# Studying the Effect of Data Structures on the Efficiency of Collaborative Filtering Systems

Pablo Sánchez, Alejandro Bellogín, Iván Cantador
Universidad Autónoma de Madrid, Spain
pablo.sanchezp@estudiante.uam.es,
{alejandro.bellogin, ivan.cantador}@uam.es

## ABSTRACT

Recommender systems is an active research area where the major focus has been on how to improve the quality of generated recommendations, but less attention has been paid on how to do it in an efficient way. This aspect is increasingly important because the information to be considered by recommender systems is growing exponentially. In this paper we study how different data structures affect the performance of these systems. Our results with two public datasets provide relevant insights regarding the optimal data structures in terms of memory and time usages. Specifically, we show that classical data structures like Binary Search Trees and Red-Black Trees can beat more complex and popular alternatives like Hash Tables.

## Keywords

Collaborative Filtering; Data Structures; Efficiency

## 1. INTRODUCTION

Quality and efficiency are the two main aspects to consider when deciding which algorithm is more suitable to a particular recommendation task. However, most of the papers in the area have focused on the first aspect, addressing the second one if it is required due to external constraints, e.g., quick response to the user, limited hardware, etc. Besides this, a recent trend in recommendation, and especially in Collaborative Filtering (CF), is to perform in-memory data processing [11], which makes efficiency even more important than before. This limits the application of works where CF data is indexed [6, 1].

Even when efficiency is considered as a system goal, there are not so many publications where more efficient data structures have been proposed [3]; this is because in general efficiency constraints are tackled by means of parallel processing or by designing recommendation algorithms that achieve high quality with less amount of data [8].

In this work, we study how different data structures actually affect the efficiency of recommendation algorithms, both in terms of memory usage and time. We consider classical data structures like Binary Search Trees and Hash Tables, and compare their performance on two public datasets when Collaborative Filtering algorithms are used.Hence, we empirically show that each recommendation stage – mainly, data loading and algorithm training – requires a different optimal data structure, where Binary Search Trees, Double Linked Lists, and Red-Black Trees turn out to show a good tradeoff between memory and time usage in two datasets with different characteristics.

## 2. BACKGROUND

### 2.1 Collaborative Filtering

The aim of Recommender Systems (RS) is to assist users in finding their way through huge databases and catalogs, by filtering and suggesting relevant items taking into account the users' preferences (i.e., tastes, interests, or priorities). Collaborative Filtering systems can be considered as the earliest and most widely deployed recommendation approach [8], suggesting interesting items to users based on the preferences from "similar" people [9]. These algorithms compute similarities between users or items and take those similarities into account when producing the recommendations. More specifically, this is the *standard* definition for a user-based nearest neighbour algorithm (UB):

$$\hat{r}_{ui} = \frac{\sum_{v \in N(u)} r_{vi} w_{uv}}{\sum_{v \in N(u)} |w_{uv}|}$$

where $N(u)$ is the neighbourhood of user $u$, representing the most similar users according to some similarity metric, $r_{vi}$ is the rating given by user $v$ to item $i$, $w_{uv}$ denotes the similarity between $u$ and $v$, and $\hat{r}_{ui}$ is the predicted rating according to this formulation. Alternatively, it is possible to predict ratings using an item-based nearest neighbour formulation (IB):

$$\hat{r}_{ui} = \frac{\sum_{j \in N(i)} r_{uj} w_{ij}}{\sum_{j \in N(i)} |w_{ij}|}$$

It is also possible to incorporate the user or item deviation in the formulation, this is called the *mean centering* formulation for UB:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in N(u)} (r_{vi} - \bar{r}_v) w_{uv}}{\sum_{v \in N(u)} |w_{uv}|}$$

where $\bar{r}_u$ is the rating average of $u$.

*CERI '16 Granada, Spain*
© 2016 ACM. ISBN 123-4567-24-567/08/06...$15.00
DOI: 10.475/123_4

This family of algorithms have two critical parameters: the size of the neighbourhood being considered and the similarity metric. Similarities are usually based on distance and correlation metrics [9]. We present here two of the most popular ones when rating data is available – Cosine and Pearson correlation coefficient:

$$\cos(u,v) = \frac{\sum_{i\in I_{uv}} r_{ui}r_{vi}}{\sqrt{\sum_{i\in I_u} r_{ui}^2 \sum_{i\in I_v} r_{vi}^2}}$$

$$\text{Pearson}(u,v) = \frac{\sum_{i\in I_{uv}} (r_{ui}-\bar{r}_u)(r_{vi}-\bar{r}_v)}{\sqrt{\sum_{i\in I_{uv}} (r_{ui}-\bar{r}_u)^2 \sum_{i\in I_{uv}} (r_{vi}-\bar{r}_v)^2}}$$

Note that in both definitions $I_{uv}$ is used, denoting the items rated by both users. Since this may be too strict in some scenarios, missing ratings can be filled with a default value by means of a process called *imputation* [8].

## 2.2 Data Structures

Data Structures allow for different ways to store data in memory. The purpose of the data structures varies in each application; actually, in [5], Falley shows a classification of data structures according to their purpose: storage structures (arrays, linked structures, hash tables), process-oriented structures (stacks, queues, priority queues, iterators), and descriptive structures (collections, sets, linear lists, binary trees). There are many other data structures that are less known or only used in very specific scenarios [4]: Fibonacci heaps, interval trees, disjoint set forests, among others.

In this section, we introduce the data structures that will be used later in the experiments. We selected these structures because they are well known and used throughout the computer science literature, but some of them have not been explicitly used in the field of Recommender Systems.

**Binary Search Tree (BST)** It is organised in a binary tree, where each node contains a key, satellite data, and pointers to its left and right child. The keys are always stored in such a way as the BST property holds: let $x$ be a node, if $y$ is a node in the left subtree of $x$ then $y.key \leq x.key$, if $y$ is a node in the right subtree of $x$ then $y.key \geq x.key$. Basic operations on a BST (search, insert, delete) take time proportional to the height of the tree, for a complete BST with $n$ nodes, such operations run in $\mathcal{O}(\lg n)$ average time and take $\mathcal{O}(n)$ space.

**Red-Black Tree (RBT)** It is a BST with one extra bit of storage per node: its color (red or black); by constraining the node colors on any simple path from root to a leaf, RBTs ensure that no such path is more than twice as long as any other, so that the tree is approximately balanced, guaranteeing that basic operations take $\mathcal{O}(\lg n)$ time in the worst case, instead of $\mathcal{O}(n)$.

**Order Statistic Tree (OST)** This data structure is built by augmenting an RBT so that finding the $i$th smallest number in a set or the rank of a given element in the total ordering of the set can be performed in $\mathcal{O}(\lg n)$.

**B-Tree (BT)** It is a balanced search tree designed to minimise disk I/O operations. They are based on the RBTs but each node may have many children (not necessary binary), having large branching factors. The time and space complexity of a BT is similar to RBT considering that the branching factor is taken as the base of the logarithm.

**Double Linked List (DLL)** It is a basic data structure where objects are arranged in linear order, the main difference in a DLL with respect to a Linked List is that it is possible to go forward and backwards in the list. Time complexity is constant for inserts and deletes unless we want to delete a particular element, where a search must be performed and, hence, it would run in $\mathcal{O}(n)$ time.

**Hash Table (HT)** It is an effective data structure for implementing dictionaries. Although searching for an element in a hash table can take as long as searching for an element in a linked list, in practice, under reasonable assumptions, the average time to search for an element is $\mathcal{O}(1)$ [4]. The space used by this structure depends on how each bucket (used to store one or multiple keys due to collisions) is implemented, but if space and time efficient structures are used, HTs would use $\mathcal{O}(n)$ space.

# 3. IMPACT OF DATA STRUCTURES IN COLLABORATIVE FILTERING

To measure the impact of different data structures in Collaborative Filtering algorithms, we have developed a framework[1] where rating prediction and item ranking tasks are available for a range of recommendation algorithms (mainly user- and item-based collaborative filtering) using different data structures transparently from inside of these algorithms. Once a data structure is selected, it is used to store the information associated to users, items, and user ratings to items, and is queried to generate recommendations. Furthermore, each data structure is initialised according to some parameters from the dataset – number of users and items – and the algorithm – user- or item-based.

The two main dimensions we are interested in analysing are memory usage and time efficiency. Effectiveness of the algorithms is not considered since it does not change when a specific data structure is selected. Moreover, the impact in testing time is also omitted because the experiments do not show significant differences between the structures being analysed, probably due to the datasets being too small. Hence, in Sections 3.2 and 3.3 we present the results related to the memory and time usages, considering the following recommendation stages: data loading and algorithm training. Before that, in Section 3.1 we provide details about the datasets used and the specific evaluation protocol followed.

## 3.1 Experimental Settings

We use two datasets, one from the movie domain (Movielens[2]) and another from the music domain (Lastfm[3]). The first one consists of $100,000$ ratings from $943$ users on $1,682$ movies. The second one contains $92,834$ user-artist relations from $1,892$ users on $17,632$ artists. Since this dataset does

[1] http://bitbucket.org/PabloSanchezP/dt4recsys

[2] Available at http://grouplens.org/datasets/movielens/

[3] Available at http://grouplens.org/datasets/hetrec-2011/

**Table 1: Aggregated memory usage (in MB) of the Movielens and Lastfm datasets. Lowest values per column in bold.**

| | Movielens | | Lastfm | |
|---|---|---|---|---|
| | Loading | Training | Loading | Training |
| BST | **61.56** | 196.84 | **51.30** | 885.79 |
| RBT | **61.56** | 296.51 | **51.30** | 1,861.37 |
| OST | **61.56** | 280.42 | **51.30** | 1,616.19 |
| BT | **61.56** | **148.29** | 61.56 | 1,406.57 |
| DLL | **61.56** | 299.78 | **51.30** | **725.98** |
| HT1 | 292.42 | 1,950.43 | 492.48 | 1,865.23 |
| HT2 | 202.47 | 2,202.69 | 317.61 | 1,011.68 |

**Table 2: Aggregated time usage (in seconds) of the Movielens and Lastfm datasets. Lowest values per column in bold.**

| | Movielens | | Lastfm | |
|---|---|---|---|---|
| | Loading | Training | Loading | Training |
| BST | 1.98 | 68.72 | 15.73 | 59.46 |
| RBT | **0.50** | **55.18** | 0.55 | **47.46** |
| OST | 0.53 | 56.97 | **0.50** | 125.55 |
| BT | 0.67 | 67.30 | 0.59 | 131.55 |
| DLL | 1.94 | 71.93 | 54.06 | 136.69 |
| HT1 | 0.80 | 349.97 | 1.94 | 1,130.53 |
| HT2 | 1.21 | 455.25 | 1.41 | 1,380.52 |

not provide explicit ratings, a normalisation is applied on a per user basis [2] to be able to apply standard collaborative filtering algorithms such as those presented in Section 2.1.

More specifically, the recommenders involved in these experiments are: baselines (user, item, and system bias [8], and combinations) and user- and item-based nearest neighbour recommenders [9] (where the standard (*Std*) and the mean-centering (*MC*) formulations were tested, using Cosine and Pearson coefficient as similarity metrics, with and without imputation, and a fixed neighbourhood size of 50).

Regarding the evaluation protocol followed, we rely on one method from the Java language to measure the time spent during loading and training (`System.currentTimeMillis()`) and on the following methods to account for the difference in the used memory before and after loading the data and training the algorithms: `Runtime.totalMemory()` and `Runtime.freeMemory()`. Note that the garbage collector may be called at any time, therefore these methods may not return the actual used memory at some point. In any case, we report results for each combination of recommendation algorithm and data structure using a 5-fold cross validation procedure.

### 3.2 Impact on Memory Usage

The results about memory usage of the experimental setup previously described are summarised in Table 1. Here, two implementations of Hash Tables are reported: HT1 uses DLL as the internal structure, and HT2 uses RBT. We observe that it is difficult to have a good performance in the two recommendation stages being analysed: loading and training; this is true in general except for BST and DLL, which perform well in both datasets, an interesting feature since each dataset has a different distribution of users and items.

To further understand in which situations each data structure may be more useful, we present in Table 3 the memory usage of the training stage in an algorithm basis (only for Movielens due to space constraints). Although these results are affected by how and when the garbage collector is called, we observe some trends that are worth analysing. Based on these results we first notice that, obviously, each recommendation algorithm requires different amount of memory to work, the baselines being the best for saving memory. Second, each algorithm and family of algorithms perform better with some data structures; in some situations the optimal structure may save up to a 97% of the memory used by another structure (compare BT with HT1 or HT2 for UB+Std+Cos). In general, it seems that when Pearson correlation is used, the algorithms impose less constraints in terms of memory. Hash Tables, on the other hand, perform worse than the other structures in most of the situations. This is because how they were implemented: they always take as much space as the number of users or items, even if some buckets are never used. We have performed additional tests with different parameters and found values where memory is optimised, but we leave the analysis of the optimal parameter (on a dataset basis) as future work.

### 3.3 Impact on Time Efficiency

Table 2 summarises the results about time efficiency for the two datasets presented before. We observe a tradeoff between time and memory usage: data structures optimised for memory need more time to load and train the algorithms. Although the differences are not large (especially in Movielens) we believe this might be an important issue if large datasets are used and want to be stored completely in memory.

As before, we present in Table 4 the time usage in an algorithm basis for Movielens. We observe that imputed versions of the algorithms require more time (reasonable because the similarities will use more ratings in their computations) and item-based recommenders take longer than user-based ones (probably because this dataset has more items than users). Here the differences between the data structures are not as evident as before – except for the Hash Tables – and the optimal data structure allows to save a minimal amount of time with respect to other data structures for the same type of algorithm (the largest improvement being a 39% for UB+MC+Pearson comparing DLL against RBT). Like in the previous scenario, Hash Tables obtain the worst performance in most cases, but this situation may change dramatically if the number of buckets are tuned to optimise this data structure.

## 4. CONCLUSIONS AND FUTURE WORK

In this paper we have analysed the performance of data structures when Collaborative Filtering algorithms keep their data in memory. We have shown that a balance should be met between memory and time usage, since efficient data structures in terms of the former perform worse for the latter. Our results evidence that B-Trees provide a good balance between memory and time usage in the two public datasets used in our experiments, a data structure not very common in public implementations of recommender systems.

In the future, we will explore compressing techniques before data are stored in the data structures to check if the results from the literature remain the same or change depending on the underlying data structure [11, 6]; we also aim to investigate the use of compact data structures [7],

**Table 3: Memory usage (in MBs) of different recommendation algorithms in the Movielens dataset for the training stage. In bold, the best result for each algorithm.**

| | | | BST | RBT | OST | BT | DLL | HT1 | HT2 |
|---|---|---|---|---|---|---|---|---|---|
| Baseline | | | 61.56 | 61.56 | 61.56 | 61.56 | 61.56 | 61.56 | 61.56 |
| IB | MC | Cos | 308.83 | 308.82 | 309.33 | **114.05** | 314.64 | 259.17 | 300.92 |
| | | Cos Imp | 308.89 | 308.95 | 309.55 | **114.08** | 314.56 | 274.74 | 279.83 |
| | | Pearson | **205.93** | 661.91 | 1,074.41 | 435.64 | 818.83 | 467.35 | 503.91 |
| | | Pearson Imp | 169.87 | 171.41 | 172.02 | 152.81 | **121.32** | 2,220.15 | 2,396.86 |
| | Std | Cos | 308.81 | 308.87 | 309.54 | **114.09** | 314.59 | 282.11 | 349.84 |
| | | Cos Imp | 308.86 | 309.03 | 309.50 | **114.10** | 314.61 | 198.40 | 343.91 |
| | | Pearson | **128.92** | 1,048.82 | 450.74 | 215.34 | 1,185.58 | 809.77 | 838.60 |
| | | Pearson Imp | 169.86 | 171.43 | 172.08 | **152.80** | 160.35 | 471.97 | 508.03 |
| UB | MC | Cos | 172.18 | 172.18 | 172.80 | **107.27** | 172.50 | 4,888.11 | 4,662.75 |
| | | Cos Imp | 172.13 | 172.19 | 172.75 | **107.29** | 172.49 | 3,959.31 | 3,545.73 |
| | | Pearson | **163.98** | 275.67 | 164.09 | 333.30 | 164.18 | 2,212.05 | 3,226.90 |
| | | Pearson Imp | 116.67 | 116.64 | 117.38 | **32.61** | 116.72 | 2,493.56 | 3,395.48 |
| | Std | Cos | 172.15 | 172.16 | 172.77 | **107.32** | 172.55 | 4,621.12 | 4,468.72 |
| | | Cos Imp | 172.14 | 172.15 | 172.73 | **107.26** | 172.55 | 4,499.40 | 3,351.26 |
| | | Pearson | **163.92** | 257.17 | 289.68 | 168.84 | 164.19 | 3,075.34 | 2,056.81 |
| | | Pearson Imp | 106.36 | 116.72 | 117.35 | **34.83** | 116.75 | 4,144.32 | 5,013.45 |

**Table 4: Time usage (in seconds) of different recommendation algorithms in the Movielens dataset for the training stage. In bold, the best result for each algorithm.**

| | | | BST | RBT | OST | BT | DLL | HT1 | HT2 |
|---|---|---|---|---|---|---|---|---|---|
| Baseline | | | 0.63 | 0.63 | 0.63 | 0.63 | 0.62 | 0.64 | 0.64 |
| IB | MC | Cos | 74.98 | **67.44** | 70.42 | 83.40 | 79.89 | 424.35 | 553.42 |
| | | Cos Imp | 104.50 | **92.51** | 94.73 | 105.70 | 107.77 | 448.43 | 570.05 |
| | | Pearson | 47.08 | **40.92** | 42.34 | 55.71 | 51.03 | 558.41 | 690.08 |
| | | Pearson Imp | 118.76 | 104.56 | **104.13** | 117.13 | 122.13 | 981.67 | 1,494.49 |
| | Std | Cos | 75.38 | **68.92** | 70.24 | 83.25 | 80.14 | 432.88 | 564.33 |
| | | Cos Imp | 101.70 | **86.40** | 93.87 | 102.90 | 106.99 | 440.29 | 573.59 |
| | | Pearson | 49.56 | **41.99** | 42.98 | 60.20 | 52.28 | 572.47 | 744.42 |
| | | Pearson Imp | 129.20 | **100.33** | 105.54 | 116.35 | 121.06 | 793.33 | 974.41 |
| UB | MC | Cos | 39.25 | **30.46** | 31.14 | 38.80 | 44.87 | 86.33 | 106.91 |
| | | Cos Imp | 63.62 | **39.39** | 43.21 | 50.13 | 66.62 | 96.70 | 116.43 |
| | | Pearson | 28.86 | **19.99** | 20.71 | 29.52 | 32.90 | 110.99 | 133.83 |
| | | Pearson Imp | 67.70 | **48.15** | 48.70 | 58.55 | 71.75 | 154.39 | 195.22 |
| | Std | Cos | 38.47 | **30.27** | 31.17 | 39.19 | 44.00 | 86.81 | 106.74 |
| | | Cos Imp | 61.78 | 43.41 | **42.90** | 50.35 | 66.29 | 99.23 | 120.44 |
| | | Pearson | 28.42 | **19.60** | 20.94 | 29.22 | 33.11 | 108.04 | 141.79 |
| | | Pearson Imp | 70.23 | **48.45** | 48.48 | 56.43 | 70.10 | 205.16 | 197.87 |

which will obtain a better memory usage but possibly also a worse time performance. We are also interested in extending these analyses to datasets with a different ratio of users/items – for instance, when there exist many more users in the system than items in the catalog.

## 5. REFERENCES

[1] A. Bellogín, J. Wang, and P. Castells. Bridging memory-based collaborative filtering and text retrieval. *Inf. Retr.*, 16(6):697–724, 2013.

[2] Ò. Celma. *Music Recommendation and Discovery - The Long Tail, Long Fail, and Long Play in the Digital Music Space*. Springer, 2010.

[3] S. H. S. Chee, J. Han, and K. Wang. Rectree: An efficient collaborative filtering method. In Y. Kambayashi, W. Winiwarter, and M. Arikawa, editors, *Data Warehousing and Knowledge Discovery, Third International Conference, DaWaK 2001, Munich, Germany, September 5-7, 2001, Proceedings*, volume 2114 of *Lecture Notes in Computer Science*, pages 141–151. Springer, 2001.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

[5] P. Falley. Categories of data structures. *J. Comput. Sci. Coll.*, 23(1):147–153, Oct. 2007.

[6] V. Formoso, D. Fernández, F. Cacheda, and V. Carneiro. Using rating matrix compression techniques to speed up collaborative recommendations. *Inf. Retr.*, 16(6):680–696, 2013.

[7] G. Jacobson. Space-efficient static trees and graphs. In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 549–554. IEEE Computer Society, 1989.

[8] Y. Koren and R. M. Bell. Advances in collaborative filtering. In Ricci et al. [10], pages 77–118.

[9] X. Ning, C. Desrosiers, and G. Karypis. A comprehensive survey of neighborhood-based recommendation methods. In Ricci et al. [10], pages 37–76.

[10] F. Ricci, L. Rokach, and B. Shapira, editors. *Recommender Systems Handbook*. Springer, 2015.

[11] S. Vargas, C. Macdonald, and I. Ounis. Analysing compression techniques for in-memory collaborative filtering. In P. Castells, editor, *Poster Proceedings of the 9th ACM Conference on Recommender Systems, RecSys 2015, Vienna, Austria, September 16, 2015.*, volume 1441 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.