Succinct representations in collaborative filtering: A case study using wavelet tree on 1,000 cores

Xiangjun Peng; Qingfeng Wang; Xu Sun; Chunye Gong; Yaohua Wang

University of Nottingham Ningbo China, 199 Taikang East Road, Ningbo, 315100, Zhejiang, China.

First published 2019

University of Nottingham
UK | CHINA | MALAYSIA

Succinct Representations in Collaborative Filtering: A Case Study using Wavelet Tree on 1,000 Cores

Xiangjun Peng*, Qingfeng Wang*, Xu Sun*, Chunye Gong‡, Yaohua Wang†
* *User-Centric Computing Group, University of Nottingham Ningbo China*
†*Department of Computer Science and Technology, National University of Defense Technology*
‡*Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology*
xu.sun@nottingham.edu.cn

***Abstract*—User-Item (U-I) matrix has been used as the dominant data infrastructure of Collaborative Filtering (CF). To reduce space consumption in runtime and storage, caused by data sparsity and growing need to accommodate side information in CF design, one needs to go beyond the *U-I Matrix*. In this paper, we took a case study of *Succinct Representations in Collaborative Filtering*, rather than using a *U-I Matrix*. Our key insight is to introduce Succinct Data Structures as a new infrastructure of CF. Towards this, we implemented a User-based K-Nearest-Neighbor CF prototype via *Wavelet Tree*, by first designing a *Accessible Compressed Documents* (*ACD*) to compress U-I data in *Wavelet Tree*, which is efficient in both storage and runtime. Then, we showed that *ACD* can be applied to develop an efficient intersection algorithm without decompression, by taking advantage of *ACD*'s characteristics. We evaluated our design on 1,000 cores of Tianhe-II supercomputer, with one of the largest public data set *ml-20m*. The results showed that our prototype could achieve 3.7 minutes on average to deliver the results.**

1. Introduction

We argue that Collaborative Filtering systems demand a more efficient infrastructure, since maintaining *U-I Matrix* is no longer as valuable as it used to be. For decades, *U-I Matrix* has been used as the dominant infrastructure in Collaborative Filtering (CF) [1], [2]. However, there are two major issues, which motivated us to rethink this very fundamental design.

First, due to the data sparsity problem, it is inevitable that too much costs were caused to maintain *U-I Matrix* in runtime, because users are very unlikely to rate all items in their collections. Data sparsity could lead more serious space efficiency problem, as the size of data grow larger. For instance, in one of the largest public data set from *Movielens* (i.e. *ml-20m*), *U-I Matrix* would be maintained with 138,489 rows (i.e. users) and 27,278 columns (i.e. items), which is quite significant. And it would become much more serious in commercialized services.

Second, collaborative filtering systems start to use more and more data, rather than just User-Item data in the matrix. For example, side information of Users and/or of Items provide promising information which goes beyond the *U-I Matrix* [3]. The ever-increasing demand for personalized recommendations and, to utilize the promising new information provided by side information, introduce more complexities in the implementation of CF. Furthermore, to satisfy performance needs in online services, storing and maintaining all those data could be too expensive.

Unlike *U-I Matrix* with its associated databases, data which are presented in document format is less constraint by its internal structure, and can contain far more rich information with less space consumption. More importantly, documents could be easily compressed and meanwhile support fast queries, like Succinct Data Structures. As one of them, *Wavelet Tree* stores strings in a highly compressed space and efficiently support fast-query [4], [5]. Moreover, such kinds of techniques have already been utilized to develop general-purpose data systems. For instance, a high-performance data store ***Succinct*** has been proposed to enable data queries on compressed data using *Skewed Wavelet Tree*.

Although previous work have demonstrated that building CF systems based on Succinct Data Structures is a promising direction to go beyond the *U-I Matrix*, there are no existing work to successfully apply it together with the document analyzing technique in CF, which could be summarized into two outstanding challenges. First, there is no existing method to organize collaborative filtering-related data into sequences to enable the use of runtime technique like *Wavelet Tree*. Second, beyond queries, there is a lack of CF-based strategy to support such applications without decompression, which required significant costs to do so.

To address these issues, we first designed a ***Accessible Compressed Documents*** (***ACD***), and then took advantage of ***ACD***'s characteristics to develop a User-based K-Nearest-Neighborhood (KNN) CF system. More specifically, this paper makes two contributions to the existing literature. First, the proposed ***ACD*** is proved able to substantially improve space efficiency for data storage and, while using *Wavelet Tree*, significant space savings in runtime, with support of fast information retrieval. Second, we showed that ***ACD*** can be used to implement a User-based KNN CF directly on compressed data, by utilizing inverted indexes and their ascending order. We evaluated our design on 1,000 cores of Tianhe-II supercomputer, with one of the largest public data

set *ml-20m*. The results showed that our prototype could achieve 3.7 minutes per user on average. It outperformed than User-based KNN in *Apache Mahout* (which required 5.1 minutes per user on average) with significant space savings.

2. Background

In this section, we provided several background information around Succinct Data Structures and Collaborative Filtering Systems. We first describe the growing trends of CF systems. Then we introduced *Wavelet Tree* in details. Finally, we provided some example data systems, which utilized Succinct Representations/Structures for significant performance benefits.

2.1. Collaborative Filtering Systems

Collaborative Filtering has utilized an elegant and proven infrastructure, *U-I Matrix* for long [1], [2]. However, since data sparsity problem is inevitable with the *U-I Matrix* (i.e. users are very unlikely to be able to rate all items in a collection), more space waste could be caused as the size of data grow larger, which further brought significant performance challenges. This is the major concern to motivate our design. Also side information becomes more and more important, which has motivated our design as well. Side information of Users and/or of Items (e.g. genres of Movies, ages of users and so on) provides promising information which goes beyond the *U-I Matrix* [3]. Many algorithms have been proposed to generate recommendations using side information [6], [7], [8]. The increasing importance of the side information in generating recommendations make such high cost to maintain *U-I Matrix* in runtime less valuable.

2.2. Succinct Data Structures

Succinct Data Structures represent a set of data structures, which uses an amount of space that is close to the information-theoretic lower bound (i.e. the maximum ratios of compression), meanwhile allows for fast-query operations. As one of Succinct Data Structures, *Wavelet Tree* is a highly compressed data structure with supports high-efficient queries without decompression [4], [5]. Also, Brisaboa et al. encodes text into *Wavelet Tree* called Reshaping to fasten queries [9]. Except these benefits, we chose *Wavelet Tree* as the core idea to develop our design because it now allowed parallel constructions and operations, which significantly enhanced its extendibility [10]. In addition, a variant of *Wavelet Tree* (i.e. *Wavelet Trie*) allows update operations without decompression, which showed its promising potentials in online services as well [5].

2.3. Wavelet Tree

Wavelet Tree is a kind of Succinct Data Structures to store strings in compressed space, which was introduced to represent compressed suffix arrays. The tree is defined by recursively partitioning the alphabet into pairs of subsets; the leaves correspond to individual symbols of the alphabet, and at each node a bitvector stores whether a symbol of the string belongs to one subset or the other. As shown in Figure 1, an example of *Wavelet Tree* has been given.

We theoretically examined the benefits of *Wavelet Tree*. Let Σ be a finite alphabet with $\sigma = [\Sigma]$. By using succinct dictionaries in the nodes, a string $s \in \Sigma^*$ can be stored in $nH_0(s) + O(|s| \log \sigma)$, where $H_0(s)$ is the order-0 empirical entropy of s. If the tree is balanced, the operations $access$, $rank_q$ and $select_q$ can be supported in $O(\log \sigma)$ time. $lookup_q$ can be combined with those operations, so that its time complexity is proportional to the height of the tree.

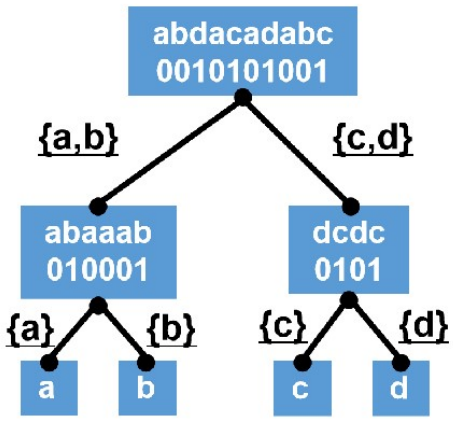

Figure 1. An example of Wavelet Tree.

2.4. Wavelet Tree Applications

The usages of *Wavelet Tree* could be found in many disciplines, because its related operations are only sensitive to how many types of characters in text (i.e the size of Succinct Dictionary), rather than full-text length. Grossi et al. are one of the first ones who apply *Wavelet Tree* technique to achieve high-performance full-text search [9], [11]. This is the most well-known usage of *Wavelet Tree*, which was covered by earlier survey from Navarro et al. as well [12]. And Claude et al. show that *Wavelet Tree*-based design for graph compressions is competitive with previous ones in web [13]. Also, Arroyuelo et al. has extended Inverted Positional Indexes functionality into Document Retrieval using *Wavelet Tree* [14].

2.5. Data Systems with Succinct Compression

Applying Succinct Compression is a growing new perspective to develop high-performance techniques for general purposes. Benefiting from Succinct Compression directly (i.e. fast queries and significant space savings meanwhile), data management systems are developed such as ***Succinct***

and ***Surf*** [15], [16]. Beyond queries, studies on how to analyze without decompression also drew attentions. *Text Analysis Directly on Compressed Data* has been proposed, along with ***Zwift*** programming framework [17], [18]. We believed collaborative filtering systems demand similar techniques, to utilize computing resources more efficiently, which further unleash the designs of CF from performance constraints.

3. Succinct Representations in CF

In this section, we introduce key design choices for *Succinct Representations in Collaborative Filtering*. We first introduce our ***Accessible Compressed Documents*** data model, which could store large volumes of data and meanwhile support queries while compressed in *Wavelet Tree*. And then, to effectively obtain the intersection between users (i.e. necessary resources for similarity computations, item selections and rating predictions), we propose an efficient algorithm design to obtain the intersection.

3.1. Accessible Compressed Documents

In this section, we introduce our data model, ***Accessible Compressed Documents*** (***ACD***), in details. In the data store ***Succinct***, to form *Skewed Wavelet Tree* in runtime, different symbols have been used as terminators to accurately locate required information, which has largely inspired our design of the ***ACD***. Different symbols are used to make distinctions between data, which ensures runtime queries could locate them correctly. For example, different data category is separated by the dots, and data records within the same data category are distinguished by symbols other than it.

In ***ACD***, data is firstly divided by categories, compression happened inside each category by simply forming a sequence. There are three major differences between *U-I Matrix* and ***ACD***. First, ***ACD*** could store greater variety of data than *U-I matrix* without increasing dimensionality of the data set. Second, unlike *U-I matrix*, ***ACD*** stores different data categories in a compressed format with different data categories being split by certain symbols. Third, data records from different categories can be retrieved by different symbols in different layers, which means that the matching between different data categories could be one-to-one, multiple-to-one, one-to-multiple or even multiple-to-multiple. Figure 2 demonstrates a ***ACD*** example, which consists of three categories (User, Item and Rating) and how they are matched.

Figure 3 demonstrates an actual ***ACD*** example, which are the format used in our implementations. However, in order to implement an efficient intersection algorithm between two item lists, we sorted our item lists increasingly based on item IDs, which would be illustrated in details next.

3.2. Efficient Intersection Computations

Maintaining the above document with *Wavelet Tree* ensured fast access towards related information, now the problem lies on how to obtain the intersection is what we needed. We utilized the order of *Inverted Index* in the item lists: we first initialize two pointers, in the corresponding item lists. Since all items in each item list are sorted in ascending order, the pointer, which points to the bigger item ID is set as the reference pointer. We fixed the reference pointer, which points to an index which is bigger than the number associated with the other pointer, the algorithm then move the other pointer forward until it reaches an index which is larger than the index associated with the previous reference pointer and update this pointer as the reference pointer. If the two pointers represent the same ID number, their corresponding ratings will be retrieved for computation and move any one of the two pointers forward, which ensures the algorithm can carry on until one of the two pointers reaches the end of its own list.

Users	Items	Ratings
1,	8|4|17|13|… ,	5.0|3.0|4.5|2.0|… ,
2,	2|13|1|… ,	3.5|4.0|1.5|… ,
3,	3|12|9|7|… ,	4.0|3.0|3.5|2.5|… ,

Figure 2. High-level Abstraction of ***Accessible Compressed Documents*** (User Item). In this figure, users items could be located according to the number of commas before it and retrieve corresponding users item list, so as ratings. For example, the "one-comma" right before the first number 3 in line two of the Items column separates the data of the user 2 from the user 3. Data from the side information for each user-item can be easily augmented to the sequence for each user-item accordingly in the Item Column of ***ACD***.

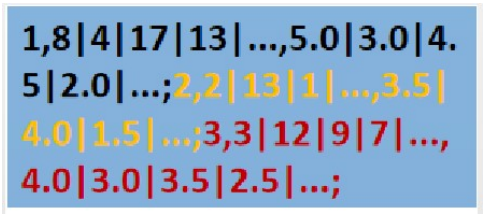

Figure 3. An Actual ***ACD*** Document Example based on the Figure 2.

Figure 4 presents a working example of our intersection algorithm implemented using two item lists. This algorithm improves the optimal time complexity from *O(nm)* (i.e. without ascending order) to *O(min(n,m))* theoretically, where *n* and *m* represent total numbers of items in different lists respectively.

4. Implementation Details

In this section, we illustrate implementation details while developing. We first describe our implementations to generate ***ACD*** fast. Then we explained how similarity values and rating prediction has been supported in our prototype.

4.1. ACD Generation

We generate our ***ACD*** demo from a item-ratings log file by applying a *sort-and-then-compress* method, which effec-

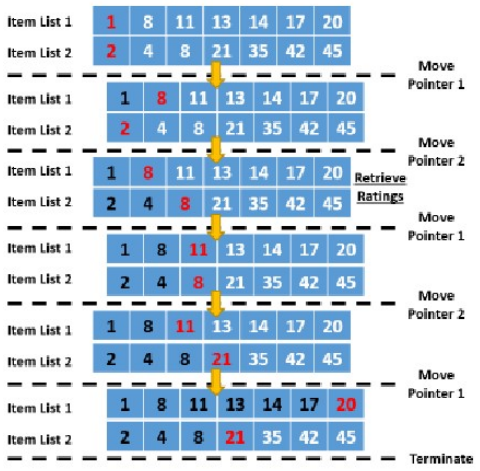

Figure 4. An example to illustrate how our intersection algorithm functions. The operations are directly operated on some parts of compressed document in *Wavelet Tree* (i.e. Item Lists).

tively speed up the data compression. Normally, gathering same-key logs and compressing via double-checking is quite straightforward but inefficient. Its time complexity is $O(n^2)$, where n represents the sum of the rating records.

To reduce the time consumption, we sort all ratings based on the identifiers (i.e. User ID, Item ID) in an ascending order by binary tree sorting method. The compression process starts right after the sorting is completed. Time complexity has been significantly reduced to $O(nlog_2n)$. This is a significant reduction in time complexity given that we are processing a large data set with millions of ratings. Our practice of the two methods shows that the former method would take about one hour to form the ***ACD***, but the latter one only take about one minute. Items in our ***ACD*** are sorted in an ascending order.

4.2. Similarity Metric

During our implementation, *Pearson Correlation Coefficient*, which is the most common similarity metric in *Recommender Systems*, has been selected as the metric to evaluate similarity values between users, as shown in Equation (1).

$$R(x,y) = \frac{N\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{N\sum x_i^2 - (\sum x_i)^2}\sqrt{N\sum y_i^2 - (\sum y_i)^2}} \quad (1)$$

Two thing needs to be highlighted: 1) in Collaborative Filtering systems, Pearson Correlation Coefficients are absolute to represent the similarity values. 2) In Equation (1), only those items, which are co-rated by two users, participated in the calculations.

4.3. Rating Prediction

To support rating prediction, a conventional method in User-based Collaborative Filtering has been selected, as shown in Equation 2.

$$P_{a,i} = \bar{r_a} + \frac{\sum_{u\in U}(r_{u,i} - \bar{r_u})}{\sum_{u\in U}|w_{a,u}|} \quad (2)$$

In Equation (2), *r* stands for the average rating from the corresponding user, and w stands for the Pearson Correlation Coefficient between corresponding users.

5. Experimental Study

In this section, we demonstrate details about our experimental study. We first introduce how we set up this experiment. Then we provide details around the experimental methodology. Next, we present the details during our experimental procedure. Finally, we show experimental results, which showed the benefits of our approach.

5.1. Experimental Setup

Data Set. We chose one of the largest open-source data set, *ml-20m* data set from Movielens, which contained 20 million ratings from 27,278 movies and 138489 users [19]. After data pre-processing, we observed that, there are only 138,483 who have co-rated items with others, which means they have the intersection with others to be retrieved. We ignored the rest of users since they are not capable to be provided recommendation results through this approach.

System Configuration. We utilized 100 computing nodes[1] of Tianhe-II Supercomputer to conduct our experiments [20]. The reason why We used a supercomputer is that, with lightweight tests, we found that it would be too long to finish all users in a reasonable time limit. For example, assuming 5 minutes for a user, there are 600,000 minutes (i.e. around 2 years) for *ml-20m* data set to finish all users' computations on a single core.

Benchmark System. We benchmarked our design with one of the state-of-the-art, open-source recommender framework *Apache Mahout* [21]. And the User-based KNN CF from Apache Mahout library was used to compare.

5.2. Methodology

Job[2] parallelism was used to speed up the overall progress of our experiment. First, all users were randomly allocated into one thousand groups and each group contains around 138 users. Then, each batch[3] contains ten groups of users which were randomly selected.

We submitted our batches of jobs to one hundred computing nodes. Batches are submitted to different cores and processed simultaneously. System Monitor records their start and end time for further analysis.

1. Each node consisted of 10 cores (i.e. Intel Xeon E5-2680V2 CPU), 64GB memory.

2. Each job represents one of our workloads, more specifically, a User-based KNN CF for one user.

3. Each batch consists of a number of jobs.

Since formatting U-I Data into documents and then compress them via *Wavelet Tree*, the space consumption has been reduced significantly.[4] During this experimental study, we focused on the time consumption of our design, since our major concern is whether such method can deliver results in a reasonable time.

5.3. Procedure

The procedure was divided into five periods, but this divisions are not done intentionally. We have further confirmed that such failures could be caused by load unbalance. Since operating system scheduler tends to keep workload balanced in different cores, distributing those jobs with different resource requirements lead to different levels of progress, and further switched the remaining ones among different cores [22]. We have eliminated time consumption of context switches in our final measurement.

We presented only the completion percentages of the first four periods here because the last one period only involves the computations of twelve users. As Figure 5 shown, the completion rate of the first two period are around 80%. The first two period combined complete the computation of users' recommendation between 97% of all users. The incomplete similarity computations in the third period, were largely due to the 'outlier users' namely those *"movie enthusiasms"*[5]. Low-performance in the third period is because of workload gap in different cores have grown significantly large, compared with previous two periods, which happens when multiple *"movie enthusiasms"* are stored in a node and distributed into different jobs. Jobs in the fourth and fifth period mainly suffers from the similarity computation of either *"super-enthusiasm"*[6] or contains multiple *"movie enthusiasms"*[5] which is usually around ten to twelve of such users.

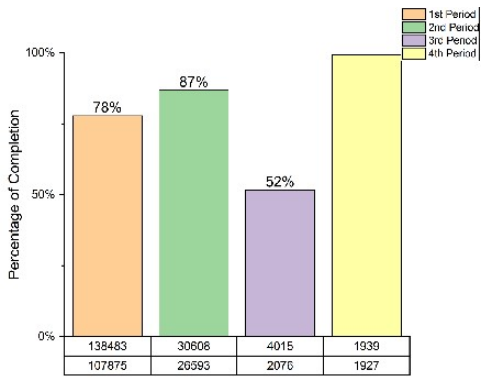

Figure 5. Details about our Large-scale Experiment Process

4. We have provided theoretical analysis in Section 2.3 and 4.1, with partial experimental results reported.

5. those users who rated around hundreds of movies

6. those users who rated almost a thousand movies

TABLE 1. DETAILS ABOUT TIME CONSUMPTION IN PROCEDURE

# of Finished Jobs	Minutes	Minutes per Users
107875	389132	3.60725
26593	98572	3.70669
2076	12217	5.88487
1927	17583	9.12455
12	134	11.16667

5.4. Results

Details of the time consumption and completed similarities computations between different numbers of users for different periods is presented in Table 1. As the results shown, our ***ACD***-based implementations is able to effectively dealing with 97% of all users below four minutes on average. However, starting from third period, average computation time per user increased to between six and eleven minutes as the multiple *"movie enthusiasms"* and the *"super-enthusiasm"*[6] enters in the computations. As the last three periods only account for less than 3% of total users, therefore, these computation time of our method is just a very small fraction of the total ones.

We also have repeated the same procedure with our benchmark system. The result showed that *Apache Mahout* has a better scalability in this case and provided results in 5.1 minutes per user on average. Our prototype has slightly outperformed than it, with significant space savings.

6. Discussions

In this section, we discussed some limitations and possible extensions of the presented promising methods. There are three outstanding issues that we believe it's worthy to explore further.

Extensions for other space-efficient designs. So far, we only explored the design of a simple intersection algorithm. However, there are many other existing applications of *Wavelet Tree* (e.g Graph) [23], which can be used similarly in Recommender System Pre-calculations.

***Wavelet Tree* construction and operations in parallel**. Existing work already demonstrated that the construction and operations of *Wavelet Tree* can be paralleled [10]. We believed it would bring more performance benefits to explore how to develop such effective methods in parallel.

Update-able Succinct Data Structure. The major bottleneck of embedding such technique into online services is that, it can't be updated once compressed. However, *Wavelet Trie*, a variant of *Wavelet Tree* with similar characteristics but capability to be updated, has been proposed [5]. This demonstrates its promising future for online services.

7. Conclusions

In this paper, we introduced our vision to introduce Succinct Data Structures into Collaborative Filtering Systems as a new perspective of infrastructure. By applying a customized data model and an efficient algorithm design,

we adapted *Wavelet Tree* into User-based KNN CF and implemented a prototype. Through 1,000 cores of Tianhe-II supercomputer, we have evaluated our design and the results demonstrated it's slightly faster than the state-of-the-art benchmark system, with significant space savings.

Acknowledgments

The authors would like to thank for anonymous reviewers for their valuable feedback. This research is generously supported by National Youth Science Program (71401085) and Summer Research Program in University of Nottingham Ningbo China. Chunye Gong is supported by National Numerical Windtunnel project (NNW2019ZT5-A10, NNW2019ZT6-B20, NNW2019ZT6-B21), Foundation of PDL (6142110180203) and National Science Foundation of China (61902411). Yaohua Wang is supported by The Science and Technology Planning Project of Hunan Province (2019RS2027).

References

[1] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, "An algorithmic framework for performing collaborative filtering," *SIGIR Forum*, vol. 51, no. 2, pp. 227–234, 2017. [Online]. Available: https://doi.org/10.1145/3130348.3130372

[2] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, 2001, pp. 285–295. [Online]. Available: https://doi.org/10.1145/371920.372071

[3] Y. Shi, M. Larson, and A. Hanjalic, "Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges," *ACM Comput. Surv.*, vol. 47, no. 1, pp. 3:1–3:45, 2014. [Online]. Available: https://doi.org/10.1145/2556270

[4] G. Navarro, "Wavelet trees for all," in *Combinatorial Pattern Matching - 23rd Annual Symposium, CPM 2012, Helsinki, Finland, July 3-5, 2012. Proceedings*, 2012, pp. 2–26. [Online]. Available: https://doi.org/10.1007/978-3-642-31265-6_2

[5] R. Grossi and G. Ottaviano, "The wavelet trie: maintaining an indexed sequence of strings in compressed space," in *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, 2012, pp. 203–214. [Online]. Available: https://doi.org/10.1145/2213556.2213586

[6] D. Agarwal and B. Chen, "Regression-based latent factor models," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, 2009, pp. 19–28. [Online]. Available: https://doi.org/10.1145/1557019.1557029

[7] N. Koenigstein, G. Dror, and Y. Koren, "Yahoo! music recommendations: modeling music ratings with temporal dynamics and item taxonomy," in *Proceedings of the 2011 ACM Conference on Recommender Systems, RecSys 2011, Chicago, IL, USA, October 23-27, 2011*, 2011, pp. 165–172. [Online]. Available: https://doi.org/10.1145/2043932.2043964

[8] Y. Moshfeghi, B. Piwowarski, and J. M. Jose, "Handling data sparsity in collaborative filtering using emotion and semantic based features," in *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2011, Beijing, China, July 25-29, 2011*, 2011, pp. 625–634. [Online]. Available: https://doi.org/10.1145/2009916.2010001

[9] N. R. Brisaboa, A. Fariña, S. Ladra, and G. Navarro, "Reorganizing compressed text," in *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2008, Singapore, July 20-24, 2008*, 2008, pp. 139–146. [Online]. Available: https://doi.org/10.1145/1390334.1390360

[10] J. Labeit, J. Shun, and G. E. Blelloch, "Parallel lightweight wavelet tree, suffix array and fm-index construction," *J. Discrete Algorithms*, vol. 43, pp. 2–17, 2017. [Online]. Available: https://doi.org/10.1016/j.jda.2017.04.001

[11] R. Grossi and J. S. Vitter, "Compressed suffix arrays and suffix trees with applications to text indexing and string matching," *SIAM J. Comput.*, vol. 35, no. 2, pp. 378–407, 2005. [Online]. Available: https://doi.org/10.1137/S0097539702402354

[12] G. Navarro and V. Mäkinen, "Compressed full-text indexes," *ACM Comput. Surv.*, vol. 39, no. 1, p. 2, 2007. [Online]. Available: https://doi.org/10.1145/1216370.1216372

[13] D. Arroyuelo, S. González, and M. Oyarzún, "Compressed self-indices supporting conjunctive queries on document collections," in *String Processing and Information Retrieval - 17th International Symposium, SPIRE 2010, Los Cabos, Mexico, October 11-13, 2010. Proceedings*, 2010, pp. 43–54. [Online]. Available: https://doi.org/10.1007/978-3-642-16321-0_5

[14] F. Claude and G. Navarro, "Extended compact web graph representations," in *Algorithms and Applications, Essays Dedicated to Esko Ukkonen on the Occasion of His 60th Birthday*, 2010, pp. 77–91. [Online]. Available: https://doi.org/10.1007/978-3-642-12476-1_5

[15] R. Agarwal, A. Khandelwal, and I. Stoica, "Succinct: Enabling queries on compressed data," in *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, May 4-6, 2015*, 2015, pp. 337–350. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/agarwal

[16] H. Zhang, H. Lim, V. Leis, D. G. Andersen, M. Kaminsky, K. Keeton, and A. Pavlo, "Surf: Practical range query filtering with fast succinct tries," in *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, 2018, pp. 323–336. [Online]. Available: https://doi.org/10.1145/3183713.3196931

[17] F. Zhang, J. Zhai, X. Shen, O. Mutlu, and W. Chen, "Efficient document analytics on compressed data: Method, challenges, algorithms, insights," *PVLDB*, vol. 11, no. 11, pp. 1522–1535, 2018. [Online]. Available: http://www.vldb.org/pvldb/vol11/p1522-zhang.pdf

[18] ——, "Zwift: A programming framework for high performance text analytics on compressed data," in *Proceedings of the 32nd International Conference on Supercomputing, ICS 2018, Beijing, China, June 12-15, 2018*, 2018, pp. 195–206. [Online]. Available: https://doi.org/10.1145/3205289.3205325

[19] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *TiiS*, vol. 5, no. 4, pp. 19:1–19:19, 2016. [Online]. Available: https://doi.org/10.1145/2827872

[20] Top-500-Supercomputer-List. (2019) Tianhe-2 (milkyway-2) supercomputer. [Online]. Available: https://www.top500.org/featured/systems/tianhe-2/

[21] A. Musselman, "Apache mahout," in *Encyclopedia of Big Data Technologies.*, 2019. [Online]. Available: https://doi.org/10.1007/978-3-319-63962-8_144-1

[22] J. Lozi, B. Lepers, J. R. Funston, F. Gaud, V. Quéma, and A. Fedorova, "The linux scheduler: a decade of wasted cores," in *Proceedings of the Eleventh European Conference on Computer Systems, EuroSys 2016, London, United Kingdom, April 18-21, 2016*, 2016, pp. 1:1–1:16. [Online]. Available: https://doi.org/10.1145/2901318.2901326

[23] G. Navarro, "Wavelet trees for all," *J. Discrete Algorithms*, vol. 25, pp. 2–20, 2014. [Online]. Available: https://doi.org/10.1016/j.jda.2013.07.004