Active Matrix Completion for Algorithm Selection

Mustafa Mısır[1,2](✉)

[1] Department of Computer Engineering, Istinye University, Istanbul, Turkey
mustafa.misir@istinye.edu.tr

[2] College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China

Abstract. The present work accommodates active matrix completion to generate cheap and informative incomplete algorithm selection datasets. Algorithm selection is being used to detect the best possible algorithm(s) for a given problem ($\sim$ instance). Although its success has been shown in varying problem domains, the performance of an algorithm selection technique heavily depends on the quality of the existing dataset. One critical and likely to be the most expensive part of an algorithm selection dataset is its performance data. Performance data involves the performance of a group of algorithms on a set of instance of a particular problem. Thus, matrix completion [1] has been studied to be able to perform algorithm selection when the performance data is incomplete. The focus of this study is to come up with a strategy to generate/sample low-cost, incomplete performance data that can lead to effective completion results. For this purpose, a number of matrix completion methods are utilized in the form of active matrix completion. The empirical analysis carried out on a set of algorithm selection datasets revealed significant gains in terms of the computation time, required to produce the relevant performance data.

1 Introduction

The experimental studies comparing different algorithms on a set of problem instances usually report that while a particular algorithm works well on a group of instances, it fails to outperform the competing algorithms on the other instances. In other words, there is no one algorithm that always perform the best, as also referred in the No Free Lunch (NFL) theorem [2]. Algorithm Selection [3] is an automated way of aiming at choosing the (near-) best algorithm(s) for solving a given problem instance. Thus, the target instances can be solved with the help of multiple algorithms rather than just one. In that respect, algorithm selection can offer an opportunity of defeating any given problem specific algorithm designed by the domain experts, as in the SAT competitions[1].

[1] http://www.satcompetition.org/.

G. Nicosia et al. (Eds.): LOD 2019, LNCS 11943, pp. 321–334, 2019.
https://doi.org/10.1007/978-3-030-37599-7_27

A traditional algorithm selection approach derives performance prediction models [4] that can tell the performance of a given set of algorithms on a new problem instance. For generating such models, the performance of these algorithms on a suite of training instances should be known. Besides, a set of features that can effectively characterize the target problem's instances is essential. Then, an algorithm selection model can simply map these instance features to the algorithms' performance. However, it can be challenging to generate the performance data in the first place. Finding representative instance features can also be complicated depending on the available problem domain expertise. Regarding the performance data, the main issue is the cost of generating it. Especially, if the computational resources are limited, it could take days, months, even years [5] to generate a single algorithm selection dataset. This drawback can restrict the use of algorithm selection by anyone who would like to perform algorithm selection on a new problem domain.

A collaborative filtering based algorithm selection technique, i.e. ALORS [1], was introduced to resolve this issue to a certain extent. Collaborative filtering [6] is a popular field of recommender systems to predict the interest of a user on an item such as a book or a movie, s/he haven't seen yet. The prediction process particularly relies on the user's preferences on multiple items, such as the scores given by her/him on these items. By taking all the other users' partial preferences into account, the existing partial preference information of this user can be utilized to determine whether s/he will like those items. From this perspective, if two users share similar preferences, their preferences on the unobserved items are also expected to be similar. The ALORS' collaborative filtering capability comes from the use of matrix completion, which is the performance data in this context. As its name suggests, matrix completion is about filling the unavailable entries of an incomplete matrix. In relation to the matrix completion task, ALORS showed that it can outperform the single best algorithm on varying problems with up to the 90% sparsity. Its matrix completion component also showed success in process mining [7].

The present study focuses on incorporating Active Matrix Completion (AMC) [8,9] into algorithm selection considering the data generation cost and quality. The AMC problem is defined as: for a given matrix $\mathcal{M}$ with unobserved entries, determine new entries to be queried, Q so that $\mathcal{M}' = \mathcal{M} \cup Q$ carries sufficient information for successful completion compared to $\mathcal{M}$. Minimizing $\mid Q \mid$ can also be targeted during querying. AMC is practical to determine the most profitable entries to add into a matrix for improved matrix completion. For this purpose, I apply various matrix completion techniques and together as ensembles (Ensemble Matrix Completion) in the form of AMC to specify the most suitable instance-algorithm entries to be sampled. In [10], a matrix completion method was used to perform both sampling and completion in ALORS. This study extends [10] by considering a set of existing matrix completion methods while examining their combined power as AMC ensembles. For analysis, a series of experiments is performed on the Algorithm Selection library (ASlib)[2] [5] datasets.

[2] http://aslib.net.

In the remainder of the paper, Sect. 2 gives background information both on algorithm selection and matrix completion. The proposed approach is detailed in Sect. 3. Section 4 provides an empirical analysis through the computational results. The paper is finalized with a summary and discussion in Sect. 5.

2 Background

2.1 Algorithm Selection

Algorithm Selection has been applied to various domains such as Boolean Satisfiability [11], Constraint Satisfaction [12], Combinatorial Optimization [13], Machine Learning [14] and Game Playing [15]. The majority of the methods called algorithm selection is considered working *Offline*. This means that the selection process is being performed before the target problem ($\sim$ instance) is being solved. This study is related to the Offline algorithm selection. Among the existing offline algorithm selection studies, SATZilla [11] is known as one of the premier works. It incorporates pre-solvers and backup solvers to address easy and hard instances, respectively. A rather traditional runtime prediction strategy is utilized to determine the best possible algorithm on the remaining instances. Instance features are used together with the performance data are used to generate the prediction models. SATZilla was further extended [16] with cost-sensitive models, as also in [17]. In [18], a collaborative filtering based recommender system, i.e. [19], was accommodated for AS. Hydra [20], inspired from ISAC [21], was developed to apply parameter tuning for generating algorithm portfolios/diverse algorithm sets, using a single algorithm. 3S [22] was studied particularly to offer algorithm schedules for determining which algorithm to run how long on each target problem instance. For this purpose, the resource constrained set covering problem with column generation was solved. In [23], deep learning was applied for algorithm selection when the instance features are unavailable. The corresponding instance files are converted into images for directly being used with the deep learning's feature extraction capabilities. As a high level strategy on algorithm selection, AutoFolio [24] was introduced to perform parameter tuning to come up with the best possible algorithm selection setting. Some of these AS systems as well as a variety of new designs were competed in the two AS competitions [25], took place in 2015 [26] and 2017[3] [27]. Algorithm scheduling was particularly critical for the successful AS designs like ASAP [28].

AS has been also studied as the *Online* operating methods for choosing algorithms on-the-fly while a problem instance is being solved. Thus, it is particularly suitable for optimization. Selection hyper-heuristics [29] mainly cover this field by combining selection with additional mechanisms. Adaptive Operator Selection (AOS) [30] has been additionally referred purely for Online selection.

[3] https://www.coseal.net/open-algorithm-selection-challenge-2017-oasc/.

2.2 Matrix Completion

Although data might be present in different ways, it is usually not perfect. The being perfect relates to quality and availability. Referring to the data quality, the data at hand might be enough in terms of size yet misleading or involving limited useful information. For availability, some data entries might be missing, requiring to have complete data. Thus, in reality, the systems focusing on data might be required to deal with both issues. In recommender systems, the latter issue is mainly approached as the matrix completion problem, for collaborative filtering [6]. The Netflix challenge[4] [31] was the primary venue made collaborative filtering popular. In the challenge, the goal was to predict missing 99% data from 1% observed entries of a large group of the user scores on many movies. The data was involving the scores regarding the preferences of the users on the target movies. In the literature, the methods addressing this problem are categorized as the *memory-based* (neighbourhood methods) and *model-based* (latent factor models) approaches. The memory-based approaches solely depend the available incomplete data to fill the missing entries. The model-based approaches look for ways to generate models that can perform matrix completion. The model-based approaches are known to be effective especially for the high incompleteness cases.

Matrix factorization [32] has been widely handled by the model-based approaches in collaborative filtering. It has been used to extract latent (hidden) factors that can characterize the given incomplete data such that the missing entries could be effectively predicted. A Probabilistic Matrix Factorization (PMF) method [33] that scales linearly with the number of observations was introduced. CofiRank [34] was devised as a matrix factorization method optimizing an upper-bound on the Normalized Discounted Cumulative Gain (NDCG) criterion for given rank matrices. ListRank-MF [35] was offered as an extension of matrix factorization with a list-wise learning-to-rank algorithm. In [36], matrix completion was studied to show its capabilities under noise. As a growing research direction in matrix factorization, non-negative matrix factorization [37,38] has been studied mainly to deliver meaningful and interpretable factors.

Related to the present work, active learning has been incorporated into matrix factorization [8]. An AMC framework was proposed and tested with various techniques in [8]. In [39], AMC was studied to address the completion problem for the low-rank matrices. An AMC method called `Order&Extend` was introduced to perform both sampling and completion together.

3 Method

A simple data sampling approach for algorithm selection, exemplified in Fig. 1, was investigated in [10]. The idea is to apply matrix completion to predict both cheap and informative unobserved entries. As detailed in Algorithm 1, a given completion method θ is first applied. Then, for each instance (matrix row), at most n number of entries to be sampled are determined. n is set for maintaining

[4] http://www.netflixprize.com.

balanced sampling between the instances. This step is basically performed by running the algorithms on the corresponding instances based on their expected performance. The top n, expectedly, the best missing entries are chosen for this purpose. Considering the target dataset with the performance metric of runtime, the best entries here also mean that the ones requiring the least computation effort for sampling. After running those selected algorithms for their target instances, a partial matrix is delivered, assuming that some entries are initially available.

$$\mathcal{M} = \begin{pmatrix} - & - & 7 & \cdots & - & - & 1 \\ 1 & - & - & \cdots & - & 6 & - \\ - & 6 & - & \cdots & 1 & - & 5 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ - & - & 3 & \cdots & - & 1 & 5 \\ - & 4 & - & \cdots & 5 & - & - \\ - & 1 & - & \cdots & - & - & 3 \end{pmatrix} \xrightarrow[\text{run algorithm } j \text{ on instance } i]{\text{sample } \mathcal{M}_{i,j} \text{ entries with } \theta} \begin{pmatrix} - & \textcolor{red}{5} & 7 & \cdots & - & - & 1 \\ 1 & - & \textcolor{red}{2} & \cdots & - & 6 & \textcolor{red}{4} \\ - & 6 & - & \cdots & 1 & - & 5 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \textcolor{red}{8} & - & 3 & \cdots & - & 1 & 5 \\ - & 4 & - & \cdots & 5 & \textcolor{red}{2} & - \\ \textcolor{red}{5} & 1 & - & \cdots & - & \textcolor{red}{7} & 3 \end{pmatrix}$$

Fig. 1. An active matrix completion example

For evaluation, the matrix is needed to be completely filled, following this active matrix completion sampling step. As in [1,10], the completion is carried out using the basic cosine-similarity approach, commonly used in collaborative filtering. The completed matrix can then be utilized to determine the best/strong algorithm on each instance that is partially known through the incomplete, observed entries. Besides that, it can also be used to perform traditional algorithm selection which additionally requires instance features. The instance features are ignored as the focus here is solely on generating sufficiently informative algorithm selection datasets.

Algorithm 1. Active Matrix Completion for Algorithm Selection

```
  Input: An incomplete performance matrix $\mathcal{M}$ of $I$ instances and $J$ algorithms,
         Matrix completion method $\theta$, Number of samples to be added: $N$
1 Matrix Completion: $\theta(\mathcal{M}) \rightarrow \widehat{\mathcal{M}}$
2 Per Instance Sample Size: $n = N / \mid I \mid$
3 foreach $\widehat{\mathcal{M}}_i$ where $i \in I$ do
4     $\arg\min_{J'} (\widehat{\mathcal{M}}_i, n)$ where $\mathcal{M}_{i,j} = \texttt{NaN}, \ \mid J' \mid \leq n$
5     foreach $j \in J'$ do
6         $\mathcal{M}_{i,j} \leftarrow$ run algorithm $j$ on instance $i$
      end
  end
```

The same completion approach is followed yet with different matrix completion methods as the sampling strategies, as listed in Table 1. The first five algorithms[5] (ϑ) perform rather simple and very cheap completion. Referring to [1], more complex approaches, in particular the model-based ones or the ones performing heavy optimization, are avoided. Besides these methods, three simple ensemble based approaches, i.e. MeanE, MedianE and MinE using ϑ, are implemented. Finally, pure random sampling (Random) from [1] and the matrix completion method (MC) [10] used in the same study are utilized for comparison.

Table 1. Matrix completion algorithms, θ (all in ϑ with their default values from fancyimpute 0.1.0, $\vartheta = \{$SimpleFill, KNN, SoftImpute, IterativeSVD, MICE$\}$)

Name	Type
SimpleFill	Column mean
KNN	k-nearest neighbour ($k = 5$)
SoftImpute [40]	Iterative soft thresholding of SVD
IterativeSVD [41]	Iterative low-rank SVD
MICE [42]	Multiple imputation by chained equations
MeanE	$\sum_i^n \vartheta_i(\mathcal{M})/n$ for $n =\mid \vartheta \mid$
MedianE	$\mu_{1/2}\vartheta_i(\mathcal{M})$ for $\forall i$
MinE	$\min \vartheta_i(\mathcal{M})$ for $\forall i$
Random [1]	Random sampling
MC [10]	Row-based cosine similarity

4 Computational Results

The computational experiments are performed on 13 Algorithm Selection library (ASlib)[6] [5] (Table 2) datasets. The performance data of each ASlib dataset is converted into a rank form as in [1]. For each dataset, AMC is utilized to sample data for the decreasing incompleteness levels, from 90% incompleteness to 10% incompleteness. As the starting point, AMC is applied to each dataset after randomly picking the first 10% of its entries. AMC is tested by gradually sampling new 10% data. For the sake of randomness, initial 10% random sampling is repeated 10 times on each dataset.

Figure 2 reports the performance of each active matrix completion method in terms of time which is the data generation cost. Time (t_C) is normalized w.r.t. picking out the least costly ($t_{C_{best}}$) and the most costly ($t_{C_{worst}}$) samples for the given incompleteness levels, as in $\frac{t_C - t_{C_{best}}}{t_{C_{worst}} - t_{C_{best}}}$. For the SAT12 datasets, all the methods can significantly improve the performance of random sampling (Random) by choosing less costly instance-algorithm samples. For instance,

[5] From fancyimpute 0.1.0: https://pypi.python.org/pypi/fancyimpute.
[6] http://aslib.net.

adding the 10% performance data after the initially selected 10% for SAT12-RAND with MICE can save ∼565 CPU hours (∼24 days) compared to Random. The time gain can reach to ∼1967 CPU hours (∼82 days) (Table 3), on PROTEUS-2014. Significant performance difference is also achieved for the SAT11 and ASP-POTASSCO datasets.

Table 2. The ASlib benchmarks, with the number of CPU days required to generate

Dataset	#Instances	#Algorithms	#CPU
ASP-POTASSCO	1294	11	25
CSP-2010	2024	2	52
MAXSAT12-PMS	876	6	56
PRE.-ASTAR-2015	527	4	28
PROTEUS-2014	4021	22	596
QBF-2011	1368	5	163
SAT11-HAND	296	15	168
SAT11-INDU	300	18	128
SAT11-RAND	600	9	158
SAT12-ALL	1614	31	415
SAT12-HAND	767	31	234
SAT12-INDU	1167	31	284
SAT12-RAND	1362	31	447

Table 3. Average time gain achieved in hours by MICE compared to Random (0.9 is ignored since it is the initial random sampling for both methods)

Dataset	Incompleteness levels							
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
ASP-POTASSCO	33	30	33	39	40	31	19	6
CSP-2010	−107	−21	−78	−83	−60	−31	−1	12
MAXSAT12-PMS	−21	37	28	60	72	53	32	−27
PRE.-ASTAR-2015	−95	−58	−26	−55	−55	−46	−67	−58
PROTEUS-2014	−516	242	1106	1702	1967	1932	1634	1095
QBF-2011	−169	−101	−74	−53	−41	−94	−118	−160
SAT11-HAND	74	108	135	127	125	122	67	6
SAT11-INDU	76	98	103	108	72	58	11	1
SAT11-RAND	47	134	156	169	170	113	80	1
SAT12-ALL	140	526	964	1156	1177	1096	817	403
SAT12-HAND	97	202	313	345	322	277	185	67
SAT12-INDU	138	507	910	1005	886	712	481	225
SAT12-RAND	137	313	560	801	1004	1160	1008	565

For PROTEUS-2014, MC, SimpleFill and KNN start choosing costly samples after the observed entries exceed 60% of the whole dataset. The remaining algorithms show similar behavior when the observed entries reaches 80%. For MAXSAT12-PMS, only SimpleFill, MICE and MeanE are able to detect cheap entries while the rest consistently picks the costly entries. On CSP-2010, PRE.-ASTAR-2015 and QBF-2011, the methods query the costly instance-algorithm pairs. The common characteristic between these three datasets is having a limited number of algorithms: 2, 4 and 5. The reason behind this issue is that for matrix completion, having high-dimensional matrices can increase the chance of high quality completion. For instance, in the aforementioned Netflix challenge, the dataset is composed of ∼480K users and ∼18K movies. Since both dimensions of the user-movie matrix are large, having only 1% of the complete data can be more practical in terms of matrix completion than a small scale matrix. Still, choosing costly entries can be helpful for the quality of the completion due to having misleading observed entries. Referring to the performance of the ensemble methods, despite the success of MeanE, MinE delivers poor performance while MedianE achieves average quality performance compared to the rest. The performance of MC, as the method of running the exact same completion method for both sampling and completion, it comes with the best performance on SAT12-INDU and SAT12-RAND together with the other matrix completion methods. However, it shows average performance on the remaining datasets except CSP-2010 where it delivers the worst performance.

Figure 3 presents the matrix completion performance in terms of average ranks, following each AMC application. The average rank calculation refers to choosing the lowest ranked algorithm for each instance w.r.t. the filled matrix, then evaluating its true rank performance. For SAT12-RAND, all the methods significantly outperform Random except KNN and MinE which provide similar performance to Random. SimpleFill, IterativeSVD, MC, MICE and MeanE particularly shows superior performance compared to Random. On SAT12-INDU, the results indicate that especially SimpleFill, MC, MICE and MeanE are able to provide significant performance. Yet, when the observed entries reach 60%, their corresponding rank performance starting to degrade. For SAT12-HAND, the significant improvement in terms of time doesn't help to reach to outperform Random. However, IterativeSVD, MICE, MeanE and MedianE are able to deliver similar performance to Random when the observed entries is below 50%. For the ASP-POTASSCO, PROTEUS-2014 and SAT11 datasets, similar rank performance can be seen for varying matrix completion methods. The majority of the methods is able to catch the performance of Random throughout all the tested incompleteness levels. On the CSP-2010 dataset, significantly better rank performance is achieved compared to Random yet as mentioned above more computationally costly samples are requested. Apart from preferring the expensive entries, the results indicate that these samples are able to elevate the quality of the matrix completion process. On the remaining datasets with the relatively small algorithm sets, i.e. PRE.-ASTAR-2015 and QBF-2011, similar performance is achieved against Random. However, it should be noted that this average rank performance is delivered

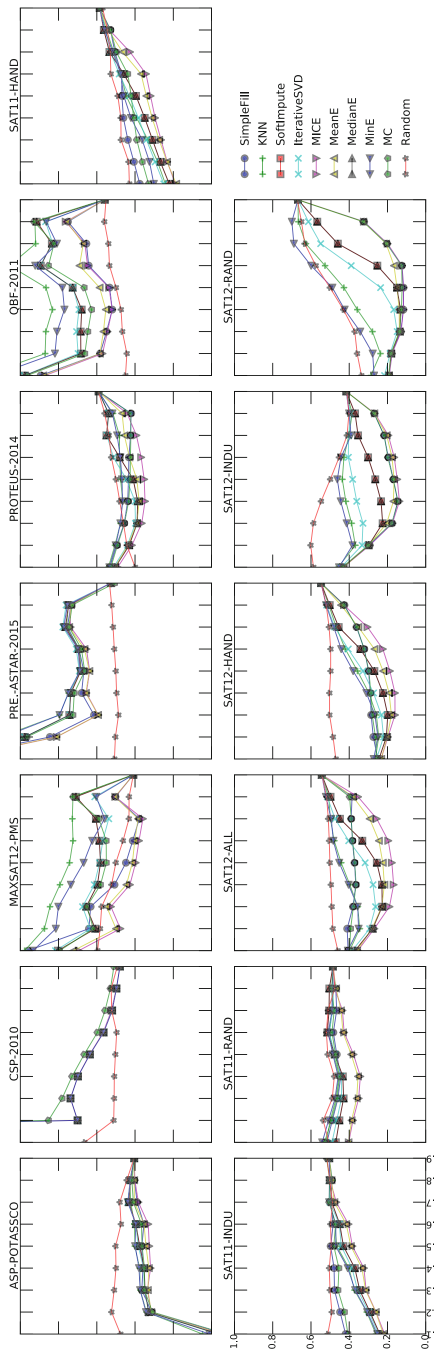

Fig. 2. Normalized time ratio (smaller is better) for varying incompleteness levels, 0.1 → 0.9 (0.9 refers to the initial 10% data case)

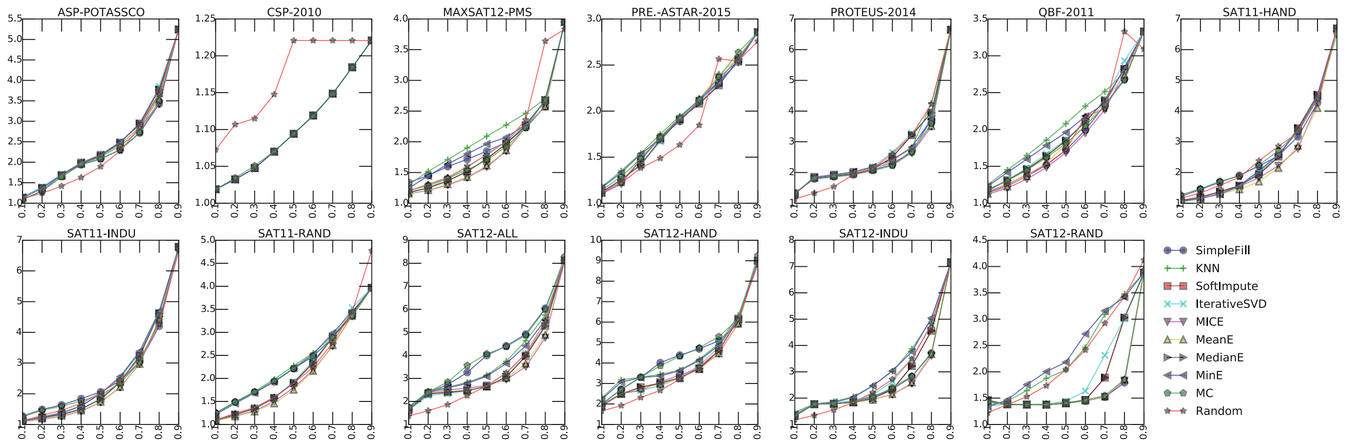

Fig. 3. Average rank (smaller is better) for varying incompleteness levels, 0.1 → 0.9 (0.9 refers to the initial 10% data case)

with more costly entries than Random. For the ensemble methods, their behavior on the cost of generating the instance-algorithm data is reflected to their average rank performance. Similar to the MC's average cost saving performance on the data generation/sampling, its rank performance is either being outperformed or matched with the other tested matrix completion techniques.

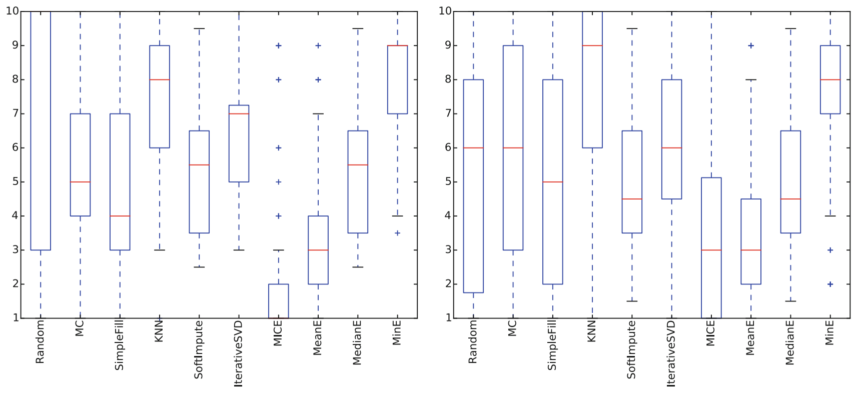

Fig. 4. Overall rank of the matrix completion methods as AMC, based on (a) normalized time ratio and (b) average rank

Figure 4 shows the overall performance of AMC across all the ASlib datasets with the aforementioned incompleteness levels. In terms of generating cheap performance data, MICE comes as the best approach, followed by an ensemble method, i.e. MeanE. Random delivers the worst performance. MinE, another ensemble method, also performs poorly together with KNN. For their average rank performance, MICE and MeanE come as the best performing methods. The overall rank performance of KNN and MinE are even worse than Random. Besides that, MC performs similarly to Random.

5 Conclusion

This study applies active matrix completion (AMC) to algorithm selection for providing high quality yet computationally cheap incomplete performance data. The idea of algorithm selection is about automatically choosing algorithm(s) for a given problem ($\sim$instance). The selection process requires a set of algorithms and a group of problem instances. Performance data concerning the success of these algorithms on the instances plays a central role on the algorithm selection performance. However, generating this performance data could be quite expensive. ALORS [1] applies matrix completion to perform algorithm selection with limited/incomplete performance data. The goal of this work is deliver an informed data sampling approach to determine how the incomplete performance data is to be generated with the cheap to calculate entries while maintaining the data quality. For this purpose, a number of simple and fast matrix completion methods is utilized. The experimental results on the algorithm selection library (ASlib) benchmarks showed that AMC can provide substantial time gain on generating performance data for algorithm selection while delivering strong matrix completion performance, especially for the datasets with large algorithm sets.

The follow-up research plan covers investigating the effects of matrix completion on cold start which is the traditional algorithm selection task, i.e. choosing algorithms for unseen problem instances. Next, the problem instance features will be utilized for further improving the AMC performance. Afterwards, algorithm portfolios [43] across many AMC methods will be explored by extending the utilized AMC techniques. Finally, AMC will be targeted as a multi-objective optimization problem for minimizing the performance data generation time while maximizing the commonly used algorithm selection performance metrics.

Acknowledgement. This study was partially supported by an ITC Conference Grant from the COST Action CA15140.

References

1. Mısır, M., Sebag, M.: ALORS: an algorithm recommender system. Artif. Intell. **244**, 291–314 (2017)
2. Wolpert, D., Macready, W.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**, 67–82 (1997)
3. Kerschke, P., Hoos, H.H., Neumann, F., Trautmann, H.: Automated algorithm selection: survey and perspectives. Evol. Comput. **27**(1), 3–45 (2019). MIT Press
4. Hutter, F., Xu, L., Hoos, H.H., Leyton-Brown, K.: Algorithm runtime prediction: methods & evaluation. Artif. Intell. **206**, 79–111 (2014)
5. Bischl, B., et al.: ASlib: a benchmark library for algorithm selection. Artif. Intell. **237**, 41–58 (2017)
6. Su, X., Khoshgoftaar, T.M.: A survey of collaborative filtering techniques. Adv. Artif. Intell. **2009**, 4 (2009)
7. Ribeiro, J., Carmona, J., Mısır, M., Sebag, M.: A recommender system for process discovery. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 67–83. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10172-9_5
8. Chakraborty, S., Zhou, J., Balasubramanian, V., Panchanathan, S., Davidson, I., Ye, J.: Active matrix completion. In: The 13th IEEE ICDM, pp. 81–90 (2013)
9. Ruchansky, N., Crovella, M., Terzi, E.: Matrix completion with queries. In: Proceedings of the 21th ACM SIGKDD KDD, pp. 1025–1034 (2015)
10. Mısır, M.: Data sampling through collaborative filtering for algorithm selection. In: the 16th IEEE CEC, pp. 2494–2501 (2017)
11. Xu, L., Hutter, F., Hoos, H., Leyton-Brown, K.: SATzilla: portfolio-based algorithm selection for SAT. J. Artif. Intell. Res. **32**(1), 565–606 (2008)
12. Yun, X., Epstein, S.L.: Learning algorithm portfolios for parallel execution. In: Hamadi, Y., Schoenauer, M. (eds.) LION 2012. LNCS, pp. 323–338. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34413-8_23
13. Beham, A., Affenzeller, M., Wagner, S.: Instance-based algorithm selection on quadratic assignment problem landscapes. In: ACM GECCO Comp, pp. 1471–1478 (2017)

14. Vanschoren, J.: Meta-learning: a survey. arXiv preprint arXiv:1810.03548 (2018)
15. Stephenson, M., Renz, J.: Creating a hyper-agent for solving angry birds levels. In: AAAI AIIDE (2017)
16. Xu, L., Hutter, F., Shen, J., Hoos, H., Leyton-Brown, K.: SATzilla2012: improved algorithm selection based on cost-sensitive classification models. In: Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions, pp. 57–58 (2012)
17. Malitsky, Y., Sabharwal, A., Samulowitz, H., Sellmann, M.: Algorithm portfolios based on cost-sensitive hierarchical clustering. In: The 23rd IJCAI, pp. 608–614 (2013)
18. Stern, D., Herbrich, R., Graepel, T., Samulowitz, H., Pulina, L., Tacchella, A.: Collaborative expert portfolio management. In: The 24th AAAI, pp. 179–184 (2010)
19. Stern, D.H., Herbrich, R., Graepel, T.: Matchbox: large scale online Bayesian recommendations. In: The 18th ACM WWW, pp. 111–120 (2009)
20. Xu, L., Hoos, H., Leyton-Brown, K.: Hydra: automatically configuring algorithms for portfolio-based selection. In: The 24th AAAI, pp. 210–216 (2010)
21. Kadioglu, S., Malitsky, Y., Sellmann, M., Tierney, K.: ISAC-instance-specific algorithm configuration. In: The 19th ECAI, pp. 751–756 (2010)
22. Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., Sellmann, M.: Algorithm selection and scheduling. In: Lee, J. (ed.) CP 2011. LNCS, vol. 6876, pp. 454–469. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23786-7_35
23. Loreggia, A., Malitsky, Y., Samulowitz, H., Saraswat, V.A.: Deep learning for algorithm portfolios. In: The 13th AAAI, pp. 1280–1286 (2016)
24. Lindauer, M., Hoos, H.H., Hutter, F., Schaub, T.: AutoFolio: an automatically configured algorithm selector. JAIR **53**, 745–778 (2015)
25. Lindauer, M., van Rijn, J.N., Kotthoff, L.: The algorithm selection competitions 2015 and 2017. arXiv preprint arXiv:1805.01214 (2018)
26. Kotthoff, L.: ICON challenge on algorithm selection. arXiv preprint arXiv:1511.04326 (2015)
27. Lindauer, M., van Rijn, J.N., Kotthoff, L.: Open algorithm selection challenge 2017: setup and scenarios. In: OASC 2017, pp. 1–7 (2017)
28. Gonard, F., Schoenauer, M., Sebag, M.: Algorithm selector and prescheduler in the ICON challenge. In: Talbi, E.-G., Nakib, A. (eds.) Bioinspired Heuristics for Optimization. SCI, vol. 774, pp. 203–219. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-95104-1_13
29. Pillay, N., Qu, R.: Hyper-Heuristics: Theory and Applications. Natural Computing Series. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96514-7
30. Da Costa, L., Fialho, A., Schoenauer, M., Sebag, M.: Adaptive operator selection with dynamic multi-armed bandits. In: GECCO, pp. 913–920 (2008)
31. Bennett, J., Lanning, S., et al.: The netflix prize. In: Proceedings of KDD Cup and Workshop, New York, NY, USA, vol. 2007, p. 35 (2007)
32. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)
33. Salakhutdinov, R., Mnih, A.: Probabilistic matrix factorization. In: Platt, J.C., Koller, D., Singer, Y., Roweis, S.T. (eds.) The 21st NIPS, pp. 1257–1264 (2007)
34. Weimer, M., Karatzoglou, A., Le, Q.V., Smola, A.: CofiRank - maximum margin matrix factorization for collaborative ranking. In: The 21st NIPS, pp. 222–230 (2007)
35. Shi, Y., Larson, M., Hanjalic, A.: List-wise learning to rank with matrix factorization for collaborative filtering. In: The 4th ACM RecSys, pp. 269–272 (2010)

36. Candes, E.J., Plan, Y.: Matrix completion with noise. Proc. IEEE **98**(6), 925–936 (2010)
37. Wang, Y.X., Zhang, Y.J.: Nonnegative matrix factorization: a comprehensive review. IEEE TKDE **25**(6), 1336–1353 (2013)
38. Gillis, N., Vavasis, S.A.: Fast and robust recursive algorithms for separable non-negative matrix factorization. IEEE TPAMI **36**(4), 698–714 (2014)
39. Sutherland, D.J., Póczos, B., Schneider, J.: Active learning and search on low-rank matrices. In: Proceedings of the 19th ACM SIGKDD KDD, pp. 212–220 (2013)
40. Mazumder, R., Hastie, T., Tibshirani, R.: Spectral regularization algorithms for learning large incomplete matrices. JMLR **11**, 2287–2322 (2010)
41. Troyanskaya, O., et al.: Missing value estimation methods for DNA microarrays. Bioinformatics **17**(6), 520–525 (2001)
42. Azur, M.J., Stuart, E.A., Frangakis, C., Leaf, P.J.: Multiple imputation by chained equations: what is it and how does it work? Int. J. Methods Psych. Res. **20**(1), 40–49 (2011)
43. Gomes, C., Selman, B.: Algorithm portfolio design: theory vs. practice. In: The 13th UAI, pp. 190–197 (1997)