Juggle: Hybrid Large-Scale Music Recommendation

José Devezas

Departamento de Ciência de Computadores,
Faculdade de Ciências, Universidade do Porto
Rua do Campo Alegre s/n, 4169-007 Porto, Portugal
&
Laboratório SAPO/U.Porto,
Faculdade de Engenharia, Universidade do Porto
Rua Dr. Roberto Frias, s/n, 4200-465 Porto, Portugal

joseluisdevezas@gmail.com

May 17, 2013

Content

Introduction

Who and When: Road as a Researcher

My name is José Devezas. I have been a research scientist for over three years, working at Laboratório SAPO/U.Porto, as well as CRACS/INESC TEC in projects involving:

- Community Detection
- Network Analysis
- Information Retrieval
- Data Visualization
- Text Mining
- Machine Learning
- Recommender Systems

Why: The Love for Connecting Knowledge Using Graphs

I've worked in some different areas of knowledge around intelligent information systems, but I have always focused on using graphs to solve problems:

- I have directly analyzed networks and their community structure.
- In information retrieval, I have focused on link analysis, using link-based features to improve search (e.g. worked with Sérgio Nunes and Cristina Ribeiro on applying the h-index from bibliometrics to blogs, comparing it with the in-degree as baseline).

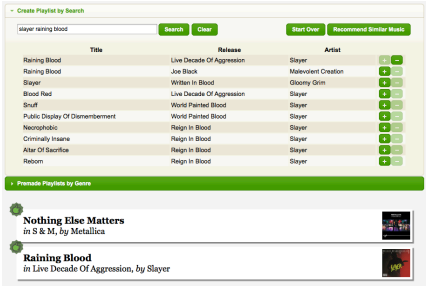
- I have worked on data visualization, applying some of the state of the art methodologies to the visualization of networks.
- I worked with multidimensional networks defined by the co-occurrence of entities, mined from online news, and experimented with community detection in multidimensional networks.
- Lately, I've been working with multimodal networks, implementing a knowledge-based recommender system: Juggle.

PhD in the Context of Laboratório SAPO/U.Porto

- I'm also a first year Computer Science PhD student at the Faculty of Sciences of the University of Porto.
- My PhD work is on **community-driven music discovery**.
- The main goal we've set for the first year is to **understand how people listen to and discover music, not only as an individual but also as a group**.
- For this, we've been using Last.fm public datasets, but it would be a lot more interesting to explore a different dataset, specifically MusicBox, which should be representative of the Portuguese people's tastes.

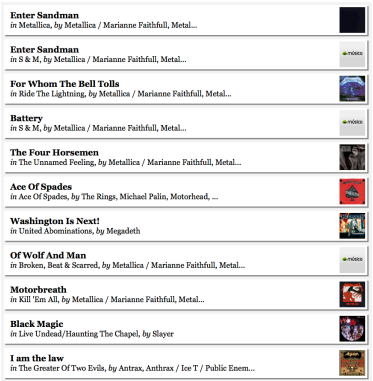
Hybrid Large-Scale Music Recommendation

Data and Goals



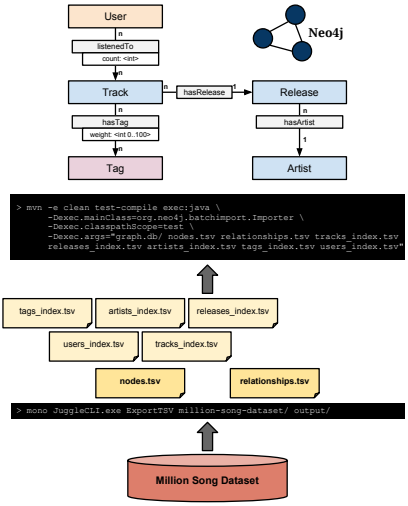
- Music collection: Million Song Dataset.
- But the goal is to adapt for MusicBox (12 million songs).
- In our demo, users decide on a playlist reflecting their tastes by either:
  - Selecting a premade playlist based on musical genre.
  - Searching for music based on available textual features, using an adaptation of Dpikt's retrieval methodology.
- Juggle uses its knowledge-based recommender to suggest the best songs for the given user playlist.

Basic Approach



- The focus is on music discovery, so we boost results in the long tail.
- The first algorithm we've come up with uses two features: tags and users.
- The IDF is precomputed for each tag and, during recommendation, tags are weighted according to their TF-IDF within each track.
- The tag-based recommendation module takes two parameters:
  - Fraction of top tags to consider for each input track.
  - Fraction of top tracks to consider for the filtered tags of each input track.
- The user-based recommendation module takes advantage of user play counts to recommend similar songs (traditional collaborative filtering).
- The results of the two modules are combined using a frequentist approach.

Our Setup



- Million Song Dataset is mostly **text files and SQLite** databases containing track metadata, tags and audio features.
- We used **Mono/C#** to prepare our starting graph.
  - At first, we tried directly reading the data as we loaded it to **Neo4j** (our graph database).
  - This was not working, so we decided to first create a nodes and relationships TSV file, along with one TSV files for each index associated with different types of nodes or relationships.
  - This way, we could skip index querying during insertion, since node IDs were precomputed during TSV generation, which enabled a much faster batch insertion using the batch-import tool available online.

Preparing the Music Graph

```
/* ALGORITHM 1: Precomputing the tags IDF. */

D = g.idx("tracks")[[trackId:"%query%*"]].count();
g.idx("tags")[[name:"%query%*"]].sideEffect{
  c = it.inE("hasTag").count();
  it.setProperty("idf", c == 0 ? 0 : Math.log(D / c))
};
```

- This is Gremlin, a DSL (domain-specific language) for graph traversal.
- It's an implementation of Tinkerpop's Pipes for Blueprints-accessible graph databases, written in Groovy.
- If you like the Unix command line, you'll like Gremlin. It's similarly designed as pipeline, so your train of thought should be the same, assuming you know what you want to do with the graph.

- The Inverse Document Frequency (IDF) is higher for tags that are less common.
  - $IDF(tag, tracks) = log \frac{|tracks|}{|t \in tracks : tag \in tags(t)|}$
  - Since, this value is query-independent, we precompute it for each tag using ALGORITHM 1.
  - First, we count the number of documents (tracks).
  - Then, we count the number of tracks that use the given tag.
  - And finally, for each tag, we calculate the corresponding IDF.

```
/* ALGORITHM 2: Community detection with label propagation. */

// Each node starts as a single member of its own community.
g.idx("tracks")[[trackId:"%query%*"]].sideEffect{
  it.setProperty("propagationLabel", it.getId()) };

// We need at least 5 iterations to get 95% accuracy.
limit = 10;
rand = new Random();
(1..limit).each{ i ->
  g.idx("tracks")[[trackId:"%query%*"]].sideEffect{
      neighborLabelCount = [:];
      it.outE("hasTag").has("weight", T.gte, 80.0f).inV
        .inE("hasTag").has("weight", T.gte, 80.0f).outV
        .except([it]).propagationLabel
        .groupCount(neighborLabelCount).iterate();
      max = neighborLabelCount.values().max();
      topLabels = neighborLabelCount.grep{it.value == max};
      newLabel = topLabels.size() == 0 ?
          it.getProperty("propagationLabel") :
        topLabels[rand.nextInt(topLabels.size())].key;
      it.setProperty("propagationLabel", newLabel);
    }.iterate();
}
```

- Community membership according to label propagation defined in ALGORITHM 2.
  - The label of each node is initialized with its ID.
  - Then we iterate through all nodes and choose the most frequent label among their direct neighbors (you can think of it as a voting system or simply as “peer pressure”).
  - Ties are broken randomly and uniformly.
  - Two variants of this algorithm exist:
    1. Iterate through the graph, but store the new labels separately to be updated at the end of the iteration.
    2. Or iterate through the graph and update labels directly.
  - They both converge to the same result, but we used the second variant since it costs less memory and is faster to converge.

- For each iteration, approximately 60M relationships are traversed more than once. This is clearly a problem. The good news is that this algorithm is “stupidly parallel”, since independent ego networks can be processed simultaneously in each iteration.
- According to the algorithm’s author, there should be a converge verification, so that it can be terminated before the defined number of iterations, however this is pointlessly costly and we didn’t do it.

Recommendation Algorithms

```
/* ALGORITHM 3: Recommending using tag relations of weight > 50. */

m = [:];
input = [] as Set;
query = 'TRGJQAP128F427B732 OR TRVATYZ128F427B733'

// Query tracks index, using Lucene syntax, to get starting tracks.
g.idx('tracks')[[trackId:'%query%' + query]].aggregate(input)
  .outE('hasTag').filter{it.getProperty('weight') > 50.0f}.inV
  .inE('hasTag').filter{it.getProperty('weight') > 50.0f}.outV
  .except(input).groupCount(m).iterate();

m.sort{-it.value}.take(20).collect{
  t = it.key;
  r = t.out('hasRelease').next();
  a = r.out('hasArtist').next();
  aName = r.out('hasArtist').collect{it.name}.join(', ');
  [
    [t.trackId, t.songId, t.title],
    [r.releaseId, r.name],
    [a.artistId, aName]
  ]
};
```

- Our baseline approach was to use similarity by tags and users to recommend new songs:
  - The total universe of tags used in the input playlist appears frequently in the recommended tracks.
  - Users who listened to each of the input playlist's tracks also listened to the recommended tracks.
- Recommender systems are information filtering systems, thus we gain from limiting data.
- Gremlin works with iterators, so it's lazy. For example, if you run a Gremlin query in the Gremlin/Groovy console, it returns an iterator, however, by default, the console executes iterators if they are the last command provided during input (hence the call to *iterate()* you see on the left).

Recommendation Algorithms

```
/* ALGORITHM 3: Recommending using tag relations of weight > 50. */

m = [:];
input = [] as Set;
query = 'TRGJQAP128F427B732 OR TRVATYZ128F427B733'

// Query tracks index, using Lucene syntax, to get starting tracks.
g.idx('tracks')[[trackId:'%query%' + query]].aggregate(input)
  .outE('hasTag').filter{it.getProperty('weight') > 50.0f}.inV
  .inE('hasTag').filter{it.getProperty('weight') > 50.0f}.outV
  .except(input).groupCount(m).iterate();

m.sort{-it.value}.take(20).collect{
  t = it.key;
  r = t.out('hasRelease').next();
  a = r.out('hasArtist').next();
  aName = r.out('hasArtist').collect{it.name}.join(', ');
  [
    [t.trackId, t.songId, t.title],
    [r.releaseId, r.name],
    [a.artistId, aName]
  ]
};
```

- We start from the queried track nodes.
- We traverse outwards through the *hasTag* relationships with weight higher than 50, thus returning tags that are used more than 50% of the times to represent a track.
- We traverse inwards through the *hasTag* relationships with weight higher than 50, thus returning tracks that use the given tag more than 50% of the times compared to other tags.
- We rank tracks by the number of times our walker passes by them.

```
/* ALGORITHM 4: Recommending using top 50% TF-IDF tags/tracks. */

Gremlin.defineStep('topTags', [Vertex, Pipe], {
  Float topPercent -> _().transform{
    n = it.outE('hasTag').count();
    it.outE('hasTag').sideEffect{w = it.getProperty('weight')}
      .inV.filter{it.getProperty('idf') != null}
      .sort{-(w / 100.0f * it.getProperty('idf'))}
      .take((int)Math.round(n * topPercent))}.scatter});

Gremlin.defineStep('topTracks', [Vertex, Pipe], {
  Float topPercent -> _().filter{it.getProperty('idf') != null}
    .sideEffect{idf = it.getProperty('idf')}.transform{
      n = it.inE('hasTag').count();
      it.inE('hasTag').sideEffect{w = it.getProperty('weight')}
        .outV.sort{-(w / 100.0f * idf)}
        .take((int)Math.round(n * topPercent))}.scatter});

m = [:]; input = []; results = [];
query = 'TRGJQAP128F427B732 OR TRVATYZ128F427B733'

g.idx('tracks')[[trackId:'%query%' + query]].aggregate(input)
  .topTags(0.5).topTracks(0.5).except(input)
  .groupCount(m).iterate();

m.sort{-it.value}.take(20).collect{
  t = it.key;
  r = t.out('hasRelease').next();
  a = r.out('hasArtist').next();
  aName = r.out('hasArtist').collect{it.name}.join(', ');
  [
    [t.trackId, t.songId, t.title],
    [r.releaseId, r.name],
    [a.artistId, aName]
  ]
};
```

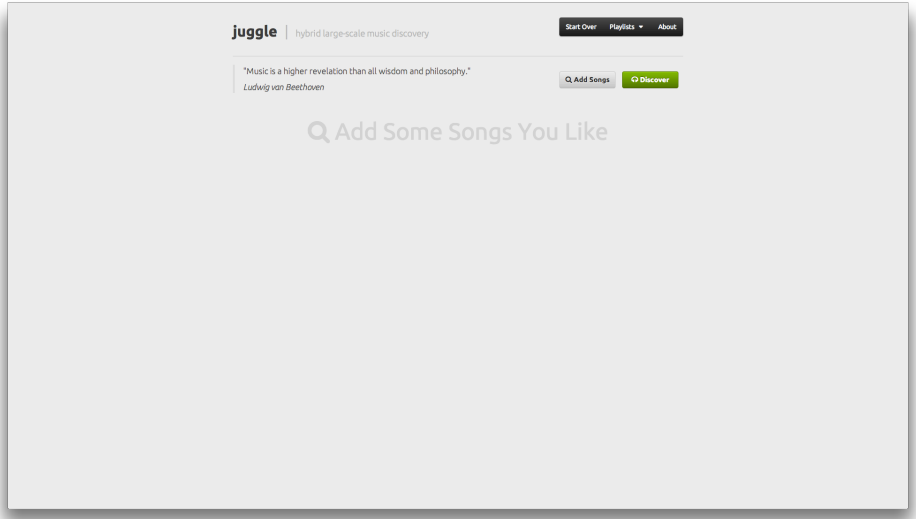
- The final algorithm we are going to present is similar to the previous one, except now the traversals are filtered in a different way, using custom pipes.
- In ALGORITHM 4, you can see the pipes *topTags* and *topTracks* which take a floating value between 0 and 1, representing the fraction of items to traverse.
- So, only the top fraction of tags or tracks, according to the TF-IDF, is traversed.
- For example, if a track has 20 tags and we use 0.1 in the *topTags* pipe, then only the top 10 tags according to the TF-IDF are returned and traversed.
- The results of this algorithm are quite different from ALGORITHM 3, the main differences being the lack of artist repetition (previous results mostly focused on other tracks from the same artists), and the boost of rarer tracks to the top, which is better for discovery, as less obvious results are returned.

Latest Additions to the Graph

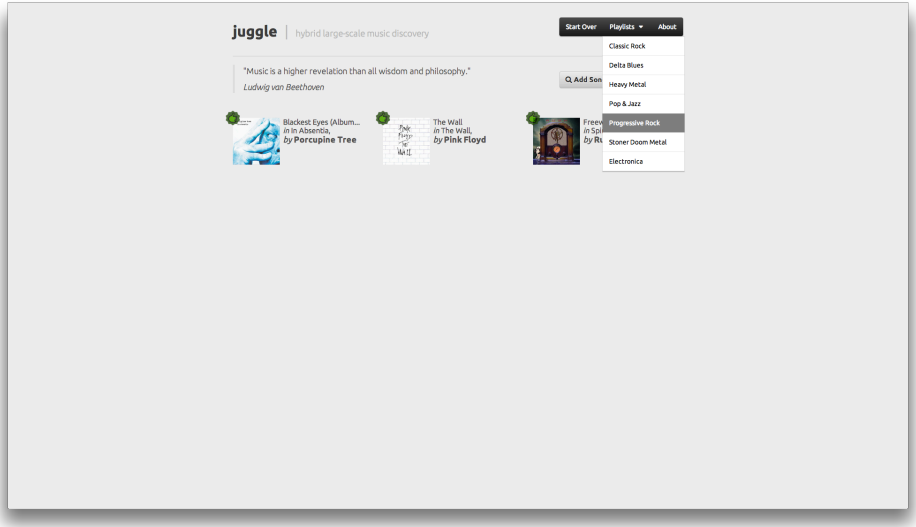
- When you visited with us at Porto, we were about to start working on using the audio features for music recommendation.
- As we expected, this was a bit of a challenge, mainly because of scale and format.
  - Scale $\Rightarrow$ A different strategy was needed to compare songs among each other, using audio features, since this data is dense, as opposed to the tag and user information, which is sparse and therefore resulted in a more treatable problem.
  - Format $\Rightarrow$ Our knowledge model was supported on a graph database, which essentially means that it's formatted as linked data, therefore requiring us to generate some kind of relationship that is usable during recommendation.
- We decided to create new relationships in the graph to illustrate direct connections between tracks by comparing them through audio similarity.

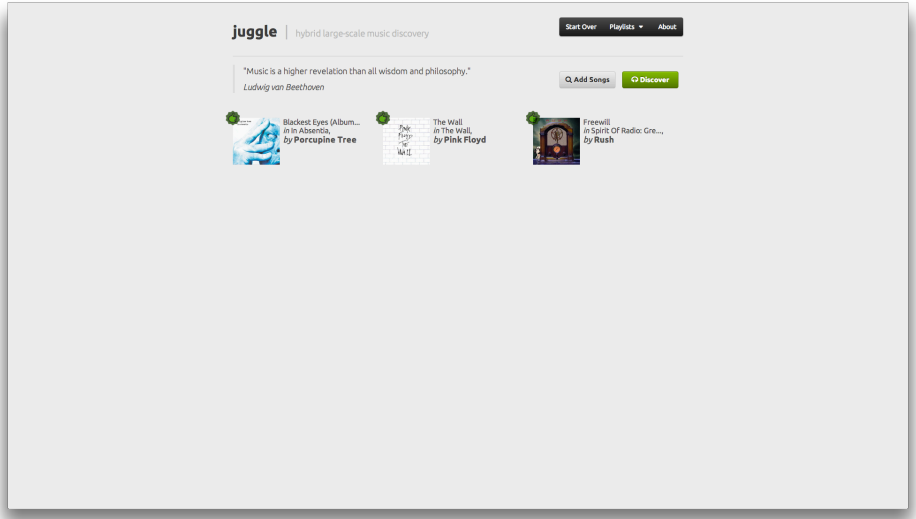
- Naively doing this would result in about $1\text{M}^2 = 1,000,000,000,000$ calculations of similarity.
- Assuming each operation takes about 0.2 seconds, the whole process would take nearly $2,315,000$ days to complete.
- For this, we've adapted Filipe Coelho's PhD work on text illustration and image retrieval to the context of music retrieval.
- We generated a trie structure supported on MongoDB, based on track pivoting, to prepare the workbench for quickly obtaining the 10 tracks with the most similar audio signal to a given track.
- This enabled us to generate a *k*-Nearest Neighbor Graph, establishing a ranked relationship between tracks according to their audio features.
- Since any relationship can easily be incorporated into the recommender engine, we believe this is an extensible and interesting way, based on a knowledge graph, to implement music recommendation.

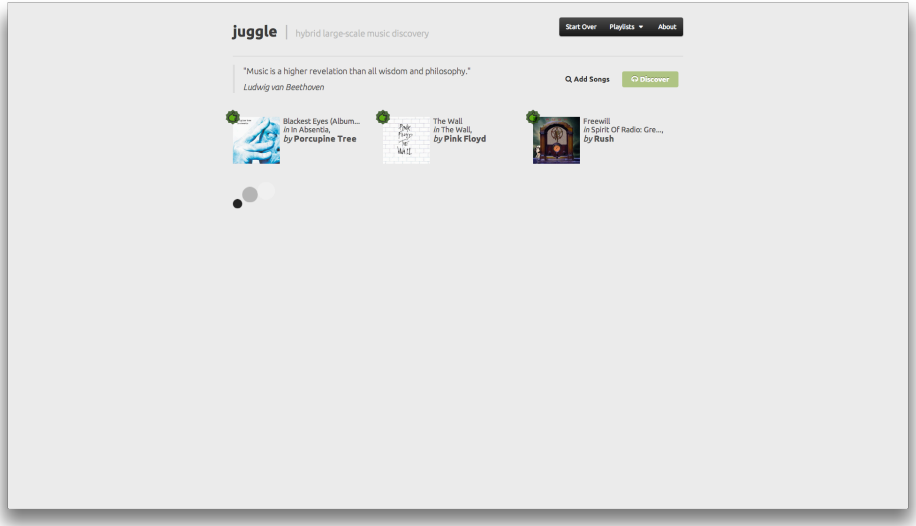
The New UI (Built with SAPO Ink)

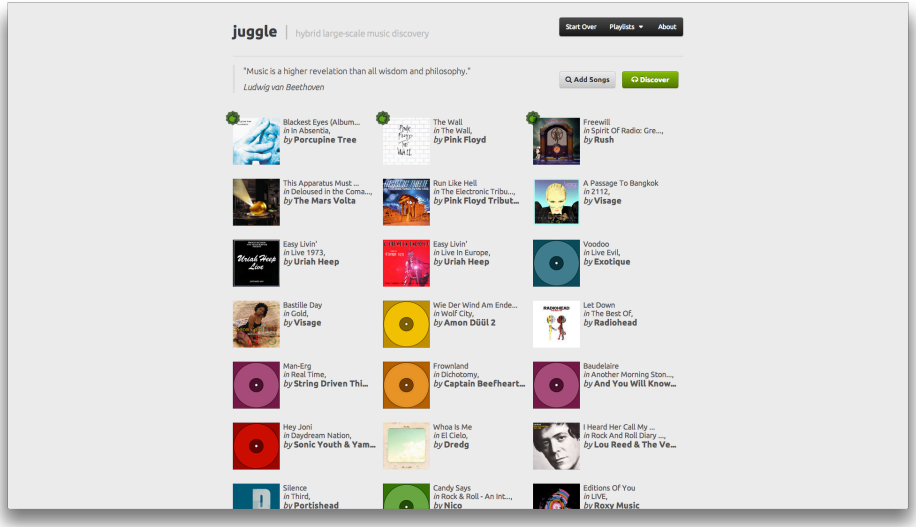

juggle
hybrid large-scale music discovery
Start Over
Playlists
About
"Music is a higher revelation than all wisdom and philosophy."
Ludwig van Beethoven
Add Songs
Discover
Add Some Songs You Like

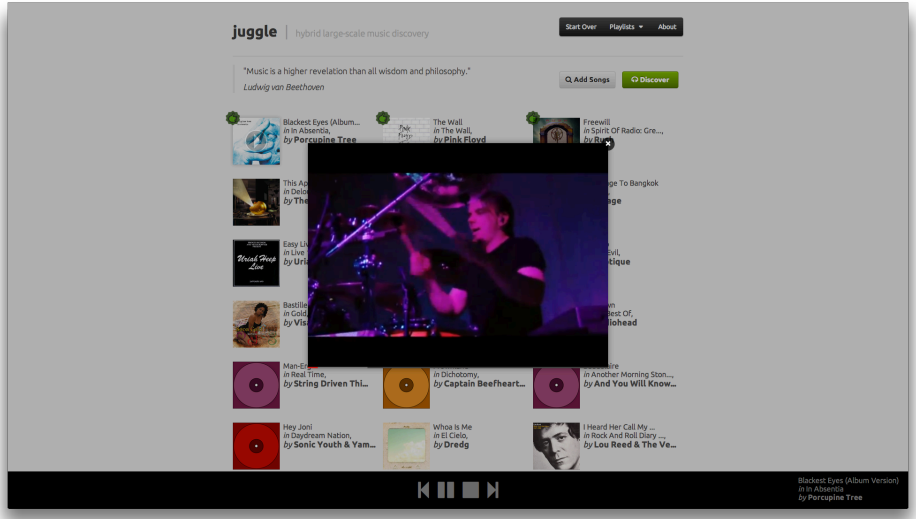
Starting From a Curated Playlist


juggle
hybrid large-scale music discovery
Start Over
Playlists
About
Classic Rock
Delta Blues
Heavy Metal
Pop & Jazz
Progressive Rock
Stoner Doom Metal
Electronica
"Music is a higher revelation than all wisdom and philosophy."
Ludwig van Beethoven
Add Son
Blackest Eyes (Album...
in In Absentia,
by Porcupine Tree
The Wall
in The Wall,
by Pink Floyd

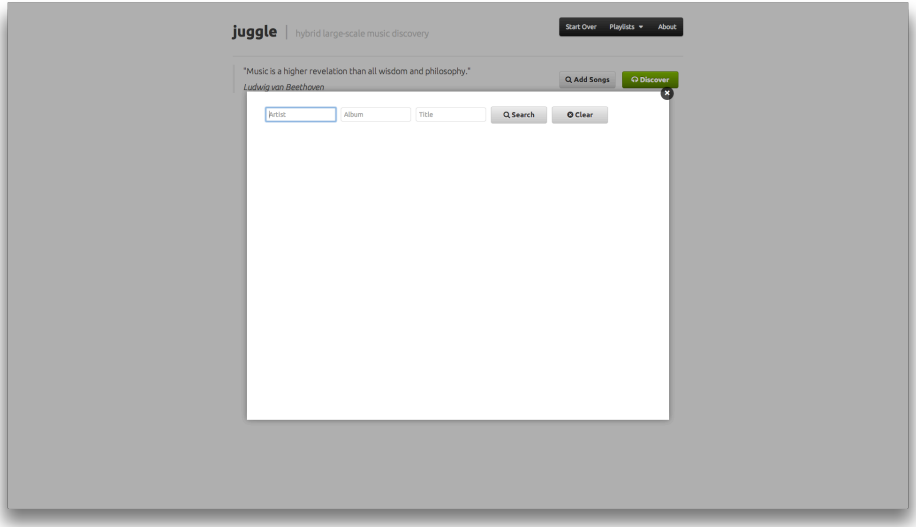

juggle
hybrid large-scale music discovery
Start Over
Playlists
About
"Music is a higher revelation than all wisdom and philosophy."
Ludwig van Beethoven
Add Songs
Discover
Blackest Eyes (Album...
in In Absentia,
by Porcupine Tree
The Wall
in The Wall,
by Pink Floyd
Freewill
in Spirit Of Radio: Gre...,
by Rush

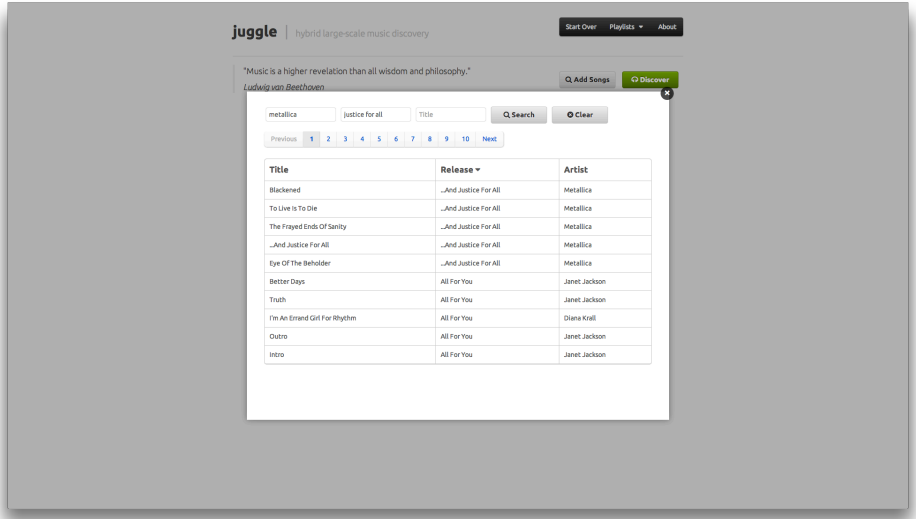

juggle
hybrid large-scale music discovery
Start Over
Playlists
About
"Music is a higher revelation than all wisdom and philosophy."
Ludwig van Beethoven
Add Songs
Discover
Blackest Eyes (Album...
in In Absentia,
by Porcupine Tree
The Wall
in The Wall,
by Pink Floyd
Freewill
in Spirit Of Radio: Gre...,
by Rush

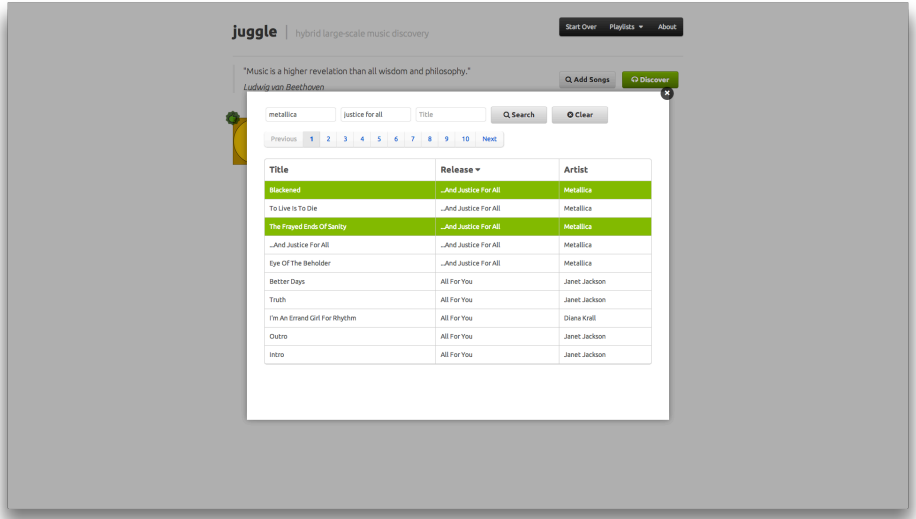

juggle
hybrid large-scale music discovery
Start Over
Playlists
About
"Music is a higher revelation than all wisdom and philosophy."
Ludwig van Beethoven
Add Songs
Discover
Blackest Eyes (Album...
in In Absencia,
by Porcupine Tree
The Wall
in The Wall,
by Pink Floyd
Freewill
in Spirit Of Radio: Gre...,
by Rush
This Apparatus Must ...
in Deloused in the Coma...,
by The Mars Volta
Run Like Hell
in The Electronic Tribu...,
by Pink Floyd Tribut...
A Passage To Bangkok
in 2112,
by Visage
Easy Livin'
in Live 1973,
by Uriah Heep
Easy Livin'
in Live In Europe,
by Uriah Heep
Voodoo
in Live Evil,
by Exotique
Bastille Day
in Gold,
by Visage
Wie Der Wind Am Ende...
in Wolf City,
by Amon Düül 2
Let Down
in The Best Of,
by Radiohead
Man-Erg
in Real Time,
by String Driven Thi...
Frownland
in Dichotomy,
by Captain Beefheart...
Baudelaire
in Another Morning Ston...,
by And You Will Know...
Hey Joni
in Daydream Nation,
by Sonic Youth & Yam...
Whoa Is Me
in El Cielo,
by Dredg
I Heard Her Call My ...
in Rock And Roll Diary ...
by Lou Reed & The Ve...
Silence
in Third,
by Portishead
Candy Says
in Rock & Roll - An Int...,
by Nico
Editions Of You
in LIVE,
by Roxy Music

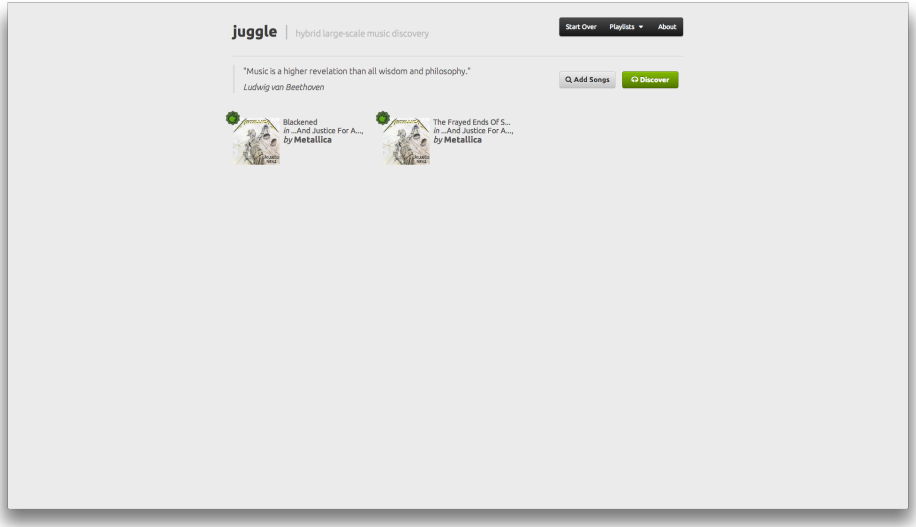

juggle
hybrid large-scale music discovery
Start Over
Playlists
About
"Music is a higher revelation than all wisdom and philosophy."
Ludwig van Beethoven
Add Songs
Discover
Blackest Eyes (Album...
in In Absencia,
by Porcupine Tree
The Wall
in The Wall,
by Pink Floyd
Freewill
in Spirit Of Radio: Gre...,
Man-E...
in Real Time,
by String Driven Thi...
in Dichotomy,
by Captain Beefheart...
in Another Morning Ston...,
by And You Will Know...
Hey Joni
in Daydream Nation,
by Sonic Youth & Yam...
Whoa Is Me
in El Cielo,
by Dredg
I Heard Her Call My ...
in Rock And Roll Diary ...
by Lou Reed & The Ve...
Blackest Eyes (Album Version)
in In Absencia
by Porcupine Tree

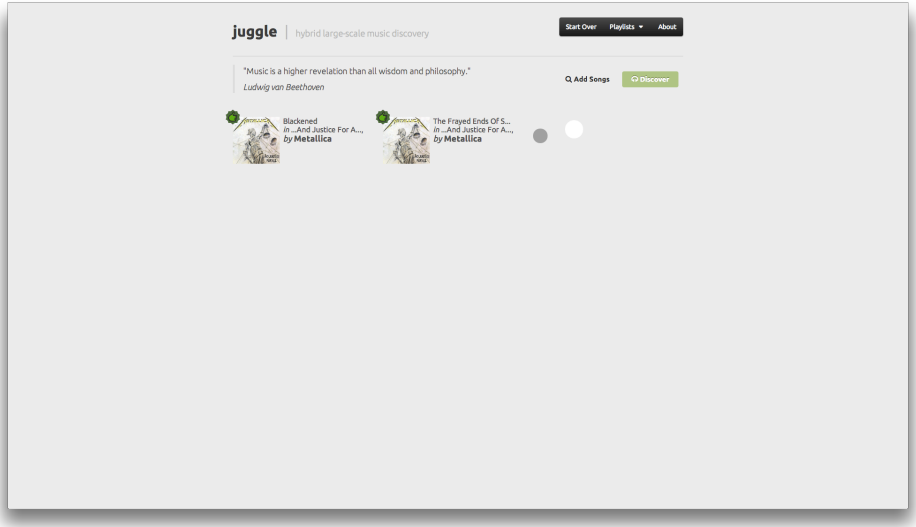
Searching and Adding Songs to Start Playlist

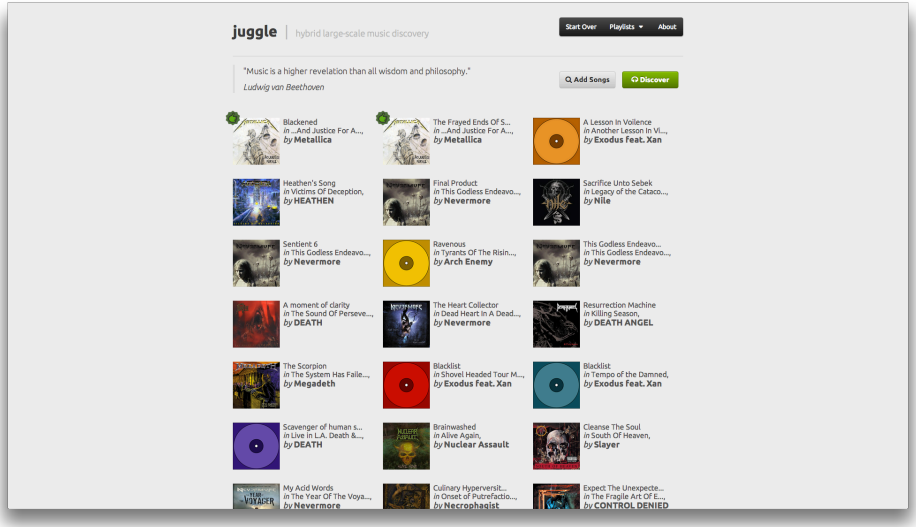

juggle
hybrid large-scale music discovery
Start Over
Playlists
About
"Music is a higher revelation than all wisdom and philosophy."
Ludwig van Beethoven
Add Songs
Discover
Artist
Album
Title
Search
Clear

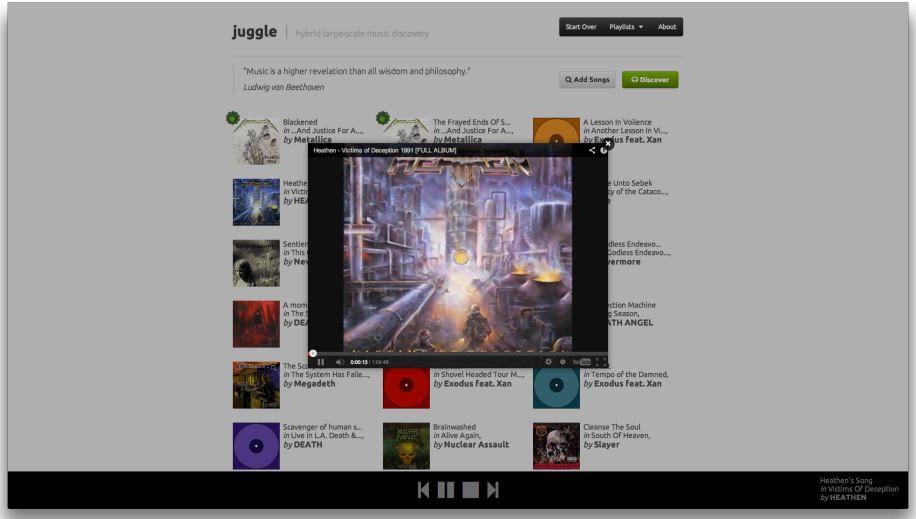

juggle
hybrid large-scale music discovery
Start Over
Playlists
About
"Music is a higher revelation than all wisdom and philosophy."
Ludwig van Beethoven
Add Songs
Discover
metallica
justice for all
Title
Search
Clear
Previous
1
2
3
4
5
6
7
8
9
10
Next
Title
Release
Artist
Blackened
...And Justice For All
Metallica
To Live Is To Die
...And Justice For All
Metallica
The Frayed Ends Of Sanity
...And Justice For All
Metallica
...And Justice For All
...And Justice For All
Metallica
Eye Of The Beholder
...And Justice For All
Metallica
Better Days
All For You
Janet Jackson
Truth
All For You
Janet Jackson
I'm An Errand Girl For Rhythm
All For You
Diana Krall
Outro
All For You
Janet Jackson
Intro
All For You
Janet Jackson


juggle
hybrid large-scale music discovery
Start Over
Playlists
About
"Music is a higher revelation than all wisdom and philosophy."
Ludwig van Beethoven
Add Songs
Discover
metallica
justice for all
Title
Search
Clear
Previous
1
2
3
4
5
6
7
8
9
10
Next
Title
Release
Artist
Blackened
...And Justice For All
Metallica
To Live Is To Die
...And Justice For All
Metallica
The Frayed Ends Of Sanity
...And Justice For All
Metallica
...And Justice For All
...And Justice For All
Metallica
Eye Of The Beholder
...And Justice For All
Metallica
Better Days
All For You
Janet Jackson
Truth
All For You
Janet Jackson
I'm An Errand Girl For Rhythm
All For You
Diana Krall
Outro
All For You
Janet Jackson
Intro
All For You
Janet Jackson


juggle
hybrid large-scale music discovery
Start Over
Playlists
About
"Music is a higher revelation than all wisdom and philosophy."
Ludwig van Beethoven
Add Songs
Discover
Blackened
In ...And Justice For A...,
by Metallica
The Frayed Ends Of S...
In ...And Justice For A...,
by Metallica


juggle
hybrid large-scale music discovery
Start Over
Playlists
About
"Music is a higher revelation than all wisdom and philosophy."
Ludwig van Beethoven
Add Songs
Discover
Blackened
In ...And Justice For A...,
by Metallica
The Frayed Ends Of S...
In ...And Justice For A...,
by Metallica


juggle
hybrid large-scale music discovery
Start Over
Playlists
About
"Music is a higher revelation than all wisdom and philosophy."
Ludwig van Beethoven
Add Songs
Discover
Blackened
in ...And Justice For A...,
by Metallica
The Frayed Ends Of S...
in ...And Justice For A...,
by Metallica
A Lesson In Vollence
in Another Lesson In Vi...,
by Exodus feat. Xan
Heathen's Song
in Victims Of Deception,
by HEATHEN
Final Product
in This Godless Endeavo...,
by Nevermore
Sacrifice Unto Sebek
in Legacy of the Cataco...,
by Nile
Sentient 6
in This Godless Endeavo...,
by Nevermore
Ravenous
in Tyrants Of The Risin...,
by Arch Enemy
This Godless Endeavo...
in This Godless Endeavo...,
by Nevermore
A moment of clarity
in The Sound Of Perseve...,
by DEATH
The Heart Collector
in Dead Heart In A Dead...,
by Nevermore
Resurrection Machine
in Killing Season,
by DEATH ANGEL
The Scorpion
in The System Has Faile...,
by Megadeth
Blacklist
in Shovel Headed Tour M...,
by Exodus feat. Xan
Blacklist
in Tempo of the Damned,
by Exodus feat. Xan
Scavenger of human s...
in Live In L.A. Death &...,
by DEATH
Brainwashed
in Alive Again,
by Nuclear Assault
Cleanse The Soul
in South Of Heaven,
by Slayer
My Acid Words
in The Year Of The Voya...,
by Nevermore
Culinary Hyperversit...
in Onset of Putrefactio...,
by Necrophagist
Expect The Unexpecte...
in The Fragile Art Of E...,
by CONTROL DENIED


juggle
hybrid large-scale music discovery
Start Over
Playlists
About
"Music is a higher revelation than all wisdom and philosophy."
Ludwig van Beethoven
Add Songs
Discover
Blackened
in ...And Justice For A...,
by Metallica
The Frayed Ends Of S...
in ...And Justice For A...,
by Metallica
A Lesson In Vollence
in Another Lesson In Vi...,
by Exodus feat. Xan
Heathen - Victims of Deception 1991 [FULL ALBUM]
0:00:13 / 1:04:48
in The System Has Falle...,
by Megadeth
in Shovel Headed Tour M...,
by Exodus feat. Xan
in Tempo of the Damned,
by Exodus feat. Xan
Scavenger of human s...
in Live In L.A. Death &...,
by DEATH
Brainwashed
in Alive Again,
by Nuclear Assault
Cleanse The Soul
in South Of Heaven,
by Slayer
Heathen's Song
in Victims Of Deception
by HEATHEN

Conclusions

Additional Knowledge

- The complete graph, containing track metadata, tags, user information and audio features, just takes about 10 GB of space out of the 300 GB for the whole music collection.
  - Loading this into memory should be trivial, which ought to translate into a considerable speedup.
  - (We weren't able to test this with our graph for lack of resources and lack of time to generate a smaller subset.)
- According to other authors (e.g. Ben Fields), **community structure** is good to predict genre.
  - We have computed the community membership for the track–track graph induced by tag co-occurrence according to the **label propagation algorithm**.
  - This hasn't been further explored due to lack of time, but the algorithm has been shown before and is available to use in the **Gremlin language**.
- For **audio features**, we focused on the pitch and timbre, which are represented by an array of twelve real numbers for each segment in a song.
  - The problem is that the number of segments is variable for each song.
  - To calculate similarity, vectors need to be the same dimension.
  - One good way to solve this is to calculate the **mean pitch and timbre along with 78 covariances for each** ($\frac{12\times(12-1)}{2} + 12$), representing all the meaningful combinations of pitches and timbres (including the covariance of the variable with itself, which represents the variance).

Closing Remarks

- Even though this is not our main focus, we point out that the performance of recommendation algorithms still needs to be improved.
  - We note however that, since a graph database possesses index-free adjacency, once these problems are solved, increasing the size of the collection should only be a matter of adding resources (disk space and additional computers for a Neo4j highly available cluster).
  - Our algorithms are based on traversals, so solving the problem is a matter of decreasing the number of visited nodes (a more aggressive filtering technique could be an option).
- As future work, we would like to focus on community-driven music discovery, which is an idea we are developing for my PhD.
  - The first approach is to understand how people listen to music and identify discovery patterns.
  - For that, we need access to user data letting us know which user listened to what song when.
  - We'd love to study MusicBox data instead of the traditional Last.fm dataset, which generates value for our work as well as knowledge that will contribute to your work.

Thank you!