Submodularity on Hypergraphs: From Sets to Sequences

Marko Mitrovic
Yale University

Moran Feldman
Open University of Israel

Andreas Krause
ETH Zurich

Amin Karbasi
Yale University

Abstract

In a nutshell, submodular functions encode an intuitive notion of diminishing returns. As a result, submodularity appears in many important machine learning tasks such as feature selection and data summarization. Although there has been a large volume of work devoted to the study of submodular functions in recent years, the vast majority of this work has been focused on algorithms that output sets, not sequences. However, in many settings, the order in which we output items can be just as important as the items themselves.

To extend the notion of submodularity to sequences, we use a directed graph on the items where the edges encode the additional value of selecting items in a particular order. Existing theory is limited to the case where this underlying graph is a directed acyclic graph. In this paper, we introduce two new algorithms that provably give constant factor approximations for general graphs and hypergraphs having bounded in or out degrees. Furthermore, we show the utility of our new algorithms for real-world applications in movie recommendation, online link prediction, and the design of course sequences for MOOCs.

1 Introduction

1.1 Preliminaries and Related Work

Intuitively, submodularity describes the set of functions that exhibit diminishing returns. Mathematically, a set function $f : 2^V \to \mathbb{R}$ is **submodular** if, for every two sets $A \subseteq B \subseteq V$ and element $v \in V \setminus B$, we have $f(A \cup \{v\}) - f(A) \geq f(B \cup \{v\}) - f(B)$. That is, the marginal contribution of any element v to the value of $f(A)$ diminishes as the set A grows.

As such, submodularity commonly appears in a wide variety of fields including machine learning, combinatorial optimization, economics, and beyond. Sample applications include variable selection (Krause and Guestrin, 2005), data summarization (Mirzasoleiman et al., 2016; Lin and Bilmes, 2011; Kirchhoff and Bilmes, 2014), recommender systems (Gabillon et al., 2013), crowd teaching (Singla et al., 2014), neural network interpretability (Elenberg et al., 2017), network monitoring (Gomez Rodriguez et al., 2010), and influence maximization in social networks (Kempe et al., 2003).

A submodular function f is said to be **monotone** if $f(A) \leq f(B)$ for every two sets $A \subseteq B \subseteq V$. That is, adding items to a set cannot decrease its value. A seminal result in submodularity states that if our utility function f is monotone submodular (and non-negative), then the classical greedy algorithm maximizes f subject to a cardinality constraint up to an approximation ratio of $1 - 1/e$ (Nemhauser et al., 1978). Since then, the study of submodular functions has been extended to a broad variety of different settings, including non-monotone submodularity (Feige et al., 2007; Buchbinder et al., 2014), adaptive submodularity (Golovin and Krause, 2011), weak submodularity (Das and Kempe, 2011), and continuous submodularity (Wolsey, 1982; Bach, 2015), just to name a few.

Despite the above, the vast majority of existing results are limited to the scenario where we wish to output sets, not sequences. Alaei and Malekian (2010) and Zhang et al. (2016) consider functions they call string- or sequence-submodular, but it is in a different context. In this paper, we use a directed graph on the items where the edges encode the additional value of selecting items in a particular order.

Proceedings of the 21^{st} International Conference on Artificial Intelligence and Statistics (AISTATS) 2018, Lanzarote, Spain. PMLR: Volume 84.

The only known theoretical result for this setting is limited to the case where the underlying graph is a directed acyclic graph (Tschiatschek et al., 2017). Considering sequences instead of sets causes an exponential increase in the size of the search space, but it allows for much more expressive models.

For example, consider the problem of recommending movies to a user. A recommendation system could determine that the user might be interested in The Lord of the Rings franchise. However, if the model does not consider the order of the movies it recommends, the user may watch The Return of the King first and The Fellowship of the Ring last, which is likely to lead make the user totally unsatisfied with an otherwise excellent recommendation. With this example as motivation, the next section gives a more detailed description of the problem we consider.

1.2 Problem Description

Tschiatschek et al. (2017) was the first to consider this particular submodular sequence setting and we will closely follow their setup of the problem. Recall that the goal is to select a **sequence** of items that will maximize some given objective function. To generalize the problem description, we will refer to items as vertices from now on.

Let $V = \{v_1, v_2, \ldots, v_n\}$ be the set of n vertices (items) we can pick from. A set of edges E encodes the fact that there is additional value in picking certain vertices in a certain order. More specifically, an edge $e_{ij} = (v_i, v_j)$ encodes the fact that there is additional utility in selecting v_j after v_i has already been chosen. Self-loops (i.e., edges that begin and end at the same vertex) encode the fact that there is some individual utility in selecting a vertex.

In general, our input consists of a directed graph $G = (V, E)$, a non-negative monotone submodular set function $h\colon 2^E \to \mathbb{R}_{\geq 0}$, and a parameter k. The objective is to output a non-repeating sequence σ of k unique nodes that maximizes the objective function:

$$f(\sigma) = h\big(E(\sigma)\big) \ ,$$

where

$$E(\sigma) = \big\{(\sigma_i, \sigma_j) \mid (\sigma_i, \sigma_j) \in E, i \leq j\big\} \ .$$

We say that $E(\sigma)$ is the set of edges induced by the sequence σ. It is important to note that the function h is a submodular set function over the edges, not over the vertices. Furthermore, the objective function f is neither a set function, nor is it necessarily submodular on the vertices.

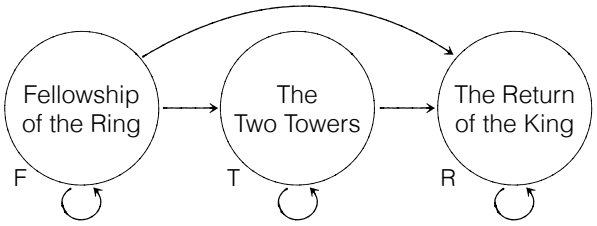


Figure 1: Graph for The Lord of the Rings franchise. The self-loops encode the fact that each movie has some individual value. The edges encode the fact that there is additional utility in watching the movies in the correct order. Notice that the utility of watching The Return of the King after having already seen both The Fellowship of the Ring and The Two Towers is higher than the utility of watching The Return of the King after having seen just one of the two.

For example, consider the graph in Figure 1, and let $h\big(E(\sigma)\big) = |E(\sigma)|$. That is, the value of a sequence is simply the number of edges induced by that sequence. Consider the sequence $\sigma_A = (F)$ where the user has watched only The Fellowship of the Ring, the sequence $\sigma_B = (T)$ where the user watched only The Two Towers, and the sequence $\sigma_C = (F, T)$ where the user watched The Fellowship of the Ring and then The Two Towers:

$$f(\sigma_A) = f(F) = h\big((F, F)\big) = 1 \ .$$
$$f(\sigma_B) = f(T) = h\big((T, T)\big) = 1 \ .$$
$$f(\sigma_C) = f(F, T) = h\big((F, F), (F, T), (T, T)\big) = 3 \ .$$

This example shows that although the marginal gain of the edges is non-increasing in the context of a growing set of edges (i.e., the function h is submodular on the edges), it is clear that the function f is *not* submodular on the vertices. In particular, the marginal gain of The Two Towers is larger once the user has already viewed The Fellowship of the Ring.

Furthermore, just to fully clarify the concept of edges being induced by a sequence, consider the sequence $\sigma_D = (T, F)$ where the user watched The Two Towers and then The Fellowship of the Ring.

$$f(\sigma_D) = f(T, F) = h\big((T, T), (F, F)\big) = 2 \ .$$

Notice that although sequences σ_C and σ_D contain the same movies, the order of σ_D means that the edge (F, T) is not induced, and thus, the value of the sequence is lower.

1.3 Our Contributions

Throughout this paper we use the notation $\Delta = \min\{d_{\text{in}}, d_{\text{out}}\}$, where $d_{\text{in}} = \max_{v \in V} d_{\text{in}}(v)$ and $d_{\text{out}} = \max_{v \in V} d_{\text{out}}(v)$. The previous work on our problem, due to Tschiatschek et al. (2017), presented an algorithm (OMegA) enjoying a $(1-e^{-\frac{1}{2\Delta}})$-approximation guarantee when the underlying graph G is a directed acyclic graph (except for self-loops).

In this paper, we present two new algorithms: Sequence-Greedy and Hyper Sequence-Greedy, which also provably achieve constant factor approximations (when Δ is constant), but their guarantees hold for general graphs and hypergraphs, respectively. Although the example given in Figure 1 is indeed a directed acyclic graph, many real-world problems require a general graph or hypergraph.

We showcase the utility of our algorithms on real world applications in movie recommendation, online link prediction, and the design of course sequences for massive open online courses (MOOCs). Furthermore, we show that even when the underlying graph is a directed acyclic graph, our general graph algorithm performs comparably well. Our experiments also demonstrate the power of being able to utilize hypergraphs and hyperedges.

2 Theoretical Results

All proofs are given in the Appendix.

2.1 General Graphs

In this section, we present our first algorithm, Sequence-Greedy. Sequence-Greedy is essentially the same as the classical greedy algorithm, but instead of choosing the most valuable vertex at each step, it chooses the most valuable valid edge.

More specifically, we start off with an empty sequence σ. At each step, we define $\mathcal{E}$ to be the set of all edges whose end point is not already in σ. We then greedily select the edge $e_{ij} \in \mathcal{E}$ with maximum marginal gain $h\big(e_{ij} \mid E(\sigma)\big)$, where

$$h\big(e_{ij} \mid E(\sigma)\big) = h\big(E(\sigma) \cup e_{ij}\big) - h\big(E(\sigma)\big) \ .$$

Recall that $e_{ij} = (v_i, v_j)$. That is, v_i is the start point of e_{ij} and v_j is the endpoint. If e_{ij} is a self-loop, then $j = i$ and we append the single vertex v_j to σ. Similarly, if $j \neq i$, but v_i is already in σ, then we still only append v_j. Finally, if e_{ij} has two distinct vertices and neither of them is already in the sequence, we append v_i and then v_j to σ. This description is summarized in pseudo-code in Algorithm 1.

Algorithm 1: Sequence-Greedy (Forward)

Input: Directed graph $G = (V, E)$
Monotone submodular function $h : 2^E \to \mathbb{R}$
Cardinality parameter k

Let $\sigma \leftarrow ()$.
while $|\sigma| \leq k - 2$ **do**
  $\mathcal{E} = \{e_{ij} \in E \mid v_j \notin \sigma\}$. // $e_{ij} = (v_i, v_j)$
  if $\mathcal{E} = \varnothing$ **then** Exit the loop.
  $e_{ij} = \arg\max_{e \in \mathcal{E}} h(e \mid E(\sigma))$.
  if $v_j = v_i$ ***or*** $v_i \in \sigma$ **then**
    $\sigma = \sigma \oplus v_j$. // $\oplus$ means concatenate
  else
    $\sigma = \sigma \oplus v_i \oplus v_j$.
return σ.

Theorem 2.1. *The approximation ratio of Algorithm 1 is at least* $\frac{1-e^{-(1-\frac{1}{k})}}{2d_{in}+1}$.

Notice that the approximation guarantee of Algorithm 1 depends on the maximum in-degree d_{in}. Intuitively, this is because Algorithm 1 builds σ by appending vertices to the end of the sequence. This means that each vertex we add to σ decreases the size of $\mathcal{E}$ by at most d_{in}.

However, one can easily modify Algorithm 1 to build σ backwards by *prepending* vertices to the start of the sequence at each step. More specifically, we redefine $\mathcal{E}$ to be the set of all edges whose *start point* is not already in σ. Again we greedily select the edge $e_{ij} \in \mathcal{E}$ that maximizes $h\big(e_{ij} \mid E(\sigma)\big)$. Now, if e_{ij} is a self-loop or v_j is already in σ, we prepend the single vertex v_i to the start of σ. Otherwise, if e_{ij} has two distinct vertices and neither of them is already in the sequence, we prepend v_j to σ first, and then prepend v_i (thus, maintaining the order). This description is summarized in pseudo-code in Algorithm 2 with the main differences noted as comments.

Algorithm 2 gives the same approximation ratio as Algorithm 1, but with a dependence on d_{out} instead of d_{in}. Thus, if we run both the forwards and backwards version of Sequence-Greedy and take the maximum, we get an approximation ratio that depends on $\Delta = \min\{d_{\text{in}}, d_{\text{out}}\}$. Furthermore, notice that the approximation ratio improves as k increases. Therefore, we can summarize the approximation ratio of Sequence-Greedy as follows.

Theorem 2.2. *As* $k \to \infty$*, the approximation ratio of Sequence-Greedy approaches* $\frac{1-\frac{1}{e}}{2\Delta+1}$.

Algorithm 2: Sequence-Greedy (Backward)

Input: Directed graph $G = (V, E)$
Monotone submodular function $h : 2^E \to \mathbb{R}$
Cardinality parameter k

Let $\sigma \leftarrow ()$.
while $|\sigma| \leq k - 2$ **do**
  $\mathcal{E} = \{e_{ij} \in E \mid v_i \notin \sigma\}$. `// different set` $\mathcal{E}$
  if $\mathcal{E} = \varnothing$ **then** Exit the loop.
  $e_{ij} = \arg\max_{e \in \mathcal{E}} h(e \mid E(\sigma))$.
  if $v_i = v_j$ ***or*** $v_j \in \sigma$ **then**
    $\sigma = v_i \oplus \sigma$.
  else
    $\sigma = v_i \oplus v_j \oplus \sigma$.
  `// vertices appended to beginning of` σ
return σ.

This is comparable to the $(1 - e^{-\frac{1}{2\Delta}})$-approximation guarantee that is achieved by the existing algorithm OMegA, except that our guarantee is valid on general graphs, not just directed acyclic graphs.

In addition to this provable approximation ratio, Sequence-Greedy has the strong advantage of being computationally efficient. Both finding $\mathcal{E}$ and identifying the most valuable edge in $\mathcal{E}$ can be done in $O(m)$ time, where $m = |E|$. Thus, Sequence-Greedy runs in $O(km)$ time. This is faster than OMegA, which runs in $O(m\Delta k^2 \log k)$.

2.2 Extension to Hypergraphs

Extending our results to hypergraphs allows us to encode increasingly sophisticated models. For example, looking back on Figure 1, we see that the value of watching all three movies is just the sum of the pairwise additional values. However, hyperedges allow us to encode the fact that there is even further utility in watching the entire franchise in order.

From this point on, we replace the directed graph G with a directed hypergraph $H = (V, E)$. Each edge $e \in E$ of this directed hypergraph is a non-empty non-repeating sequence of vertices from V. Let $V(e)$ be the set of vertices found in the hyperedge e. We assume that the intersection of a sequence and a set maintains the order of the sequence, which allows us to redefine $E(\sigma)$ as

$$E(\sigma) = \{e \in E \mid \sigma \cap V(e) = e\} \ .$$

Informally, $E(\sigma)$ contains an edge $e \in E$ if and only if all the vertices of e appear in σ in the proper order.

We also need to explain how the concept of in-degrees and out-degrees extends to hypergraphs. Self-loops contribute 1 to both the in-degree and the out-degree of that vertex. For all other edges $e \in E$ such that $v \in V(e)$, they will contribute 1 to $d_{\text{in}}(v)$ if v is not the first vertex of e, and 1 to $d_{\text{out}}(v)$ if v is not the last vertex of e. Finally, we define r as the maximum size of any edge in E. More formally, $r = \max_{e \in E} |e|$.

Aside from the above redefinition of $E(\sigma)$, there is no need to make other changes in the definition of the objective function f. Specifically, it is still defined as $f(\sigma) = h(E(\sigma))$, where $h \colon 2^E \to \mathbb{R}_{\geq 0}$ is a non-negative monotone submodular function.

Our algorithm for hypergraphs, Hyper Sequence-Greedy, is an extension of the original Sequence-Greedy. Again, we start off with an empty sequence σ. This time, at each step we define $\mathcal{E}$ to be the set of all hyperedges $e \in E$ such that $\sigma \cap V(e)$ is a prefix of e. The idea is that we can only select a hyperedge e if the vertices of e included in our sequence σ form a prefix of e, and they appear in σ in the right order. We then select the hyperedge $e^* \in \mathcal{E}$ that has the maximum marginal gain, and append the vertices of e^* (that are not already in our sequence) to σ without changing their order. This description is summarized in pseudo-code in Algorithm 3.

Algorithm 3: Hyper Sequence-Greedy (Forward)

Input: Directed hypergraph $H = (V, E)$
Monotone submodular function h
Cardinality parameter k

Let $\sigma \leftarrow ()$.
while $|\sigma| \leq k - r$ **do**
  Let $\mathcal{E} = \{e \in E \mid \sigma \cap V(e)$ is a prefix of $e\}$.
  if $\mathcal{E} = \varnothing$ **then** Exit the loop.
  $e^* = \arg\max_{e \in \mathcal{E}} h(e \mid E(\sigma))$.
  for *every* $v \in e^*$ *in order* **do**
    if $v \notin \sigma$ **then** $\sigma = \sigma \oplus v$.
return σ.

Theorem 2.3. *The approximation ratio of Algorithm 3 is at least* $\frac{1 - e^{-(1 - \frac{r}{k})}}{r d_{\text{in}} + 1}$.

As with Sequence-Greedy, we can also run Hyper Sequence-Greedy backwards and take the maximum of the two results. In the backwards version, we *prepend* the vertices to the start of the sequence and we can only select a hyperedge e if $V(e) \cap \sigma$ is a suffix of e. Once more, this improves the approximation ratio in the sense that the dependence on d_{in} is replaced with a dependence on $\Delta = \min\{d_{\text{in}}, d_{\text{out}}\}$. Additionally notice that, as before, our approxi-

mation ratio improves as k increases. Thus, we can summarize the performance guarantee of Hyper Sequence-Greedy as follows.

Theorem 2.4. *As $k \to \infty$, the approximation ratio of Hyper Sequence-Greedy approaches $\frac{1-\frac{1}{e}}{r\Delta+1}$.*

Remarks: One can observe that this hypergraph setting is a generalization of the previous directed graph setting. Specifically, Sequence-Greedy and the associated theory is a special case of Hyper Sequence-Greedy for $r = 2$. Furthermore, if $r = 1$ (i.e., our graph has only self-loops) then Hyper Sequence-Greedy is the same as the classical greedy algorithm.

We also note that while Algorithm 3 may select fewer than k vertices, the theoretical guarantees still hold. Furthermore, since we assume that h is monotone, we can safely select k vertices in practice every time. One simple heuristic for extending σ to k vertices is to only consider hyperedges with at most $k - |\sigma|$ vertices.

3 Applications

3.1 Movie Recommendation

In this application, we use the *Movielens 1M* dataset (Harper and Konstan, 2015) to recommend movies to users based on the films they have reviewed in the past. This dataset contains 1,000,209 anonymous, time-stamped ratings made by 6,040 users for 3,706 different movies. As in Tschiatschek et al. (2017), we do not want to predict a user's rating for a given movie, instead we want to predict which movies the user will review next.

One issue with this dataset is that the distribution of the number of ratings per user (shown in Figure 2a) has a very long tail, with the most prolific reviewer having reviewed 2,314 movies. In order for our data to be representative of the general population, we remove all users who have rated fewer than 20 movies or more than 50 movies. We also remove all movies with fewer than 1,000 reviews. This leaves us with 67,757 ratings made by 2,047 users for 207 different movies.

We first group and sort all the reviews by user and time-stamp, so that each user i has an associated sequence σ^i of movies they have rated, where σ^i_j refers to the j^{th} movie that user i has reviewed. We use a 90/10 training/testing split of the data and 10-fold cross validation.

For each user i in the test set (D_{test}), we use their first 8 movies as a given starting sequence $S_i = \{\sigma^i_1 \dots \sigma^i_8\}$. We want to use S_i to select k movies that we think user i will review in the future. Therefore, for each user i, we build a hypergraph $H_i = (V, E_i)$, where $V = \{v_1, \dots, v_n\}$ is the set of all movies, and E_i is a set of hyperedges. Each hyperedge e_s has value p_s, where s is a movie sequence of length at most 3. Intuitively, p_s is the conditional probability of reviewing the last movie in s given that the rest of the movies in s have already been reviewed in the proper order.

Since we use empirical frequencies in the training data to calculate these conditional probabilities, we may run into the issue of overfitting to rare sequences. To avoid this, we add a parameter d to the denominator of our calculation of each edge value. This will increase the relative value for sequences that appear more often. In this experiment, we use $d = 20$.

More formally, define N_s to be the number of users in the training set (D_{train}) that have reviewed all the movies in the sequence s in the proper order. Also define s_l to be last element in s, and s' to be s with s_l removed. Now we can define the value of each edge e_s as follows:

$$p_s = \begin{cases} \dfrac{N_s}{N_{s'} + d} & s' \subseteq S_i \ , \\ p_{s'} \dfrac{N_s}{N_{s'} + d} & \text{otherwise} \ . \end{cases} \tag{1}$$

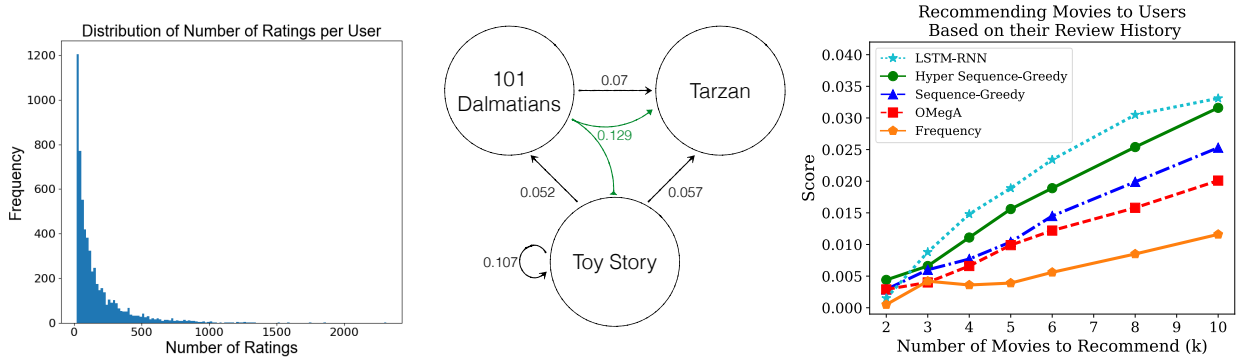
As mentioned above, the idea is that p_s represents the conditional probability of reviewing s_l given that all the movies in s' have already been reviewed in the proper order. If user i has not reviewed all the movies in s', then we scale down the value of that edge by $p_{s'}$ (i.e., the conditional probability of reviewing all the movies in s').

Note that if $s' = \varnothing$, then we define $N_{s'} = |D_{train}|$, thus ensuring that this definition also applies for self-loops. A small subgraph of a fully trained hypergraph is shown in Figure 2b.

We use a probabilistic coverage utility function as our non-negative monotone submodular function h. Mathematically,

$$h(E) = \sum_{v \in nodes(E)} \Big[1 - \prod_{s \in E | s_l = v} (1 - p_s)\Big]$$

We compare the performance of our algorithms, Sequence-Greedy and Hyper Sequence-Greedy,



(a) (b) (c)

Figure 2: (a) Shows the long-tailed distribution of the number of ratings per user in the *Movielens 1M* dataset. (b) Shows a small subgraph of the overall hypergraph H that we train. For clarity, we only show edges with value $p_s > 0.05$, as defined in equation (1). We also highlight the size 3 hyperedge in green. (c) Shows the performance of our algorithms against existing baselines for various cardinalities k.

to the existing submodular sequence baseline (OMegA), as well as a naive baseline (Frequency), which just outputs the most popular movies that the user has not yet reviewed.

We also compare to a simple long short-term memory (LSTM) recurrent neural network (RNN). In addition to tuning parameters, we experimented with various frameworks such as training on uniform vs. variable-sized sequences. In the end, we obtained the best results when we trained the neural network on the first k movies of each σ^i, where the target is to predict the next k movies that the user i will review. In terms of the architecture, we use one layer of 512 LSTM nodes (with a dropout of 0.5) followed by a dense layer with a softmax activation that returns a 207×1 vector P, where entry P_i is the probability that movie i will be reviewed. For each k, we simply return the k highest values in P. As before, we used a 90/10 training/testing split with 10-fold cross validation.

With enough data, neural networks will likely significantly outperform our algorithms. However, with this comparison, we would like to show that in situations where data is relatively scarce, our algorithms are competitive with existing neural network frameworks.

To measure the accuracy of a prediction, we use a modified version of the Kendall tau distance (Kendall, 1938). First, for any sequence σ, we define $T(\sigma)$ to be the set of all ordered pairs in σ. For example, if $\sigma = \{1, 3, 2\}$, then $T(\sigma) = \big[(1, 3), (1, 2), (3, 2)\big]$.

Let P_i be our predicted sequence for the next k movies that user i will review, and let Q_i be the next k movies that user i actually reviewed. Then, we define the accuracy of the prediction P_i as follows.

$$\tau(P_i, Q_i) = \frac{|T(P_i) \cap T(Q_i)|}{|T(Q_i)|}$$

In other words, $\tau(P_i, Q_i)$ is the fraction of ordered pairs of the true answer Q_i that appear in our prediction P_i. Our experimental results in terms of this accuracy measure are summarized in Figure 2c.

These results showcase the power of using hypergraphs, as Hyper Sequence-Greedy consistently outperforms Sequence-Greedy. We also notice that Hyper Sequence-Greedy outperforms the score of the existing baseline OMegA by roughly 50%.

3.2 Online Link Prediction

In this application, we consider users who are searching through Wikipedia for some target article. Given a sequence of articles they have previously visited, we want to predict which link they will follow next. We use the Wikispeedia dataset (West et al., 2009), which consists of 51,138 completed search paths on a condensed version of Wikipedia that contains 4,604 articles and 119,882 links between them.

The setup for this problem is similar to that of section 3.1, so we will only go over the main differences. Again we will use a 90/10 training/testing split of the data with 10-fold cross validation.

For each training set D_{train}, we build the underlying hypergraph $H = (V, E)$. This time, V is the set of all articles, and E is a set of hyperedges e_s where p_s is the conditional probability of moving to article s_l given that the user had just visited s' in succession.

For each testing set D_{test}, we will use the last article in each completed path as the target, and the previous 3 articles as the given sequence. This means we will be able to use hyperedges of up to size 4. We employ the same probabilistic coverage function h and the same baseline comparisons as in Section 3.1. For this application, our neural network was most effective when we used a single layer of 32 LSTM nodes (with a dropout of 0.2). Our results are shown in Figure 3.

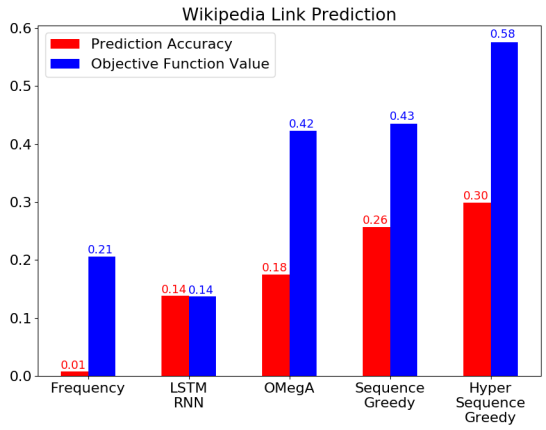


Figure 3: Given a sequence of articles a Wikipedia user has visited, we want to predict the next link they will click. This bar chart shows the prediction accuracy, as well as the objective function value, of various algorithms.

In this case, Hyper Sequence-Greedy exhibits the best performance. We see that the simple neural network implementation is outperformed by Hyper Sequence-Greedy as well as by some of the baselines. This is likely a result of the data in this experiment being more sparse. Although in this application we technically have more data than in the previous one, here we attempt to choose between 4,604 articles, rather than just 207 movies.

We also show the results that the various algorithms achieve when evaluated on our objective function $f(\sigma) = h\big(E(\sigma)\big)$. Asides from the LSTM-RNN, which doesn't consider the objective function at all, we see that the objective function values are relatively in line with the prediction accuracy. This demonstrates that the probabilistic coverage function was a good choice for the objective function.

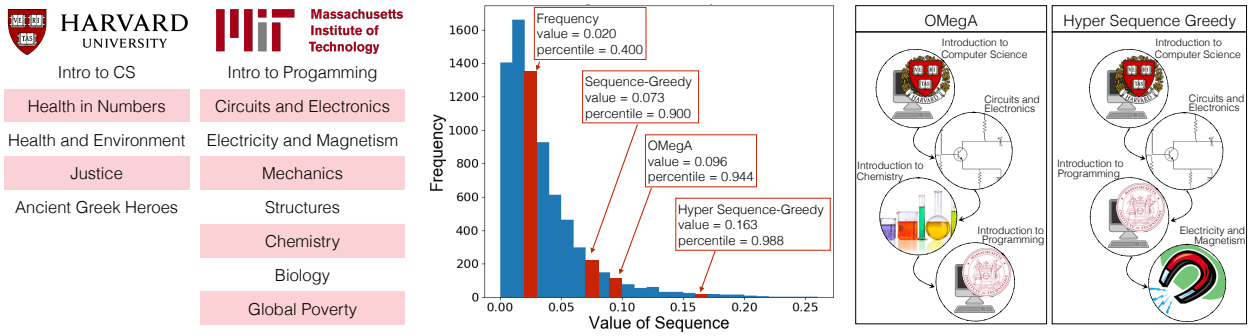
3.3 Course Sequence Design

In this final application we want to use historical enrollment data in Massive Open Online Courses (MOOCs) to generate a sequence of courses that we think would be of interest to users. We use a publicly available dataset (Ho et al., 2014) that covers the first year of open online courses offered by edX. The dataset consists of 641,139 registrations from 476,532 unique users across 13 different online courses offered by Harvard and MIT. Amongst a plethora of other statistics, the data contains information on when each user first and last accessed each course, how many course chapters they accessed, and the grade they achieved if they were ultimately certified (i.e., fully completed) in the course.

One natural way to think about the value of a sequence of courses is in terms of prerequisites. That is, in what order should we offer courses to students in order to help them learn as much as possible. This model comes with a natural measure of success as well, which is the grade each student gets in each course. Unfortunately, out of the 476,532 unique users in this dataset only 180 were certified (and thus, received grades) in 3 or more courses. Furthermore, this dataset only contains 13 different courses (shown in Figure 4a), none of which are logical prerequisites for each other.

Instead, we can think about a sequence of courses being valuable if they will all be interesting to a user who registers for them. Similarly to the prerequisites model where the order of courses affects the user's grade, the order in which a user registers for courses should also affect their interest. In this dataset, we can measure interest by the percentage of the course that the user accessed. In particular, we say that if a user was interested in a course i if she accessed at least one-third of all the chapters for course i.

As always, we need to build the underlying hypergraph $H = (V, E)$ for each training set. In this case, V is the set of all courses and E is a set of hyperedges of form e_s, where s is a sequence of at most 3 courses and p_s is the probability that a user will be interested in s_l given that she previously showed interest in s' in the proper order. Recall that s_l is the last course in s, and s' is the sequence obtained from s after deleting s_l. As in section 3.1, we also use a parameter d to avoid overfitting to rare sequences. In this case we use $d = 100$. However, unlike Section 3.1, we are not making recommendations based on a user's history. Instead each algorithm will use the underlying hypergraph to build a single sequence σ. Since we are not starting with any given sequence, we can



(a) (b) (c)

Figure 4: (a) shows the 13 different courses that were available to students in this dataset. (b) is a histogram of the value of every 4 course sequence that appears in the dataset. The values of the courses selected by the various algorithms are overlayed on top of the corresponding bar in the histogram. (c) shows representative course sequences selected by OMegA and Hyper Sequence-Greedy.

finally run Sequence-Greedy and Hyper Sequence-Greedy both forwards and backwards, and take the maximum of the two results.

Different users will naturally have different interests, so it is unreasonable to expect that any single sequence σ will work for all users. However, if σ is a "good" sequence, we could expect that users who start all the courses in σ in the correct order ultimately end up showing interest in those courses. Intuitively, the idea is that σ should capture a sequence of courses with some common theme and present them in the best possible order. Therefore, if a user begins all the courses in σ they likely have some interest in this common theme. Hence, if σ is a good sequence, it will present these courses in a good order and properly pique the interest of these users.

Mathematically, we define S_σ to be the set of users who started all the courses in σ in the proper order, and c_{ij} to be the percentage of course j that user i completed. Therefore, the value of σ for a given test set D_{test} is defined as:

$$D_{test}(\sigma) = \frac{\sum_{i \in S_\sigma} \sum_{j \in \sigma} c_{ij}}{\big(|S_\sigma| + d\big)|\sigma|}$$

Using a 75/25 training/testing split of the data and 4-fold cross validation, we compare the effectiveness of Hyper Sequence-Greedy, Sequence Greedy, OMegA, and Frequency for the task of selecting a sequence of 4 courses. Note that due to the inherent randomness in the training/testing split, there is some variance in the results. To be conservative, the results shown in Figure 4b are actually on the lower end of the performance we see from our algorithms. Figure 4c shows some representative sequences.

We see that Hyper Sequence-Greedy outperforms the other algorithms, as expected. From the histogram, we also see that Hyper Sequence-Greedy tends to select one of the best possible sequences, with Sequence-Greedy and OMegA both performing in the 90^{th} percentile. Somewhat surprisingly, OMegA (which has to use a random topological order in the absence of a directed acyclic graph) outperforms Sequence-Greedy. However, this may be explained by the fact that $k = 4$ is relatively small. Unfortunately, only 1,153 users even started more than 4 courses, meaning that we cannot effectively test sequences of larger length with this dataset.

4 Conclusion

Building on existing work, this paper extended results on submodular sequences from directed acyclic graphs to general graphs and hypergraphs. Our theoretical results showed that both our algorithms, Sequence-Greedy and Hyper Sequence-Greedy, approach a constant factor approximation to the optimal solution (for constant Δ). Furthermore, we demonstrated the utility of our algorithms, in particular the power of using hyperedges, on real world applications in movie recommendation, online link prediction, and the design of course sequences for MOOCs.

Acknowledgements We acknowledge support from DARPA YFA (D16AP00046), AFOSR YIP (FA9550-18-1-0160), and ISF grant 1357/16.