APPEARS IN THE PROCEEDINGS OF THE TWENTY-NINTH CONFERENCE ON INNOVATIVE APPLICATIONS OF ARTIFICIAL INTELLIGENCE (IAAI-17), SAN FRANCISCO, FEBRUARY 2017

Designing Better Playlists with Monte Carlo Tree Search

Elad Liebman[‡], Piyush Khandelwal[‡], Maytal Saar-Tsechansky[†] & Peter Stone[‡]
[‡]CS Department, The University of Texas at Austin, {eladlieb, piyushk, pstone}@cs.utexas.edu
[†]McCombs School of Business, The University of Texas at Austin, Maytal.Saar-Tsechansky@mccombs.utexas.edu

Abstract

In recent years, there has been growing interest in the study of automated playlist generation - music recommender systems that focus on modeling preferences over song *sequences* rather than on individual songs in isolation. This paper addresses this problem by learning personalized models on the fly of both song *and* transition preferences, uniquely tailored to each user's musical tastes. Playlist recommender systems typically include two main components: i) a preference-learning component, and ii) a planning component for selecting the next song in the playlist sequence. While there has been much work on the former, very little work has been devoted to the latter. This paper bridges this gap by focusing on the planning aspect of playlist generation within the context of DJ-MC, our playlist recommendation application. This paper also introduces a new variant of playlist recommendation, which incorporates the notion of diversity and novelty directly into the reward model. We empirically demonstrate that the proposed planning approach significantly improves performance compared to the DJ-MC baseline in two playlist recommendation settings, increasing the usability of the framework in real world settings.

Introduction

Individual songs are seldom listened to in isolation, and it is well established that music is experienced in a temporal manner [7]. Good music recommendation systems can take advantage of this fact while generating customized playlists. Such playlists need to take into account not only the enjoyment experienced by a given listener when listening to a certain song, but also how songs can be put in sequence appropriately to provide greater enjoyment than simply listening to songs in an arbitrary order.

While several recent papers have looked into the algorithmic generation of music playlists, they have predominantly focused on the *learning* aspect of modeling user preferences, rather than the *planning* aspect of utilizing learned knowledge effectively to generate good sequences. In this paper, we address this gap by applying the adaptation of an advanced planning approach, Upper Confidence Bound in Trees (UCT) [10], to generate better song sequences.

UCT is a member of the Monte Carlo Tree Search (MCTS) family of planning algorithms, which approximately solve sequential decision making problems. These algorithms are anytime, meaning that they can iteratively improve results given additional computational time. MCTS algorithms execute a number of planning simulations, i.e. Monte Carlo rollouts, and keep track of encountered states and actions within a tree structure. UCT was first popularized in Computer Go [6], and in the past few months played a key role in enabling a computerized Go program, AlphaGo, to surpass the highest echelons of human level performance [19].

In a recent paper, we have proposed a music recommender system called DJ-MC [12], which treats playlist generation as a sequential decision making task, and applies techniques from the Reinforcement Learning (RL) literature to learn and model user preferences over both songs and song transitions on the fly. In this paper, we build on this approach and demonstrate how more sophisticated Monte Carlo planning, based on Upper Confidence Bound in Trees (UCT) [10], can improve performance over the naive planning method used in the original DJ-MC framework. Then, to further understand the importance of advanced planning techniques in generating meaningful song sequences, we introduce an alternative framework for music playlist recommendation, which aims to maximize diversity over a sequence of preferred songs.

This paper makes two main contributions. First, we demonstrate the effectiveness of a UCT planning approach in a music recommendation platform that has been tested on human participants. To our knowledge, this is the first application of such approaches in a music recommendation domain. This extension makes DJ-MC practical for recommending songs using realistically large music corpora. Second, we extend the previous framework by introducing a new recommendation objective, namely novelty search, and show that our proposed approach is well suited for this setting as well.

RL Applied to Playlist Recommendation

This section provides the technical details of the application considered in this paper. Specifically, it introduces how playlist recommendation can be modeled as an MDP, describes the DJ-MC framework, and introduces our adaptation of UCT planning to playlist recommendation.

Playlists as Markov Decision Processes

If we consider playlists as depending on the specific sequence of songs, such that the enjoyment of each song is dependent on the songs chosen before it, then at each point in the playlist generation process, the selection of a new song affects the choice of possible songs in the future. From this perspective, playlist recommendation is a sequential decision-making task, and as such, is suitably formulated as a Markov Decision Process (MDP) [20].

An episodic MDP M is represented as $\langle S, A, P, R, T\rangle$ where S is the set of states an agent can be in, A is the set of actions that the agent can take at a state, P is the transition function that gives the transition probability of reaching a particular next state s' from state s after taking action a ($P : S \times A \times S \to \mathbb{R}$; $\sum_{s'} P(s, a, s') = 1$), R is the reward received given a transition ($R : S \times A \times S \to \mathbb{R}$), and T is the set of terminal states which end the episode. To perform optimally in a task that an MDP represents, an agent must find a policy $\pi : S \to A$ such that from any given state s, executing action $\pi(s)$ and then continuing to act optimally (that is, following the optimal policy π^*) would yield in the highest expected sum of rewards over the length of the episode. This is traditionally referred to as "solving" an MDP.

DJ-MC formulates the playlist generation problem as an MDP as follows. Given a finite set of songs $\mathcal{M} = \{a_1, a_2, \ldots, a_n\}$, and the assumption that playlists are of length k, the state representation captures a list of all the songs that have been heard by a listener. Thus, the set of MDP states can be constructed as follows:

$$S = \{(a_1, a_2, \ldots, a_i) \mid 1 \leq i \leq k, \forall_{j \leq i}[a_j \in \mathcal{M}]\}.$$

At any state, the agent's action directly corresponds to the next song played by the system, and consequently the action set is the entire set of songs, i.e. $A = \mathcal{M}$. Given these S and A, the MDP transition function P is trivial and deterministic, i.e. each state-action pair maps to exactly one state. Given a state $s = (a_1, a_2, \ldots, a_i)$ and action a, the next state $s' = (a_1, a_2, \ldots, a_i, a)$. Furthermore, since playlists are of length k, any state s that contains k songs is terminal.

On the other hand, the definition of an appropriate MDP reward function R is far from trivial, and represents a key challenge in tackling the playlist recommendation problem. Intuitively, R models a listener's enjoyment while listening to the generated playlist. The goal of the music recommendation system is to select songs in order to maximize the cumulative reward across an entire episode. More exactly, a reward function $R_u(s, a)$ needs to capture the enjoyment of listener u hearing song a after listening to song sequence $s = (a_1, a_2, \ldots, a_i)$. In other words, the reward function needs to model not only the utility of playing a particular song to the listener, but also reflect the significance of the sequential decision making aspect of the playlist generation problem.

The DJ-MC Architecture

In this section, we summarize DJ-MC, a reinforcement learning approach to a playlist-oriented, personalized music recommendation system [12].

The DJ-MC architecture expresses the reward function as a linear combination of both song and transition utility. To simplify learning, DJ-MC decouples the reward derived from the choice of songs from transitions as follows:

$$R(Seq_t, a, Seq') = \phi_s(u) \cdot \theta_s(a) + \phi_t(u) \cdot \theta_t(Seq, a) \quad (1)$$

Here, the state is the song sequence Seq, and a is the song played at state $Seq = \{a_1, \ldots, a_n\}$ leading the system to a future state $Seq' = \{a_1, \ldots, a_n, a\}$.

$\theta_s(a)$ and $\theta_t(Seq, a)$ are finite length feature vectors representing song a and the transition from $a_n \to a$, respectively. $\phi_s(u)$ and $\phi_t(u)$ are vectors which represent a particular listener's preferences for song a and the transition from $a_k \to a$, respectively.

This song and transition representation is constructed as follows. Each song is analyzed for its acoustic properties, and properties such as amplitude, pitch, timbre, and tempo are used to construct a real-valued song descriptor of length 34. A binned representation of size 10 per feature is then constructed from each real-value feature by indicating the relative position of a song across the entire song set $\mathcal{M}$. Across all 34 features, this binary representation leads to the 340-dimensional representation θ_s. As described by Liebman et al., the transition representation θ_t can be modeled in a similar way as a 3400-dimensional binary vector. Correspondingly, a listener's preferences over songs and transitions are modeled as a 340-dimensional weight vector ϕ_s and a 3400-dimensional weight vector ϕ_t, respectively. The high-level DJ-MC architecture pseudocode is presented in Algorithm 1.

The values of ϕ_s and ϕ_t for a particular user are initially unknown, and are updated as DJ-MC learns the user's preference from experience. Should ϕ_s and ϕ_t be fully known, and consequently the reward function R be known, every element of the playlist MDP in which DJ-MC operates becomes well-defined, and the MDP can be solved as a pure planning problem. Given a well-defined MDP reward function, DJ-MC needs to plan in order to select the next song in the playlist. Since the problem space is too large to be able to plan optimally, DJ-MC instead uses a heuristic approach, running multiple Monte Carlo simulations from the current state, where actions within each rollout are selected randomly.

Algorithm 1 DJ-MC Architecture

1: **Input:** $\mathcal{M}$ - song corpus, K - planned playlist duration, k_s - number of steps for song preference initialization, k_t - the number of steps for transition preference initialization
Initialization:
2: Initialize individual song weights to obtain ϕ_s
3: Initialize song transition weights to obtain ϕ_t
Planning and Model Update:
4: **for** K `steps` **do**
5: Run Monte-Carlo search to select the next song
6: Update ϕ_s, ϕ_t
7: **end for**

Subsequently, the first action from the trajectory obtaining the highest cumulative reward is selected. This is a relatively unsophisticated approach, as planning was not the main focus in the original DJ-MC formulation. A main contribution of this paper is a significant improvement in the DJ-MC planner, which we motivate next.

Upper Confidence Bound in Trees (UCT)

In this section, we discuss how DJ-MC planning can be improved using the more sophisticated UCT algorithm. There are several approaches to solving an MDP optimally, assuming the transition function P and reward function R are known. For example, Value Iteration [3] solves discrete MDPs directly with dynamic programming. However, in many domains, including the playlist generation problem described in the previous sections, the state-action space is sufficiently large such that optimally solving the MDP is infeasible. In these situations, it is necessary to use approximate solvers that restrict search to more relevant regions of the state action space. One family of such approaches is Monte Carlo Tree Search (MCTS). In this paper, we focus on one variant of this approach called Upper Confidence Bound in Trees (UCT) [10].

In UCT, planning is performed by simulating a number of state-action trajectories from the current MDP state, i.e. Monte Carlo rollouts. In the playlist generation domain, the current MDP state reflects the songs that have already been played by the system. For each state-action pair encountered within this trajectory, UCT stores the number of visits for that pair as well as the long term expected reward of choosing that action at that state in a tree structure. Each node represents a state, with edges representing actions leading from one state to another.

Given information collected in prior simulations, UCT uses the UCB1 algorithm [2] for action selection, allowing the algorithm to spend more time in areas of the state-action space that seem more promising. The UCB1 decision criterion is defined as $a = \arg\max_a \left(Q(s,a) + c_p\sqrt{\ln(n_s)/n_a}\right)$, where n_s is the number of visits to the state, n_a is the number of times action a was selected in previous simulations at this state, $Q(s,a)$ is the current expected long term reward for taking action a at this state, and c_p is tuned empirically to better balance exploration versus exploitation.

This paper applies a parametrized UCT variant called MaxMCTS(λ). MaxMCTS(λ) employs a more complex Q-value backpropagation strategy than that used in the original UCT algorithm. This variant was previously used for multi-robot coordination problems [8], and is studied extensively along with other variants by [9]. In MaxMCTS(λ), Q-values are estimated using an eligibility trace mechanism used in Peng's $Q(\lambda)$ reinforcement learning algorithm. This estimation process is summarized in Algorithm 2.

Algorithm 2 Eligibility trace backpropagation

1: **Input:** *trajectory* - Stack of $\langle$state, action, reward$\rangle$, populated during planning simulation.
2: $q \leftarrow 0$ # Backpropagated value
3: **for** $\langle s, a, r\rangle$ = *trajectory.pop*() **do**
4: $q \leftarrow q + r$, $n_s \leftarrow n_s + 1$, $n_a \leftarrow n_a + 1$
5: $Q(s,a) \leftarrow Q(s,a) + (q - Q(s,a))/n_a$
6: $q \leftarrow (1-\lambda)\max_{a'|n_{a'}\neq 0}[Q(s,a')] + \lambda q$
7: **end for**

In Algorithm 2, q is a value backpropagated up the tree, and used to update Q-value estimates in Line 5. The key update rule for this backpropagation strategy is the update rule on line 6, which uses parameter λ to interpolate between the current backpropagated value and the maximum Q-value estimate at that state. Intuitively, when λ values are close to 0, even when exploratory actions are taken further down in the tree, the value of the action with the highest expected reward is propagated higher up in the tree. This technique minimizes the risk of exploratory actions taken further down the tree, but increases the likelihood of finding suboptimal policies. The value of λ must be selected empirically, and intermediate values between 0 and 1 can often provide significantly better performance in some domains.

UCT for Playlist Generation

As pointed out above, while the original DJ-MC architecture puts a great deal of emphasis on effectively learning user preferences from limited information, when it comes to leveraging the learned model to select the next song, a relatively naive planning heuristic was employed. However, especially given the complex nature of generating playlists using a large song database, it stands to reason that a stronger heuristic, better suited for balancing the exploration-exploitation tradeoff with limited information, is a more appropriate choice. On the other hand, more sophisticated methods hold the risk of requiring more experience to be effective, which may be a problem if only a limited number of simulations is possible.

Two main adaptations need to be made to make UCT-based approaches applicable in the playlist recommendation setting. First, as mentioned above, the more sophisticated parameterized backup strategy of MaxMCTS(λ) has to be used, since given the difficulty of the search problem pure Monte Carlo backups aren't likely to find good enough trajectories. Second, to make DJ-MC with MaxMCTS(λ) applicable to huge song corpora, we introduce hierarchy into the song selection step. One of the key determining factors in the efficiency of MCTS methods is the branching factor induced by the domain at each node. The branching factor is determined by the number of available actions at each node of the UCT tree. Since by default the set of actions available at each state of the music playlist MDP is the entire set of songs M, the default branching factor for this domain is prohibitively high for even moderately sized music databases. To mitigate this issue, we use the structure of the song space to cluster songs into subsets. Each subset represents an abstract song type. Then, we alternate between choosing song types and choosing specific songs at each step of the trajectory, dramatically reducing the branching factor (the lowest branching factor in expectation is achieved when the number of clusters is $\sqrt{M}$).

Algorithm 3 MaxMCTS(λ) for music applications starting at playlist s

1: *input: current playlist s, song corpus* $\mathcal{M}$
2: Cluster $\mathcal{M}$ to obtain song types C and mapping to concrete songs $S_{\mathcal{M}}(C)$
3: $rootNode \leftarrow initNode(s)$ # Root node represents the current playlist state
4: **for** $sim \in \{1, \ldots, numSimulations\}$ **do**
5: $node \leftarrow rootNode$
6: $trajectory \leftarrow$ **new** Stack
7: **while** $notTerminal(node)$ **do**
8: **if** $node.parent \in \mathcal{M}$ **then** $a \in$ *Song Types* # If last song in the trajectory is instantiated, plan over abstract songs.
9: **else** $a \in S_{\mathcal{M}}(node.parent)$
10: **if** $node.n_s = 0$ **then** $a \leftarrow$ *default song selection*
11: **else** $a \leftarrow selectNextActionWithUCB1(node)$
12: $\langle ns, reward \rangle \leftarrow simulate(node, a)$
13: $nextNode \leftarrow getOrInitNode(node, a, ns$ # Next node represents the playlist after selecting a new song.
14: $trajectory.push(node, a, reward)$
15: $node \leftarrow nextNode$
16: **end while**
17: BACKPROPAGATE($trajectory$) using Algorithm 2
18: **end for**

The pseudocode for MaxMCTS(λ) adapted to the playlist generation domain is presented in Algorithm 3. Line 2 preprocesses the song set by clustering the song corpus to obtain abstract song types and a mapping from each song type to a set of concrete songs that comprise that type. Clustering is done via the canonical K-Means algorithm [15]. Then line 3 initializes the root of the search tree to be the current playlist state. The main loop in lines 4-18 generates one MCTS simulation. Lines 7-12 descend down the tree using either the UCB1 criterion for action selection or a default song exploration policy if not all actions have been tried at least once. Lines 9-10 decide whether we're currently selecting a song type or a concrete song of the song type specified. Line 13 runs a simulation from the new song selected to obtain an estimated user reward this song choice will accrue. Once the rollout is complete line 17 backpropagates the overall reward up the search tree, using Algorithm 2.

Planning for Personalization

In this section we empirically evaluate the benefit of adapting MaxMCTS(λ) to the DJ-MC framework, which aims to maximize playlist personalization. We compare the performance of the proposed algorithm, DJ-MC + MaxMCTS, with that of "vanilla" DJ-MC, measuring user reward over 30-song sequences. We also compare it to the benchmarks DJ-MC was originally compared against: a greedy heuristic which selects the favorite song not played yet irrespective of sequence, thus mimicking a more traditional music recommendation algorithm, and a random baseline. the results are presented in Figure 1.

Results indicated that using MaxMCTS(λ) with $\lambda = 0.5$ statistically significantly outperforms the standard DJ-MC, and that with all chosen λ values MaxMCTS(λ) did as well as or better than the standard planning technique used in DJ-MC.

Planning for Diversity

DJ-MC is an interesting and robust framework for learning playlist preferences and generating playlists efficiently on the fly. However, it is not the only

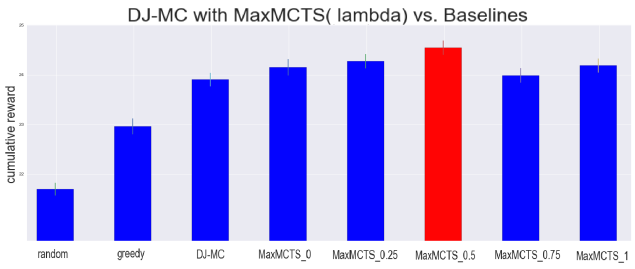

Figure 1: Average reward for 30-song sequences, comparing standard DJ-MC to DJ-MC with MaxMCTS(λ) planning using varying values of λ, a greedy system which attempts to maximize song enjoyment regardless of sequence, and a random sequence generator. Results are obtained over 30 repetitions with a corpus size of 5000 and 70 song types (leading to an average branching factor of 70). Songs are randomly taken from the Million Song Dataset [4]. Best performing algorithm marked red. Monte-Carlo approaches ran 5000 simulations.

playlist recommendation framework where planning is useful. Indeed, any playlist recommendation framework which takes transitions into account should benefit from the usage of advanced planning techniques such as MaxMCTS(λ). To illustrate this point, we study the application of MaxMCTS(λ) with various λ values in a different playlist recommendation setting, with different transition mechanics and a different reward function - diversity (or novelty) based playlist generation.

The idea of algorithmic novelty search in playlist generation has been proposed before [22, 11, 13, 21] and it seems intuitive - even once we gain some knowledge user preferences, they wouldn't want to listen to very similar songs one after another in sequence, leading to a potentially tedious experience.

To this end, we propose a novel diversity generation framework. Given a song database $\mathcal{M}$ and some similarity metric D between songs, we are tasked with finding a sequence of songs the listener enjoys, but penalize for the amount of similarity between each song chosen and the songs which preceded it. Similar to the temporal discounting approach adopted by [12], we can assume the penalty for similarity across songs in the sequence decays as songs progress. Formally, in the novelty detection setting, we assume the same general playlist problem MDP as in DJ-MC, but propose an alternative approach for modeling R. Instead of the decomposition proposed in Eq. 1, we propose an alternative formulation, as presented in Eq. 2. Assuming a utility function for individual songs R_s and a similarity measure between songs D, the reward function representing a listener u for a song sequence $Seq = \{s_0, \ldots, s_t\}$ is modeled as:

$$R^u_{novelty}(Seq) = \sum_{i=0}^{t} (R_s(s_i) + \sum_{j=0}^{i} \frac{1}{i-j} D(s_i, s_j) \quad (2)$$

Given this setting, and assuming the user's individual song preferences are known, the planning problem - generating a good sequence of songs - is purely a combinatorial search problem. However, it is intractable for even moderately sized song sets. Indeed, it can be shown to be NP-hard via a reduction from the weighted max-clique problem [18]. However, using MaxMCTS(λ) with the same two-stage song type abstraction suggested in Section lends itself directly to this setting as well.

As in the DJ-MC extension case, we wish to empirically test MaxMCTS(λ) in the novelty maximization setting described above. We choose an experimental setting similar to that employed in the previous section, but in order to isolate the planning aspect, we assume the individual song preferences are roughly known - the listener provides a list of 100 liked songs and preferences are inferred from these songs based on the similarity function D which is also assumed to be known. For the purpose of this section, D is assumed to be the Euclidean distance between the 34-dimensional song representation vectors used by DJ-MC (this is similar in spirit to [13]), i.e. for two songs s_1, s_2, $D(s_1, s_2) = \sqrt{\sum_{i=1}^{34} (s_{1i} - s_{2i})^2}$.

The results in the diversity-based playlist recommendation domain are provided in Figure 2.

As evident from the results, in this setting, MaxMCTS(λ) with $\lambda = 0$ significantly outperforms the other approaches. We note that the relative improvement here is higher than that observed for the DJ-MC setting, but unlike that case, this result is sensitive to an apt choice of λ. It can nonetheless be seen as a positive result, especially given that [9] suggest a straightforward approach for roughly tuning λ. This outcome yet again illustrates how RL applied to playlist recommendation can improve performance compared to our baselines. Looking comparatively at the two different playlist generation settings, the novelty detection setting induces a considerably sparser search problem. This provides a relative advantage to planning methods which tend to retain relatively good observed trajectories, which is effectively what MaxMCTS($\lambda = 0$) does.

Related Work

Generally speaking, there has been substantial research on modeling song similarity towards music playlists [1]. Some previous work attempted to model playlists directly. [16] treated the playlist prediction problem as a supervised binary classification task, with pairs of songs in sequence as positive examples. [17] trained a bigram model for transitions. [5] took a similar Markov approach, treating playlists as Markov chains in some latent space, and utilized this to learn a metric representation for each song. [23] adapted a Latent Dirichlet Allocation model to capture music taste from listening activities across users and songs. Recent work by [22] also borrows from the reinforcement learning literature, and considers the problem of song recommendations as a bandit problem, balancing exploration and exploitation to identify novel. Novelty and diversity in them-

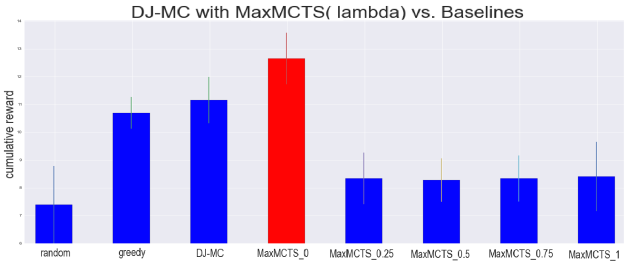

Figure 2: Average reward for 20-song long sequences, comparing a naive Monte-Carlo search approach to MaxMCTS(λ)planning using varying values of λ, a greedy system which attempts to maximize song enjoyment regardless of sequence, and a random sequence generator. Results are obtained over 30 repetitions with a corpus size of 1000 and 30 song types (leading to an average branching factor of 30). Songs are randomly taken from the Million Song Dataset. Best performing algorithm marked red. Monte-Carlo approaches ran 5000 simulations.

selves have also been a studied objective of playlists. [14] considered novelty in song trajectories via spectral song similarity. [11] used context-aware cues to better tailor a mobile music streaming service to user needs. More recently, [21] used a combination of Latent Dirichlet Allocation with graph search to produce more diversified playlists. All these papers, however, focused primarily on learning models for listener preferences. They did not address the complexity of generating good playlists, which can be decoupled from the learning aspect. In this paper, building on the approach suggested by [12], we treat the music playlist generation task as an AI planning problem, and show that using better planning leads to better playlists.

Summary & Conclusion

In this paper we study the application of reinforcement learning and Monte Carlo Tree Search (MCTS) to playlist recommendation. Even provided with useful learning algorithms, leveraging learned knowledge effectively is nontrivial in large song spaces or when the application is expected to work in real time. To this end we introduce MaxMCTS(λ) to effectively decide on which song to play next on the fly. We show that in two separate playlist generation settings using MaxMCTS(λ) holds potential of significantly improving the quality of generated playlists. We also show how maximizing diversity can be directly integrated into the sequential decision-making process. We believe this work is the first step in connecting the learning aspect of music playlist recommendation with better playlist planning techniques, resulting in better, more expressive music recommendation and user experience in real-world systems.

Acknowledgments

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (CNS-1330072, CNS-1305287, IIS-1637736, IIS-1651089), ONR (21C184-01), and AFOSR (FA9550-14-1-0087). Peter Stone serves on the Board of Directors of, Cogitai, Inc. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

References

[1] N. Aizenberg, Y. Koren, and O. Somekh. Build your own music recommender by modeling internet radio streams. In *Proceedings of the 21st international conference on World Wide Web*, pages 1–10. ACM, 2012.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[3] R. Bellman. Dynamic programming, 1957.

[4] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere. The million song dataset. In *ISMIR 2011: Proceedings of the 12th International Society for Music Information Retrieval Conference, October 24-28, 2011, Miami, Florida*, pages 591–596. University of Miami, 2011.

[5] S. Chen, J. L. Moore, D. Turnbull, and T. Joachims. Playlist prediction via metric embedding. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 714–722. ACM, 2012.

[6] S. Gelly and Y. Wang. Exploration exploitation in Go: UCT for Monte-Carlo Go. In *Conference on Neural Information Processing Systems (NIPS)*, 2006.

[7] B. Kahnx, R. Ratner, and D. Kahneman. Patterns of hedonic consumption over time. *Marketing Letters*, 8(1):85–96, 1997.

[8] P. Khandelwal, S. Barrett, and P. Stone. Leading the way: An efficient multi-robot guidance system. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1625–1633. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[9] P. Khandelwal, E. Liebman, S. Niekum, and P. Stone. On the analysis of complex backup strategies in monte carlo tree search. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1319–1328, 2016.

[10] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning (ECML)*, 2006.

[11] A. Lehtiniemi. Evaluating supermusic: streaming context-aware mobile music service. In *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, pages 314–321. ACM, 2008.

[12] E. Liebman, M. Saar-Tsechansky, and P. Stone. Dj-mc: A reinforcement-learning agent for music playlist recommendation. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 591–599. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[13] B. Logan. Content-based playlist generation: Exploratory experiments. In *ISMIR*, 2002.

[14] B. Logan and A. Salomon. A music similarity function based on signal analysis. In *null*, page 190. IEEE, 2001.

[15] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[16] F. Maillet, D. Eck, G. Desjardins, P. Lamere, et al. Steerable playlist generation by learning song similarity from radio station playlists. In *ISMIR*, pages 345–350, 2009.

[17] B. McFee and G. R. Lanckriet. The natural language of playlists. In *ISMIR*, pages 537–542, 2011.

[18] P. M. Pardalos and J. Xue. The maximum clique problem. *Journal of global Optimization*, 4(3):301–328, 1994.

[19] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[20] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[21] M. Taramigkou, E. Bothos, K. Christidis, D. Apostolou, and G. Mentzas. Escape the bubble: Guided exploration of music preferences for serendipity and novelty. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 335–338. ACM, 2013.

[22] X. Wang, Y. Wang, D. Hsu, and Y. Wang. Exploration in interactive personalized music recommendation: A reinforcement learning approach. *arXiv preprint arXiv:1311.6355*, 2013.

[23] E. Zheleva, J. Guiver, E. Mendes Rodrigues, and N. Milić-Frayling. Statistical models of music-listening sessions in social media. In *Proceedings of the 19th international conference on World wide web*, pages 1019–1028. ACM, 2010.