

MovieMash: Providing Movie Schedules Using Search

Thomas McDonald	Troy Fulton	Mitchell Eldridge
Texas A&M University	Texas A&M University	Texas A&M University
College Station, TX USA	College Station, TX USA	College Station, TX USA

Abstract—Recently, the COVID-19 epidemic and the policies that followed the outbreak have forced nearly everyone on Earth indoors with two things: a medium for watching movies and time. We present MovieMash as a searching and scheduling tool to help movie “bingers” who have a vague idea of what kinds of movies they want to see and some free time on their hands plan out their weekend. Users input free text that expresses interest(s) in topics, the amount of time they have available, and other preferences, and MovieMash returns a schedule that best fills up their available time with the most relevant movies to their query.

We use a bag-of-words representation of queries and the unstructured data from IMDb to score and filter movies. To evaluate the relevance of our system, we also introduce NDCG-Partial to formulate about the relevance of our schedule to the user query. Using keywords available from IMDb as queries, our public web application often provides users with mostly relevant results (as measured by an NDCG-Partial score of 0.9) with a relatively full schedule (only about 20 minutes left) when entering a time window above about 10 hours. Overall, the results show that as long as groups have more than 10 hours to “mash up” movies, they are likely to find a schedule everyone will like.

I. INTRODUCTION

In our current society, much of our day is watching television and consuming other media. The U.S. Bureau of Labor Statistics published that the average American in 2018 watched 2.8 hours of television per day, accounting for over half of the available leisure time for most Americans [1], and the current state of society (even before COVID-19) suggests that the numbers will increase. For example, the Los Angeles Times reports that Americans consume about 3 hours and 35 minutes of each day watching television [2]. If the trend continues, it is safe to say that Americans will continue to consume a significant amount of

media and television.

In addition, the recent outbreak of the coronavirus (COVID-19) has forced everyone indoors. Due to the highly contagious nature of the airborne virus, the CDC has advised [3] that everyone remain indoors and practice social distancing indefinitely to prevent overloading our medical system. Despite the unfortunate effects of the deadly virus, such as its effects on the world economies, it seems that the best thing we can do for one another is stay isolated at home. Streaming services providing online entertainment have accelerated significantly as a result, and since most people are sheltered in place for non-essential needs, many streaming service subscribers and TV watchers have nothing but time to use movies to distract themselves.

There are multiple methods to fill in the time that Americans have to consume media. Some may randomly select movies or television shows using a streaming provider, such as Netflix. Others may search websites like IMDb to identify movies to watch. For the latter, many factors may influence their decision of what to watch such as the rating of the content, how new it is, or whether the individual will be interested in it. Even for a small group of viewers or an individual, it can be challenging to match your preferences to movies you might find through any online search.

To combat this difficulty, we introduce MovieMash as a searching and scheduling tool for finding relevant movies to spend hours watching. The goal of MovieMash is to match collections of movies with users based on a free-form query and other input parameters, such as how much time the individual has to fill up. MovieMash is also a solution that can be used to generate a list of movies for movie marathons. To implement

such a solution, we discovered data in the form of scenarios on websites across IMDb and scraped them from the web to insert them into our instance of ElasticSearch, which we use for our information storage. Information retrieval is done via queries to ElasticSearch from the server of our hosted web application. Our application normalizes results and tries to maximize the relevant results before presenting the user with their best schedule.

We present our implementation of MovieMash as a project for our Information Storage and Retrieval Class, and we evaluate our system using our own predefined metrics. Since our information retrieval involves both a ranking and a filtering, we consider an evaluation like NDCG catered for our problem size. Our evaluations show that with increasing time windows, our schedules represent increasingly relevant results, as ranked by BM25 in ElasticSearch. We show that after about 15 hours of movies, the user will most likely have all the most relevant movies returned in their schedule.

II. APPROACH

To complete the application, we broke it down into two phases. The first phase involves collecting and sanitizing unstructured data and inserting them into a database. The second phase involves the interactions of the user with the data, including querying and filtering of results. To clarify, these two phases are not mutually exclusive. Due to the short-term (one semester) nature of this project, we only collected data once, but it is conceivable (and desirable) for longer-term production systems to constantly be scraping for new data. Here, we describe both phases and the associated design choices we made.

A. Phase 1: Data Retrieval

1) *Unstructured Movie Pages:* The first challenge of Phase 1 was determining how to represent data in our system. An IMDb page for any movie (like the one in Figure 1) will include a lot of scattered information, such as reviews, the cast, links to where to watch the movie and trailers, and much more. Since the IMDb site is often crowd sourced, there also may be entries present for only some movie entries. The entries for fields

like reviews and summaries may also vary widely in length, character sets, etc.

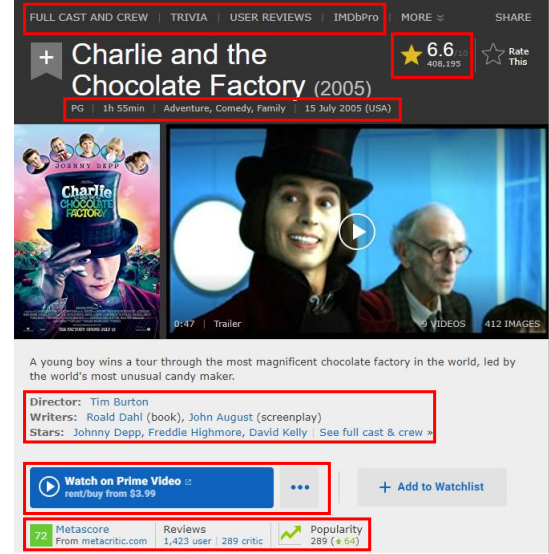


Fig. 1: Example IMDb Page with unstructured data boxed in red.

We decided that some of the most relevant features for each movie include the average rating, year, genres, and "votes" (a metric unique to IMDb). We also included a few textual features, namely the title and the synopses for each movie, which we use to match with textual queries.

2) *Pre-processing of Data:* One other design decision we made was that we would pre-process the data before entering it into the database. We considered making our application so that the data is retrieved at the time that the user makes the query, but because retrieving the information from the Internet is often an expensive and slow (although parallelizable) operation, we decided to pre-process as many movies into storage as we could. This gives users the illusion of a faster runtime experience.

B. Phase 2: Data Storage

Storing the movie data was originally going to be done using CouchDB but later was switched to Elasticsearch. In our initial draft of the project, the relevant movie information was simply the movie name, run time, rating, and genre. This made CouchDB a ideal choice of database due to its simplicity, Document-based storage, and intuitive RESTful HTTP API. Before much progress

was made using CouchDB, our project pivoted towards using movie synopses (a potentially long string) to further rank movie results to users. After this addendum, Elasticsearch was the ideal data storage candidate because of it's ability to do some additional processing for us discussed later in this paper.

1) *Types of User Query Interaction:* Our goal for the interface was to make it simple and easy to use as possible. In fact, the original proposal did not include any free-form queries or use of unstructured data. However, after a brief discussion, the addition of using movie synopses and free-form queries to affect results was made. This allows us to exercise the techniques we learned in class for retrieving and ranking documents. To maintain simplicity, we offer the user just one text box where they can enter keywords (matched with the free text options in the documents) and filters on the results.

2) *Forming a Schedule:* The scheduler uses the results from an Elasticsearch query to form a schedule that is most likely to meet the user's need. Forming a schedule has two important components. First, the schedule should have as many of the most relevant movies as possible. Second, the schedule should fit in the time window given by the user as well as possible. This is often a tradeoff because often the most relevant movies will not fit in the schedule given by the user. We decided that, for the purposes of this project, we will enforce that no schedule shall exceed the time window given by the user. This allows us the luxury of taking a greedy approach to put movies into files based on their scores in the collection matching.

Initially, we did not have a concrete way of how we wanted to score the documents and we were thinking of a variety of different methods to fill in the time requested by the user. The original plan for scoring just included using genres, movie rating, and amount of time available to return a list of movies. Once the free-form user query and preference for newer movies were added, the scoring was modified. Each user query field has its own individual scoring (or filtering, in the case of genres), and each scored field is given a weight to appropriately deliver results.

III. IMPLEMENTATION

In this section, we describe the tools we used to build and implement MovieMash. The overall system architecture for MovieMash described in this section can be see in Figure 2.

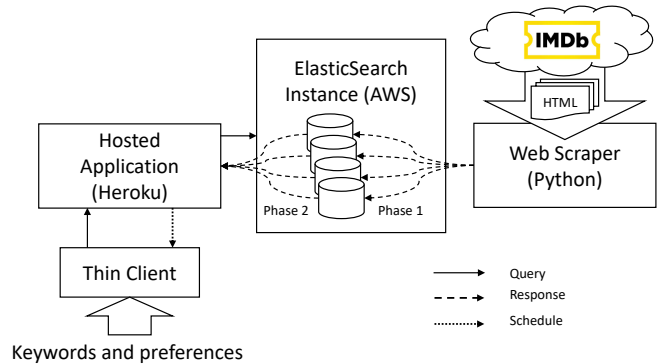


Fig. 2: System Architecture

A. Data Storage

1) *Web Scraping*: Flat files available on IMDb’s website provide a unique ID for thousands of entries on IMDb. We downloaded a file containing various small pieces of information for each ID, such as the fields of interest (title, run time, etc.), but the file did not contain the scenarios needed for relevance feedback. Using the `requests` library, we wrote a Python script that uses the ID of each movie in the flat file to form a URL that links to the web page with the synopsis for that particular entry and requests the page. We keep only entries whose pages are available based on the return code from `requests`. Some movies contain no synopses, as the synopses are often crowd sourced. We find that the synopsis for a movie often contains relevant information, such as director and actor names, names of characters, plot points, etc.

The HTML documents retrieved are then parsed using `BeautifulSoup`. Luckily, the synopses are positioned in consistent locations on the HTML page. Thus, objects returned from `BeautifulSoup` can all be accessed the same way to find the string representing the synopses.

2) *Database Software*: Once the synopsis for a particular index has been scraped (or has failed to do so), it is indexed in our instance of ElasticSearch which we host on Amazon Web Services (AWS) for increased storage capacity. We elect

MovieMash

Brought to you by Thomas McDonald, Mitchell Eldridge, Troy Fulton

What are some actors, locations, characters, or other keywords for movies that may interest you?

How much time do you have?

Hours: 10
Minutes: 0

What genres would you like?

Drama
Documentary
Comedy
Action

☒ Prefer Newer Movies

Find Movies

New York Mixed Martial Arts	60 min	2011	Open in IMDb
New York Decalogue	80 min	2011	Open in IMDb
August Rush	114 min	2007	Open in IMDb
Gangs of New York	167 min	2002	Open in IMDb
The Tree of Life	139 min	2011	Open in IMDb

Fig. 3: MovieMash in use

not to do any sanitizing on the input string from IMDb because the issue is complex and has been well researched [4]. In addition, the added layer of inversion would add to the runtime complexity of a significant portion of the movies. The process of retrieving HTML documents from IMDb and indexing in our ElasticSearch can be seen in Phase 1 (right) of Figure 2.

B. Data Retrieval

We now discuss the rest of Figure 2 for Phase 2. We host our web application on Heroku. We have implemented a Flask application running a Python script that simply passes the user's query received from the front end thin client to ElasticSearch. When results are returned to the web application, the weights are used to determine which movies should be included in the schedule and returned to the client for displaying.

1) *Web Application:* As shown in Figure 3, the user interface on the thin client at the bottom left of Figure 2 is a simple single-column page. The first card contains all of the query options. The first input is the main query for retrieving the movies. The input can be comma separated but it does not make a difference as the punctuation is stripped and non-contextual ranking is done.

Next on the figure is the amount of time that a user has. The input has a selection of hour increments and 5 minute intervals for the minutes. This makes the experience very simple and impossible to put bad input.

Lastly, the bottom half has a multi-select list of genres that are retrieved from the database and a checkbox that modifies the scoring to prefer "newer" movies. These selections are more for the user to have some more control over what results are shown. However, the preference for newer movies does modify the scoring of movies.

When the results are displayed, the run-time, year, and title of the movies are displayed. This gives the user some idea of whether the movies might be a good fit from just looking at them. We also display a generated IMDb link so the user can get more information and possibly identify where they can watch the film, if IMDb has the information for the specific film.

2) *Scoring and Filtering:* Similar to the implementation in the first programming assignment, scoring in ElasticSearch is done using BM25 [5]. The modification will initially filter out any movies that do not have any of the query terms in the movie title, genre, or synopsis. These are deemed

irrelevant movies for our purposes. We also combine the query to filter out movies that do not contain any of the selected genres specified on user input. These two steps will immediately reduce the number of movies we consider in our results and provide for faster result generation.

Next, BM25 to calculate a similarity score for each movie that remains after the initial filtering. The score that results from this is then normalized by the maximum score and multiplied by a weight for how much of a role in plays in the final scoring.

Once the initial scores from the document matching are set, we then normalize each movie's rating and also multiply by a weight for how much the rating will play a role in the final results. We included rating into document ranking since we wanted to return "good" movies to the user and not ones that are deemed "bad".

The last main aspect to the numerical score is dependent on whether a user prefers newer movies or not. If so, we normalize the movie release year with a base year and also multiply by a weight. The weight in this case is very high as we want to heavily influence the results to more recent releases.

All normalization that is done is simply by finding the maximum value and doing the following: $\log(value) - \log(max_value)$.

Tuning of the weights is discussed in the Evaluation section. Modifications and future ideas to improve the querying are discussed in the "Future Work" section.

Finally, we had to fill up the time available given from the user to generate a list of movies. We accomplished this by implementing a simple greedy approach. We started with total time available and would iterate through our sorted list of movies by the calculated score from high to low. We then check if the current movie can fit in the leftover time. If so, add the movie to the results list. If not, move on to the next movie and repeat. Each time a movie is added as a result, the time remaining is decremented by the movie length.

As one can imagine, the average score of the returned movies could be lower than possibly using a different approach that optimizes the average score while also filling up the time, however, that was not implemented. An evaluation on the greedy

approach to filling time is done in the next section.

IV. METHODOLOGY

When evaluating our system, we needed to find a new metric for determining the score of a given schedule. Each schedule can be score based on two different metrics. First, it is important to the user that their time is filled appropriately. There should not be much down time in the schedule, although it may be beneficial for users to have time to get snacks and take breaks during their movie marathon. Second, it is important that the schedule contain as many of the most relevant movies returned from ElasticSearch as possible. For this metric, we calculate a score similar to NDCG, which we describe in this section. We also describe how we tuned the weights of each result.

1) *NDCG-Partial*: For a given schedule, we calculate the NDCG-Partial score by looking at the full list of normalized results returned by ElasticSearch sorted by score, as discussed in the scoring subsection in the last section. Since it is possible for these values to be negative, we always add the lowest score + 1 to every score for use in the DCG formula, which is reproduced below for a position p and documents of relevance rel_i :

$$DCG_p = \sum_{i=1}^p \frac{2^{rel_i} - 1}{\log_2(i + 1)}$$

Although it may not be perfect, we use the results from ElasticSearch as our ground truth for calculating *IDCG*. Since the user can always reorder their schedule after we have given them the list of movies, we claim that the order is not important for the schedule generated from the ground truth. Thus, we treat the schedule as a subset of the returned movies from ElasticSearch. This subset's relevance is only hurt by leaving out movies in the ground truth ranking. Thus, when we calculate DCG for the schedule, we do not count the contributions of the movies left out of the schedule. We find the movie in the last position of the (sorted) schedule, and we calculate *IDCG* by setting that movie's position in the ground truth ranking as p in the formula above. Thus, this gives us a new DCG for a schedule $S = \{p_1, p_2, \dots, p_n\}$ of positions of movies in the ground truth:

$$DCG_p = \sum_{i \in S} \frac{2^{rel_i} - 1}{\log_2(i + 1)}$$

We illustrate with an example. Consider the results returned from ElasticSearch and their normalized scores in Figure 4 and the schedule returned in Figure 5.

Indians & Cowboys	1.45836
CowJews and Indians	1.187038
Cowboys	1.146719
Cowboys	1.146719
Cowboys and Idiots	1.142443
Disappointment Valley: A Modern Day Western	1.127599
Imagining Indians	1.063601
Drugstore Indians	1.063601
Virgin Cowboys	1.057215
Cowboys & Communists	1
Pordenone Cowboys	1

Fig. 4: Ground Truth for IDCG from query "Cowboys and Indians"

Indians & Cowboys	96 min	2016	Open in IMDB
Cowboys	90 min	2019	Open in IMDB
Cowboys	92 min	2015	Open in IMDB
Disappointment Valley: A Modern Day Western	105 min	2017	Open in IMDB
Imagining Indians	60 min	1992	Open in IMDB
Drugstore Indians	146 min	2016	Open in IMDB
Pordenone Cowboys	7 min	2015	Open in IMDB

Fig. 5: Schedule generated from query "Cowboys and Indians"

The DCG scores for the two rankings are identical, only the schedule is penalized for skipping over the two more relevant movies "Virgin Cowboys" and "Cowboys and Communists" to fit the 7-minute "Pordenone Cowboys" movie in the schedule. This will lead to a better time difference in the schedule (fills up more time), but gives a NDCG-Partial score of 0.795.

2) *Tuning*: To tune the weights for the results, we started with our base weights. With a specified query string, the scores of the resulting movies were analyzed and the weights were manually modified until we were content with the order/final scores of each movie. The tuning was a very elementary process but could potentially be done using machine learning in the future.

V. EVALUATION

To evaluate the product, we wanted to verify both that the results represented the input query and also that the time availability input was mostly, if not completely filled. Using time differences as a metric for time utilization and NDCG-Partial as a metric for relevance of a schedule, we present the results of our evaluations.

A. Time Usage

When proposing the project, we wanted to have the resulting movies to ideally be within 30 minutes of the time available input. To evaluate the results of our application, we created a script that would find the average time difference for a sequence of times with multiple different queries. Once the average time differences were calculated for each time available input, we were then able to calculate a total average time difference.

A point worth note is that it is almost impossible to guarantee to completely fill up the input time while also ranking documents based on the input query. It is possible to try to fill up time with movies without considering query as long as there exists a combination of movies that add up to that time. For this reason, we wanted the results to be within a certain time range of the input.

The results of running the time difference evaluation function can be seen above in Table I. The average between all the results is 17.338 minutes, which is significantly less than our original goal of 30 minutes, although this was a less formal goal for the evaluation. The greedy approach to filling time seems to work well but there is some room for improvement.

Improvements to the time usage evaluation could include a different approach to filling up the time with perhaps a dynamic programming approach. This could potentially give us better relevance results.

Available Time	100	150	200	250	300	350	400	450	500	550	600	650	700
Time Difference	23.6	27.4	10.0	18.4	3.8	8.0	14.2	28.2	17.0	19.8	13.2	27.0	14.8

TABLE I: Time Difference Evaluation Results. Time in Minutes

B. Result Relevance

As mentioned in the Methodology section, we use our NDCG-Partial to evaluate the relevance of a given schedule to the user’s query. Figure 6 shows the change in NDCG-Partial results of two randomly chosen queries: “Cowboys and Indians” and “sherlock holmes mystery crime murder investigation hounds of baskerville” (several words having to do with Sherlock Holmes films) when given the query and the number of hours specified with no additional inputs (genres or the “new movies” selection). The results show that scores for a schedule do not necessarily strictly increase when given a larger time window (as one might think).

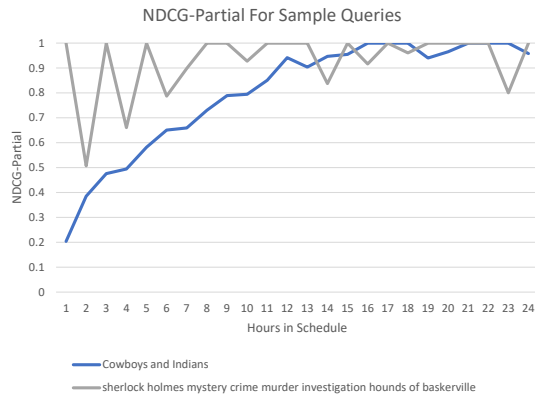


Fig. 6: Trends in the NDCG-Partial Scores as the user’s time window increases

Readers should note that the query with much more specific keywords (the latter query) tends to quickly reach 1.0 but sporadically dip every few hours. We have observed that with specific queries like this one, there are often only a few mostly relevant movies. In this case, the schedule will most likely capture all of them, but when given an awkward time window where the schedule needs to skip one of the more relevant movies in favor of a shorter, less relevant movie to fill the schedule, it is greatly penalized. In the former query, there are many movies matching with more equally distributed scores, so the curve is much smoother.

To generalize our evaluations, we draw queries from a list of 221 keywords found immediately on IMDb’s website used for both users and developers to search the site. Some keywords include “action,” “satire,” and “virtual reality.” These keywords include both queries that might be more specific to certain movies and those that should draw in a large pool of diverse movies. We ran the same experiments as those aforementioned on these keywords with the same inputs for hours, genres, and the “new movies” selector. The results of the evaluation, seen in Figure 7, reveal that after about 10 or 11 hours, the average NDCG-Partial score (the red trend line in the figure) tends to level off above 0.9 (shown in black). Thus, with a sufficiently large time window, users can be sure that this algorithm will find most of the most relevant movies that match their query.

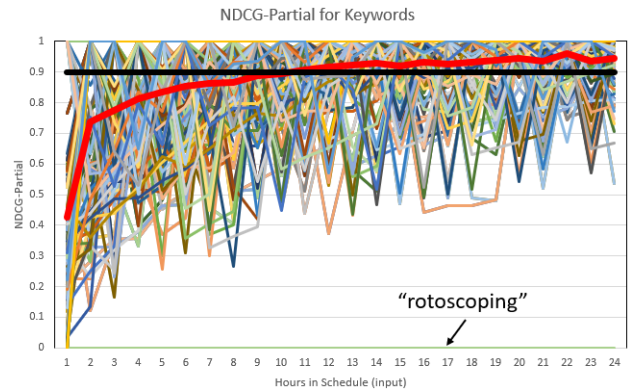


Fig. 7: NDCG-Partial trends using IMDb keywords as queries

We note two more results from Figure 7 worth investigating. First, notice that when given only 1 hour as input, the data vary widely, mostly from 0 to about 0.65 and at 1.0. Assuming that the majority of the top relevant results are scored the same, this shows us that some keywords have to search much “deeper” in the results to find a relevant movie that will fit in just one hour. Second, we observe that some terms, like “rotoscoping” (shown in Figure 7) can never score higher than 0 because movies whose synopses or

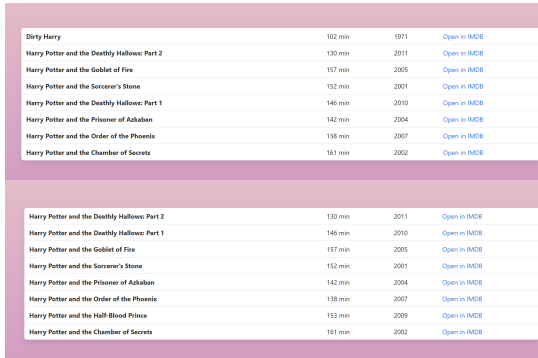
title do not contain that phrase are filtered out completely. Rotoscoping refers to an animation *method* for drawing frames used in movies such as *Snow White*, but it is not likely to appear in the summaries or titles of movie entries, since most writers are not aware of these techniques. This particular result exposes a flaw in using only keywords in queries to match queries to synopses. We leave analysis of better alternative methods of movie-query matching as future work.



Harry Potter and the Deathly Hallows: Part 2	130 min	2011	Open in IMDB
Harry Potter and the Goblet of Fire	157 min	2005	Open in IMDB
Harry Potter and the Sorcerer's Stone	152 min	2001	Open in IMDB
Harry Potter and the Deathly Hallows: Part 1	146 min	2010	Open in IMDB
Harry Potter and the Prisoner of Azkaban	142 min	2004	Open in IMDB
Harry Potter and the Order of the Phoenix	138 min	2007	Open in IMDB
Harry Potter and the Chamber of Secrets	161 min	2002	Open in IMDB
Harry Potter and the Half-Blood Prince	153 min	2009	Open in IMDB

Fig. 8: Harry Potter Results

A simple test of verifying that relevant results would appear based on the query was done. An example would be searching for "Harry Potter" in the input query. With all of the other query aspects set to their defaults and a large time interval given, one can see that the results are very highly relevant to the input query. We did not have a list of queries and manual rankings of which movies should be returned with that query so it is difficult to get a numerical evaluation on the query results. However, a qualitative analysis can be done to verify that the results make sense for the input query, as shown with the "Harry Potter" results in Figure 8.



Dirty Harry	102 min	1971	Open in IMDB
Harry Potter and the Deathly Hallows: Part 2	130 min	2011	Open in IMDB
Harry Potter and the Goblet of Fire	157 min	2005	Open in IMDB
Harry Potter and the Sorcerer's Stone	152 min	2001	Open in IMDB
Harry Potter and the Deathly Hallows: Part 1	146 min	2010	Open in IMDB
Harry Potter and the Prisoner of Azkaban	142 min	2004	Open in IMDB
Harry Potter and the Order of the Phoenix	138 min	2007	Open in IMDB
Harry Potter and the Chamber of Secrets	161 min	2002	Open in IMDB

Harry Potter and the Deathly Hallows: Part 2	130 min	2011	Open in IMDB
Harry Potter and the Deathly Hallows: Part 1	146 min	2010	Open in IMDB
Harry Potter and the Goblet of Fire	157 min	2005	Open in IMDB
Harry Potter and the Sorcerer's Stone	152 min	2001	Open in IMDB
Harry Potter and the Prisoner of Azkaban	142 min	2004	Open in IMDB
Harry Potter and the Order of the Phoenix	138 min	2007	Open in IMDB
Harry Potter and the Half-Blood Prince	153 min	2009	Open in IMDB
Harry Potter and the Chamber of Secrets	161 min	2002	Open in IMDB

Fig. 9: Prefer New Movie Results. Query: "Harry Potter, dirty"

The above figure (9) shows the results before the "Prefer Newer Movies" option was selected and the results after the the option was selected.

The specific query used was "Harry Potter, dirty". The initial results recommended the highly acclaimed film "Dirty Harry" from 1971. To verify the scoring and preference of newer movies, we selected the option and wanted to verify that the Harry Potter films were rated higher than "Dirty Harry". The results seemed to be successful for the accomplished task.

VI. FUTURE WORK

A. Contextual Queries

A very desirable next step would be to add support for contextual queries. Currently queries are stripped down into a unordered list of keywords used to create a list of relevant titles. By taking into account the ordering of query words and their correlation, we can further narrow down related movies to the query. For example, the query 'harry potter, dirty' will currently return the movie *dirty harry* as the most relevant title before any Harry Potter film despite the fact that contextually, the query is clearly pointing to harry potter. By taking into account the connection between these words we can further optimize our results and avoid this issue.

B. User Ranking and Evaluation

A very useful improvement would be to introduce some form of positive feedback. This would likely take the form of an option for users to vote on movies they felt were good or satisfied their query, or mark movies as bad or unrelated to their query. This feedback would then be stored in our database to further refine our results by putting a heavier ranking weight to titles that have performed well in the past.

C. Randomized Results

Sometimes a list of the most popular movies for a user's query might not be what they truly want. A frequent movie watcher would likely have seen many of the top titles for a specific genre and would gain little from receiving a list of what we think are highly rated movies. By giving an option to add some randomization to the results, users that do not like the results for their query can get a new list of movies that might contain more titles that would satisfy them without having to change their query.

D. Expansion to other media

A point of future work could be to further add to our database various other types of media such as TV episodes, shorts, or YouTube videos. While a method of comparing these documents to uniformly rank them against a user query would need to be created, it would allow users to mix up their entertainment experience with a wider variety of options that meet their interests. In addition to this, having a larger variance of run time media would allow our program to better match the users given free time and reduce the time difference originally shown in Table I.

E. Time Allocation

Our project used a greedy algorithm in choosing titles to fill a users given free time. A possible avenue going forward would be to revamp this algorithm to instead use a more dynamic programming approach to try to maximize the average score of a given list while still meeting the time constraints.

F. Automated Weight Tuning

Instead of doing a greedy or dynamic programming algorithm, yet another approach would be to use machine learning to fine tune the results of a user query. This would require coming up with a training set of query:expected-result pairs in order to train a machine learning program how to score titles. This would likely entail a lot of manual labeling or storing user actions in order to come up with the set but could prove to be an extremely flexible and dynamic approach for scoring titles.

G. Text Sanitization

One final area for improvement is in sanitizing any text input from IMDb. The scenarios we received did not always translate well to bag-of-words representation because of punctuation, lack of stemming, differences in languages, etc. Use of a language model or an external library would improve the scoring of the results if we continue to implement this service with bag-of-words representation.

VII. CONCLUSION

We introduce MovieMash, a searching and scheduling tool for movie "bingers," as a way to help groups with varying interests "mash up" their movies on long weekends under COVID-19 shelter-in-place policies. Our software architecture employs the help of Elasticsearch's BM25 [5] document-query matching to help users with a vague idea of what they want to watch search through our index of pre-processed movie data. Our greedy scheduling algorithm quickly searches through many ranked results to a query to give the user a schedule that both fills up their time the best and retrieves as many of the most relevant results as it can.

We evaluate both the effectiveness of the scheduler to utilize the time block given and its ability to retrieve documents with sampled differences and our new NDCG-Partial metric, respectively. We show that often, regardless of how big of a time window is given, users will get a schedule that leaves only about 20 minutes of down time. We also introduce NDCG-Partial as a way to favor schedules that do not "skip" over more relevant movies in favor of filling up time in the schedule. We evaluate our system by running several searches on IMDb's public set of keywords, which reveal that users can often expect to see mostly relevant movies and a nicely filled schedule for any scheduled time over about 10 hours.

REFERENCES

- [1] "American time use survey summary." <https://www.bls.gov/news.release/atus.nr0.htm>, Jun 2019.
- [2] W. Lee, "People spend more time on mobile devices than tv, firm says." <https://www.latimes.com/business/la-fi-ct-people-spend-more-time-on-mobile-than-tv-20190619>, Jun 2019.
- [3] O. US Department of Labor and various others, "Interim guidance for businesses and employers to plan and respond to coronavirus disease 2019 (covid-19)." <https://www.cdc.gov/coronavirus/2019-ncov/community/guidance-business-response.html>, Apr 2020.
- [4] W. Jiang, M. Murugesan, C. Clifton, and L. Si, "t-plausibility: Semantic preserving text sanitization," in *2009 International Conference on Computational Science and Engineering*, vol. 3, pp. 68–75, 2009.
- [5] "similarity: Elasticsearch reference [7.6]." <https://www.elastic.co/guide/en/elasticsearch/reference/current/similarity.html>.