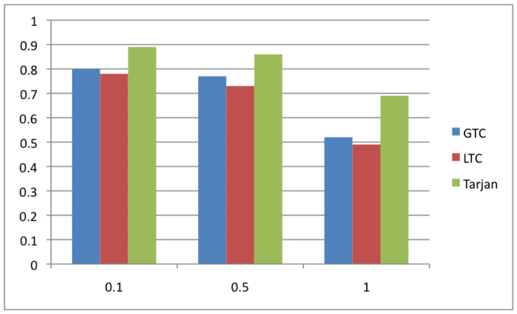

Figure 4: Overall recall for GTC, LTC, and TRJ, for varying overlap threshold.

6.3 Entity-Based Evaluation

We now discuss results from entity-based evaluation, which “drills” deeper to understand better the effectiveness of various algorithms at detecting events for individual entities. For these experiments, we adopt the *overlap threshold* when matching gold events with DROP events. Specifically, so far for each gold event, we identified an event generated by our algorithms if the identified event had *at least* one entity in common with the gold event. We now generalize this matching criteria to incorporate the overlap threshold, which is defined as the fraction of entities in the gold event the two events must share to be considered a match. (The original criteria can be considered as having an overlap threshold of greater than 0.)

We begin by first examining overall recall values for each algorithm computed as follows. For each gold event, we consider it being *identified* if there is at least one DROP event produced by our algorithms that matches it given the overlap threshold. The recall is then computed as the percentage of gold events that are identified against the entire set of gold events. Figure 4 illustrates the results for varying threshold. Interestingly, TRJ, exhibits the highest recall among the three algorithms. This is due to the fact that TRJ generates large events that contain several entities which are more likely to match any gold event (see Table 1). However, it should be noted that the TRJ events matched are often undesirable in practice since they contain many more other unrelated entities as well.

Figures 5 and 6 show the average precision and recall, respectively, computed using Equations 1 and 2, respectively. As an interesting exception to our expectation, TRJ resulted in precision values higher than that for GTC and LTC. To gain insights into this, we closely looked at the pairs of a gold event and a TRJ event, as compared to the other algorithms. Interestingly, TRJ generates a few (one or two) large events and the remaining events are relatively small in size. While the precision for cases where large events were matched is small, many other cases had smaller matching events produced by TRJ, which resulted in perfect or close to perfect precision (note that the denominator in Equation 1 is the size of the TRJ event). The precisions are aggregated across all matched events, and the slightly higher precision value is observed due to the influences of small events in TRJ.

[6] Some of those involve well over 1000 entities and are clearly not reflecting real world events.

[7] Obviously, attempting to create a true gold-set requires prohibitive amount of manual effort, and hence wasn’t carried out in this paper.

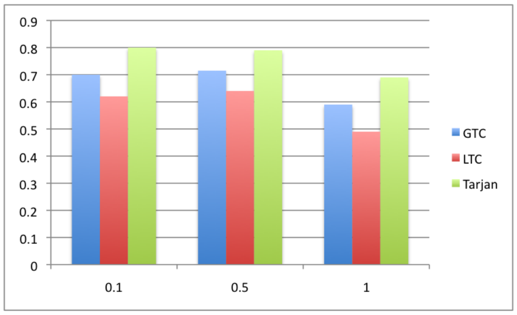

Figure 5: Precision of GTC, LTC, and TRJ over entire gold set, varying match threshold

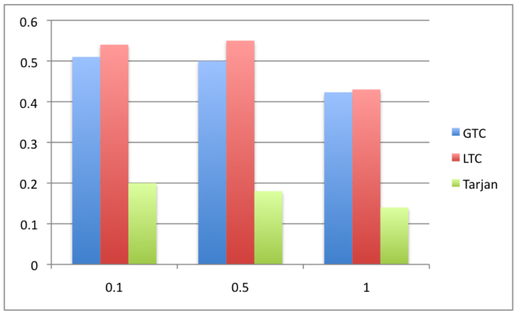

Figure 6: Average recall of GTC, LTC, and TRJ over entire gold set, varying match threshold

6.4 Performance of Peak Detection

A fundamental step of our system is the peak detection techniques, which we evaluate next. In particular, the main objective of our experiment is to examine the performance of the simple technique used in our approach (Section 3). For this experiment, we randomly sampled 10 entities and manually identified all peaking time periods for each of these entities. We then compared the peaks generated by our Rapid Rising technique against these manually generated peaks, allowing a disparity of ±3 weeks to account for delayed news availability, or delayed popularity rise, of peaking entities. Specifically, we compute *precision*, i.e., fraction of identified peaks that are in the manual set, and *recall*, i.e., fraction of peaks in manual set identified by our algorithm. Our algorithms obtained a **precision of** 0.78 and **recall of** 0.82, thus demonstrating that simple peak detection techniques suffice for determining candidates pairs of co-peaking and co-occuring entities.

6.5 Impact of Corroboration by Co-Occurrence

Finally, we studied the impact of parameters used by the co-occurrence based corroboration technique on the quality of the events generated. In particular, we varied the threshold of the co-occurrence scores (see Section 3) and ran the entity-based evaluation. For each threshold, we ran the consolidation algorithms, LTC and GTC and examined the overall recall, i.e., fraction of entities for which an event was identified, and average recall derived using Equation 2.

Figure 7 illustrates the overall recall for varying co-occurrence thresholds (which was set at 0.03 for all previous experiments). As expected, as we increase the co-occurrence threshold, the total number of events decreases, which in turn reduces the fraction of entities for which an event is identified. Figure 8 illustrates the average recall for varying co-occurrence thresholds. Interestingly, for low thresholds, multiple events are combined together resulting in fewer events; similarly, for high thresholds, individual events are split into multiple sub-events. Both these phenomenon naturally result in a lower recall value.

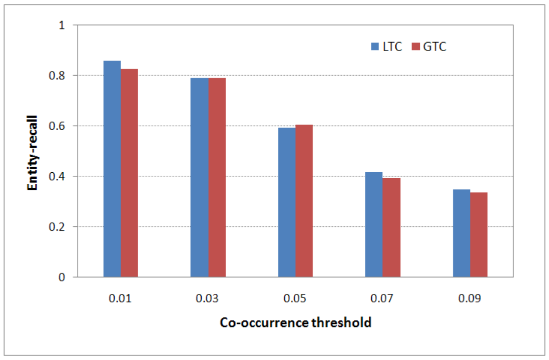

Figure 7: Overall recall for LTC and GTC for varying co-occurrence threshold.

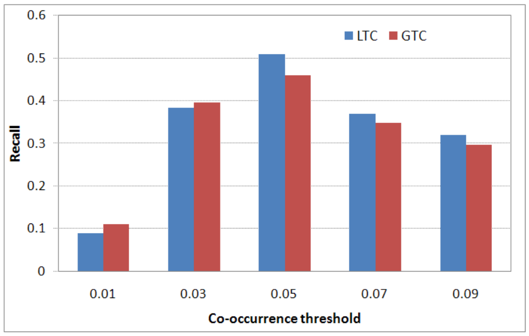

Figure 8: Average recall for LTC and GTC for varying co-occurrence threshold.

6.6 Qualitative Discussion of DROP

To give an anecdotal insight into our results, we provide some sample events detected by the three algorithms, namely, LTC, GTC, and TRJ. The first event (a boating incident involving famous football players) shows the case where all algorithms produce the same event, including all the entities that participated. The second event (a fund-raising concert involving many celebrities) highlights a case where LTC and GTC produce a subset of the entities that participated in the real-world event. Interestingly, this case also highlights the challenges faced due to sparsity of evidence data: in reality, the event included more than 30 people and many of them are not popular in the search user space and further, news articles that discuss this real-world event may not list all the participants. Thus, some of the participating events are eliminated by our peak detection and co-occurrence techniques. Meanwhile, TRJ can not produce a meaningful event since many of those celebrities are buried within a giant cluster of entities. The third event (a broadway musical starting its national tour) shows another example where LTC and GTC is able to discover the event while Tarjan can only produce a giant cluster containing the participants.

We now discuss some interesting insights from our dataset. We observed a larger than expected proportion of events involving celebrities. Some of these were valid (e.g., '{Cameron Diaz; Guy Ritchie; Robert Downey}' attending Comic Con around '2009-06-26'), however, often they were incorrect. These invalid events were formed due to the nature of news reports that cover celebrities: frequently,

T	Alg.	Involved entities	Event
Week of 2010-02-26	LTC	{ corey smith; marquis cooper; nick schuyler }	*boating incident*
	GTC	{ corey smith; marquis cooper; nick schuyler }	
	Tarjan	{ corey smith; marquis cooper; nick schuyler }	
Week of 2010-01-29	LTC	{ edward norton; gloria estefan; derek jeter; alicia keys }	*concert for Haiti*
	GTC	{ edward norton; gloria estefan; derek jeter; alicia keys }	
	Tarjan	*giant event with 2630 entities*	
Week of 2009-10-30	LTC	{ cloris leachman;gene wilder; mel brooks; peter boyle }	*broadway musical tour*
	GTC	{ cloris leachman;gene wilder; mel brooks; peter boyle }	
	Tarjan	*giant event with 2630 entities*	

Table 3: Examples of events detected by DROP.

celebrities that are not necessarily involved in an event are cited together in an article rendering a high co-occurrence measure. We observed this phenomenon in the case of sports athletes as well. These cases were somewhat alleviated due to the fact that our consolidation algorithms take temporal constraints into account, however, given the constant "buzzy" nature of some celebrities, not all such events were eliminated and we observed giant events with several celebrities and sports athletes. Since the consolidation algorithms differ in the extent to which they apply temporal constraints, they differ in the extent to which they can avoid such giant events. In particular, as discussed in Section 5, TRJ produced most of such giant events.

7. RELATED WORK

Our work is mainly related to three fields of study: *relation extraction*, *buzzy entity detection*, and *event detection*. We briefly describe them in this section.

Relation or entity co-occurrence extraction: Relation extraction has received significant attention recently. The typical techniques begin with a predefined schema and build instances of this schema from semi- or unstructured data. As extensions, the problem of identifying co-occurring entities (without assuming a predefined schema) has been studied. For this, Banko et al. [2] introduces Open Information Extraction, which mines co-occurring entities from a large collection of documents while being agnostic to the real world relationships causing the co-occurrence. Very recently, Sarkas et al. [12] presents efficient algorithms to identify *strongly* associated entities over a large document collection, as well as over subsets of documents generated by imposing meta-data constraints. In this paper, we not only eliminate the requirement of a pre-defined schema for relationship discovery, but also expand beyond relationships to discover dynamic events, which are consolidations of multiple relationships based on temporal constraints and more reflective of real world events.

Detecting buzzy entities: Research on burst detection has largely focused on effectively determining a burst in the general behavior of an entity. Various methods for modeling the general temporal behavior have been proposed [9, 15, 6], some of which are extensions of the seminal work by Kleinberg [8]. Bursts are typically detected by examining the divergence from a background model. Existing techniques differ in supporting evidence used to build the background model or in algorithms for detecting a burst. In this paper, we implement a simple peak detection technique that works well in practice, and we believe that many of the proposed techniques can be used in place.

Event detection: Event detection has been studied in the past [7, 11, 16, 3]. Many of those works focus on the problem of detecting events from documents based on language patterns, with the emphasis on extracting location and time information about the events. Perhaps the closest related work is by Zhao et al. [16], where the authors aim to detect events as represented by keywords using a variety of signals including temporal and social constraints. While similar in spirit, our problem setting differs in one important aspect: the events we aim to discover are entity-based and therefore not easily subject to detection by their approach.

8. CONCLUSIONS

In this paper, we presented an approach to detect dynamic relationships between entities that are formed due to real-world events. These relationships differ from static relationships in that they may not conform to any pre-defined schema. Our approach begins with identifying buzzy entities likely to be a part of an event and then identifies a collective set of entities participating in an event. We presented two novel algorithms for generating events while considering temporal constraints and corroborating a variety of evidence such as, user-interest, textual, and temporal. Over an extensive set of experiments, we established the effectiveness of our algorithms and highlighted the tradeoffs exhibited by them. Further, we carried out an in-depth qualitative analysis of the output generated by these algorithms.

9. REFERENCES

[1] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *DL*, 2000.

[2] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *Proceedings of IJCAI-07*, 2007.

[3] H. Becker, M. Naaman, and L. Gravano. Learning similarity metrics for event identification in social media. In *WSDM*, 2010.

[4] O. Etzioni, M. J. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in KnowItAll (preliminary results). In *WWW*, 2004.

[5] D. Freitag. Information extraction from html: Application of a general machine learning approach. In *AAAI/IAAI*, 1998.

[6] G. P. C. Fung, J. X. Yu, P. S. Yu, and H. Lu. Parameter free bursty events detection in text streams. In *VLDB*, 2005.

[7] Q. He, K. Chang, and E.-P. Lim. Analyzing feature trajectories for event detection. In *SIGIR*, 2007.

[8] J. Kleinberg. Bursty and hierarchical structure in streams. In *KDD*, 2002.

[9] J. Leskovec, L. Backstrom, and J. Kleinberg. Meme-tracking and the dynamics of the news cycle. In *SIGKDD*, 2009.

[10] M. Paşca, D. Lin, J. Bigham, A. Lifchits, and A. Jain. Organizing and searching the world wide web of facts - step one: The one-million fact extraction challenge. In *AAAI*, 2006.

[11] T. Sakaki, M. Okazaki, and Y. Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *WWW*, 2010.

[12] N. Sarkas, A. Angel, N. Koudas, and D. Srivastava. Efficient identification of coupled entities in document collections. In *ICDE*, 2010.

[13] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[14] P. D. Turney. Mining the web for synonyms: PMI-IR versus LSA on TOEFL. In *ECML*, 2001.

[15] Z. Wang and S. Hu. Mining correlated bursty topic patterns from coordinated text streams. In *KDD*, 2007.

[16] Q. Zhao, P. Mitra, and B. Chen. Temporal and information flow based event detection from social text streams. In *AAAI*, 2007.