# Matrix Completion Under Interval Uncertainty: Highlights

Jakub Marecek[1(✉)], Peter Richtarik[2,3], and Martin Takac[4]

[1] IBM Research – Ireland, Damastown, Dublin 15, Ireland
jakub.marecek@ie.ibm.com
[2] School of Mathematics, University of Edinburgh, Edinburgh EH9 3FD, UK
[3] KAUST, 2221 Al-Khwarizmi Building, Thuwal 23955-6900, Kingdom of Saudi Arabia
[4] Department of Industrial and Systems Engineering, Lehigh University, Bethlehem 18015, USA

**Abstract.** We present an overview of inequality-constrained matrix completion, with a particular focus on alternating least-squares (ALS) methods. The simple and seemingly obvious addition of inequality constraints to matrix completion seems to improve the statistical performance of matrix completion in a number of applications, such as collaborative filtering under interval uncertainty, robust statistics, event detection, and background modelling in computer vision. An ALS algorithm MACO by Marecek et al. outperforms others, including Sparkler, the implementation of Li et al. Code related to this paper is available at: http://optml.github.io/ac-dc/.

## 1 Introduction

Matrix completion is a well-known problem: Given dimensions of a matrix $X$ and some of its elements $X_{i,j}, (i,j) \in \mathcal{E}$, the goal is to find the remaining elements. Without imposing any further requirements on $X$, there are infinitely many solutions. In many applications, however, the matrix completion that minimizes the rank:

$$\min_Y \operatorname{rank}(Y), \text{ subject to } Y_{i,j} = X_{i,j}, (i,j) \in \mathcal{E}, \tag{1}$$

often works as well as the best known solvers for problems in the particular domain. There are literally hundreds of applications of matrix completion, especially in recommender systems [3], where the matrix is composed of ratings, with a row per user and column per product.

Two major challenges remain. The first challenge is related to data quality: when a large proportion of data is missing and one uses matrix completion for data imputation, it may be worth asking whether the remainder data is truly known exactly. The second challenge is related to the rate of convergence and run-time to a fixed precision: many solvers still require hundreds or thousands of CPU-hours to complete a $480189 \times 17770$ matrix reasonably well.


U. Brefeld et al. (Eds.): ECML PKDD 2018, LNAI 11053, pp. 621–625, 2019.
https://doi.org/10.1007/978-3-030-10997-4_38

The first challenge has been recently addressed [8] by considering a variant of the problem with explicit uncertainty set around each "supposedly known" value. Formally, let $X$ be an $m \times n$ matrix to be reconstructed. Assume that elements $(i,j) \in \mathcal{E}$ of $X$ we wish to fix, for elements $(i,j) \in \mathcal{L}$ we have lower bounds and for elements $(i,j) \in \mathcal{U}$ we have upper bounds. The variant [8] is:

$$\begin{aligned} \min_{X \in \mathbb{R}^{m \times n}} \quad & \text{rank}(X) \\ \text{subject to} \quad & X_{ij} = X^{\mathcal{E}}_{ij}, (i,j) \in \mathcal{E} \\ & X_{ij} \geq X^{\mathcal{L}}_{ij}, (i,j) \in \mathcal{L} \\ & X_{ij} \leq X^{\mathcal{U}}_{ij}, (i,j) \in \mathcal{U}. \end{aligned} \tag{2}$$

We refer to [8] for the discussion of the superior statistical performance.

## 2 An Algorithm

The second challenge can be addressed using the observation that a rank-$r$ $X$ is a product of two matrices, $X = LR$, where $L \in \mathbb{R}^{m \times r}$ and $R \in \mathbb{R}^{r \times n}$. Let $L_{i:}$ and $R_{:j}$ be the $i$-th row and $j$-h column of $L$ and $R$, respectively. Instead of (2), we shall consider the *smooth, non-convex* problem

$$\min\{f(L,R) \ : \ L \in \mathbb{R}^{m \times r}, \ R \in \mathbb{R}^{r \times n}\}, \tag{3}$$

where

$$\begin{aligned} f(L,R) := \tfrac{\mu}{2}\|L\|_F^2 + \tfrac{\mu}{2}\|R\|_F^2 \\ + f_{\mathcal{E}}(L,R) + f_{\mathcal{L}}(L,R) + f_{\mathcal{U}}(L,R), \end{aligned}$$

$$\begin{aligned} f_{\mathcal{E}}(L,R) &:= \tfrac{1}{2}\sum\nolimits_{(ij) \in \mathcal{E}} (L_{i:}R_{:j} - X^{\mathcal{E}}_{ij})^2 \\ f_{\mathcal{L}}(L,R) &:= \tfrac{1}{2}\sum\nolimits_{(ij) \in \mathcal{L}} (X^{\mathcal{L}}_{ij} - L_{i:}R_{:j})^2_+ \\ f_{\mathcal{U}}(L,R) &:= \tfrac{1}{2}\sum\nolimits_{(ij) \in \mathcal{U}} (L_{i:}R_{:j} - X^{\mathcal{U}}_{ij})^2_+ \end{aligned}$$

and $\xi_+ = \max\{0, \xi\}$. The parameter $\mu$ helps to prevent scaling issues[1]. We could optionally set $\mu$ to zero and then from time to time rescale matrices $L$ and $R$, so that their product is not changed. The term $f_{\mathcal{E}}$ (resp. $f_{\mathcal{U}}$, $f_{\mathcal{L}}$) encourages the equality (resp. inequality) constraints to hold.

Subsequently, we can apply an alternating parallel coordinate descent method called MACO in [8]. This is based on the observation that although $f$ is not convex jointly in $(L,R)$, it is convex in $L$ for fixed $R$ and in $L$ for fixed $R$. We

[1] Let $X = LR$, then also $X = (cL)(\frac{1}{c}R)$ as well, but we see that for $c \to 0$ or $c \to \infty$ we have $\|L\|_F^2 + \|R\|_F^2 \ll \|cL\|_F^2 + \|\frac{1}{c}R\|_F^2$.

can hence alternate between fixing $R$, choosing $\hat{r}$ and $\hat{S}$ of rows of $L$ uniformly at random, updating $L_{i\hat{r}} \leftarrow L_{i\hat{r}} + \delta_{i\hat{r}}$ in parallel for $i \in \hat{S}$, and the respective steps for $L$. Further, notice that if we fix $i \in \{1, 2, \ldots, m\}$ and $\hat{r} \in \{1, 2, \ldots, r\}$, and view $f$ as a function of $L_{i\hat{r}}$ only, it has a Lipschitz continuous gradient with constant $W^{\mathcal{L}}_{i\hat{r}} = \mu + \sum_{v\,:\,(iv)\in\mathcal{E}} R^2_{\hat{r}v} + \sum_{v\,:\,(iv)\in\mathcal{L}\cup\mathcal{U}} R^2_{\hat{r}v}$. That is, for all $L$, $R$ and $\delta \in \mathbb{R}$, we have $f(L + \delta E_{i\hat{r}}, R) \leq f(L, R) + \langle \nabla_L f(L, R), E_{i\hat{r}} \rangle \delta + \frac{W^{\mathcal{L}}_{i\hat{r}}}{2} \delta^2$, where $E$ is the $n \times r$ matrix with 1 in the $(i\hat{r})$ entry and zeros elsewhere. Likewise, one can define $V$ for $R_{\hat{r}j}$. The minimizer of the right hand side of the bound on $f(L + \delta E_{i\hat{r}}, R)$ is hence

$$\delta_{i\hat{r}} := -\tfrac{1}{W^{\mathcal{L}}_{i\hat{r}}} \langle \nabla_L f(L, R), E_{i\hat{r}} \rangle, \tag{4}$$

where $\langle \nabla_L f(L, R), E_{i\hat{r}} \rangle$ equals

$$\begin{aligned} &\mu L_{i\hat{r}} + \textstyle\sum_{v\,:\,(iv)\in\mathcal{E}} (L_{i:}R_{:v} - X^{\mathcal{E}}_{iv}) R_{\hat{r}v} \\ &+ \textstyle\sum_{v\,:\,(iv)\in\mathcal{U}\ \&\ L_{i:}R_{:v} > X^{\mathcal{U}}_{iv}} (L_{i:}R_{:v} - X^{\mathcal{U}}_{iv}) R_{\hat{r}v} \\ &+ \textstyle\sum_{v\,:\,(iv)\in\mathcal{L}\ \&\ L_{i:}R_{:v} < X^{\mathcal{L}}_{iv}} (X^{\mathcal{L}}_{iv} - L_{i:}R_{:v}) R_{\hat{r}v}. \end{aligned}$$

The minimizer of the right hand side of the bound on $f(L, R + \delta E_{\hat{r}j})$ is derived in an analogous fashion.

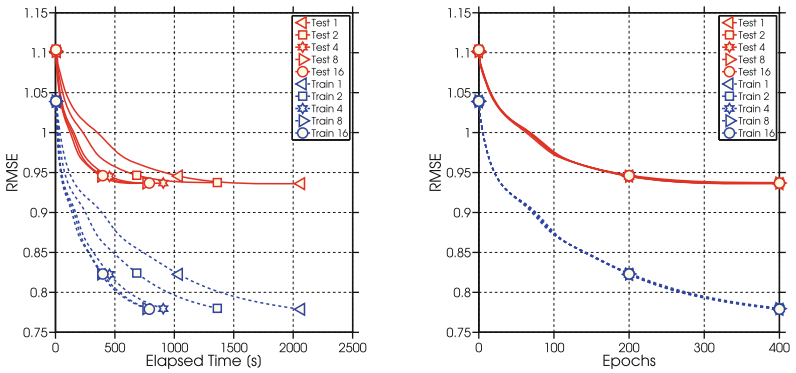


**Fig. 1.** RMSE as a function of the number of iterations and wall-clock time, respectively, on a well-known $480189 \times 17770$ matrix, for $r = 20$ and $\mu = 16$.

## 3 Numerical Experiments

A particular care has been taken to produce a numerically stable and efficient implementation. Algorithmically, the key insight is that Eq. (4) does not require as much computation as it seemingly does. Let us define matrix $A \in \mathbb{R}^{m\times r}$ and $B \in \mathbb{R}^{r\times n}$ such that $A_{iv} = W^{\mathcal{L}}_{iv}$ and $B_{vj} = V^{\mathcal{U}}_{vj}$. After each update of the solution, we also update those matrices. We also store and update sparse residuals, where

$(\Delta_{\mathcal{E}})_{i,j}$ is $L_{i:}R_{:j} - X^{\mathcal{E}}_{ij}$ for $(ij) \in \mathcal{E}$ and zero elsewhere, and similarly for $\Delta_{\mathcal{U}}$, $\Delta_{\mathcal{L}}$. Subsequently, the computation of $\delta_{i\hat{r}}$ or $\delta_{\hat{r}j}$ is greatly simplified.

Our C++ implementation stores all data stored in shared memory and uses OpenMP multi-threading. Figure 1 presents the evolution of RMSE over time on the well-known $480189 \times 17770$ matrix of rank 20 on a machine with 24 cores of Intel X5650 clocked at 2.67 GHz and 24 GB of RAM. There is an almost linear speed-up visible from 1 to 4 cores and marginally worse speed-up between 4 and 8 cores. The comparison of run-times of algorithms across multiple papers is challenging, especially when some of the implementations are running across clusters of computers in a distributed fashion. Nevertheless, the best distributed implementation, which uses a custom matrix-completion-specific platform for distributed computing [4], requires the wall-clock time of 95.8 s per epoch on a 5-node cluster, for rank 25, and 121.9 s per epoch on a 10-node cluster, again for rank 25, which translates to the use of 47900 to 121900 node-seconds, on the same $480189 \times 17770$ matrix (denoted N1). For a recent Spark-based implementation [4], the authors report the execution time of one epoch of 500 s for rank between 25 and 50 on a 10-node cluster, with 8 Intel Xeon cores and 32 GB of RAM per node. A run of 100 epochs, which is required to obtain an acceptable precision, hence takes 50000 to 300000 node-seconds. As can be seen in Fig. 1, our algorithm processes the 100 epochs within 500 node-seconds, while using 8 comparable cores. This illustration suggests an improvement of two orders of magnitude, in terms of run-time.

## 4 Conclusions

In conclusion, MACO makes it possible to find stationary points of an NP-Hard problem in matrix completion under uncertainty rather efficiently. The simple and seemingly obvious addition of inequality constraints to matrix completion seems to improve the statistical performance of matrix completion in a number of applications, such as collaborative filtering under interval uncertainty, robust statistics, event detection [7,9], and background modelling in computer vision [1,2,5,6]. We hope this may spark further research, both in terms of dealing with uncertainty in matrix completion and in terms of the efficient algorithms for the same.

**Acknowledgement.** The work of JM received funding from the European Union's Horizon 2020 Programme (Horizon2020/2014-2020) under grant agreement No. 688380. The work of MT was partially supported by the U.S. National Science Foundation, under award numbers NSF:CCF:1618717, NSF:CMMI:1663256, and NSF:CCF:1740796. PR acknowledges support from KAUST Faculty Baseline Research Funding Program.

## References

1. Akhriev, A., Marecek, J., Simonetto, A.: Pursuit of low-rank models of time-varying matrices robust to sparse and measurement noise. Preprint arXiv:1809.03550 (2018, submitted)

2. Dutta, A., Li, X., Richtarik, P.: Weighted low-rank approximation of matrices and background modeling. Preprint arXiv:1804.06252 (2018, submitted)
3. Jahrer, M., Töscher, A., Legenstein, R.: Combining predictions for accurate recommender systems. In: KDD, pp. 693–702. ACM (2010)
4. Li, B., Tata, S., Sismanis, Y.: Sparkler: supporting large-scale matrix factorization. In: EDBT, pp. 625–636. ACM (2013)
5. Li, X., Dutta, A.: Weighted low rank approximation for background estimation problems. In: ICCVW, pp. 1853–1861, October 2017
6. Li, X., Dutta, A., Richtarik, P.: A batch-incremental video background estimation model using weighted low-rank approximation of matrices. In: ICCVW, pp. 1835–1843, October 2017
7. Marecek, J., Maroulis, S., Kalogeraki, V., Gunopulos, D.: Low-rank methods in event detection. Preprint arXiv:1802.03649 (2018, submitted)
8. Marecek, J., Richtarik, P., Takac, M.: Matrix completion under interval uncertainty. Eur. J. Oper. Res. **256**(1), 35–43 (2017)
9. Marecek, J., Simonetto, A., Maroulis, S., Kalogeraki, V., Gunopulos, D.: Low-rank subspace pursuit in event detection (2018, submitted)