# Paper Recommendation using GraphX

Jeremy Chen
University of Waterloo
jeremy.chen@uwaterloo.ca

Junyi Zhang
University of Waterloo
j823zhan@uwaterloo.ca

Yuqing Xie
University of Waterloo
yuqing.xie@uwaterloo.ca

## ABSTRACT

Data in graph structure has become popular nowadays. Graph processing framework, therefore, becomes an essential tool to perform scalable computation. GraphX is one of the top choices because of its easiness, scalability, and performance. In this project, we explore GraphX in a deeper way and gain some insights. We achieve this by using GraphX to build an academic paper recommendation system. Without using the trivial way, we implement PageRank, keyword filtering, and pattern finding algorithms in GraphX. We apply them on a citation network to give recommendations of papers, with taking users' interest into account. We have two metrics to evaluate our recommendations: inversion count and ranking distance, and both have given us positive feedback. Through doing this project, we have not only studied GraphX in a practical way but performed data analytics on a citation network.

## 1 INTRODUCTION

With the rapid publication of scientific literature, conducting a comprehensive literature review has become more challenging. Hundreds of papers in computer science are published every day. Keeping up with current development of a certain area also requires huge effort. With limited time and energy, researchers would like to invest their time on the papers that can give them the most inspiration. However, without any assistance on filtering papers, finding representable papers in a particular field seems impossible.

More and more information retrieval method and graph processing method are developed during the past few years. What if we could combine the ability of distributed system with the advanced graph processing algorithms together with the text processing techniques? If there is a system that can automatically extract interesting or classic papers to researchers, we can save more time from only manually filtering the papers that we are interested in, based on the title and the abstracts. In this project, we proposed a pipeline that could take interested keywords or a list of past read papers as the input and then recommend related papers that the user will be interested in. We combined the crawled abstracts of papers with the citation network to help the paper recommendation. We learnt and explore the new framework **GraphX**, which makes graph computation easy and scalable. Since a citation network is in graph structure, we can leverage existing graph data processing framework to help us analyze the importance of each paper.

Our system is able to give interesting results based on both keywords mode and reading history mode. The recommended papers are most classic ones. This could help the researchers to trace back to the most important moment in the history, which is valuable both for new researchers and experienced researchers who need an exhausted literature survey.

Project source code can be downloaded here

The rest of the paper is structured as follows. Section 2 reviews related work on recommendation systems. We define the problem that we need to tackle in Section 3 and also give the formal notation that we need. Section 4 discusses the framework we explored (i.e. GraphX), and we describe the algorithms we use in GraphX in Section 5. In Section 6, we analyze the advantages and disadvantages of GraphX and discuss what we have learned while exploring the framework. We present our experimental results in Section 7 and further analyze them. Section 8 states some future work of this project and concludes the paper.

## 2 RELATED WORK

There are number of researches on recommendation system, and the methods generally could be divided into machine learning and graph. Our approach is based on graph, one relevant work would be a service that provides real-time recommendations to tens of millions of mobile users in Twitter by implementing real-time motif detection on large dynamic graphs [5]. In addition, similarity measures [3] have been extensively studied in the area of information retrieval and networks, i.e. user similarity computation and the effects of user similarity in social media. Similarities between two entities are used for community detection, similarity search and recommendations, and most of them proposed for networks are based on graph structure, i.e. SimRank [6], Penetrating Rank [11]. We proposed to use classic graph algorithms and network pattern mining methods for paper recommendation.

The papers mentioned above and our project are all based on large-scale graph processing. One suitable system for this kind of problem is Pregel [9], where programs are expressed as a sequence of iterations. In each iteration, a vertex could receive the message sent from the previous iteration, then it sends messages to other vertices, updates its own state and outgoing edges. Since large-scale graphs that with millions of vertices and billions of edges are difficult to analyze, researchers have usually turned to distributed solutions, i.e. MapReduce [8], or reduce the size of the graph by partitioning [7]. In this work, we tried to explore GraphX, a new framework for distributed graph computation.

Furthermore, there is a pretty related work to our project [2], which is about content-based citation recommendation. However, the method used in this paper is neural model instead of graph. They first embed all available documents into a vector space, and encoded text content of each document. Then, chose the nearest neighbours of the query document as candidates and reranked the candidates by another model trained, where this model takes a pair of documents as input and estimates the probability that $document_2$ should be cited in $document_1$.

The evaluation methods [1] for recommendation system include online evaluations and offline evaluations. Online evaluation means that recommendations are shown to users as they use the real-world system, while offline evaluation [10] includes precision, recall

and F-measure. A good recommender system contributes to three features, recommendation accuracy, user satisfaction, and provider satisfaction, and leads to the question how these three features are to be quantified and compared. However, in our dataset, we do not have any of the needed data for evaluation: we do not have a rating for the predicted recommendations, we do not have a history data of users and we don't know the most important papers within a certain field. So in this work, we also proposed a novel way to measure the effectiveness of our recommendation system.

## 3 PROBLEM DESCRIPTION

The goal of the project is to recommend papers to researchers according to their interest. There are two main features to represent users' interest. The first will be keywords. The second will be researcher's reading history.

We will have an offline dataset consisting of paper citation relation and papers' content.

Next, we give formal notations to describe our problem. We separate the problem into two part: given the interested keywords then the system output a list of recommending papers, or given a list of read papers in user's history then the system output a list of recommended papers.

### 3.1 First Part: Recommend According to Keywords

For the first part, the input will be a list of $n$ strings $K = [k_1, k_2, ..., k_n]$, where each string is an interested field, e.g. ["machine learning", "computer vision"]. The desired paper list will be $R = [r_1, r_2, ..., r_m]$, in the form of indexs of $m$ papers. Our system will recommend papers in an ordered way, which means $r_i$ is a better paper than $r_{i+1}, \forall i$.

### 3.2 Second Part: Recommend According to Reading History

For the second part, the input will be a list of papers $P = [p_1, p_2, ..., p_n]$, where each paper is represented as a unique id, e.g. ["journals/-cacm/Szalay08"]. This represents user's reading history ignoring the time sequence. The desired paper list will be $R = [r_1, r_2, ..., r_m]$, in the form of indexs of $m$ papers. Our system will also recommend papers in an ordered way, which means $r_i$ is a better paper than $r_{i+1}, \forall i$.

### 3.3 Graph representation

First of all, we represent the citation relations among papers to be a directed graph $G = (V, E)$. $V$ is the vertex set, each vertex represents a paper, we use the index to represent each paper. We also stored the detailed information of papers in a separated map from index to information. $E$ is the directed edge set. $\exists e = (v- > u) \in E$ means the paper $v$ cites the paper $u$.

*3.3.1 Subgraph.* We also need the concept of subgraph $G_s = (V_s, E_s)$. In the subgraph, we ensure that $V_s \subseteq V$, meaning we keep a subset of the original graph and get rid of the other vertexes. Then we compute the sub-edge set according to the sub-vertexes set: $E_s = \{e = (u- > v) | u \in V_s and v \in V_s \ and e \in E\}$. In this work, we do not need other subgraphs, which means a subgraph is determined by the original graph $G$ and the sub-vertexes set $V_s$.

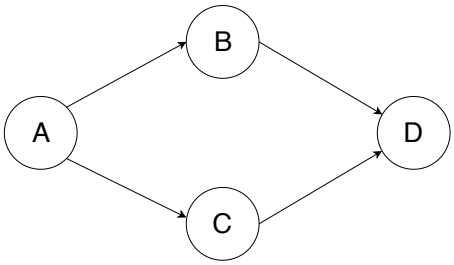


**Figure 1: Example of a diamond pattern.**

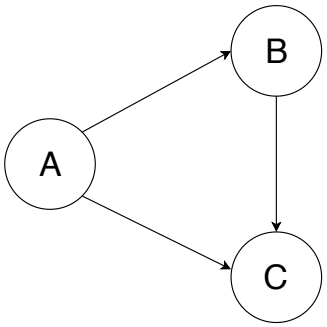


**Figure 2: Example of a triangle pattern.**

*3.3.2 Reverse Graph.* Since we are working on a directed graph, we can define the concept of a reversed graph. $G^* = (V^*, E^*)$ is a reversed graph of $G = (V, E) \iff$

$$V^* = V, \text{ and } E^* = \{(u- > v) | (v- > u) \in E\}. \quad (1)$$

*3.3.3 Patterns.* According to previous network studies [5], several common patterns are extremely useful for recommendation on networks. We address two main patterns: the diamond pattern and the triangle pattern.

*Diamond pattern.* As shown in Figure 1, a diamond pattern is that if a user has read paper $A$, and paper $A$ cites both paper $B$ and $C$, paper $B$ and $C$ both cite paper $D$. Then, we will recommend paper $D$ to the user if the paper has not been read. This is easy to understand because paper $D$ may be an important work in the reader's interested field.

*Triangle pattern.* Triangle is also easy to understand. As shown in Figure 2, a triangle pattern is that if a user has read paper $A$, and paper $A$ cites both paper $B$ and $C$, paper $B$ cites paper $C$. Then, we will recommend paper $C$ to the user if the paper has not been read.

## 4 GRAPHX

In this section, we discuss the big data processing framework for graphs, GraphX, that we use to tackle our problem. We will highlight some of the features in GraphX and go through some of the essential details of Pregel API.

## 4.1 Highlights

GraphX is a component built on top of Spark therefore has adopted many benefits and features from Spark. In Spark, data to be processed is represented as Resilient Distributed Dataset (RDD), which is immutable and distributed. In GraphX, graph data is abstracted as a directed graph allowing properties associated with vertices and edges. GraphX extends Spark RDD to represent a graph. In particular, a graph is represented as VertexRDD and EdgeRDD. Since VertexRDD and EdgeRDD are they are simply extensions of RDD, operations that can be applied on RDD can also be applied on them.

## 4.2 Pregel API

In terms of the richness of operators, GraphX provides a set of fundamental operators such as "subgraph," "reverse edges," etc. as well as a few common graph algorithms such as PageRank, ConnectedComponents, etc.. One feature that we use heavily in GraphX is the Pregel API. Pregel API uses the concept of "thinking like a vertex." In other words, the API provides a vertex-centric way of processing graphs. Users are required to specify the following functions for a Pregel API call.

- **Vertex Program** (denoted as *vProg*): This function runs on every vertex of the graph. In this function, each vertex $V$ has the access of the aggregated message (see below) delivered to $V$ as well as the attribute of $V$. The function returns the new state (i.e. attribute) of the vertex.
- **Send Message Program** (denoted as *sendMsg*): This function runs on the edges in the EdgeRDD. The input of this function is an edge triplet, which is a view logically joining the vertex and edge properties The function returns a message to be sent to the neighbors of each vertex.
- **Merge Message Program** (denoted as *mergeMsg*): This function combines two messages that are sent to the same vertex in an iteration into one. The function returns the combined message. Because of this aggregation, when a vertex executes the vertex program in the next iteration, only the aggregated message is passed in.

In addition, users need to specify the following parameters.

- **Initial Message** (denoted as *initialMsg*): This is the message passed to every vertex in the first iteration. The purpose of this message is to make sure each vertex executes the vertex program at least once.
- **Max Iteration** (denoted as *maxIterations*): This specifies the maximum number of iterations that can be run for a Pregel API call.
- **Edge Direction** (denoted as *activeDirection*): This specifies the direction of the edges on which the *sendMsg* function will run.

As a summary of how Pregel works, there are two major parts of the computation in each iteration.

(1) **Execution of Vertex Program**: On each vertex, the vertex program would be run sequentially. Note that only those vertices that have incoming messages from previous iteration will be run.
(2) **Message Preparation**: A message that will be sent would be constructed and delivered at the beginning of next iteration.

We express the execution of a Pregel API call at a high level in Algorithm 1.

**Input :** Initial Graph: $G = (V, E)$
**Output :** Updated Graph: $G = (V, E)$
**Function *Pregel***(*initialMsg, maxIterations, activeDirection, vProg, sendMsg, mergeMsg*)**:**
  numIterations := 0 ;
  initialMsg is sent to each vertex ;
  **while** *# vertices that will receive msg > 0 AND numIterations* $\leq$ *maxIterations* **do**
    **vProg**(Vertex, sumMsg) // run in parallel on each vertex ;
    **sendMsg**(Edge) // run in parallel on each edge satisfying activeDirection ;
    **mergeMsg**(aMsg, bMsg) // run in parallel on each set of messages sent to the same vertex ;
  **end**
**end**

**Algorithm 1:** Pregel API

# 5 ALGORITHMS

For the separated two parts of programs, we develop our algorithm differently.

## 5.1 Keywords recommendation algorithm

Before implement recommendation algorithm, we first implemented the TF/IDF algorithm to preprocessing papers' abstracts into their keywords. Then we implemented the classic PageRank algorithm using GraphX for recommendation score computation.

*5.1.1 Preprocessing.* For computation effectiveness, we only keep the abstract of each paper. We first tokenize the abstract, then implemented the TF/IDF algorithm. Now we acquired the TF/IDF score for each token in each paper. Then we filter $k$ tokens with the highest score in each paper to be the paper's keywords. Then we construct an inverse map from keywords to paper index, meaning the token is a keyword for those papers.

*5.1.2 PageRank for recommendation.* After we get the keywords to papers map, we implement the classic page rank algorithm to compute the score.

First, according to the keywords from the input, we filter out the subgraphs. Each paper in this subgraph have the target tokens as their keywords.

Next, we will first use GraphX to construct the citation network. After we constructed the graph, we will implement the classic PageRank algorithm to establish a paper ranking. You can refer to 3 to see detailed GraphX inplementation

Then we give recommendation paper list according to the ranking score, from the highest to the lowest.

*5.1.3 Pattern mining for recommendation.* To take advantage of the reading history for recommendation, we use pattern mining algorithms.

We will find common patterns such as diamond pattern and triangle pattern to do recommendation. To find the interested classic

**Input** : Keywords list: $K = [k_1, k_2, ..., k_n]$, $G = (V, E)$, Term2PapersIndex: $T$ -> {paper IDs with $T$ as a keyword}
**Output**: Filtered graph: $G = (V, E)$
**Function** *FilterGraph(Keywords)*:
    intersectedPaperIds := Intersection(
    Term2PapersIndex[$k_1$],
    Term2PapersIndex[$k_2$],
    ...,
    Term2PapersIndex[$k_n$]
    )
    return subgraph of G containing IDs in intersectedPaperIds and associated edges
**end**

**Algorithm 2:** Keyword Filtering Algorithm

**Input** : Keywords list: $K = [k_1, k_2, ..., k_n]$, $G = (V, E)$
**Output**: Recommended paper list: $R = [r_1, r_2, ..., r_m]$
**Function** *InitialGraph*:
    JoinVertices
    map(edge.Attr => $\frac{1}{edge.srcAttr}$)
    map(Vertice => 1.0)
**end**
Init.Message ← 0.0
G ← **InitialGraph**.Filter(K)
set resetProb
set numIteration
**Function** ***vProg****(Vertex, sumMsg)*:
    **return** *resetProb + (1.0 - resetProb) * msgSum*
**end**
**Function** ***sendMsg****(Edge)*:
    **return** *edge.srcAttr * edge.Attr*
**end**
**Function** ***mergeMsg****(aMsg, bMsg)*:
    **return** *aMsg + bMsg*
**end**
G ← G.Pregel(initialMsg, maxIteration=numIteration, activateDirection=out, vProg, sendMsg, mergeMsg)
R ← G.Vertex.SortedBy(Vertex.Attr)
**return** *R*

**Algorithm 3:** Paper Rank Algorithm

papers, we find patterns on the original graph. To find the interested recent paper, we find the patterns on the reversed graph.

To simplify the description, we first consider the original graph. The recent paper version will be the same after we turn the graph into a reversed version.

Because the restrictions from GraphX, we design the algorithm to be a two-iteration algorithm. And the operations of each iteration will be different. In the first iteration, we filter out the papers cited by the papers in the reading history. We call them referred papers. In the second iteration, we mine the patterns out. You can refer to the detailed algorithm in Algorithm 4

**Input** : Paper list: $P = [p_1, p_2, ..., p_n]$, $G = (V, E)$
**Output**: Recommended paper list: $R = [r_1, r_2, ..., r_m]$
**Function** ***InitialGraph****(P)*:
    **if** *Vertex.Id* $\in P$ **then**
        Vertex.Attr ← 1.0
    **else**
        Vertex.Attr ← 0.0
    **end**
**end**
Init.Message ← 0.0
G ← InitialGraph(P)
**Function** ***vProg1****(Vertex, sumMsg)*:
    **return** *Vertex.Attr + sumMsg*
**end**
**Function** ***sendMsg1****(Edge)*:
    **if** *src.Attr > 0.0* **then**
        **return** *2.0*
    **else**
        **return** *0.0*
    **end**
**end**
**Function** ***mergeMsg1****(aMsg, bMsg)*:
    **return** max*(aMsg, bMsg)*
**end**
G ← G.Pregel(initialMsg, maxIteration=1, activateDirection=out, vProg1, sendMsg1, mergeMsg1)
**Function** ***mapVertex****(Vertex)*:
    **if** *Vertex.Attr > 1.0* **then**
        Vertex.Attr←(1.0, 0.0)
    **else**
        Vertex.Attr←(0.0, 0.0)
    **end**
**end**
G ← G.mapVertex(mapVertex,Vertex)
**Function** ***vProg2****(Vertex, sumMsg)*:
    **return** *(Vertex.Attr._1, sumMsg)*
**end**
**Function** ***sendMsg2****(Edge)*:
    **if** *src.Attr._1 > 0.0* **then**
        **return** *1.0*
    **else**
        **return** *0.0*
    **end**
**end**
**Function** ***mergeMsg2****(aMsg, bMsg)*:
    **return** *aMsg + bMsg*
**end**
G ← G.Pregel(initialMsg, maxIteration=1, activateDirection=out, vProg2, sendMsg2, mergeMsg2)
R ← G.Vertiex.filter(Vertex.Attr._1 + Vertex.Attr._2 ≥ 2.0)
**return** *R*

**Algorithm 4:** Pattern Mining Algorithm

## 6 INSIGHTS

Since the main component of this project consists of learning the framework GraphX, we list some insights and what we have learned whiling exploring it.

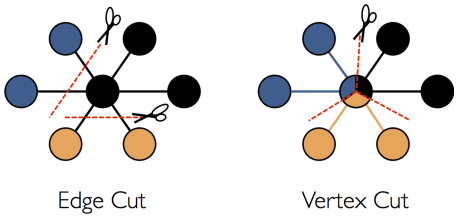


**Figure 3: Edge-based partitioning VS Vertex-based partitioning.**

## 6.1 Advantages of GraphX

Since GraphX is built on top of Spark, it adopts a lot of advantages from Spark. One of the biggest advantages of GraphX is the easiness of implementation. As the purpose of this project is to learn the framework, we choose not to use the PageRank function provided by GraphX. Instead, we implement the PageRank algorithm by ourselves using Pregel API in GraphX. However, the setup in order to initiate the execution of PageRank using Pregel only takes approximately 20 lines of code to complete. Then, GraphX handles everything else for us, including fault-tolerance, concurrency control, etc..

There is one significant difference between the implementation of Pregel API in GraphX and standard Pregel implementation. The Pregel API in GraphX only allows sending messages to the neighbors, whereas standard Pregel implementation can send messages to any vertex in the graph. Although this sounds like a disadvantage, this "limitation" helps GraphX improve the efficiency. Since in GraphX a graph is an extension of RDD, a graph dataset would be distributed across multiple machines. Intuitively, we could partition a graph based on the vertices(Figure 3(a))[4], for example, by hashing vertex IDs. However, this would cause large communication overhead as a lot of messages need to be sent to across machines, which becomes a bottleneck of the computation. Instead, GraphX handles graph partitioning in a edge-based manner (Figure 3(b)). Edges are partitioned onto different machines and vertices are allowed to span on multiple machines. This way, with the "limitation" that messages can only be sent to direct neighbors, we can partition the graph in a way that the communication overhead is very small.

## 6.2 Disadvantages of GraphX

While implementing algorithms in GraphX, we have also found some disadvantages of the framework. First, we are not allowed to have different initial messages of Pregel API in GraphX for different types of vertices. This has made the design of our pattern finding algorithm more challenging as we need to think of an initial message that would not mess up the attribute of any vertex. The flexbility of initial messages may ease the implementation in some cases. Secondly, since GraphX uses Pregel API, which is a synchronous model, the framework may not be the best fit for some algorithms, especially in machine learning and/or data mining. In a synchronous model like Pregel, every vertex executes the same iteration. That is, in the case that some of the vertices need more time to finish their vertex programs than the others, the entire computation is blocked. An asynchronous model can make the computation a lot fast as the most recent information is being used, as opposed to previous iteration.

# 7 EXPERIMENTAL EVALUATION

Although the focus of this project is to learn a new graph processing framework, we still describe a little bit how we are going to evaluate our proposed solution.

## 7.1 Dataset

We use the ACM Citation Network as our graph dataset, which can be downloaded here. The raw dataset is in text file, and it includes the information we need such as references of papers, abstracts, etc.. As the raw dataset uses strings as paper unique IDs, we first transform the dataset to have integer IDs starting from 1 to ease our processing. We then extract the graph topology (for PageRank and pattern finding algorithms) and the paper abstracts (for TF-IDF) to construct the datasets in our desired format.

## 7.2 Evaluation Methodology

The evaluation methodologies for both keyword-filtering and reading-hisotry recommendation features are similar. Since evaluating the effectiveness of the recommendations by automatic testing is challenging, one way to evaluate our results is manual evaluation. In this work, we face the classic cold start problem in recommendation system: there is no good data for interested papers according to keywords or reading history. We will use both manually evaluation and citation rank evaluation for results measuring. We are going to perform our proposed algorithms on the DBLP dataset described in Section 7.1. For keyword-filtering recommendation, a keyword from a list of randomly generated keywords is going to be an input. For reading-history recommendation, a reading list is going to be an input. Then, we manually evaluate how relevant the results are to the input keyword or reading history, respectively. The other way to evaluate our results is based on the number of citations of a paper. The recommendations are effective if they are among the most-cited papers. Given a recommendation ranking list $r_1, r_2, ..., r_m$, we crawl their citations $c_1, c_2, ..., c_m$. Then we re-rank the papers according to the citation, from the most to the least. The ranked papers will be $r_{i_1}, r_{i_2}, ..., r_{i_m}$. Then we count the number of inversion in the new index ranking. We also measure the rank distance: $D(i_1, ..., i_m) = \frac{1}{m} \sum_{j=1}^{m} |i_j - j|$.

## 7.3 Experimental Results

Because of the cold start problem, we display out results through examples. For the keywords recommendation, we select the top 8 papers to be our recommendation. For the reading history recommendation, we randomly select paper in related field and feed these index to be the input. We select the top 5 papers from pattern to be our recommendation.

*7.3.1 Keywords recommendation results.* Table 1, showed the five queries we sent to our system. We display the inverse counts and the rank scores. We also provide the first recommended paper title and it's citation number. The average inverse counts is 8 while the average rank distance is 1.68. The results are quite interesting.

| Keywords | Inverse count | Rank distance | Top one title | Top one citation |
|---|---|---|---|---|
| Machine Learning | 8 | 1.75 | Support-Vector Networks | 33820 |
| Cryptography Security | 5 | 1.125 | State of the Art in Ultra-Low Power Public Key Cryptography for Wireless Sensor Networks | 268 |
| Distributed System | 8 | 1.75 | Distributed Systems: Principles and Paradigms | 4038 |
| Deep Learning | 11 | 2.25 | Deep learning via semi-supervised embedding | 610 |
| Computer Vision | 8 | 1.5 | OpenVIDIA: parallel GPU computer vision | 281 |

**Table 1: Results for selected keywords recommendation.**

**Keywords** Machine Learning

| Rank | Paper Title | Citation | Publish year |
|---|---|---|---|
| 1 | Support-Vector Networks | 33820 | 1995 |
| 2 | Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond | 10802 | 2001 |
| 3 | Machine learning in automated text categorization | 9095 | 2002 |
| 4 | Application of argument based machine learning to law | 3 | 2005 |
| 5 | Data Mining: Practical Machine Learning Tools and Techniques | 35014 | 2004 |
| 6 | Very Simple Classification Rules Perform Well on Most Commonly Used Datasets | 2060 | 1993 |
| 7 | Investigating statistical machine learning as a tool for software development | 59 | 2008 |
| 8 | Machine Learning for User Modeling | 474 | 2001 |

**Table 2: An example for keywords recommendation.**

| Input paper | Recommend Paper Title | Recommend Paper Citation |
|---|---|---|
| State of the Art in Ultra-Low Power Public Key Cryptography for Wireless Sensor Networks | A method for obtaining digital signatures and public-key cryptosystems | 20024 |
| Cryptography and Network Security: Principles and Practice | Handbook of Applied Cryptography | 18008 |
| Computational soundness for standard assumptions of formal cryptography | Securing ad hoc networks | 3625 |
| Minimalist cryptography for low-cost RFID tags (extended abstract) | The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks | 1639 |
| Securing Mobile Ad Hoc Networks with Certificateless Public Keys | Mitigating routing misbehavior in mobile ad hoc networks | 4594 |
| BeeHiveGuard: a step towards secure nature inspired routing algorithms | | |
| A proposed curriculum of cryptography courses | | |
| Integration of Quantum Cryptography in 802.11 Networks | | |

**Table 3: An example for reading history recommendation.**

For keywords that have relatively long research history, the recommended paper are pretty good. The top one recommendation of machine learning is a work with up to 33,820 citations. As we all know, the support vector related methods was a heat back to the 90's.

Here we can take a closer look at the machine learning query's return in Table 2. The selected papers are almost all classic works with high citations. This is a natural result since we propagate the weights to the citation source. Classic papers will finally receive almost all scores from the later papers. We also examined the papers in the middle of the recommendation list. However, these papers still contain some relatively old papers. The main reason may due to the delay of the ACM dataset. We also manually checked several classic but recent papers from our knowledge, but many them either

| Exp ID | Inverse count | Rank distance | Top one citation |
|---|---|---|---|
| 1 | 2 | 0.8 | 75881 |
| 2 | 4 | 1.6 | 96 |
| 3 | 1 | 0.4 | 20024 |
| 4 | 3 | 1.6 | 2796 |
| 5 | 3 | 1.2 | 10141 |

**Table 4: Results for reading history recommendation.**

are not in the original dataset or was discarded because of missing of the abstract.

*7.3.2 Reading History recommendation results.* Table 3 shows an example of recommendation query according to reading history. In the first column are the papers in the reading history, while in the second column are the papers we recommended. The input papers are selected from the cryptography security. The average citations of the input papers is 183.9. We applied the original direction pattern mining. As you can see, the recommended papers are all closely related to cryptography and their citations are all above 1,500, which is much more higher than the input papers.

Table 4 shows statistics on 5 reading history queries. The inverse count and the rank distance are all low. This means the effectiveness of the pattern mining method.

## 7.4 Experimental Analysis

We can see from the previous results, our system works especially well on mining the classic papers. It can easily find the most popular and ancient papers. This is very important for researchers to find the source and follow the history within a field. Our system may not recommend recent papers. We next analyze the source of these disadvantages.

We first check the keywords selection part. After we extract the top 20 keywords that appears the most in the keywords list, we find an interesting phenomenon. Some common words may appear as keywords frequently, which is not what we expected. You can see from Table 5, the token "you" even appear as a keyword in up to 14,912 papers. We then go back to some of these papers. We found that some researchers prefer using "you" a lot while the other do not. This leads to the relatively small document frequency of the token while large term frequencies in some papers. This will cause our system not to identify the keywords precisely.

Since the most papers we recommended are all "ancient" papers. We dig deep into the dataset. Although the dataset was collect until 2016, the most papers are published before 2010. So the fashion topics such as computer vision and natural language processing will not receive a good performance, especially when people prefer more recent papers. Also, the recommended papers tend to be old due to the processing logic. We also applied the reverse directional reading history recommendation, but the test results are not desirable because of the cold start. That is to say, we do not have an actual reading history which can be applied within this dataset. So the pattern mining becomes difficult due to the poor input selection.

| Token | # Appearance as keywords | Token | # Appearance as keywords |
|---|---|---|---|
| web | 23293 | service | 17037 |
| image | 23082 | d | 15955 |
| network | 22428 | algorithm | 15906 |
| data | 22134 | video | 15294 |
| software | 21866 | services | 15019 |
| learning | 21299 | search | 14931 |
| mobile | 18176 | you | 14912 |
| fuzzy | 17559 | students | 14236 |
| control | 17389 | sensor | 13855 |
| security | 17060 | graph | 13686 |

**Table 5: Top 20 keywords appears in papers.**

# 8 CONCLUSION

In this project, we aim to learn and have some insights about GraphX, a graph processing framework built on top of Spark. We learn GraphX by leveraging it to easily accomplish paper recommendation at scale. In GraphX, we implement PageRank algorithm without using the provided PageRank API and apply it on a citation network. This would help us rank the importance of the papers. Besides direct execution of PageRank, we allow users to provide keywords to narrow the search space. Users can also choose to provide their reading history. Based on the history, we derive a list of papers that the users might be interested by pattern finding. Through the project, we have gained a number of insights and understood the advantages and disadvantages of GraphX.

Future work: According to the previous analysis, we could improve the system in several ways. First we can include external TF/IDF score or compute the score according to the whole paper to eliminate problem from the language unicity. In this way, we can acquire better keywords. Second, we can try other valid citation network dataset for more updated data. Also, we can explore other pattern mining method and try different recommendation algorithms. Another very important work should be done is that we have to solve the cold start problem and develop auto evaluation process for unlabeled recommendation problem.

# REFERENCES


[1] Joeran Beel, Stefan Langer, Marcel Genzmehr, Bela Gipp, Corinna Breitinger, and Andreas Nürnberger. 2013. Research Paper Recommender System Evaluation: A Quantitative Literature Survey. In *Proceedings of the International Workshop on Reproducibility and Replication in Recommender Systems Evaluation (RepSys '13)*. ACM, New York, NY, USA, 15–22. https://doi.org/10.1145/2532508.2532512

[2] Chandra Bhagavatula, Sergey Feldman, Russell Power, and Waleed Ammar. 2018. Content-Based Citation Recommendation. *CoRR* abs/1802.08301 (2018). arXiv:1802.08301 http://arxiv.org/abs/1802.08301

[3] Ashish Goel, Aneesh Sharma, Dong Wang, and Zhijun Yin. 2013. Discovering Similar Users on Twitter.

[4] Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. 2014. GraphX: Graph Processing in a Distributed Dataflow Framework. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI'14)*. USENIX Association, Berkeley, CA, USA, 599–613. http://dl.acm.org/citation.cfm?id=2685048.2685096

[5] Pankaj Gupta, Venu Satuluri, Ajeet Grewal, Siva Gurumurthy, Volodymyr Zhabiuk, Quannan Li, and Jimmy Lin. 2014. Real-time Twitter Recommendation: Online Motif Detection in Large Dynamic Graphs. *Proc. VLDB Endow.* 7, 13 (Aug. 2014), 1379–1380. https://doi.org/10.14778/2733004.2733010

[6] Glen Jeh and Jennifer Widom. 2002. SimRank: A Measure of Structural-context Similarity. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '02)*. ACM, New York, NY, USA, 538–543. https://doi.org/10.1145/775047.775126

[7] George Karypis and Vipin Kumar. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.* 20, 1 (Dec. 1998), 359–392. https://doi.org/10.1137/S1064827595287997

[8] Jimmy Lin and Michael Schatz. 2010. Design Patterns for Efficient Graph Algorithms in MapReduce. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs (MLG '10)*. ACM, New York, NY, USA, 78–85. https://doi.org/10.1145/1830252.1830263

[9] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: A System for Large-scale Graph Processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD '10)*. ACM, New York, NY, USA, 135–146. https://doi.org/10.1145/1807167.1807184

[10] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. 2010. *Recommender Systems Handbook* (1st ed.). Springer-Verlag, Berlin, Heidelberg.

[11] Peixiang Zhao, Jiawei Han, and Yizhou Sun. 2009. P-Rank: A Comprehensive Structural Similarity Measure over Information Networks. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM '09)*. ACM, New York, NY, USA, 553–562. https://doi.org/10.1145/1645953.1646025