UNIVERSITY OF CALIFORNIA

Los Angeles

Graph-based Recommender System using Reinforcement Learning

A thesis submitted in partial satisfaction
of the requirements for the degree Master of Science
in Statistics

by

Diana L Zhang

2021

ABSTRACT OF THE THESIS

Graph-based Recommender System using Reinforcement Learning

by

Diana L Zhang

Master of Science in Statistics
University of California, Los Angeles, 2021
Professor Mark S. Handcock, Chair

Traditional recommender systems, such as collaborative filtering, content-based filtering, and hybrid approaches, are limited by challenges including data sparsity and cold start. In order to alleviate these issues, graph-based systems have been increasingly developed for serving recommendations. We build on these existing graph-based approaches and further increase recommendation quality by reflecting the dynamically changing and sequential nature of the recommendation problem and by training prediction models using reinforcement learning (RL). We implement this system using the widely known Netflix Prize data set and build a movie recommender system as a case study. We present results and challenges and discuss how these recommendations can be easily adapted for other user-item interactions as well.

The thesis of Diana L Zhang is approved.

Hongquan Xu

Tao Gao

Mark S. Handcock, Committee Chair

University of California, Los Angeles

2021

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

We interact with recommender (or recommendation) systems on a frequent basis. They are a subset of artificial intelligence (AI) algorithms that make predictions about what we may like. For digital services and apps, including Amazon, Facebook, and Netflix, to name a few giants, recommendation engines have become necessary due to the explosive growth of available content. Personalized recommendations are key to user engagement and retention. [8]

Historically, recommender systems have generally been classified on the basis of "rating" estimation technique. A rating models the extent to which a user is interested in an item. Traditional recommender systems are matrix factorization-based collaborative filtering processes, content-based processes, or hybrid processes. [3] Collaborative filtering approaches model a user's social environment and attempt to predict utility of items based on items previously rated by other similar users. Content-based approaches rely on modeling characteristics of items and predict utility based on items previously rated by the same user. Hybrid approaches combine both collaborative and content-based methods in order to boost prediction accuracy. However, these systems run into limitations with over-specification, sparsity, and cold start of new users and new items. They are also often implemented "offline" because of long training times and fail to update in-moment user interactions. [7] Alternatively, graph-based recommender systems can model complex relationships, provide explanations for recommended content, and produce relevant and novel predictions in real-time. [10]

Graph-based systems continue to be an emerging research trend, increasingly adopted by industrial platforms. [16, 19] These approaches, including PageRank [1] and Stochastic Approach for Link-Structure Analysis (SALSA) [2] algorithms, focus on various random walks to harness the graph structure for recommendation. Link prediction approaches from network modeling have also played a role in extending these algorithms. [4] These systems are semantically rich representations of interactions and, unlike traditional matrix factorization-based approaches, do not need to infer data connections. Relationship information between users and items can be updated and accessed naturally by design of graph-based systems. However, a limitation to these systems is that the recommendation algorithms are static and fail to adapt to dynamic and changing user interests.

We attempt to further improve these systems by using reinforcement learning to train our prediction models. Reinforcement learning is a computational approach focused on goal-directed learning from interaction. Serving recommendations includes trial-and-error search and delayed rewards, which are both important characteristics of RL. In addition, user interactions with recommender systems have fundamental temporal order [9]. Accordingly, this makes RL a good match for the recommendation problem. We formulate it as a Markov decision process (MDP), a formalization of sequential decision making, and explore a reinforcement learning based solution that can continuously evaluate user acceptance and rejection of recommended items and shape increased recommendation quality.

CHAPTER 2

Related Work

Recommender systems are a large and diverse research field. We divide the related work into two sections and focus on current graph-based systems as well as reinforcement learning based methods.

2.1 Graph-based Recommender Systems

The usefulness of graph-based recommendations has been studied broadly. [24] Compared to traditional matrix factorization-based techniques, the graph structure more accurately captures the relationships between and preferences of users. [21] Existing graph-based recommender systems generally harness the graph structure by focusing only on random walks and other Monte Carlo approaches to provide recommendations. Embedding-based methods use the information from the graph to learn low-dimensional vector representations for users and items in order to augment latent factor models for common matrix factorization-based algorithms such as collaborative filtering, and path-based methods generate recommendations based on searching for reasonable paths between users and items. However, these systems are static and limited in their ability to handle the dynamic and sequential nature of user interactions. [6] Instead, the recommendation problem is better formulated as an MDP and solved using reinforcement learning.

2.2 Reinforcement Learning

In recent years, various techniques applying reinforcement learning to recommender systems have been developed. Afsar et al. [26] provide a comprehensive survey of current algorithms that use RL for the recommendation problem. In general, these systems can be distinctly classified based on the specific algorithm used to optimize recommendations, including traditional and deep reinforcement learning methods.

The closest to our work here is the system proposed by Xian et al. [23], utilizing reinforcement learning over graphs to deliver explainable recommendations. In particular, there is a focus on reasoning about and selecting paths between users and candidate items. Beyond a graph-based framework based on reinforcement learning for the recommender task, our systems differ in problem formulation and implementation. In contrast, we attempt to learn a global representation of the graph in order to model the complete and complex interactions between users and items and from there, use reinforcement learning methods to evaluate recommendations.

CHAPTER 3

Data

Recommender systems received a great deal of attention after the announcement of the Netflix Prize in 2006. From 2006 to 2009, Netflix set to crowdsource a recommendation algorithm 10% more accurate, measured in root-mean-square error (RMSE), than the company's existing system at the time. We focus on the Netflix Prize data for a few reasons. First, it is an open data set, publicly available. [5] Second, Netflix is social. Graph-based models can capture in useful detail the complex relationships between users and the content they watch. Finally, the Netflix Prize data set has the reflection of sequential decision making (timestamps) necessary for a reinforcement learning problem.

The format of the Netflix Prize data is a set of four movie rating text file lists of

Movie ID:

User ID, Rating, Date

Combined, these files contain more than 100 million ratings from 480,189 randomly chosen, anonymous Netflix users over 17,770 movie titles. The data were collected between November 1999 and December 2005, and the ratings are on a scale from 1 to 5 (integral) stars. The movie ID values range from 1 to 17,770 sequentially, while the user ID values range from 1 to 2,649,429 with gaps. Dates have the format YYYY-MM-DD.

The distribution of the number of movies rated by any particular user is heavily right-skewed.

Mean	**SD**	**Min**	**25%**	**50%**	**75%**	**Max**
209.25	302.34	1	39	96	259	17,653

Table 1: Summary of user ratings

As shown in Table 1, the average number of ratings per user is approximately 209, while the median number is 96, and the number of movies rated by a single user ranges between 1 and 17,653.

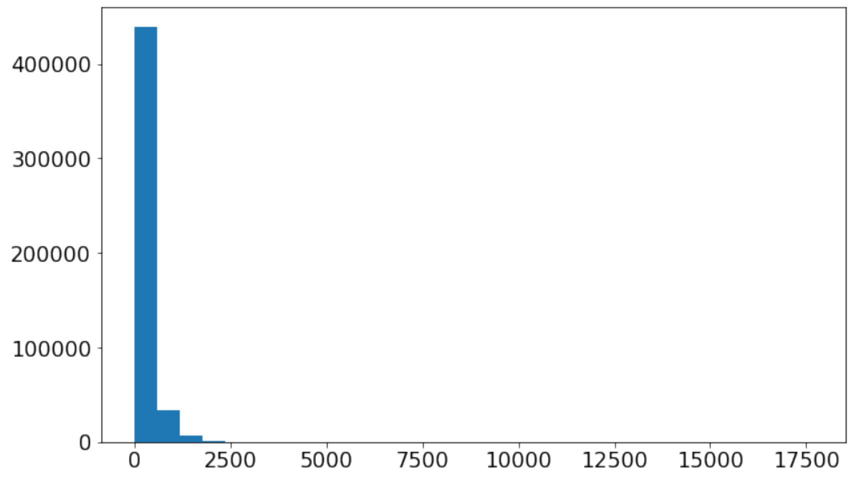

Figure 1: Distribution of ratings by users

The distribution of the number of users who rated any particular movie is also heavily right-skewed.

Mean	**SD**	**Min**	**25%**	**50%**	**75%**	**Max**
5,654.50	16,909.67	3	192	561	2,667.75	232,944

Table 2: Summary of movie ratings

As shown in Table 2, the average number of ratings per movie is approximately 5,654, while the median number is 561, and the number of users who rated a single movie ranges between 3 and 232,944.

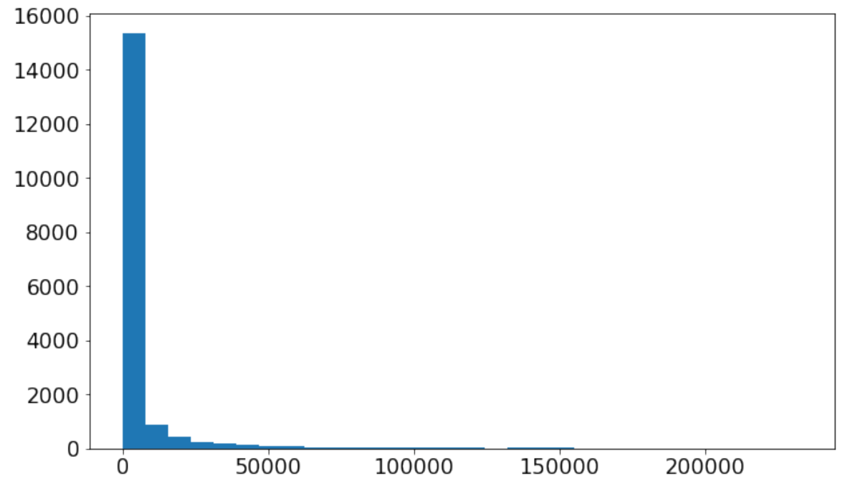

Figure 2: Distribution of ratings of movies

The distribution of ratings by year is shown in Figure 3. The exponential trend can likely be attributed to the growth of Netflix users over the span of the seven years.

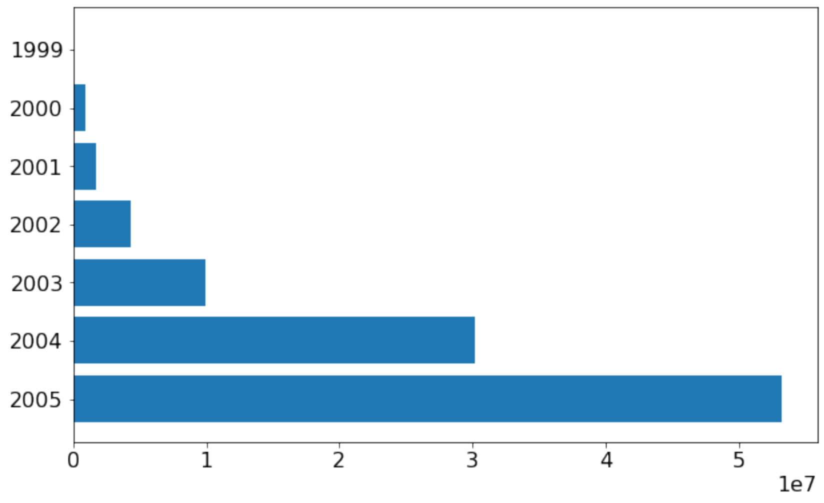

Figure 3: Distribution of ratings by year

We also look at the spread of ratings and see that ratings tend to be relatively positive (greater than 3).

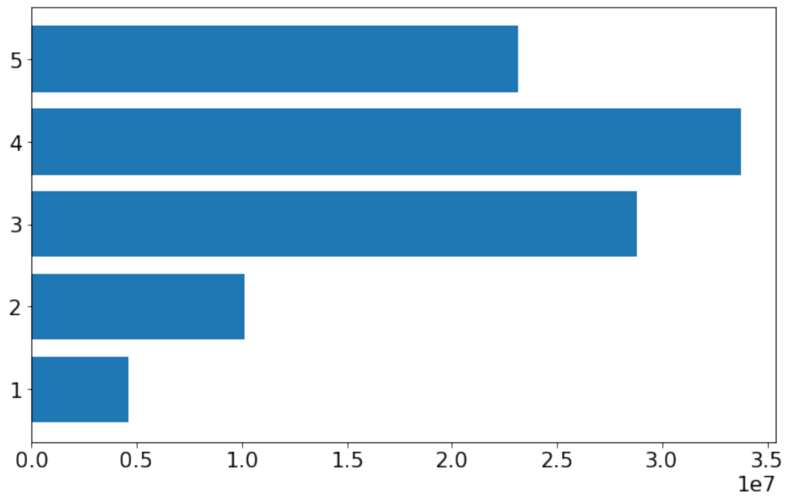

Figure 4: Distribution of ratings

For the network representation of the Netflix Prize data, we build an undirected bipartite graph with two disjoint and independent vertex sets consisting of users and movies. Rating data is included in the network as weighted edges.

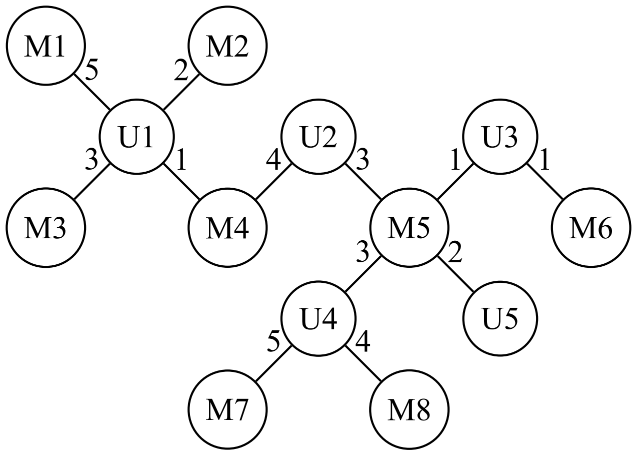

Figure 5: Network structure of Netflix Prize data

Figure 5 presents a visual example of the bipartite graph structure. User vertices are represented with a “U” prefix, while movie vertices are represented with an “M” prefix. Every edge connects a vertex in the user set to one in the movie set and the graph does not contain any odd-length cycles.

CHAPTER 4

Data Preprocessing

The Netflix Prize data is publicly available as a set of four text files, each containing a distinct subset of the complete data set. We first parse each movie list of user ratings and then combine the user, movie, rating, and date values into a single comprehensive data frame. Because the movie ID values range from 1 to 17,770 sequentially and the user ID values range from 1 to 2,649,429 with gaps, there is a subset of user ID values that overlap with movie ID values. To address this issue and allow for two disjoint and independent vertex sets for our bipartite graph, we remove duplicate movie ID and user ID values by offsetting user ID so that the minimum value is strictly greater than the maximum movie ID value.

In order to improve the quality of the data and reduce the size of the massive data set for our recommender system implementation, we keep only the 45% to 55% quantile of rating counts over movies and over users. We drop users who have rated too few movies (not very active) and movies with too few ratings (not very popular) and additionally drop users who have rated too many movies and movies with too many ratings (not representative of user behavior). This results in keeping users who have rated a minimum of 80 movies and a maximum of 117 movies and keeping movies with a minimum of 439 ratings and a maximum of 754 ratings. The reduced subset of data includes 49,530 ratings from 20,814 users over 1,788 movies. The resulting distributions of ratings by users and of movies are shown in Figure 6 and Figure 7, respectively, below.

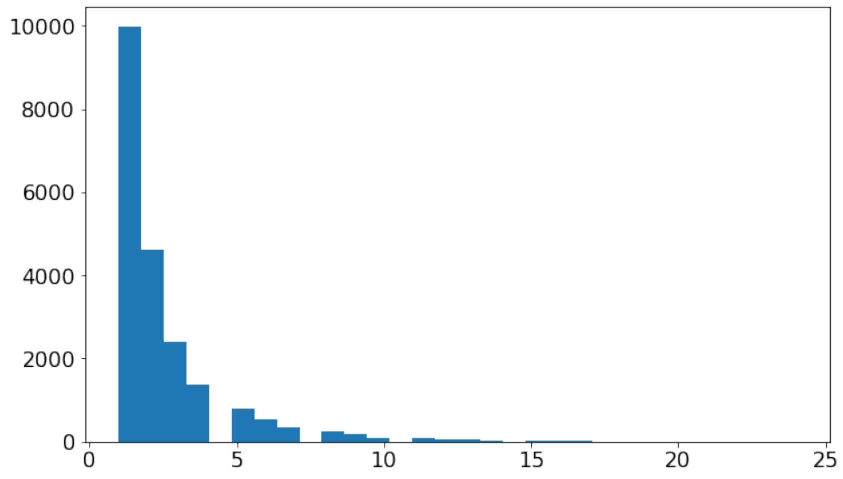

Figure 6: Distributions of ratings by users, reduced data

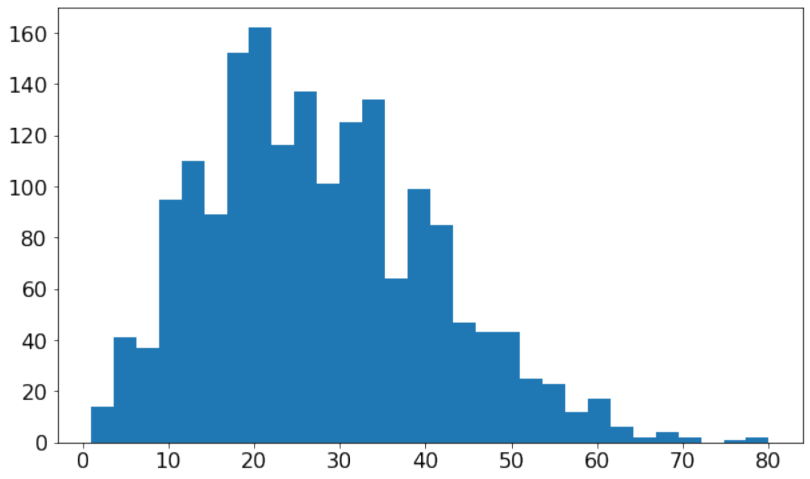

Figure 7: Distributions of ratings of movies, reduced data

From Figure 6 and Figure 7, we see that the distribution of ratings by users in the reduced data set is similar to the complete data set and that the distribution of ratings of movies is much less skewed. We also look at both the distribution of ratings by year (Figure 8) and the distribution of ratings overall (Figure 9) in the reduced data set.

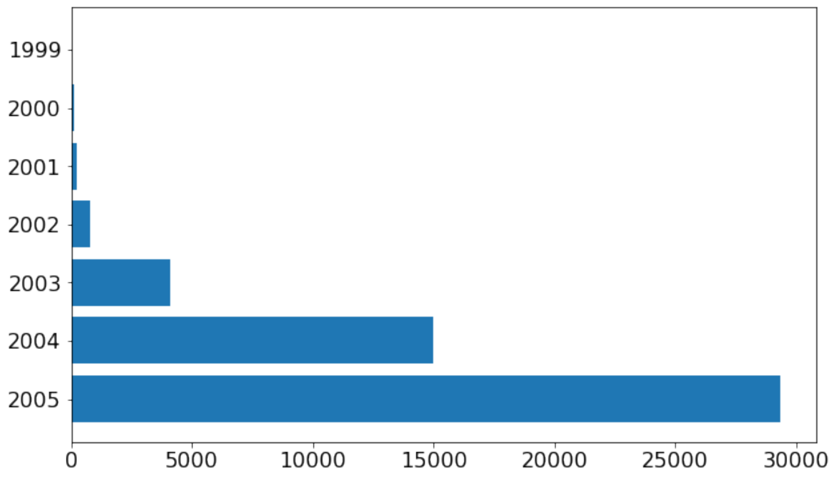

Figure 8: Distribution of ratings by years, reduced data

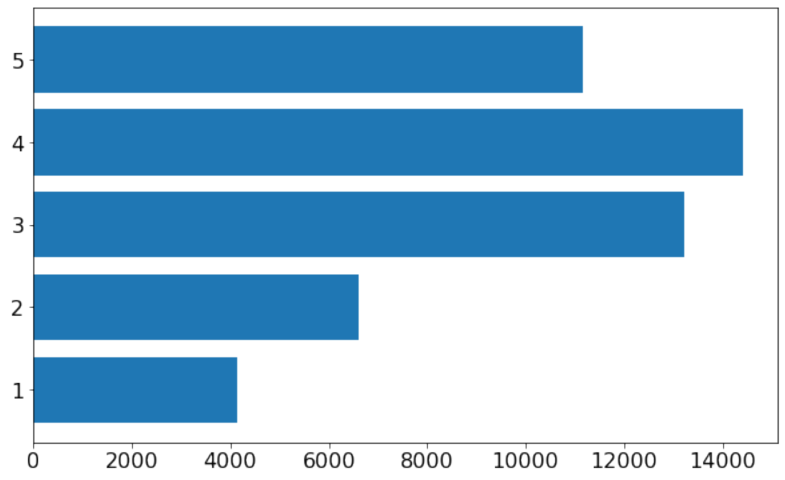

Figure 9: Distribution of ratings, reduced data

Figures 8 and 9 above show similar distributions to the complete data set. We do not lose significant information by limiting the full Netflix Prize data set to the 45% to 55% quantile of activity.

CHAPTER 5

Reinforcement Learning Specification

Reinforcement learning is a computational approach to learning from interactions between an active decision-making agent (the learner) and its environment (everything outside the learner). Formally, we can frame the RL problem as an MDP shown in Figure 10 below from Sutton and Barto's *Reinforcement Learning: An Introduction*.

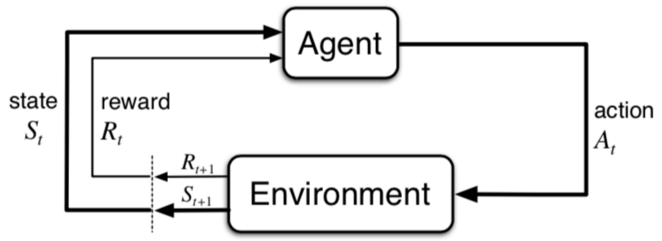

Figure 10: Agent-environment interaction in an MDP

MDPs are a formalization of sequential decision making, involving delayed rewards and the need to balance immediate and future gains. At each time step t, the agent receives some representation of the environment's state S_t from the set of all possible states, which is the state space. Based on the state, the agent selects an action A_t from the set of all possible actions, which is the action space. One time step later (as a consequence of its action and the environment), the agent receives a numerical reward R_{t+1} and finds itself in a new state S_{t+1}. The trajectory is $\{S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, \ldots, R_T, S_T, A_T\}$ where the time of termination T is a random variable defined to be the final time step of a learning task. The goal is for the agent to model good policies for sequential decision problems by maximizing the cumulative reward in the long run through its choice of actions. [18] These policies map states of the environment to actions and define an agent's behavior at any time t.

In general, the agent attempts to maximize expected discounted return

$$\max \ \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R_t\right] \tag{1}$$

where $0 \leq \gamma \leq 1$ is a parameter called the discount rate that determines the trade-off between immediate and future rewards. From the reward signals R_t, the agent learns the values of states in the state space, the long run amount of reward an agent expects to accumulate starting from a given state.

Many algorithms have been proposed to solve the RL problem and can generally be divided into tabular solution methods and approximate solution methods. Tabular solution methods can often find exact optimal policies and value functions because the state and action spaces are small enough to be represented as tables. In contrast, approximate solution methods can find only approximate solutions but can be applied to arbitrarily large state and action spaces.

Here, we specify the RL problem for a movie recommender system using the Netflix Prize data. User interactions with recommendations are iterative and have temporal order, and the sequential nature of the decision problem translates well to be modeled as an MDP and solved by RL methods. The recommender system is the agent, and its environment consists of Netflix users (and their behaviors) as well as the scope of available movies. We define the time of termination T of a trajectory to be the last movie a given Netflix user watches. The state space, action space, transition function, and reward function are defined as follows.

5.1 State Space

The state space is all possible graphs of user rating data. Recommender systems take information about users and their preferences and output predictions of what they might like. In the graph-based system, users are connected to the movies that they have watched (and rated), and are therefore also connected to other users who have done the same. These relations can capture complex user behaviors and provide good representations of the environment for the agent.

For a given graph, we use node2vec [15] to learn continuous vector feature representations for the user and movie nodes. The state at a particular time S_t is the learned vector in the current graph of the user receiving a recommendation.

5.2 Action Space

The action space is the set of all possible movies, which is the collection of movies available on Netflix. In the case of the Netflix Prize data, it is the collection of 17,770 movies in the full data and 1,788 movies in the reduced data. The action at a given time A_t of the agent (recommender system) is the t-th movie served to a user.

5.3 Transition Function

The transition function records the probability of transitioning from state S to state S' after taking action A while obtaining a scalar reward R. We use a model-free (trial-and-error learning) training method for off-policy approximation detailed in Chapter 5.5 and so do not define explicit transition probabilities. More generally, state S' is the learned vector representation of a given user in the graph after the agent takes action A, which is serving a movie recommendation. The state S' is identical to state S if the user rejects the recommendation. If the user accepts the recommendation and provides a rating, an edge between the user and movie vertices is added to the graph, and vector representations for each node are recalculated.

5.4 Reward Function

Click-through rate (CTR) is a common metric of success for digital recommender systems. For Netflix, it can be defined as the total number of views from all users divided by the total number of recommendations served. However, for the RL problem, we want to maximize life-time value (LTV). [13] LTV is the total number of views from all users divided by the total number of users. We want to maximize the total number of times any given user will view over multiple sessions of watching, instead of the greedy approach that maximizes CTR and assumes that views on Netflix are independent and identically distributed (iid). We explore the effects of a few different reward functions defined below.

5.4.1 Fixed Reward

We first investigate a fixed reward. We define R_t to be a numerical reward at time t, which is +100 if the user enjoys the (temporally relevant) movie recommendation and gives it a generally positive rating of 3 or greater, -100 if the user rates the movie poorly (a rating of 1 or 2), and -1 otherwise. In the case that the scalar reward is -1, a small penalty, a recommendation is served but ignored by the user, taking into account the user's implicit feedback.

5.4.2 Linear Reward

Next, we explore a linear reward function. In order to further encourage the agent to continuously provide good, sequential recommendations, we define R_t to be a function of a user's rating of a given movie recommendation as well as the agent's ability to consistently produce a quality recommendation. At time t,

$$R_t = Cx \tag{2}$$

where C is the fixed numerical reward defined in Chapter 5.4.1 and x is the current count of consecutive and positively rated movies the agent has recommended. If the agent recommends a poorly rated or rejected movie, x resets to 1.

5.4.3 Quadratic Reward

Finally, we examine a quadratic reward function, similar to the linear reward function, where the reward at time t is

$$R_t = Cx^2 \tag{3}$$

C and x are defined and discussed in Chapter 5.4.2.

5.5 Off-policy Approximation

To solve the sequential decision problem of movie recommendations over time, we learn estimates for the optimal value of each movie recommended (action), defined as the expected sum of future rewards when recommending that movie and following the optimal sequence of movies recommendations (policy) thereafter. The expected sum of future rewards is the expected value users derive from enjoying, or not enjoying, Netflix over the duration of their subscriptions to the service (LTV). Under a given policy π, the true value of an action A in a state S is

$$Q_\pi(S, A) := \mathbb{E}(R_1 + \gamma R_2 + \ldots | S_0 = S, A_0 = A, \pi) \tag{4}$$

The optimal value is $Q_*(S, A) = \max_\pi Q_\pi(S, A)$, and the optimal policy follows as the highest valued action in each state. Because the state and action spaces in our recommender system are too large for tabular solution methods, we use Double DQN (deep Q-learning), an off-policy approximate solution method, to learn these estimates and train the RL agent. [14]

CHAPTER 6

Recommender System Implementation

We choose an in-memory graph processing engine with no partitioning in order to avoid the overhead of distributed graph stores and to enable our real-time design. [11] We use NetworkX, a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. This allows for the simplicity and flexibility of Python for initial development of the recommendation algorithm. We choose a bipartite graph to store the Netflix user-rating-movie data with user and movie nodes connected by integer rating weighted edges.

Using the reduced data set, we load an initial graph containing user movie ratings prior to a date threshold set by a given user's first rating and predict on future recommendations and ratings to reflect the sequential decision making needed for the RL problem. To train the RL agent, we implement Double DQN with experience replay using PyTorch libraries. [22]

6.1 Double DQN

We use an approximate solution method for training our recommender system because the state space and action space are arbitrarily large. It is the combination of deep learning with traditional reinforcement learning methods that enables these solutions. We choose Double DQN to mitigate action value overestimation [14] and implement experience replay to improve the performance of the algorithm. [12] Pseudocode is provide below.

Algorithm 1 Double DQN with experience replay

```
1: Initialize primary network $Q_\theta$, target network $Q_{\theta'}$, replay buffer $\mathcal{D}$, $\tau << 1$
2: for each iteration do
3:     for each environment step do
4:         Observe state $S_t$ and select $A_t \sim \pi(A_t, S_t)$
5:         Execute $A_t$ and observe next state $S_{t+1}$ and reward $R_t = R(S_t, A_t)$
6:         Store $(S_t, A_t, R_t, S_{t+1})$ in replay buffer $\mathcal{D}$
7:     end for
8:     for each update step do
9:         sample $e_t = (S_t, A_t, R_t, S_{t+1}) \sim \mathcal{D}$
10:        Compute target $Q$ value: $Q^*(S_t, A_t) \approx R_t + \gamma Q_\theta(S_{t+1}, \arg\max_{A'} Q_{\theta'}(S_{t+1}, A'))$
11:        Perform gradient descent step on $(Q^*(S_t, A_t) - Q_\theta(S_t, A_t))^2$
12:        Update target network parameters: $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$
13:    end for
14: end for
```

6.2 Training the Recommender System

The complete set of vector representations are calculated using node2vec for the initially loaded graph with embeddings of size 128. We set the walk length for learning these embeddings to 25 (we use an odd-valued walk length to accommodate the structure of the bipartite graph) and use the integer rating values as edge weights. At each time point t, and based on the current state of the graph, the agent makes a recommendation prediction for a given user. We compare that recommendation to the next movie watched and rated by the same user, updating the environment based on the feedback loop defined in Chapter 5. We train over a sample of 100 users and 500 episodes for our reinforcement learning task.

CHAPTER 7

Analysis and Results

We implement the graph-based Netflix movie recommender system trained using reinforcement learning and present the results here.

7.1 Exploring Reward Functions

In Chapter 5.4, we defined three distinct reward functions, signals passed from the environment to the agent that formalize the goal of the reinforcement learning task. For each reward definition, we show a time series of scores for a single run as well as over an average of 20 runs, followed by a discussion below.

7.1.1 Fixed Reward

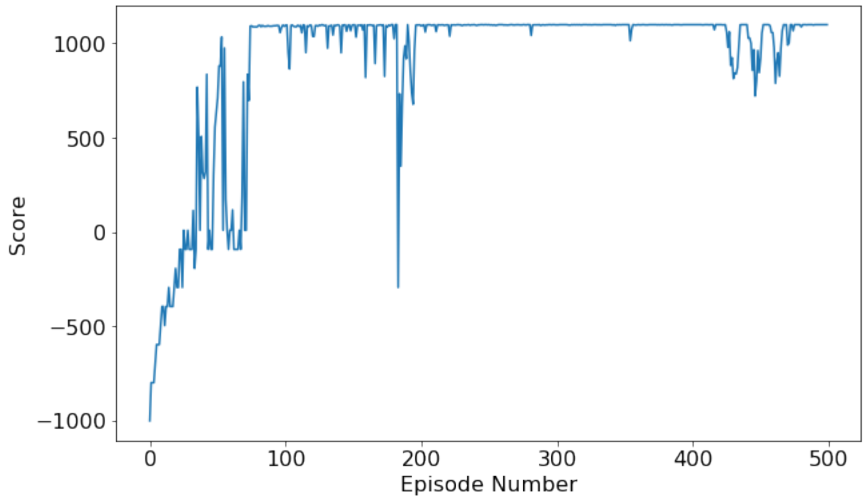

Figure 11: Time series of scores for a single run

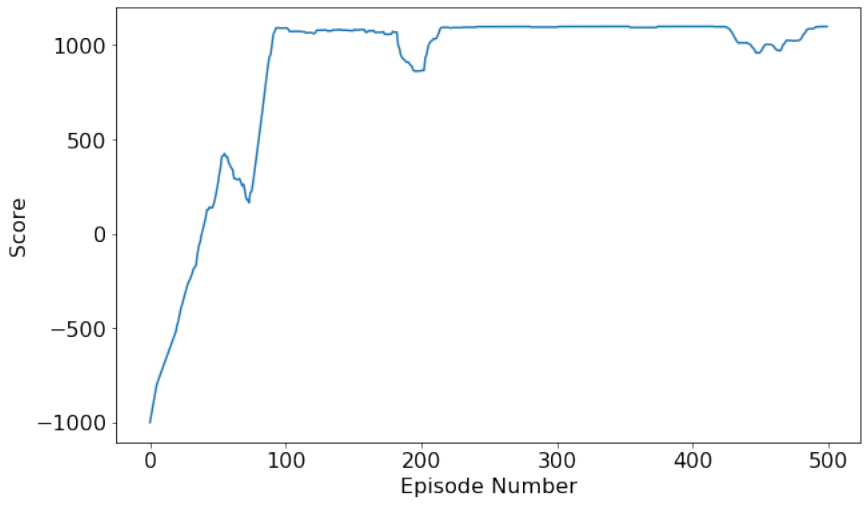

Figure 12: Time series of scores over an average of 20 runs

7.1.2 Linear Reward

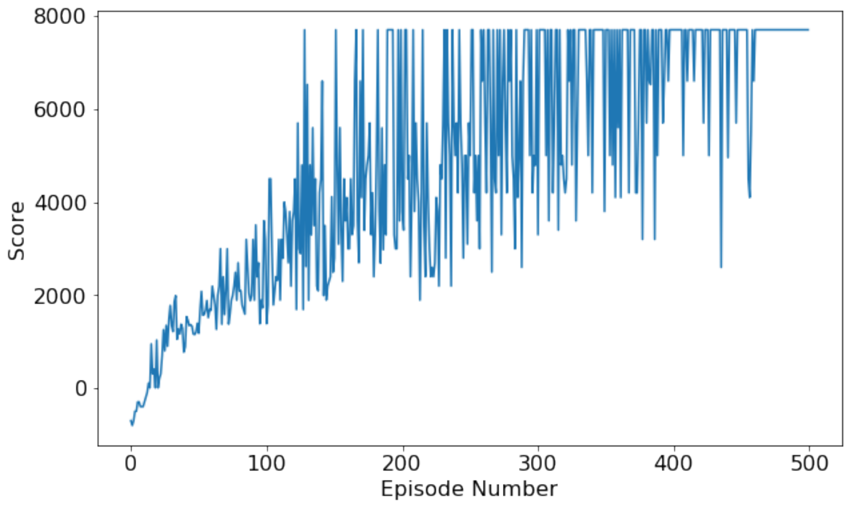

Figure 13: Time series of scores for a single run

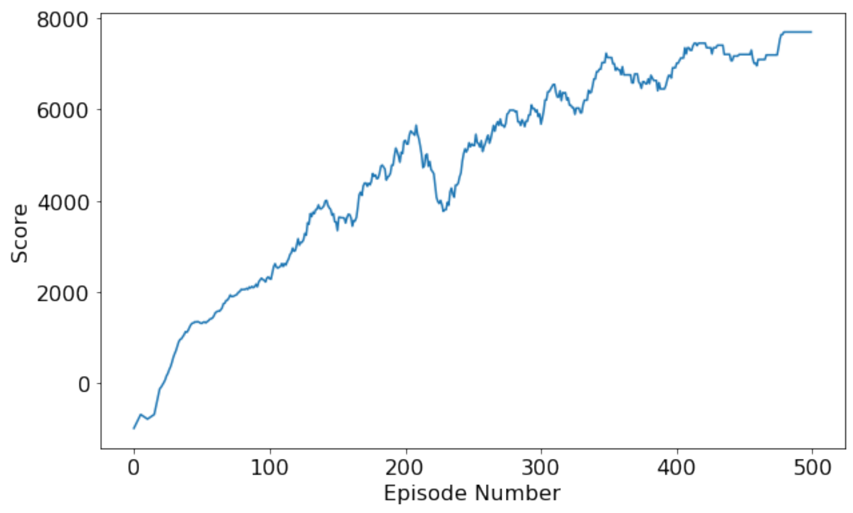

Figure 14: Time series of scores over an average of 20 runs

7.1.3 Quadratic Reward

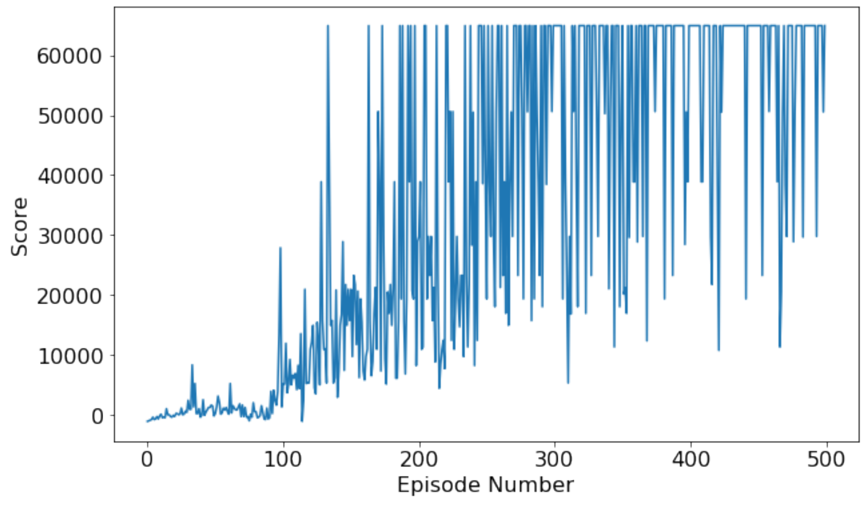

Figure 15: Time series of scores for a single run

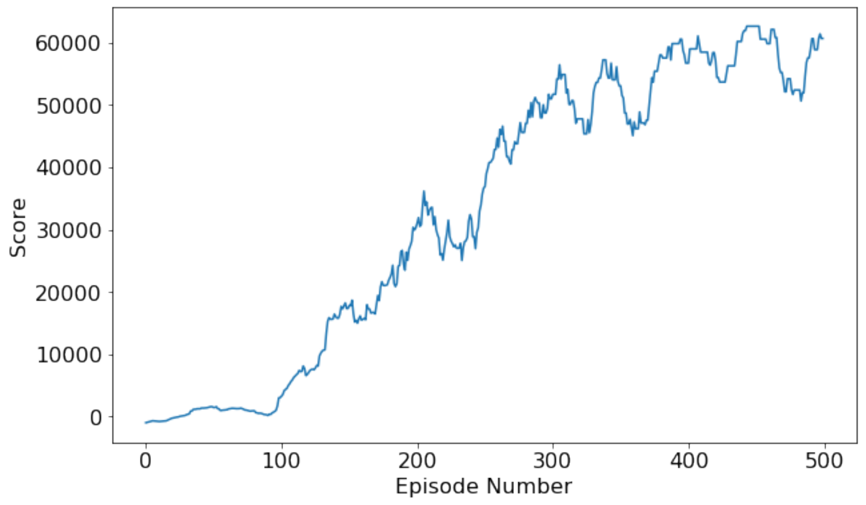

Figure 16: Time series of scores over an average of 20 runs

7.2 Summary

At each time step t, the reward R_t is a numerical value that defines the objective of the agent. The agent's goal is to maximize the total reward it receives over the long run. For our recommender system, the rewards signal user behavior and preferences, and the "long run" is the duration of a user's subscription to Netflix. The agent recommends to a given user a movie from the collection available and generally receives a large positive reward for highly rated recommendations, a large negative reward for poorly rated recommendations, and a small negative reward for rejected recommendations. We consider several different reward functions in order to explore the effects of the reward signal on recommendation quality and convergence.

With the fixed reward function, the agent receives a predetermined, fixed value based on a user's response to a provided recommendation. For the linear and quadratic reward functions, the agent receives a signal based on a user's response as well as how many consecutive, highly rated movies the agent is able to recommend at each time step. The motivation behind this is to further encourage the agent to continuously serve good recommendations.

For each reward definition, the agent is able to consistently learn the next best movie to recommend based on the current state of the graph and given a particular user. However, the agent more quickly converges on the optimal sequence with the fixed reward signal compared to both the linear and quadratic reward signals. In the linear and quadratic returns, there are a larger range of possible scores, depending on the trajectory of the agent-environment interaction. This increases the complexity of the recommendation task, and the agent requires a greater number episodes to approximate the optimal policy.

In each case, the system continuously improves the recommendation policy by making adjustments in response to user feedback, which are explicitly the ratings provided and implicitly the rejections of previous recommendations. The vector representation of the user-item interaction graph preserves complex properties, including structure and information, and conveys to the agent the state of user preferences at a particular time. Based on the estimated values of actions (movies), the agent makes recommendations. From Figures 11-16, we see that the agent is able to learn the sequential nature of user behavior and user interaction with a recommender system. Despite the extensive size of the state space and the many available Netflix movies that make up the action space, our system is able to find a good approximate solution and provide quality recommendations under the constraint of limited computational resources.

CHAPTER 8

Discussion

Formulating the recommendation problem as an MDP and solving using reinforcement learning allows us to capture the sequential nature of user-item interactions. We store these relationships in a graph and model a global representation of the complex connections to learn quality recommendations for users. The vectorized feature representations for the nodes provide useful state space information which can then be used to determine the next item to recommend.

We trained our system on a small subset of the publicly available Netflix Prize data set, keeping users who were actively providing ratings and movies that were actively being rated within the 45% to 55% quantile. While the initial implementation of our recommender system shows encouraging results, there are many future research directions we outline here.

8.1 Future Work

A limitation of the Netflix Prize data set, is that while we can reflect sequential decision making by taking advantage of the timestamped movie ratings, we do not have the continuity and feedback of a complete advertising campaign. A complete advertising campaign would reduce the number of falsely rejected recommendations we currently model and likely improve the reward signal for the reinforcement learning agent.

There are also a number of future directions related to the RL problem. One direction is to implement and evaluate other RL methods such as policy gradient methods, including REINFORCE, that learn a parameterized policy instead of the action-value function learned by deep Q-networks. Additionally, with a larger data set, the domain becomes too extensive to be exhaustively explored. To address this, we can further study the effects of reward shaping [17] (beyond the several reward functions we explored) as a technique to provide localized feedback and guide the learning process for more efficient convergence. For our proposed reinforcement learning algorithms, there is also the complex and challenging task of hyperparameter optimization. The size of the learned vector embeddings that model the state, the number of hidden layers and their dimensions in the neural network, learning rates, etc. all require more investigation.

In the current implementation of our graph-based system for recommendations, we use node2vec to learn the continuous feature vector representations for the nodes that compose the state space. The node2vec framework is well-studied and flexible, but an interesting alternative could be to instead use BiNE [20]. It is a representation learning method specifically for bipartite graphs and could potentially lead to embeddings that are more reflective of the graph structure that models user-item relationships.

Another innovation is to incorporate supplementary graphs, such as distinct user and item graphs, that can augment the state representations. Users and items can be semantically connected based on common tags that can also be learned through labeling tasks. These additional graphs can provide greater context for the recommender system and be used to define a more complete state space.

Beyond the algorithmic and implementation details, we also discuss future performance considerations related to various graph analysis libraries. NetworkX is widely used and well-supported, but the package has many performance limitations due to its Python based implementation. Other C or C++ based alternatives, including Stanford Network Analysis Platform (SNAP), graph-tool, and igraph, to name a few, provide much faster computation over large graphs. [25]

8.2 Additional Recommendation Tasks

Beyond movie recommendations, our system can easily be adapted for other content discovery applications as well, including product recommendations for e-commerce platforms and content and news recommendations for media platforms. The formal implementation changes only in the data stored in the bipartite graph and can be used across many domains for personalized recommendations.

CHAPTER 9

Conclusion

Recommender systems are extensively used to deliver personalized content to users. In this paper, we proposed a new algorithm attempting to bridge various advancements in the field, including graph-based methods and reinforcement learning methods. We implemented using the Netflix Prize data set and showed that quality and continuous recommendations can be learned through this process. It is a dynamic system that improves the static nature of traditional recommender systems, and it is a flexible approach that can also be extended to many other applications that model and predict relationships between users and items.

REFERENCES

[1] Lawrence Page et al. *The PageRank citation ranking: Bringing order to the web.* Tech. rep. Stanford InfoLab, 1999.

[2] Ronny Lempel and Shlomo Moran. "SALSA: the stochastic approach for link-structure analysis". In: *ACM Transactions on Information Systems (TOIS)* 19.2 (2001), pp. 131–160.

[3] Gediminas Adomavicius and Alexander Tuzhilin. "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions". In: *IEEE transactions on knowledge and data engineering* 17.6 (2005), pp. 734–749.

[4] Hsinchun Chen, Xin Li, and Zan Huang. "Link prediction approach to collaborative filtering". In: *Proceedings of the 5th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'05)*. IEEE. 2005, pp. 141–142.

[5] Netflix. *Netflix Prize data*. 2006. URL: `https://www.kaggle.com/netflix-inc/netflix-prize-data`.

[6] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. "Fast incremental and personalized pagerank". In: *arXiv preprint arXiv:1006.2880* (2010).

[7] Fidel Cacheda et al. "Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems". In: *ACM Transactions on the Web (TWEB)* 5.1 (2011), pp. 1–33.

[8] Lalita Sharma and Anju Gera. "A survey of recommendation system: Research challenges". In: *International Journal of Engineering Trends and Technology (IJETT)* 4.5 (2013), pp. 1989–1992.

[9] Andrew Zimdars, David Maxwell Chickering, and Christopher Meek. *Using Temporal Data for Making Recommendations*. 2013. arXiv: `1301.2320 [cs.IR]`.

[10] Kibeom Lee and Kyogu Lee. “Escaping your comfort zone: A graph-based recommender system for finding novel recommendations among relevant items”. In: *Expert Systems with Applications* 42.10 (2015), pp. 4851–4858.

[11] J. Lin. “Is Big Data a Transient Problem?” In: *IEEE Internet Computing* 19.5 (2015), pp. 86–90. DOI: `10.1109/MIC.2015.97`.

[12] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[13] Georgios Theocharous, Philip S Thomas, and Mohammad Ghavamzadeh. “Personalized ad recommendation systems for life-time value optimization with guarantees”. In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015.

[14] Hado Van Hasselt, Arthur Guez, and David Silver. “Deep reinforcement learning with double q-learning”. In: *arXiv preprint arXiv:1509.06461* (2015).

[15] Aditya Grover and Jure Leskovec. “node2vec: Scalable feature learning for networks”. In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 855–864.

[16] Aneesh Sharma et al. “GraphJet: real-time content recommendations at twitter”. In: *Proceedings of the VLDB Endowment* 9.13 (2016), pp. 1281–1292.

[17] Marek Grzes. “Reward shaping in episodic reinforcement learning”. In: (2017).

[18] Andrew Barto and Richard S. Sutton. *Reinforcement Learning: An Introduction*. The MIT Press, 2018. ISBN: 9780262039246.

[19] Chantat Eksombatchai et al. “Pixie: A system for recommending 3+ billion items to 200+ million users in real-time”. In: *Proceedings of the 2018 world wide web conference*. 2018, pp. 1775–1784.

[20] Ming Gao et al. “Bine: Bipartite network embedding”. In: *The 41st international ACM SIGIR conference on research & development in information retrieval*. 2018, pp. 715–724.

[21] Yongfeng Zhang et al. “Learning over knowledge-base embeddings for recommendation”. In: *arXiv preprint arXiv:1803.06540* (2018).

[22] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[23] Yikun Xian et al. “Reinforcement knowledge graph reasoning for explainable recommendation”. In: *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval*. 2019, pp. 285–294.

[24] Qingyu Guo et al. “A survey on knowledge graph-based recommender systems”. In: *IEEE Transactions on Knowledge and Data Engineering* (2020).

[25] Timothy Lin. “Benchmark of popular graph/network packages v2”. In: *Quasilinear Musings* (May 10, 2020).

[26] M. Mehdi Afsar, Trafford Crump, and Behrouz Far. *Reinforcement learning based recommender systems: A survey*. 2021. arXiv: `2101.06286 [cs.IR]`.