On Parallelizing SGD for Pairwise Learning to Rank in Collaborative Filtering Recommender Systems

Murat Yagci
Bogazici University
murat.yagci@boun.edu.tr

Tevfik Aytekin
Bahcesehir University
tevfik.aytekin@bahcesehir.edu.tr

Fikret Gurgen
Bogazici University
gurgen@boun.edu.tr

ABSTRACT

Learning to rank with pairwise loss functions has been found useful in collaborative filtering recommender systems. At web scale, the optimization is often based on matrix factorization with stochastic gradient descent (SGD) which has a sequential nature. We investigate two different shared memory lock-free parallel SGD schemes based on block partitioning and no partitioning for use with pairwise loss functions. To speed up convergence to a solution, we extrapolate simple practical algorithms from their application to pointwise learning to rank. Experimental results show that the proposed algorithms are quite useful regarding their ranking ability and speedup patterns in comparison to their sequential counterpart.

KEYWORDS

Learning to rank; Pairwise loss; Parallel SGD; Personalization

1 INTRODUCTION

Recommendation problem can also be seen as a learning to rank (LtR) problem [6, 9]. Among pointwise [5, 7] and listwise [16] approaches, the pairwise [2, 15] LtR approach attempts to learn from preference information between a pair of items under a certain context [13]. This type of LtR can especially be a good fit for various implicit feedback scenarios since it does not need to ignore or mislabel unknown preferences. Moreover, because it works on item pairs instead of arbitrarily large lists, it can sometimes be more efficient than listwise LtR. Our motivation in this paper is to further improve its scalability to large feedback datasets and streams.

The idea of lock-free parallel SGD has been motivated for different machine learning methods, and applied to matrix completion from sparse feedback data. Two major camps are based on *block partitioning* [12], and *no partitioning* [11]. These approaches are especially interesting for multi-core CPU or GPU processing, and they can be preferable for intensive large-scale memory-based numerical computation.

In this paper, we investigate two shared memory lock-free parallel SGD schemes for personalized pairwise LtR to improve its scalability. We first adapt a block partitioning approach to this setting, and propose the PLtR-B algorithm. We then show that the no partitioning approach is applicable to pairwise LtR as in pointwise LtR [11], and present the PLtR-N algorithm.

We structure our paper as follows: In Section 2, we analyze parallel personalized pairwise LtR, and present the two proposed algorithms. In Section 3, comparative experimental results are provided for the ranking ability of algorithms and the speedup patterns. We conclude in Section 4.

RecSys '17, August 27–31, 2017, Como, Italy

DOI: http://dx.doi.org/10.1145/3109859.3109906

2 METHODS

2.1 Preliminaries on personalized pairwise LtR

Given a set of users, U, and a set of items, I, *personalized pairwise LtR* methods commonly learn from a dataset, $\mathcal{D}$, of triples (u, i, j), where $u \in U$ is the context, and $i, j \in I$. A triple implies a partial personalized ranking such that for a given user u, the ranking $r^i \prec_u r^j$ holds. In other words, user u prefers item i over item j. In this analysis, unless otherwise stated, we assume that $i \in I_u^+$, $j \in I \setminus I_u^+$, where I_u^+ denotes the set of items interacted by user u in a positive sense. The ultimate goal is to discover full or topmost personalized rankings from available partial rankings.

The typical pairwise error function is in the form :

$$E(\Theta \mid \mathcal{D}) = \sum_{(u,i,j)\in\mathcal{D}} \left\{ L\left(\theta_u, \theta_i, \theta_j\right) + \sum_{\theta_x \in \{\theta_u, \theta_i, \theta_j\}} \lambda_{\theta_x} ||\theta_x||_2^2 \right\}, \quad (1)$$

where Θ represents all model parameters, and λ_{θ_x} is a regularization coefficient. Different loss functions, L, are possible. In this paper, we stick to sigmoid-smoothed pairwise loss and negative log likelihood described in the seminal work, Bayesian personalized ranking (BPR) [15]. The same ideas apply to learning from other differentiable pairwise loss functions, for example, based on hinge loss [2].

In case of BPR with matrix factorization, $L = -\ln \sigma(y_{uij})$, and $y_{uij} = \mathbf{p}_u^\top(\mathbf{q}_i - \mathbf{q}_j)$, where $\mathbf{p}_u, \mathbf{q}_k \in \mathbb{R}^f$ are column vectors in the low-rank component matrices of the factorized user-item preference matrix, $\mathbf{R} \approx \mathbf{P}^\top\mathbf{Q}$. The resulting error function is differentiable with respect to model parameters, and it can be optimized with SGD using a suitable learning rate, γ. The typical procedure is summarized in Algorithm 1. Note that the algorithm often requires multiple passes over the dataset prior to convergence.

Algorithm 1 Personalized pairwise learning to rank

1: **repeat**
2: Sample a tuple (u, i) from positive interaction dataset, $\mathcal{D}^+$;
3: Sample j ;
4: **for all** $\theta_x \in \{\theta_u, \theta_i, \theta_j\}$ **do**
5: $\theta_x = \theta_x - \gamma \frac{\partial}{\partial \theta_x} \{ L\left(\theta_u, \theta_i, \theta_j\right) + \lambda_{\theta_x} ||\theta_x||_2^2 \}$;
6: **until** convergence

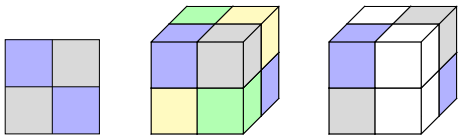


Figure 1: (Left) With 2 processing units, pointwise LtR can be parallelized with such a partitioning of R. Sets of chunks are shown with different colors. (Middle-Right) In pairwise LtR, the second item can be imagined on a third dimension. Two possible partitioning ideas are given.

2.2 Parallel pairwise LtR with block partitioning

For shared memory lock-free parallelization of matrix completion, one idea is block partitioning which partitions a user-item preference matrix, **R**, into multiple sets of ideally non-overlapping chunks [3, 12]. Then, each processing unit updates one of the chunks in every set which enables a form of parallel processing without using locks. This approach has been originally applied to personalized pointwise LtR scenarios, where updates are based on a single item interaction. In the case of pairwise LtR, a block partitioning solution can be formulated as in Figure 1 regarding the (u, i, j) updates. The block partitioning in the middle does not guarantee non-overlapping sets of chunks. Therefore, we concentrate on the rightmost partitioning approach which forces the (i, j) pairs to be always from the same chunk. This restriction guarantees mutually exclusive updates to both users and items, and it is computationally less expensive. We explain its applicability next.

By slightly modifying the pointwise scheme [12], the following is possible for pairwise LtR: Suppose we have ψ processing units. At each training epoch, we first generate a random permutation of users ($perm_U$) and items ($perm_I$) by shuffling their indexes in place. We then partition all (u, i, j) triples into ψ^2 chunks C_{abc}, where $a, b, c = 1, \ldots, \psi$ such that,

$$a = \left\lfloor \frac{\psi}{|U|} \left(perm_U(u) - 1\right) \right\rfloor + 1, b = \left\lfloor \frac{\psi}{|I|} \left(perm_I(i) - 1\right) \right\rfloor + 1,$$
$$c = \left\lfloor \frac{\psi}{|I|} \left(perm_I(j) - 1\right) \right\rfloor + 1, \quad (2)$$

and the sampled item j is forced to be in a chunk with $c = b$. Note that due to random permutations, users and items corresponding to a chunk are different at each training epoch, which makes this restrictive approach viable.

A set of chunks is updated in a single round and each update round can be decided simply by Algorithm 2. Putting it altogether, the personalized pairwise learning to rank algorithm can be parallelized as in Algorithm 3. We call this algorithm PLtR-B.

Algorithm 2 Deciding the set of chunks in a round

```
for z = 1 to ψ do
    for a = 1 to ψ do
        c = b = (a + z − 1) mod ψ ; {set b = ψ if b == 0} ;
        Add C_abc to Round[z] ;
```

Algorithm 3 Parallel personalized pairwise learning to rank with block partitioning (PLtR-B)

```
1:  for all training epochs do
2:      Generate perm_U and perm_I ;
3:      for all (u, i) in positive interaction dataset, D+ do
4:          Sample j s.t. c = b, and place (u, i, j) in corresponding C_abc ;
5:          {Since c = b, sampling of j can be postponed until line 10, if partitioning step is not performed in a dedicated thread }
6:      Assign C_abc to rounds w.r.t. Algorithm 2 ;
7:      for all Rounds do
8:          for all C_abc in current round in parallel do
9:              for all (u, i, j) ∈ C_abc do
10:                 for all θ_x ∈ {θ_u, θ_i, θ_j} do
11:                     θ_x = θ_x − γ ∂/∂θ_x {L(θ_u, θ_i, θ_j) + λ_θx ||θ_x||_2^2} ;
12:         Synchronize processing units ;
```

We note a few practical concerns regarding the block partitioning solution. First, the chunks in each round need to have a balanced number of training triples in order to fully benefit from parallelization. Random shuffling helps solve this problem, and if necessary, further amendments [18] can be useful. Second, the prior shuffling brings some extra time overhead which can optionally be overcome at the cost of dedicating separate threads [12]. Space overhead is also increased due to auxiliary structures. Third, and importantly, block partitioning imposes a predetermined sampling scheme for (u, i) and j. Therefore, we also investigate the no partitioning approach which can be more versatile for dynamic stream learning environments as well as for pairwise LtR tasks with custom biased sampling schemes [8, 10, 14, 17].

2.3 Parallel pairwise LtR with no partitioning

In this approach, parallel processes can access and write any portion of the shared memory at any time and in a lock-free fashion [11]. Nevertheless, this extreme approach still provides some theoretical guarantees for convergence when the optimization function can be defined as sparse summations, typical of SGD-based personalized LtR. In the case of pairwise LtR, the summation has the sparse form given in (1). This summation induces a hypergraph, $G = (V, E)$. However, unlike a bipartite graph for pointwise LtR, we now have a hypergraph which reflects personalized pairwise item relationships. We illustrate this new hypergraph in Figure 2.

Both $|V|$ and $|E|$ are typically very large with the latter being especially large in the case of pairwise LtR. However, as seen in (1), each summation acts on a single $e \in E$, that is, a very small subset of V, which suggests intuitively that parallel lock-free updates without any partitioning are viable. Following the theoretical analysis in [11], this notion can be formalized using the following statistics of a hypergraph, G:

$$\Omega = \max_{e \in E} |e|,$$
$$\Delta = \frac{\max_{1 \le v \le |V|} |\{e \in E : v \in e\}|}{|E|}, \quad (3)$$
$$\rho = \frac{\max_{e \in E} |\{e' \in E : e' \cap e \neq \emptyset\}|}{|E|},$$

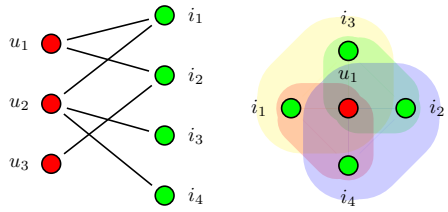


Figure 2: Example hypergraphs showing user-item relationships for pointwise and pairwise LtR respectively. For clarity, the latter is shown for a single user only, where 3-vertex hyperedges capture all possible pairwise item relationships for u_1. Here, $i_1, i_2 \in I^+_{u_1}$ and $i_3, i_4 \in I \setminus I^+_{u_1}$.

where Δ and ρ are measures of vertex regularity and hypergraph sparsity, respectively. When these values are relatively small, parallel updates can bring a highly effective speedup for convergence. In pairwise LtR, $\Omega = 3$, since each e is made up of an (u, i, j) triple. Assume every user makes m interactions, then $|E| = |U| \times m \times (|I| - m)$ triples. Since we have (u, i, j) triples, there can be user and item vertices in the hypergraph. Assume an item is interacted by at most n users. Then, it can be that $\Delta = (|I| - m) \times m/|E|$ regarding user vertices, or $\Delta = [(|I| - m) \times n + m \times (|U| - n)]/|E|$ regarding item vertices. As expected, we see that item vertices determine Δ in many real-life datasets. However, it is not straightforward to make general assumptions about m and n in the worst case. For example, in case $m \ll |I|$, $n \ll |U|$, and $m \approx n$, $\Delta \approx 1/|U| + 1/|I|$ which is close to that of pointwise LtR under similar assumptions. Since $\Omega = 3$, $\rho \leq 3\Delta$. In real-life datasets, the shared memory lock-free approach without any partitioning is effective even when the values of Δ and ρ are relatively high, as shown experimentally in [11] for different machine learning problems including matrix completion in a pointwise LtR setting. We observe in our experiments that in the personalized LtR problem setting, a possible reason behind this situation is that user and item vertex degrees often follow a power law distribution, that is, Δ (maximum normalized vertex degree) can be significantly higher than the values in the modal interval of the normalized vertex degree distribution. In Section 3, we compute these statistics for various datasets and show that the statistics of pairwise LtR are quite similar to those of pointwise LtR.

Algorithm 4 Parallel personalized pairwise LtR with no partitioning (PLtR-N)

1: {*Perform following loop in every parallel processing unit*}
2: **for all** training epochs **do**
3: **for all** iterations **do**
4: Sample a tuple (u, i) from positive interaction dataset, $\mathcal{D}^+$;
5: Sample j ;
6: **for all** $\theta_x \in \{\theta_u, \theta_i, \theta_j\}$ **do**
7: $\theta_x = \theta_x - \gamma \frac{\partial}{\partial \theta_x} \{L\left(\theta_u, \theta_i, \theta_j\right) + \lambda_{\theta_x} ||\theta_x||_2^2\}$;
8: Synchronize processing unit if learning rate, γ, changes ;

Therefore, we propose to parallelize pairwise LtR without partitioning as given in Algorithm 4. We call this algorithm PLtR-N. As an illustrative example, if we assume ψ processing units and 2ψ total epochs of the learning algorithm, then we see that every processing unit will run 2 epochs instead of a single processing unit running 2ψ epochs.

We note that PLtR-N can be conveniently extended to learn in an incremental fashion with a custom sampling scheme. This has potential benefits like taking time-dependency into account, less memory consumption, and faster convergence in real-life scenarios. For preliminary results, we refer the reader to [17].

Table 1: Basic properties of datasets

	$\|U\|$	$\|I\|$	# of Interactions
ML20M	138,493	27,278	20,000,263
LAST.FM	359,208	159,000	17,177,350

3 EXPERIMENTS

3.1 Datasets and evaluation

We use MovieLens 20M (ML20M) [4], and LAST.FM [1] datasets in our experiments. Basic properties of the final datasets are given in Table 1. Apart from user and item information, the raw interaction datasets contain confidence information in terms of ratings or interaction counts. We use this information to form a test set by leaving one random item out with maximum confidence for every user. We consider the rest as implicit feedback. We also make sure that every user interacts with at least 10 items, and for the LAST.FM dataset that a song is listened to by a user at least twice.

We measure ranking ability of the algorithms using MAP@500 [14], and area under curve (AUC) as described in [15]. Since the results show a similar pattern, we report the latter. Besides ranking evaluation, we illustrate speedup of the algorithms graphically using execution times.

We use a virtual cloud machine having a multi-core Intel Xeon CPU with 12 physical cores, and 16 GB of main memory. There are many ways to implement shared memory parallelism. In this paper, we rely on the Linux operating system and Java (1.7) threads to exploit the available cores. Although this leaves the management of cores to Java virtual machine, we monitor the effectiveness by diagnostic tools. We leave the atomicity of operations to what is available in the system. Guiding source code[1] is provided for reproducibility.

3.2 Statistics of dataset graphs

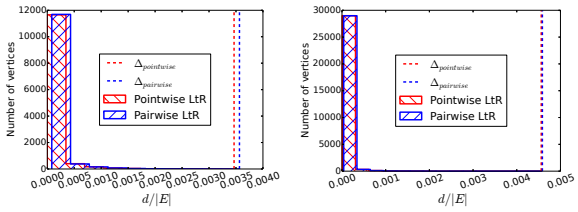
To complement analysis of PLtR-N in Section 2.3, we enumerate the (u, i, j) triples in the training sets and see that both Δ and ρ are on the order observed in real-life datasets for pointwise LtR. We provide comparative statistics in Table 2 which imply similar convergence properties for pointwise and pairwise LtR.

We also investigate vertex degree distributions of the dataset graphs and compare them with the Δ values in Figure 3. We observe that although the Δ values define an upper bound for normalized vertex degree distributions, the modal intervals of the distributions correspond to much lower values which further support parallel methods like PLtR-N.

[1] github.com/myaged

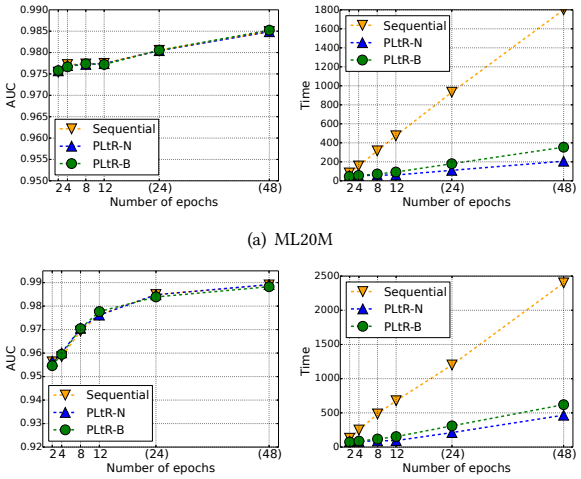
Table 2: Statistics of dataset graphs for LtR

	Δ	ρ
ML20M - Pointwise LtR	0.0035	$\leq 2\Delta$
ML20M - Pairwise LtR	0.0036	$\leq 3\Delta$
LAST.FM- Pointwise LtR	0.0045	$\leq 2\Delta$
LAST.FM - Pairwise LtR	0.0046	$\leq 3\Delta$



(a) ML20M (b) LAST.FM

Figure 3: Vertex degree (d) distributions in comparison to Δ values for the datasets



(a) ML20M

(b) LAST.FM

Figure 4: Comparing ranking ability and speedup of algorithms. For 24 and 48 total epochs, both PLtR variants utilize 12 processing units only.

3.3 Evaluation of ranking ability and speedup

Experimental results for PLtR-N up to 12 total epochs are obtained by performing a single epoch in each processing unit. Since we have 12 processing units, we then obtain results for 24, and 48 total epochs by running 2, and 4 epochs at each processing unit respectively. The results are illustrated in Figure 4. The important comparison here is to BPR which performs all epochs sequentially. For both BPR and PLtR-N, the following parameters are found by cross validation in a limited search space: For ML20M, $\gamma = 0.005$ and $f = 20$. For LAST.FM, $\gamma = 0.05$ and $f = 40$. Regularization parameters are found by cross-validation using a search space around $\lambda_{\theta_u} = \lambda_{\theta_i} = 0.0025$, and $\lambda_{\theta_j} = 0.00025$. Initial values in matrices $\mathbf{P}$ and $\mathbf{Q}$ are sampled from $\mathcal{N}(0, 0.01)$.

We observe in both datasets that the difference between ranking ability of BPR and PLtR-N is not statistically significant which shows the effectiveness of shared memory lock-free parallelization with no partitioning. On the other hand, the speedup patterns show that PLtR-N scales quite well whereas, as expected, sequential BPR has a linearly increasing execution time with the number of total epochs. Note that since we have 12 processing units in our experimental setup, the increase in PLtR-N execution time for 24 and 48 total epochs is due to the increase in number of epochs per processing unit. These results suggest that PLtR-N can converge much faster without loss of ranking ability, and by making better use of resources at hand.

Experimental results for PLtR-B up to 12 total epochs are obtained by performing ψ parallel updates in each round where ψ is equal to the number of total epochs. For 24 and 48 total epochs, $\psi = 12$. The partitioning is done in the master thread and the parallel processing in worker threads. The parameterization is the same as BPR and PLtR-N. The AUC pattern shows that its convergence is comparable to BPR and PLtR-N. On the other hand, while PLtR-B offers considerable speedup compared to sequential BPR, the speedup pattern is somewhat worse than PLtR-N. Possible reasons include imbalanced chunks in a round, and the partitioning step at master thread. We note that using separate dedicated worker threads for the partitioning step has a potential to improve this speedup.

4 CONCLUSIONS

Personalized pairwise LtR is a powerful approach for learning from relevance feedback datasets. We aim to improve its scalability using state-of-the-art shared memory lock-free parallel SGD schemes. In this direction, we propose two algorithms: PLtR-N and PLtR-B. We show that the usefulness of PLtR-N is due to graph theoretical similarities between pairwise and pointwise LtR for which the shared memory lock-free SGD scheme is known to have good convergence properties. Our experimental results support this analysis further with respect to ranking ability and speedup of the PLtR-N algorithm compared to its sequential counterpart. Block partitioning approach is inherently lock-free for pointwise LtR. We attempt to adapt it for pairwise LtR with the PLtR-B algorithm, and show comparative experimental results to BPR and PLtR-N algorithms. Both PLtR-N and PLtR-B exploit multi-core shared memory architectures, and they are easy to implement which can make them desirable in web-scale applications. They can also facilitate faster testing of pairwise LtR models with different parameterizations, which is a common bottleneck in evaluation design.

We note the following as future work: Although we provide some pointers, the mutual effect of biased sampling schemes and parallelization needs further investigation. Second, GPU-based parallel processing can be exploited for experimental limits of parallelization.

ACKNOWLEDGMENTS
We thank the anonymous reviewers for their comments. This work is supported in part by Bogazici University BAP project no. 13241.

REFERENCES

[1] Oscar Celma. 2010. *Music Recommendation and Discovery in the Long Tail.* Springer.

[2] Ernesto Diaz-Aviles, Lucas Drumond, Lars Schmidt-Thieme, and Wolfgang Nejdl. 2012. Real-time Top-n Recommendation in Social Streams. In *Proceedings of the 6th ACM Conference on Recommender Systems (RecSys '12)*. 59–66.

[3] Rainer Gemulla, Erik Nijkamp, Peter J. Haas, and Yannis Sismanis. 2011. Large-scale Matrix Factorization with Distributed Stochastic Gradient Descent. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '11)*. 69–77.

[4] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems* 5, 4, Article 19 (2015), 19 pages.

[5] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM '08)*. 263–272.

[6] Alexandros Karatzoglou, Linas Baltrunas, and Yue Shi. 2013. Learning to Rank for Recommender Systems. In *Proceedings of the 7th ACM Conference on Recommender Systems (RecSys '13)*. 493–494.

[7] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (2009), 30–37.

[8] Lukas Lerche and Dietmar Jannach. 2014. Using Graded Implicit Feedback for Bayesian Personalized Ranking. In *Proceedings of the 8th ACM Conference on Recommender Systems (RecSys '14)*. 353–356.

[9] Tie-Yan Liu. 2011. *Learning to Rank for Information Retrieval.* Springer.

[10] Babak Loni, Roberto Pagano, Martha Larson, and Alan Hanjalic. 2016. Bayesian Personalized Ranking with Multi-Channel User Feedback. In *Proceedings of the 10th ACM Conference on Recommender Systems (RecSys '16)*. 361–364.

[11] Feng Niu, Benjamin Recht, Christopher Ré, and Stephen J. Wright. 2011. Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. In *Advances in Neural Information Processing Systems*. 693–701.

[12] Benjamin Recht and Christopher Ré. 2013. Parallel stochastic gradient algorithms for large-scale matrix completion. *Mathematical Programming Computation* 5, 2 (2013), 201–226.

[13] Steffen Rendle. 2011. *Context-Aware Ranking with Factorization Models*. Studies in Computational Intelligence, Vol. 330. Springer.

[14] Steffen Rendle and Christoph Freudenthaler. 2014. Improving Pairwise Learning for Item Recommendation from Implicit Feedback. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining (WSDM '14)*. 273–282.

[15] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI '09)*. 452–461.

[16] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Nuria Oliver, and Alan Hanjalic. 2012. CLiMF: Learning to Maximize Reciprocal Rank with Collaborative Less-is-more Filtering. In *Proceedings of the 6th ACM Conference on Recommender Systems (RecSys '12)*. 139–146.

[17] Murat Yagci, Tevfik Aytekin, Hurol Turen, and Fikret Gurgen. 2016. Parallel personalized pairwise learning to rank. In *Proceedings of the 3rd EURO mini conference on From Multiple Criteria Decision Aid to Preference Learning (DA2PL '16)*.

[18] Yong Zhuang, Wei-Sheng Chin, Yu-Chin Juan, and Chih-Jen Lin. 2013. A Fast Parallel SGD for Matrix Factorization in Shared Memory Systems. In *Proceedings of 7th ACM Conference on Recommender Systems (RecSys '13)*. 249–256.