Working Paper Series
ISSN 1177-777X

GRAPH-RAT
COMBINING DATA SOURCES IN MUSIC RECOMMENDATION SYSTEMS

Daniel McEnnis & David Bainbridge

Working Paper: 07/2008
July 28, 2008

Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

GRAPH-RAT: COMBINING DATA SOURCES IN MUSIC RECOMMENDATION SYSTEMS

Daniel McEnnis & David Bainbridge
University of Waikato, Hamilton, New Zealand
dm75,davidb@cs.waikato.ac.nz

July 28, 2008

Abstract

The complexity of music recommendation systems has increased rapidly in recent years, drawing upon different sources of information: content analysis, web-mining, social tagging, *etc.* Unfortunately, the tools to scientifically evaluate such integrated systems are not readily available; nor are the base algorithms available. This article describes Graph-RAT (Graph-based Relational Analysis Toolkit), an open source toolkit that provides a framework for developing and evaluating novel hybrid systems. While this toolkit is designed for music recommendation, it has applications outside its discipline as well. An experiment—indicative of the sort of procedure that can be configured using the toolkit—is provided to illustrate its usefulness.

1 Introduction

A trend in music recommendation systems research is to work with private data sets, custom algorithms, and incompatible evaluation metrics. This is neither a new issue nor a problem restricted to music recommendation research. It is an affliction that cuts across the whole field of music information retrieval, fueled by concerns of copyright infringement [6]. Comparisons between new music recommender systems and standard 'workhorse' algorithms are typically difficult to reproduce and standard evaluation metrics for music recommendation are not well defined. Furthermore, the need to start from scratch discourages researchers from building innovative context-aware and culturally-aware systems since these features require a sophisticated toolbox of algorithms before this kind of development can begin.

In this paper we describe an open source toolkit that helps alleviate some of these concerns, and—through a suite of algorithms covering data acquisition, content analysis, collaborative filtering, clustering, and evaluation algorithms—provides a significant boost to researchers new to the field. The techniques are drawn from several disciplines, including measurements from Social Network

Analysis (prestige and centrality [7]) in addition to computer-science staples such as machine learning. In Section 2 we review existing music recommendation systems, which leads to a set of requirements for the toolbox. Next we describe the implemented toolkit, starting with the fundamental structures devised for the toolkit (Section 3), followed by the suite of algorithms provided (Section 4) grouped by category. The utility objects supporting these algorithms (and which are available for other algorithms to use) are described next (Section 5). In Section 6 we provide details of an an experiment that compares different recommendation algorithms that illustrates of the sort of evaluation that can be setup and run using the software. We conclude with a summary of the paper.

2 Related work

Generally, music recommendation systems have used either content-based analysis or collaborative filtering analysis for generating playlists. In addition, a few systems have used user-supplied metadata and web-based data to augment content-based approaches.

Logan [11] uses a purely content-based approach, producing playlists either directly from similarity measures between songs, or similarity measures of a set of songs. Pauws [18] utilizes a more formal definition of constraints based on content-based features to generate playlists. Pampalk *et al.* [16] also uses content-based analysis but the playlists are altered by users skipping songs. Pampalk and Gasser [14] extend this system by replacing skipping behavior with explicit user ratings. Pandora[1] is an example of a commercial content-based recommendation system based on hand-crafted musical descriptions in its database based on an analogy of DNA. Tiemann *et al.* [20] produced a hybrid system which uses iterative classifications to tag music for recommendation.

Two of Chen and Chen's [3] recommendation algorithms utilize purely content-based approaches. Another of Chen and Chen's algorithms, also described in [3], utilizes a pure collaborative filtering approach. Crossen [4] utilizes user recommendations to determine the music to play in a shared space, filtering hand-picked genre classifications of music. Yoshii *et al.* [21] provide a hybrid approach where content-based analysis and collaborative filtering are calculated separately and then later integrated. Yoshii *et al.* extended this in [22] with genre preferences and ability to provide recommendations for new users. Anglade *et al.* [1] produced a peer-to-peer recommender that clustered users with similar tastes and streamed music from these cluster's shared musical collection. Last.FM[2] uses collaborative filtering based on watching users' listening habits directly with modifications made through explicit negative and positive ratings.

Pauws and Eggen [17] produced a systems that generates playlists from a single song using clustering songs similar in audio content but defines these playlists by learning from what songs the user removed. Other approaches have

[1] http://www.pandora.com
[2] http://www.last.fm

supplemented the traditional pure content-based approaches with web-derived context. Celma *et al.* [2] constructs networks of artists derived from targeted web searches to provide recommendations for users based on their Friend Of A Friend (FOAF) profiles. Sandvold *et al.* [19] extended this with tagging and content-based analysis. Pampalk and Goto [15] extended Sandvold *et al.*'s work by combining three similarity metrics.

2.1 Requirements

In order to address the concerns raised in the introduction, and factoring in related prior work, a toolkit for analysis is required similar in structure to the GATE toolkit for natural language processing [5].

Requirement i. This toolkit should be scriptable so that different algorithms can share the same data set and evaluation metrics without requiring a recompilation of the underlying system.

Requirement ii. The toolkit should have both machine readable and human readable descriptions of the algorithms it contains.

Requirement iii. The toolkit should be executable from a graphical user interface, through the command-line, and through embedding.

Requirement iv. This toolkit should provide a set of data acquisition algorithms for publicly available sources of social music context such as LiveJournal.

Requirement v. The toolkit must also provide implementations of the standard recommendation algorithms currently in use.

Requirement vi. The toolkit should provide sets of algorithms for clustering data and provide mechanisms so that algorithms can be run over arbitrarily nested subsets of the original data (making culturally-aware and context-sensitive recommendations easier to implement).

A review of existing graph toolkits [12] reached the conclusion that, while some parts of some of the toolkits matched these requirements, none of them provided a good match for all the requirements. Consequently implementation of a new toolkit, Graph-RAT, was undertaken—tempered, as much as possible, with the reuse of sub-systems from related open source software.

3 Toolkit Structure

Graph-RAT is written in Java and utilizes a blackboard structure [9] for analysis—the data is kept separate from the algorithms that act upon it. A scheduler loads the different algorithms and execute them against the data structure. Graph-RAT also incorporates a number of utility structures such as a custom webcrawler and reusable core objects. Each algorithm produces graphs, paths, actors, links, or properties. These terms are best explained in context below.

3.1 Data Structure

Graph-RAT's data structure is a graph. Each graph consists of actors and the links between them. In addition, graphs have globally unique ids and can contain sub-graphs. These sub-graphs are strict subsets of the actors and links of the parent with arbitrary nesting. Actors, links, and graphs can contain properties. Actors have a mode (such as "Artist" or "User") with unique ids within the mode. Links are likewise unique using a combination of link type, source, and destination actors.

Properties are pieces of data attached to a particular graph, actor, or link. Properties are multi-valued with immutable values and again are uniquely identified with the object they are paired with. Values can be Java objects of any class. Loading properties to and from XML is accomplished by the use of custom functions stored in a property registry by Java class type.

3.2 Algorithms

Algorithms in Graph-RAT are executable objects (byte-compiled Java class files) that provide a type of operation performed against a graph. These algorithms contain metadata listing all parameters, inputs, and outputs. Each parameter contains the class of the object that the parameter uses, the name, and the keyword associated with this parameter. All algorithms include input and output descriptors that list all actors, links, graphs, and properties that the algorithm reads and produces. Combined with immutable property values, this provides the blackboard book-keeping as described by Jensen [9] in addition to design requirement (requirement i). Each algorithm must be re-executable as it could be executed any number of times over any number of graphs depending on the scheduler algorithm used. This is not difficult to achieve in practice.

3.3 Scheduler

The scheduler loads all data acquisition and algorithm modules and executes each algorithm against the main graph section or one of its sub-graphs. The scheduler is responsible for deciding which algorithms execute and against which data source. This permits applications, for example, to perform automated segmentation of users into culturally-homogeneous sub-groups and then execute algorithms against each segment in turn. Schedulers can be non-deterministic; the default one provided is deterministic.

3.4 Entry Points

Graph-RAT provides three different mechanisms for interacting with the toolkit. Applications can be constructed with XML and executed from the command line. Applications can also be constructed dynamically in a GUI. Finally, Graph-RAT can be embedded in another Java application through a well defined API. Together this satisfies requirements (ii) and (iii).

4 Algorithms

In all following descriptions, N is the number of users, L is the number of links, and A is the number of artists.

4.1 Data Acquisition Algorithms

Graph-RAT provides a number of parsing tools for acquiring data from a number of sources. Each of these sources provides information that is useful for constructing music recommendations.

Crawl Live Journal. Retrieves from the LiveJournal website[3] a network of on-line users in FOAF (Friend of a Friend) format. Each profile contains information about the person, such as geographical location and lists of interests as well as a list of people this person knows. This acquisition module spiders from an initial list of user names and downloads them to a directory (requirement iv).

Crawl LastFM. Spiders LastFM web services - a website that collects social networking, tag, and playcount information on artists and songs. Each profile contains information about what has been tagged, what tags were used, personal information about the person, what artists and songs have been listened to, and how often they are listened to.

File Reader. Reads LiveJournal FOAF pages into a graph from a local directory. This might include 'dangling' links to actors outside of the set of pages downloaded. The FileReader2 variant uses a two pass system to remove dangling links from the data set.

Read Audio Files. Reads a directory of music, using jAudio to transform these music files into a feature vector which is then transformed into Weka Instances. This acquisition module also creates an actor for each file and attaches the Weka Instance to this actor as a property.

Read LastFM Profile. Reads a tar file containing the directory created by CrawlLastFM. This is about an order of magnitude faster than reading the files from the file system.

Read MemGraph XML. Restores from disk a previously serialized MemGraph object. Usually the fastest way to load a graph.

4.2 Machine Learning Algorithms

Machine learning algorithms provide a means for predicting what new music a person will like. Graph-RAT's machine learning algorithms transform links

[3] http://www.livejournal.com

between users and artists into artist vectors and then uses Weka[4] to make the predictions.

There are two varieties: single instance uses Weka to generate a model for every artist predicting whether a given artist is recommended for a particular user; and multi instance uses Weka to train a model for every artist using the artist vector of friends but not the user to recommend music. The latter is of interest for comparing global learning against the same information as local recommendation. For either variety the classifier that is used is supplied as a parameter.

4.3 Collaborative Filtering Algorithms

Graph-RAT provides three different collaborative filtering algorithms that are executed over all actors of a given mode (requirement v). All three work by transforming links between users and artists into an artist vector before performing calculations.

User to User. This performs nearest neighbor comparisons such that all users with a similarity measure larger than the threshold to a given user recommend their artist vector to the user. This is an $O(N^2)$ algorithm in time and $O(L)$ algorithm in space.

Item to Item. This calculates the artist to artist correlations and then recommends artists for each user as the sum of all correlations for artists in the user's artist vector. This algorithm is $O(AL)$ in time and $O(L)$ in space.

Associative Mining. This calculates whether a given artist is significantly more or less likely to be present, conditional on another artist compared with all occurrences of the artist. The recommendation for an artist is the sum of all correlated artists for a given artist present in the user vector. This algorithm is $O(A^2L)$ with space $O(A'!)$ or $O(L)$ where A' is the largest number of artist with significant deviations for any artist in the set.

4.4 Prestige Algorithms

Prestige is the degree of the influence of an actor within a graph. The algorithms implemented in Graph-RAT are as follows (requirement v):

Degree. The in-degree (number of incoming links) is degree prestige, normalized by the maximum possible in-degree (all actors linking to this actor) so that the value is between 0 and 1. Similarly out-degree (the number of outgoing links) is degree centrality. This is an $O(N)$ algorithm in both space and time [7].

[4] *http://www.cs.waikato.ac.nz/ml/weka/*

Closeness. The closeness prestige is a measure of the distance from any other actor to this actor which is then normalized to be between 0 (Infinite distance to nodes) and 1 (all nodes are distance 1 away). Similarly closeness centrality is a measure of the distance from this actor to every other actor. In the case of a disconnected graph, closeness is only calculated on the component it is a member of. This algorithm is $O(N^2)$ in time and $O(N^2)$ in space if paths are calculated separately, but $O(N)$ in space if the algorithm mixes shortest-path calculations while calculating closeness [7].

$$Closeness_i = \frac{N}{\sum_{j=1}^{N} \min distance(i,j)} \tag{1}$$

where N is the total number of nodes. While this should be ∞ for graphs that do not have paths between some nodes, in practice this is set to be closeness within its component (largest connected sub-graph) instead.

Betweenness. The number of times this actor appears on the shortest path between any two actors, normalized so that the value is between 0 (not on any paths) and 1 (on every shortest path). Its time and space complexity is exactly the same as for closeness.

$$Betweenness_i = \frac{\sum_{j=1}^{N} \sum_{k=1}^{N} i \subseteq Path(j,k)}{N(N-1)} \tag{2}$$

where N is the total number of nodes and $Path(j,k)$ is the shortest path between nodes j and k.

Page Rank. This algorithm calculates prestige for an actor based on the prestige of the actors linking to it. The prestige value for each actor is the value of the first eigenvector of the normalized link matrix with an additional phantom node. The normalization is that the sum of outgoing links from every actor is 1. For this phantom actor, every actor is connected to by every other node with a weight of 15% (100% for those without outgoing links) of all links from the actor and to all nodes equally. Directly calculating the eigen-matrix, this is an $O(N^2)$ algorithm in space and time, but an $O(N)$ in time and $O(L)$ in space using the power method [10]. The $O(N^2)$ algorithm is useful because it also populates the eigen-matrix for use in clustering algorithms.

HITS. This algorithms generates Hubs (points to a prestigious page) and Authorities (is a prestigious page). The hubs are the first eigenvector of the MM^T where M is the link matrix. The authorities are the values of the first eigenvector of M^TM. Space and time complexity as for Page Rank, again with the $O(N^2)$ variant useful for clustering.

4.5 Clustering Algorithms

Clustering algorithms (requirement vi) act on all links of a graph.

Maximal Cliques. A clique is a fully connected sub-graph and a maximal clique is any clique such that no clique exists that contains the maximal clique as a strict subset. This algorithm finds all maximal cliques and runs in $O(Nlog(N)k!)$ where k is the maximum clique size in the graph. For the evaluation work presented in Section 6, which processed a graph where N=10000, k had an upper bound of 10.

Bipartite Clustering. This identifies all sub-graphs that have the property that all elements are connected by at least 2 fully independent paths (i.e. the paths share no common actor beyond the beginning and ending node). The code is based on Jung2.

Strongly Connected Components. This identifies all strongly connected sub-graphs within a graph. A sub-graph is strongly connected if $\forall a, b \in L$, $a \rightarrow b$ exists and $b \rightarrow a$ exists. Another variant is **Weakly Connected Components** which is equivalent to strongly connected if links are bidirectional.

Weka Classifier Clustering. This applies a Weka clustering algorithms that provide a unique cluster ID to all items clustered. Which clustering algorithm to use is specified by a parameter.

Weka Probabilistic Clustering. This utilizes a Weka clustering algorithm that provide a probability distribution for each actor for which cluster it belongs to. Again, the specific algorithm is specified by parameter.

Edge Betweenness Clustering. This algorithm performs hierarchical clustering on a graph. Edges are removed in order of their betweenness score until all actors are isolated. A variant—**Norman-Girvan Edge Betweenness Clustering** [13]—recalculates betweenness before each link removal.

4.6 Similarity Algorithms

These algorithms are utilized to create a pairwise distance metric between two objects. All algorithms use the Strategy pattern to make the distance metric a parameter (see Section 5). All are This is $O(n^2)$ invocations of the distance metric.

Similarity By Link. This algorithm determines the similarity between two actors of the same mode by comparing the links each possess (along some relation) by outgoing, incoming, or all links. These links are loaded into a DataVector (see Section 5) and actors compared pairwise using a distance metric (see Section 5).

Similarity By Property. This algorithm determines the similarity between two actors of the same mode using a property value as the source. They are compared pairwise using a distance metric.

Graph Similarity. This algorithm calculates the similarity between two sub-graphs using a given property. Sub-graphs are chosen as siblings of the current graph or via a regular expression match on ID's of sub-graphs of the current graph. The comparisons are performed using a distance metric and stored on a property of each graph as a map between graph ID and a double representing similarity.

4.7 Aggregator Algorithms

These algorithms take as input a collection of actors or graphs with properties and summarize the given properties in a single Weka Instance.

Aggregate By Link. This algorithm collects all links of a given actor (by incoming, outgoing, or all links). For every link, the values of the given property are aggregated by an aggregator function (see Section 5). Each of the resultant properties are collected on the actor using another (potentially different) aggregator function. This is O(V) invocations of the inner aggregator function and O(N) invocations of the outer aggregator function.

Aggregate On Actor. This algorithm collects all properties of a given actor and creates a single property representing all this information. The values of each property are aggregated, then the result is aggregated into a single property. This is O(N) invocations of both the inner and outer aggregator functions.

Aggregate By Graph. This algorithm collects a given property over all actors, aggregating first over property values for each actor, then aggregating over all actors into a single property on the graph. This is O(N) invocations of both the inner and outer aggregator functions.

4.8 Matrix Algorithms

These algorithms transform either graphs or a Colt matrix [5] and creates a new Colt matrix.

Distance Matrix. This algorithm calculates a distance matrix over all actors of a given mode using the PathBase reusable core (see Section 5). This is O(N^2) in memory and time.

Principal Component Analysis. This algorithm calculates a transform matrix for a distance matrix mapping it onto a lower-dimensional space,

[5] *http://acs.lbl.gov/ hoschek/colt/*

storing the resulting new vectors for each actor as a property. This is $O(N^2)$ in memory and time.

Graph Triples Two algorithms are present for creating an array describing the graph's structure given a single relation as defined in social network analysis. One algorithm treats all links as bidirectional. The other utilizes directional links. Both are $O(ND)$ in time where D is the average degree of the graph and the maximum of $O(D)$ and $O(N)$ in space.

4.9 Evaluation Algorithms

Graph-RAT current provides seven evaluation metrics [8]. Some only work on binary recommendations (present or not present) while others require stratified recommendations. All algorithms provide evaluation for each user and average and standard deviation for the data set.

Precision Recall FMeasure. This metric provides the three basic measures—precision, recall, and F-measure—for binary results. It is utilized in the experiment in Section 6.

Recommendation Error. This metric is the percentage of the binary recommendations that are not present in the ground truth compared to all recommendations made.

ROC Area. This operates on ranked recommendations. The recommendations are separated into those that are present in the ground truth and those not. The average percent of ground truth at each step is recorded and the average of these percentages is the metric.

Half Life. Each predicted value in the ground truth is ordered from highest score to lowest. Sequentially moving from highest predicted recommendation, the score is added divided by an exponentially increasing penalty. The result is a value between 0 and 1 where 0 contained nothing from the recommendation list while 1 contained the entire recommended list from highest score to lowest score.

Pearson's Correlation. The correlation between the predicted scores of artists and ground truth scores are calculated. This requires valued recommendations. Note that , if a ranking algorithm is applied prior to calculating this, this metric is equivalent to Spearmans' ρ [8].

Kendall's Tau. This provides another correlation metric that is based on the number of correctly ordered pairs of recommendations.

Mean Error. This metric calculates three different error metrics based on mean error between the expected recommendation value and the predicted recommendation value.

4.10 Display Algorithms

Graph-RAT provides three different algorithms for displaying graphs. One provides display of a single mode (actor) and relation (link), another provides the same view, but color codes the actors by a given property value. The third displays the entire graph with each relation and mode in a different color. All three utilize the Prefuse toolkit.

4.11 Other Algorithms

Beyond these established classes are algorithms that do not fit into a category. These include graph property algorithms such as calculating basic properties of a given relation on a graph as well as calculating path-based metrics. Others calculate the distance between two properties on a single metric or define a new property providing a ranking of actors by a given property.

5 Utility Objects

Web Crawler Graph-RAT provides three different web-crawlers—all available through the Crawler interface. The FileReader crawler reads local filesystems. The BaseCrawler provides single-threaded web crawling. The WebCrawler provides full multi-threaded crawling.

All crawlers utilize a unique structure for determining parsers which allows for arbitrary parsers to be used on arbitrary pages. When submitting a page to a crawler for parsing, for one also provides which set of parsers are to be used for the given web site. Particularly for REST-based services (such as LastFM's services), this allows the flexibility to construct the next page to spider dynamically based on the information in the current document, deciding at runtime which set of parsers will parse the submitted link. This makes spidering LastFM web services is possible.

DataVector DataVectors are an abstraction of an array of double values that can be generated from a number of different sources. The primary advantage is that algorithms can create data in any of a number of different data structures and reuse the same utility functions for performing operations against them.

Distance Functions These functions describe a suite of different methods for calculating the distance between two vectors. The methods derived from Wikipedia entries are: Manhattan, Euclidean, Chebyshev, and DotProduct. Jaccard, Pearson, WeightedKLDistance, and Cosine distance are adapted from distance methods provided by Anna Huang.

Aggregator Functions These functions define a number of methods for combining a number of different Weka Instance objects. The Null and FirstItem aggregators have no restrictions on the backing Attributes. The Sum,

Product, Max, Min, Mean, and Standard Deviation aggregators require the Instances use the same backing Attributes. The Concatenate function requires a disjoint set of Attributes over the Instance objects.

Path Base Core This component provides as a reusable module a set of Paths calculated by Djikstra's spanning tree algorithm. The module calculates trees on a per-actor basis so memory use is kept at O(N). This algorithm is guaranteed to produce accurate results only over non-negative links.

Actor Distance Matrix Core This component provides a 2D Colt matrix representing the distance between all actors of a mode as defined by the cost of the shortest path between two actors.

Link Betweenness This component provides link betweenness for each link over a given relation.

Strongly Connected Components This component provides, if multiple components exists in the graph over the given relation (by strong connectedness definition of components), sub-graphs describing these components.

6 Evaluation

To illustrate the usefulness of the toolkit an indicative experiment was undertaken that compared the ability of four algorithms—Local Recommendation, Single and Multi Instance AdaBoost, and Dispersion—to predict musical taste (like/dislike). Recall and precision (PrecisionRecallFMeasure) was used to quantify results.

First a configuration was constructed to crawl LiveJournal using a seed list of 50 LiveJournal users. This resulted in 10,000 FOAF files saved locally to disk. This was done from the command line, but could have equally been performed using the GUI. Next, the interest field of the FOAF descriptions was extracted and set as properties of actors of mode user. Each of these properties was then compared against a list of top 40 artist since 1999. For each artist found in a FOAF a new actor of mode artist was created. Links were created between users and artists. This was more convenient to do using the GUI, but as the models of computation are equal it could also have been done from the command line.

To evaluate the selected algorithms, ground truth needed to be established. This was set to be the original artist list for each user. Using a variant of leave one out cross-validations each algorithm was run using RecallPrecisionFMeasure to compute results, which are summarized in Table 1. AdaBoost had the best recall while Local Recommendation had the best precision.

7 Conclusion

Graph-RAT provides a set of tools for constructing and evaluating music recommendation systems which are also applicable for a number of other tasks as well.

Algorithm	Precision Mean	Precision Std Dev	Recall Mean	Recall Std Dev
Local Recommendation	0.020	65.8	0.061	398.9
Weka Ada Boost	0.015	34.4	0.382	2242
Multi Instance Weka	0.0029	0.533	0.297	1983
Dispersion	0.008	3.693	0.180	1308

Table 1: Existing graph toolkits

A range of algorithms are provided that reflect the multi disciplinary nature of the topic. As much as was possible, pre-existing open source software was re-used: jAudio forms the basis for content analysis; Weka for machine learning; Prefuse for the interactive graph visualizer.

Furthermore, Graph-RAT provides a blackboard framework that encourages culturally-aware recommendations by making it easy for researchers to segment data into culturally-homogenous groups—making the use of existing algorithms no longer ethnocentric.

Finally, the toolkit also provides a number of standard algorithms for comparing different approaches to music recommendation which can be combined in a Unix-shell style pipeline. Applications from this framework can be built in XML, via a GUI, or embedded.

8 Acknowledgments

The author would like to thank the University of Waikato for its generous support through the its doctoral scholarship programme.

References

[1] A. Anglade, M. Tiemann, and F. Vignoli. Virtual communities for creating shared music channels. *Int. Cont. on Music Information Retrieval*, pages 95–100, 2007.

[2] O. Celma, M. Ramìrez, and P. Herrera. Getting music recommendations and filtering newsfeeds from foaf descriptions. *Int. Cont. on Music Information Retrieval*, 2005.

[3] H.-C. Chen and A. L. P. Chen. A music recommendation system based on music data grouping and user interests. *CIKM*, pages 231–238, 2001.

[4] A. Crossen, J. Budzik, and K. J. Hammond. Flytrap: Intelligent group music recommendation. *IUI*, 2002.

[5] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the 40th Anniversary Meeting of ACL*, 2002.

[6] J. Downie, J. Futrelle, and D. Tcheng. The international music information retrieval systems evaluation laboratory: governance, access and security. In *Int. Conf. on Music Information Retrieval*, pages 9–14, 2004.

[7] L. C. Freeman. Centrality in social networks. *Social Networks*, 1:215–239, 1979.

[8] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering systems. *ACM Transactions onj Information Systems*, 22(1):5–53, January 2004.

[9] D. Jensen. Unique challenges of managing inductive knowledge. In *Proc. of the 1997 AAAI Spring Symposium on AI in Knowledge Management*, Stanford, March 1997.

[10] A. N. Langville and C. D. Meyer. Deeper inside PageRank. *Internet Mathematics*, 1(3):335–380, 2003.

[11] B. Logan. Music recommendation from song sets. *Int. Cont. on Music Information Retireval*, 2004.

[12] D. McEnnis. Towards a music recommendation system. In *New Zealand Computer Science Research Students Conference*, 2008. To appear.

[13] M. E. J. Newmann and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69, 2004.

[14] E. Pampalk and M. Gasser. An implementation of a simple playlist generator based on audio simlarity measures and user feedback. *Int. Cont. on Music Information Retrieval*, 2006.

[15] E. Pampalk and M. Goto. Musicsun: A new approach to artist recommendation. *Int. Cont. on Music Information Retrieval*, pages 101–104, 2007.

[16] E. Pampalk, T. Pohle, and G. Widmer. Dynamic playlist generation based on skipping behavior. *Int. Cont. on Music Information Retrieval*, 2005.

[17] S. Pauws and B. Eggen. Pats: Realization and user evaluation of an automatic playlist generator. *Int. Cont. on Music Information Retrieval*, 2002.

[18] S. Pauws, W. Verhaegh, and M. Vossen. Fast generation of optimal music playlist using local search. *Int. Cont. on Music Information Retrieval*, 2006.

[19] V. Sandvold, T. Aussenac, O. Celma, and P. Herrera. Good vibrations: Music discovery through personal musical concepts. *Int. Cont. on Music Information Retrieval*, 2006.

[20] M. Tiemann, S. Pauws, and F. Vignoli. Ensemble learning for hybrid music recommendation. *Int. Cont. on Music Information Retrieval*, pages 179–181, 2007.

[21] K. Yoshii, M. Goto, K. Komatani, T. Ogata, and H. G. Okuno. Hybrid collaborative and content-based music recommendation using probabilistic model with latent user preferences. *Int. Cont. on Music Information Retrieval*, 2006.

[22] K. Yoshii, M. Goto, K. Komatani, T. Ogata, and H. G. Okuno. Improving effeciency and scalability of model-based music recommender system based on incremental training. *Int. Cont. on Music Information Retrieval*, pages 89–94, 2007.