Displaying Cross-Cultural Differences in News Videos II

Dave Dirnfeld (dd2912@columbia.edu)
Supervisor: Dr. John R. Kender
Columbia University

1 Introduction

As globalization continues to grow, news about any event is likely to be reported by a diverse group of media outlets. These reports may come via News Papers, Talk Radio, and Television. We would like to study how the same events are reported by and represented in different cultures and nations. In particular, we focus on how different cultures report the same event differently through video.

In this report, I describe how I enhanced the user interface previously built to display the result of cultural differences in news videos. In particular, I would like to present the changes I made to allow the system to be scaled up by comparing different affinity groups. While I still work with the old data, the new system allows for dynamic growth and new data to be added. The goal is the develop a highly intuitive system by which a user can explore the differences and similarities between international news coverage.

2 Previous Work

In the previous semester, Jiaqi Liu [1] designed the user interface to show the result of cross-cultural differences analysis of a 2017 three-game match between the top-ranking Go player Ke Jie and the computer Go program Alpha-Go. The browser presents a timeline how this event was reported by Chinese news and U.S. news. The goal is to show the user the differences and similarities between the two affinity groups' coverage of the event.

3 Frontend Improvements

In this section, I go through the changes and improvements I made to the user interface. These changes enhance the scalability of the system while maintaining an

intuitive scheme for users to navigate and compare different news coverage.

3.1 From EventDrops to Patternfly-Timeline

Figure 1: Original UI using EventDrops

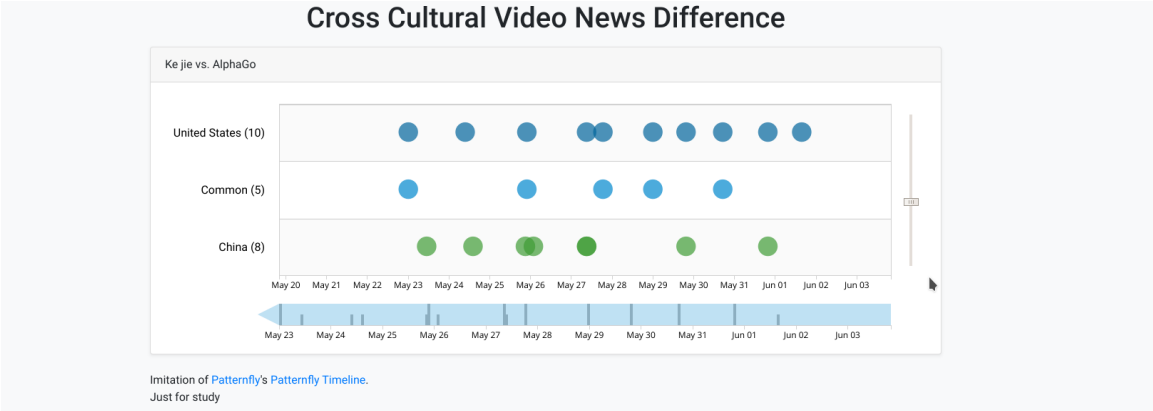

Figure 2: New UI using Patternfly-Timeline

In the previous implementation (Figure 1), the user interface used JavaScript library EventDrops to display each event as bubbles on a timeline. The event's timeline has three rows, the top row represents news coverage in the United States, the bottom row represents news coverage in China, and the middle row represents the correspondence between the two. While this system works well for our purposes, EventDrops had a few usability flaws that are fixed with the new implementation built on Patternfly-Timeline (Figure 2).

(a) Zoom level on .001 second scale

(b) Zoom level at infinity

Figure 3: Old UI Zoom Constraint

3.1.1 Zoom Control

EventDrops has two problems associated with zoom control. The first is with regard to infinite zoom levels (Figure 3a). The implementation did not set an upper and lower bound on the zoom level. A user was able to zoom in to sub-second levels and zoom out passed the common era. The second problem deals with how users zoom in or out (Figure 3b). EventDrops did not have a adequate zooming mechanism. Users needed to hover over the timeline and use the mouse wheel to zoom in and out.

Patternfly-Timeline fixed the first problem by adding two functions `minSacle()` and `maxScale()`. The `minSacle()` function allows us to set how far a user can zoom out, while the `maxScale()` function allows us to set how far a user can zoom in. To fix the second problem with EventDrops, Patternfly-Timeline also added a slider to the right side of the timeline. This feature allows the user the easily zoom in and out on the timeline (Figure 2).

3.1.2 Droplet Responsiveness

Patternfly-Timeline also fixed an issue I found with EventDrops. In the old system clicking on a drop would be inconsistent. Sometimes the information regarding the droplet would show, and other times it would not. With Patternfly-Timeline, clicking on a droplet is highly consistent, and the information regularly displayed during testing.

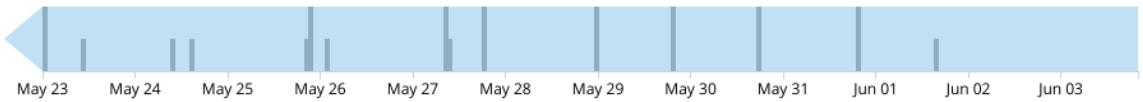

Figure 4: Histogram to display the spread of droplets for a given time frame

3.1.3 Histogram

One of the refreshing features that come with Patternfly-Timeline is the ability to show a histogram underneath the timeline (Figure 4). The histogram gives the user an intuitive approach to visualize the spread of droplets over time and to focus in on a specific date.

3.2 Tags

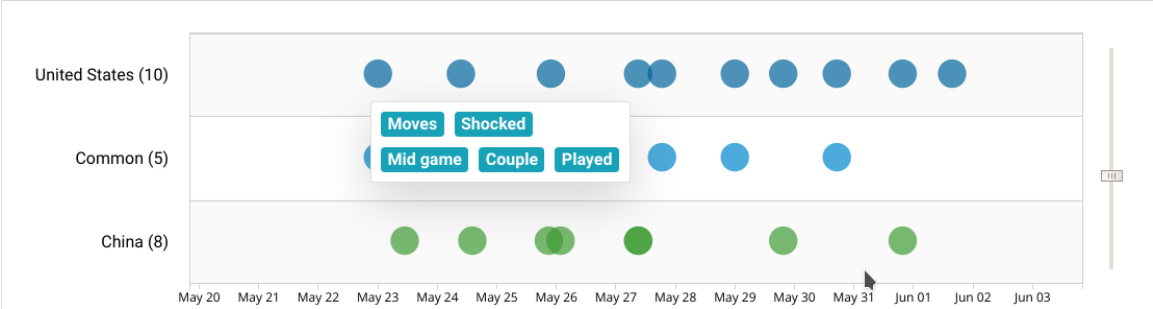

Figure 5: Tags displayed when hovering the mouse over a droplet

One of the things we worked on this semester is the addition of tags to a droplet. When a user hovers their mouse over droplet, they are presented with tags associated with the droplet (Figure 5). These tags are taking from the video description, and should help the user compare different droplets from the same affinity group.

```
eventHover(function(el){
    var tags = "";
    for (i in el.details.tags){
      tags += '<p class="badge badge-info">'
              +el.details.tags[i]+'</p>';
    }
    ...
```

```
8   })
```

Listing 1: JavaScript function to display tags on hover

The `eventHover()` (Listing 1) takes an element `el` as an argument, and loops over `el.details.tags`. The result is a list of tags associated with the droplet the user is hovering over. The list is then displayed to the user as a popup box below the hovered droplet.

4 Backend

To make the browser extendable to more affinity groups, I needed a reliable system to store the information and handle requests. One popular framework is Flask; therefore, I elected to use Flask as the core system. Flask allows me to integrate a SQL Database to store and retrieve information quickly, and for this project, I used Flask-SQLAlchemy, an extension for Flask to support SQLAlchemy relational database.

4.1 The Database Model

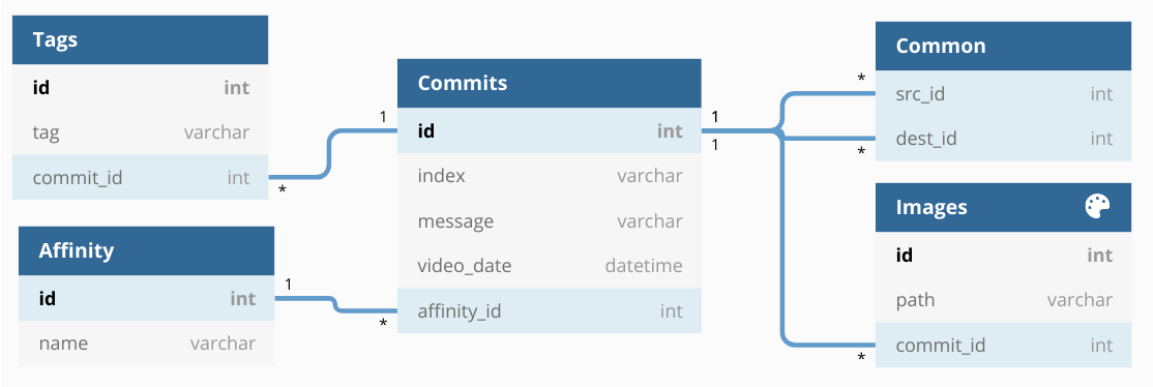

Figure 6: Database Logical Model

The databases consists of five tables: Affinity, Commits, Images, Tags, and Common. In this section, I describe each table and the data it holds (Figure 6.

The Affinity table stores information about affinity groups. Each entity has two fields, an *id*, and a *name*. In our case, the table consists of two entities, one for the United States and one for China.

The Commits table stores information regarding the timeline droplets. Each entity has the following properties: *id*, *index*, *affinity_id*, *message*, *video_date*, and *commonality*. The *id* is the primary key of the entity; the *index* is a three-letter abbreviation of the affinity group followed by a dash and a number. The *affinity_id* field links a commit to an affinity group. The *message* and *video_date* store the transcription and date of a video, and *commonality* kinks two commits if they have a correspondence.

The Images table stores the path of the images associated with a commit and has a foreign key constraint to an entity in the Commits table. Similarly, the Tags table lists the tags of a Commit, also identified by a foreign key constrain to the Commits table.

Finally, the Common table links one commit to another if the two commits are classified as being related. The table is many to many Self Referential Relationship and contains two fields, both pointing to a commits entry.

4.2 Retrieving Data

When the user requests the home page, Flask makes a call to the database to retrieve the proper information. In this section, I describe how the information is queried and returned to the user.

4.2.1 Min and Max Dates

```
1    commits = Commits.query.order_by(Commits.video_date)
2    d = [c.video_date.strftime("%Y-%m-%d") for c in commits]
3    dates = {'min':d[0], 'max':d[-1]}
```

Listing 2: Retrieving min and max dates.

JaveScript needs to know the start and end dates of the data to set up the scale properly. To get the information, I query the database the get all the commits in ascending order. I then extract the dates into a Python list so that I can get the first and last elements. I store the maximum and minimum dates in a JSON object and pass it to JavaScript.

4.2.2 Droplets Information

```
commit_data = [{"name": "United States", "data":[]},
               {"name": "Common", "data":[]},
               {"name": "China", "data":[]}]
```

Listing 3: Patternfly-Timeline's data object.

Retrieving the droplet information is a little more involved. First, Patternfly-Timeline needs each named time series (i.e., the United States, China, and Common) to be an array of JSON objects. Each object must contain two keys, *name* and *data*. The names, in our case, are the affinity groups plus an additional entry *Common*. The key *data* is an array of JSON objects containing a date and optional payload called *details*.

```
commits = db.session.query(Commits, Affinity).outerjoin(
               Affinity, Affinity.id == Commits.affinity_id)
               .all()

for c in commits:
    if c[1].name == 'United States':
        tags = Tags.query.filter_by(commit_id = c[0].id)
              .all()
        data = {
            "date": c[0].video_date.strftime(date_format),
            "details":{
                "name": c[0].index,
                "message": c[0].message,
                "tags": [t.tag for t in tags]
            }}
        commit_data[0]["data"].append(data)
    ...
```

Listing 4: Querying the Database and parse the data.

To properly construct the object Patternfly-Timeline needs, I first join the Commits table and Affinity table, and loop over the results to parse it based on the name. For each row, I extract the video_date, message, index, and tags. In order to get the tags for a commit, another query is required to get all tags where the commit_id is equal to a current commit id.

```
def get_common(self):
    commonality = Commits.query.join(common,
                    common.c.dest_id == Commits.id))
                    .filter(common.c.src_id==self.id)
```

Listing 5: Database helper function to get a correspondence table.

```
common = c.Commits.get_common().all()
for a in common:
    commit_data[1]["data"].append({
        "date": a.common.all()[0].video_date.strftime(date_format),
        "details":{
            "isCommon": 1,
            "name": a.common.all()[0].index,
            "common": a.index,
            "message": {
                "US": a.common.all()[0].message,
                "China": a.message
            }
        }
    })
```

Listing 6: Add the correspondence to data

One caveat is in regard to the correspondence. Extra logic is required in the case in which the top row (the United States) has a correspondence to the bottom row (China). In this case, one more join is required, this time it on the Commits table to itself. The results are then stored in the Common index of the data array.

4.2.3 Image Slides

```
@app.route('/img/<i>', methods=['POST'])
def img(i):
    commit = Commits.query.filter_by(index = i).first()
    imgs = Images.query.filter_by(commit_id = commit.id).all()
    paths = [i.path for i in imgs]

    return jsonify(paths)
```

Listing 7: Retrieving images for a droplet

When a user clicks on a droplet, the frontend makes an asynchronous get request to retrieve the image paths of that droplet. The function takes a commit id and returns a list of image paths from the Images table.

5 Future work

5.1 Selecting Events

In the future, I would like to add the option for the user to select which global event they would like to compare. At this time, there is only one event, namely Ke Jie vs. AlphaGo. In a future system, there should be many more events a user could choose from. Furthermore, within one event, there is likely to be more than one affinity group covering the event. Hence, the user should be able to select the affinity groups to compare.

5.2 Importing New Data

One crucial option currently missing from the system is the ability to add more data quickly. Ideally, the visual interface should expose a hook to allow the system to add more events and groups. The simplest method is to expose an API by which new data can be added to the database.

5.3 User Interface Improvements

The current system is not perfect, and there are a few things that should be improved. First of all, the line connecting correspondences are not attached to the droplets. As a result, when the user zooms in and out or moves the timeline, the connecting lines do not move with them. Second, the color scheme is not ideal. There should be clear color separation among the affinity groups. Finally, I would like to conduct a user experience study to improve the user interface further.

6 Acknowledgments

I acknowledge the use of code from Patternfly Timeline and their demo. In addition, the code is based upon the previous implementation of the browser by Jiaqi Liu.

References

[1] Jiaqi Liu. Displaying Cross Cultural Differences in News Videos. http://www.cs.columbia.edu/~jrk/NSFgrants/videoaffinity/Interim/19x_jiaqi.pdf