Exposé zur Bachelorabeit

Christopher Pahl

August 14, 2013

1 Motivation

Heutzutage gibt es viele Plattformen die dabei helfen wollen neue Musik zu entdecken. Eine dieser Plattformen ist beispielsweise `https://www.last.fm`. Nutzer bekommen dabei basierend auf ihrem Hörverhalten Vorschläge was sie als nächstes anhören könnten - leider ist die Software in ihrem Kern keine *FOSS* und zudem abhängig von den zentralen Servern des Betreibers. Daher wäre ein freies, clientseitiges System wünschenswert.

Die Zielgruppe wären hierbei Entwickler von Musicplayern oder vergleichbarer Software. Auch ein *Standalone-Tool* wäre denkbar welches von normalen Usern genutzt werden kann.

2 Themenstellung

Erstellen einer Softwarebibliothek zur automatisierten Empfehlung von Musik basierend auf der Musik-Sammlung eines Nutzers und dessen Hör-Gewohnheiten.

Im Speziellen soll das System dabei die Musik-Datenbank des Nutzers importieren und dessen Hör-Gewohnheiten beobachten können. Daraus sollen dann Empfehlungen (N ähnliche Songs zu Stück X) und Dynamische Playlisten (Playlist die zu den zuletzt Gehörtem passt) abgeleitet werden können.

Beteiligte Disziplinen der Informatik:

- Graphentheorie (Aufbau/Wartung der internen Graphenstruktur)
- Datamining-Algorithmen (Ähnlichkeitsmaß, Machinenlernende Systeme)
- Maschinelles Lernen (Prüfen ob Empfehlungen angenommen wurden)
- Audioanalyse (Moodbar (siehe Literatur [1)])

2.1 Namensgebung:

Die Library soll *libmunin* heißen, wie Odin's Rabe *Munin*:

> *Munin gehört zum altnordischen Verb muna (denken an, sich erinnern), der Name Munin bedeutet folglich ,,die Erinnerung".*

Siehe auch: `http://de.wikipedia.org/wiki/Hugin_und_Munin`

3 Geplantes Vorgehen

Es soll eine Prototyp für unixoide Betriebssysteme in *Python* implementiert werden. Wenn noch ausreichend Zeit bleibt (leider eher unwahrscheinlich) soll dieser Prototyp in eine *C-Bibliothek* umgesetzt werden. Zuerst soll die Grundfunktionalität der Bibliothek stehen, danach können dann *spezielle Provider*, welche beispielsweise eine *Moodbar-Anaylse* oder *Liedtexte vergleichen*, implementiert werden. Wenn die Library rechtzeitig in einem annehmbaren Zustand sein, so soll die Library in einem *MPD-Client zum Einsatz* kommen.

Später soll dann die *Theorie* mit Zuhilfenahme von *Visualisierungen* detailliert in der Bachelor-Arbeit beschrieben werden und, nach Möglichkeit, wird das System an *echten Menschen* und verschiedenen Musik-Sammlungen getestet um zu sehen ob es auch praxistauglich ist.

4 Aufteilung/Zeitplanung

Relative Zeitabschätzung in (Klammern):

Projektarbeit:

1. Implementierung von *libmunin*. (ca. 55%)
2. [*Optional*] Beispielanwendung in einem MPD-Client (voraussichtlich ,,*Moosecat*"). (ca. 15%)

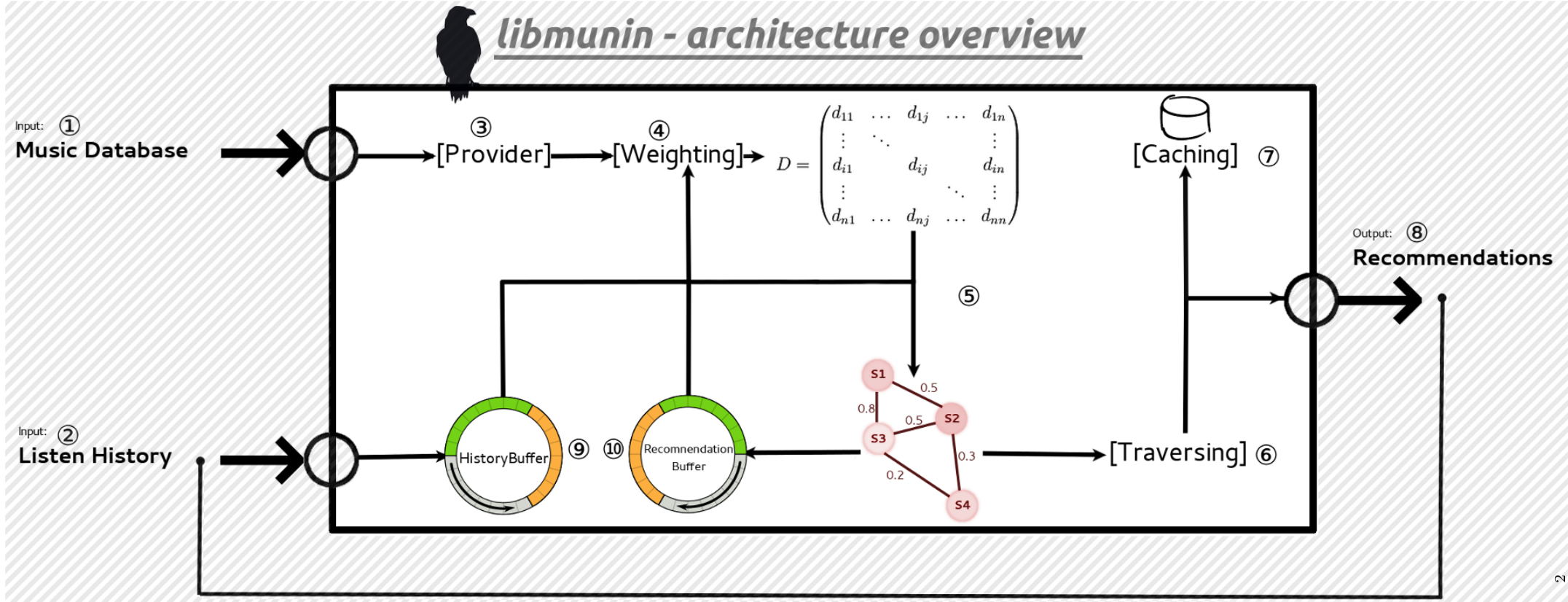
Bachelorarbeit:

1. Beschreibung der Theorie/Algorithmik mit Visualisierungen. (ca. 25%)
2. [*Optional*] Test mit echten Nutzern und verschiedenen Musik-Sammlungen. (ca. 5%)

5 Literatur

1. *Moodbar* (`http://cratoo.de/amarok/ismir-crc.pdf`)
2. *A Mood Based Music Classification and Exploration System* (`http://www.google.de`)
3. *Automatic Playlist Generation via Music Mood Analysis* (`http://www.google.de`)
4. *Polysound* (`http://grupoweb.upf.edu/~luca.chiarandini/personal/v0/index.html#projectsPolysound`)

libmunin - architecture overview



Inputs

① Music Database:

- The library user feeds songs from the music database (only their metadata).
- The library user also sets an AttributeMask ◦ a set of features most songs have (like a Artist, Title or Genre).
- The internal database can be updated at any time. Also caching it is possible.

② Listen History:

- The library user can feed the last listened songs.
- These can be used to check the Recommendations the library gives.
- This is the main input of learning for libmunin.

Outputs

⑧ Recommendations:

- Giving Recommendation for a Song
- Create dynamic Playlist based on the *Listen History*.
- Ranking of Search Results based on the Similarity of Songs.

Internal:

③ Provider:

- **Song :=** a set of attributes
- **Attribute :=** A feature specific to a Song (e.g. a *Title, Moodbar, ...*)
- **Provider** deliver these attributes (implemented by libmunin)

④ Weighting:

- **Distance** := *„Similarity"* of two songs.
- **DistanceFunction** := Computes a Distance between two Songs.
  The following must apply to a Distancefunction *D*:

  $D(i,j) = D(j,i)\ \forall\ i,j \in D$
  $D(i,i) = 1.0\ \forall\ i \in D$

- **Attributemask** := A common subset of attributes given by the user of the library, including a weighting for each attribute.

⑤ Distanzmatrix & Graph:

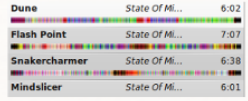
- **DistanceMatrix :=** NxN matrix of D(s1, s2) ∀ s1, s2 ∈ Songs
- Used as Lookup-Table and to build-up the graph.
- **Graph :=** Nodes are Songs; Edges are Distances; Every Song has at most X neighbors.

⑥ Traversing:

- Querying is done by traversing the graph.
- Possible queries:
  - **n**-Similar Songs to **x** (*Breadth-First Search* from **x**)
  - Simlarity of two Song **A** and Song **B** (*Shortest Distance*)
  - Ranking of Search Results (*Similarity with Search-Song* **q**)
- Graph adapts to the user's listening history by modifying edges.

⑦ Caching:

- Hard to calculate Attributes should be stored.
- This includes for example the moodbar:
- Implemented as a SQLite cache usable from the API.



An example of a moodbar

⑨ History Buffer:

- Ringbuffer with **N** Entries at max.
- Holds the latest **N** listened songs.
- Used to evaluate given Recommendations (*Followed* or *Declined*).

⑩ Recommendation Buffer:

- Ringbuffer with **M** Entries at max.
- Holds the latest **M** listened songs.
- Used to *punish* or *reward* songs in the graph.