The VLDB Journal (2020) 29:1475–1500
https://doi.org/10.1007/s00778-020-00623-8

REGULAR PAPER

Incremental preference adjustment: a graph-theoretical approach

Liangjun Song[1] · Junhao Gan[2] · Zhifeng Bao[1] · Boyu Ruan[3] · H. V. Jagadish[4] · Timos Sellis[5]

Received: 8 September 2019 / Revised: 11 May 2020 / Accepted: 19 July 2020 / Published online: 3 August 2020

Abstract
Learning users' preferences is critical to personalized search and recommendation. Most such systems depend on lists of items rank-ordered according to the user's preference. Ideally, we want the system to adjust its estimate of users' preferences after every interaction, thereby becoming progressively better at giving the user what she wants. We also want these adjustments to be gradual and explainable, so that the user is not surprised by wild swings in system rank ordering. In this paper, we support a *rank-reversal* operation on two items x and y for users: adjust the user's preference such that the personalized rank of x and y is reversed. We emphasize that this problem is orthogonal to the preference learning and its solutions can run on top of the learning outcome of any vector-embedding-based preference learning model. Therefore, our preference adjustment techniques enable all those existing offline preference learning models to incrementally and interactively improve their response to (indirectly specified) user preferences. Specifically, we define the Minimum Dimension Adjustment (MDA) problem, where the preference adjustments are under certain constraints imposed by a specific graph and the goal is to adjust a user's preference by reversing the personalized rank of two given items while minimizing the number of dimensions with value changed in the preference vector. We first prove that MDA is NP-hard, and then show that a 2.17-approximate solution can be obtained in polynomial time provided that an optimal solution to a carefully designed problem is given. Finally, we propose two efficient heuristic algorithms, where the first heuristic algorithm can achieve an approximation guarantee, and the second is provably efficient. Experiments on five publicly available datasets show that our solutions can adjust users' preferences effectively and efficiently.

Keywords Incremental preference adjustment · Rank reversals · Network flows · Graph theory · Algorithms

1 Introduction

Users are gaining access to numerous datasets nowadays. Search and recommendation systems become increasingly important as a way for users to explore data in an unfamiliar dataset. These systems usually need to develop ranked lists of items in the dataset that are *relevant* to the user's need and tailored to the user's *individual preference* [26].

In the past decades, numerous preference learning models have been developed, such as *Bayesian Personalized Ranking* (BPR) [22], *Weighted Approximate-Rank Pairwise Loss* (WARP) [29] and *Adversarial Personalized Ranking for recommendation* (APR) [14]. All these models require historical data for a user in the training phase [15]. Unfortunately, historical data is often unavailable. This is always the case when the systems are open for new users without any historical

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s00778-020-00623-8) contains supplementary material, which is available to authorized users.

✉ Junhao Gan
junhao.gan@unimelb.edu.au

Liangjun Song
liangjun.song@rmit.edu.au

Zhifeng Bao
zhifeng.bao@rmit.edu.au

Boyu Ruan
b.ruan@uq.edu.au

H. V. Jagadish
jag@eecs.umich.edu

Timos Sellis
tsellis@swinburne.edu.au

[1] RMIT University, Melbourne, Australia

[2] University of Melbourne, Melbourne, Australia

[3] University of Queensland, Brisbane, Australia

[4] University of Michigan, Ann Arbor, USA

[5] Swinburne University of Technology, Melbourne, Australia

data. As a result, preference learning should never be a one-shot process. Instead, it should be a continuous and ongoing evolution.

Consider real estate search as an example. Our goal is to develop a system that behaves like a personal real estate agent, who helps customers to find their candidate properties progressively: At first, users may have preferences on some attributes of the house (e.g. price, location) and interact with the system; then the system returns a ranked list according to users' specified criteria; users may give feedback to the system, indicating their preferences; through the interaction, the system learns and evolves, trying to provide a more personalized and better-ranked list in the next round. We refer readers to a detailed case study over real-world real estate data conducted in Sect. 2.

Motivated by this, in this paper, we study the problem of "post-learning" adjustments on user preferences. Specifically, our goal is to update the preferences learned by a *static* (or offline) model based upon new feedback from users. We emphasize that *our aim is* **neither** *to propose a new ranking model,* **nor** *to improve the subjective recommendation quality of any existing model.* Instead, the problem we study here is in a completely *orthogonal* direction (as discussed in the next paragraph). Moreover, our proposed techniques can be applied to **any** *vector embedding based* preference learning model, where (i) both the preferences of the users and the items are represented by vectors, and (ii) the extent of a user being fond of an item is computed by the dot-product between the user's preference vector and the vector of the item.

In particular, consider a preference learning model and a vector embedding that the model learned, our primary intention is to support a "rank-reversal" operation on two items x and y specified by users: adjust the user's preference such that the (personalized) rank of x and y is reversed. Among the explicit feedback that a user can provide [12], we argue that a user may naturally be curious about "why item y is not ranked higher than item x". For example, it is usually easy for a user to tell which house she likes more (between two houses), but it may be difficult for them to tell how to weight each attribute of a house they prefer. From all such rank-reversal operation requests, more and more information about a user's personal preference is revealed, and hence, the preference vector of the user can be adjusted to meet her subjective preference (under the specified preference learning model).

For this purpose, we define the *Minimum Dimension Adjustment (MDA) rank-reversal* problem, where the adjustments on users' preference vectors are under certain constraints and the goal is to adjust a user's preference vector by reversing the personalized rank of two given items while minimizing the number of dimensions with their value changed in the preference vector.

As we will see shortly in Sect. 2, although there can be multiple adjustments on the preference vector to have the rank order between two given items reversed, unrestricted adjustments will be problematic. Restrictions (or say, constraints) on the adjustments on the *weights* of attributes (i.e., the coordinate values in the corresponding dimensions) in the preference vectors are necessary. For this reason, we propose to adopt a *weight transition graph* to limit the changes happening on different attributes.

Interestingly, as we will see in the problem definition in Sect. 3, the weight transition graph in the MDA problem is reminiscent of the *propagation probability graph* in the *Influence Maximisation (IM)* problem [18] which has been widely studied in recent years [20,24,25]. In the IM problem, knowledge from domain experts is required to assign the propagation probabilities to each edge in the graph, while such graph is widely considered as an input and assumed to be known a priori in all the existing related work. Likewise, in the MDA problem, we adopt the same assumption and treat the weight transition graph as an input. Nonetheless, we introduce an alternative algorithm (in Sect. 8.1) to construct weight transition graphs from the training data only, which allows users to enjoy our techniques even though they may not have domain expertise.

Furthermore, the importance of the role of the weight transition graph, constructed by our algorithm without domain knowledge, has been evident in our empirical study (See Sect. 9.3). The results show that the weight transition graph constructed by our simple algorithm is highly effective. Therefore, it is reasonable that the effectiveness can be improved when domain knowledge is available to tune the weight transition graph. Moreover, the experimental results show that the weight transition graph is critical in preference adjustments, without which the adjusted preferences are extremely unstable.

To this end, it is worth emphasizing the following highlights of this work. (1) The MDA *rank-reversal* problem is independent of the choice of the preference learning model in search and recommendation systems. (2) Our solutions can run on top of the learning outcome of any (user and item) vector embedding based preference learning model. Therefore, our preference adjustment techniques can be applied to enable all those existing offline preference learning models to incrementally and interactively improve their users' preferences. (3) As a side product, our solutions can be used to support the cold-start scenario, which itself is a kind of preference model.

Finally, our contributions in this paper are as follows.

– We conduct a case study over a real-world real estate dataset, to justify why *rank-reversal* suffices to represent user's preference adjustment, how MDA may provide to

the user some insights behind the adjustment, and why the rank reversal under constraints are necessary.
- We formally define the MDA *rank-reversal* preference adjustment problem.
- We show the NP-hardness of MDA with a non-trivial proof connecting the Maximum Coverage problem [2] to the MDA via the notion of flows.
- We show that a 2.17-approximate solution can be obtained in polynomial time using an optimal solution to a carefully designed problem called the F-problem. Though the F-problem is also NP-hard, a local optimum can be computed efficiently by existing popular optimization techniques such as the Lagrange multipliers method [4] and the gradient descent [11].
- We propose two efficient heuristic algorithms for solving the MDA problem: (i) ItrLP and (ii) ItrSSP. In particular, we show an approximation guarantee for the ItrLP, while we prove that the running time complexity of the ItrSSP is the same as the well-known Successive Shortest Path (SSP) algorithm [9].
- We conduct experiments on five real datasets and three well-adopted preference learning models, and evaluate our methods in terms of (1) number of dimensions changed, (2) incremental adjustment effectiveness, and (3) interactive speed. The results show that our methods are both effective and efficient.

Paper organization Table 1 introduces the notations which are frequently used in the paper. In Sect. 2, we present a case study. In Sect. 3, we define the rank-reversal problem and its special variant, the minimum dimension adjustment (MDA) problem. Section 4 introduces preliminaries about flows and reveals a subtle connection between the MDA problem and min-cost flows. Section 5 shows the NP-hardness of the MDA problem and Sect. 6 proposes a 2.17-approximate algorithm. We propose two heuristic algorithms in Sect. 7. To facilitate experiment illustration, in Sect. 8 we present a method to construct the weight transition graph G when it is not available in hand. Section 9 describes experiments and Sect. 10 gives a literature review. We conclude the paper in Sect. 11.

2 Motivation and case study

Before we formally define the MDA problem, in this section, we first conduct a case study on an example in a real estate scenario. Our work in this paper indeed is inspired by our observations gained from HomeSeeker,[1] a real estate data exploration system [19] developed by one of the authors.

[1] http://civilcomputing.com/HomeSeeker

Table 1 Frequently used notations

Notation	Description
$[n]$	the set of integers in $[1, n]$
i, j	two index variables in $[n]$
w	a preference vector
Δw	a weight adjustment on w
x, y	two item vectors
$v[i]$	the ith coordinate in a vector v
$G = \langle V, E \rangle$	a weight transition graph
$G_{\text{ext}}(S, T, \text{w})$	an extended weight transition graph w.r.t. S and T
$G^{\text{cost}}_{\text{ext}}(S, T, \text{w})$	a cost-associated extended weight transition graph
$S(\Delta\text{w})$	$\{i \in [n] \mid \Delta\text{w}[i] < 0\}$
$T(\Delta\text{w})$	$\{i \in [n] \mid \Delta\text{w}[i] > 0\}$
$S_\delta(\Delta\text{w})$	$\{i \in [n] \mid \Delta\text{w}[i] \leq -\delta\}$
$T_\delta(\Delta\text{w})$	$\{i \in [n] \mid \Delta\text{w}[i] \geq \delta\}$

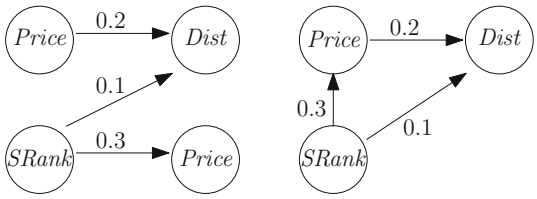
Background and basic settings HomeSeeker has collected data for over 1.5 million properties in Australia, where each property has 72 dimensions (i.e., attributes). The dimensions are categorized into *five* unique profiles: *education profile* (e.g., school zones and ranks), *transportation profile* (e.g., distance to the nearest train station, travel time to city center), *facility profile* (e.g., number of supermarkets and general practitioners (GP's) within 1 km), *suburb profile* (e.g., census data), and *in-house profile* (e.g., landsize and number of bedrooms).

To facilitate readers' understanding, we select *six dimensions* which are of most interests with our domain knowledge: (1) *Price* (in thousand AUD); (2) *Bed*, the number of bedrooms; (3) *Size*, the land size (in square meter); (4) *SRank*, the rank of the school zone; (5) *Dist*, the distance to the nearest train station (in meter); and (6) *Tran*, which is the transportation time of the train to city (in minute). Furthermore, each of these attributes is normalized to a value in the range of $[-1, 1]$ such that the larger value is more preferable. In other words, each house is represented by a 6-dimensional vector in $[-1, 1]^6$. Likewise, each user is represented by a 6-dimensional vector w in the same space indicating her preference on the corresponding attributes. Thus, the score of a house x with respect to w is measured by $\text{w} \cdot \text{x}$, and higher scores are preferred by the user. The ranked list returned to the user is essentially the list of the houses sorted by their scores in descending order.

Why are pair-wise rank reversals sufficient? In HomeSeeker, an important yet fine-grained operation is to reverse the ranks of a pair of two houses by "adjusting" the preference vector w of a user. An adjustment is conducted by transferring weights in w from dimensions to dimensions. For instance, as shown in Table 2, to reverse the ranks of the two houses x and

Table 2 Two houses x and y and the preference vector w

ID	Price	Bed	Size	SRank	Dist	Tran
x	0.4	1	1	0.6	0.2	1
y	0.2	1	1	0.4	0.4	1
x − y	0.2	0	0	0.2	−0.2	0
w	0.1	0.1	0.1	0.4	0.1	0.1



(a) Pair-wise Constraints **(b)** Flow-like Constraints

Fig. 1 The weight transition constraints in Example 1 (the number on each arrow indicates the cap of weight transition)

y under the preference vector w, one can transfer 0.2 weight from *SRank* to *Dist*, that is, the adjusted w becomes to have a weight of 0.2 on *SRank* and 0.3 on *Dist*. We call it as the *pair-wise rank reversal operation*. We note that the pair-wise rank reversals are without loss of generality, as reversing the ranks of any two groups of houses can often be broken down into a certain sequence of pair-wise rank reversals. With such (pair-wise) rank reversal operations, HomeSeeker can accordingly adjust the preference vector for the user and hence, improve the quality of its ranked list.

Why are minimum dimension adjustments helpful? When adjusting a preference vector w, it is important for HomeSeeker to keep the *number of such weight transitions* as small as possible, i.e., to adjust w with a *minimum number of dimensions changed*. There are two intuitive reasons.

First, in addition to reversing houses' ranks, HomeSeeker also aims to explain to the user what attributes play a crucial role in the reversal. On the one hand, a large number of dimensions changed in the preference adjustment often requires a big effort from the user to identify which attributes are essentially the key. On the other hand, a concise explanation would be easier for users to learn their own preference on the attributes.

Second, HomeSeeker aims to provide meaningful recommendations, such that it can *retain short-term preference while being flexible enough to capture long-term interest drift (if any)*. The minimum dimension adjustments can largely retain a user's preference on the attributes and hence, the ranked list would not change dramatically under *short-term* reversals. Meanwhile, such adjustments are still general enough to handle *long-term* interest drifts (after a certain number of rank reversals). For example, users' preferences on houses would not change frequently in a short term, but they may do over a long term due to the changes of users' circumstances, such as the increases of income and the growth of a family, in which case, a user may prefer a larger house with more bedrooms and can afford a higher price than before. For this reason, as we will see shortly in the formal problem definition, there is no *explicit* constraint on keeping the previous feedback in the rank reversal. Nonetheless, the objective of minimizing the number of dimensions adjusted can largely (in some sense) keep the short-term preference implicitly. The effectiveness of the adjustments under this objective is supported by the experimental results in Sect. 9.3.

Why are rank reversals under constraints necessary? With no restrictions, a rank reversal operation always admits a "greedy" adjustment, that is, transferring the weights between the most "effective" pair of dimensions for reversing the ranks. Unfortunately, this kind of greedy adjustments sometimes are less meaningful. This is simply because the changes of preferences on the attributes should be limited within a certain tolerance, as earlier illustrated in Sect. 1 and later evident in our empirical study (Sect. 9.3). For example, it is rare that a user can afford unlimited increase in the price for a better school rank.

Flow-like weight transition constraints In general, it is natural to specify a "cap" on the weight adjustment between any two possible pairs of attributes; and all these constraints naturally constitute a *complete bipartite graph* on the attributes. An example is shown in Fig. 1a, where all the other edges are with zero cap and thus omitted for simplicity. Since each of these constraints specifies the cap for a pair of dimensions, we call them as *pair-wise weight transition constraints*. When performing a rank reversal, the weight transition in an adjustment on w has to comply with the caps specified by the pair-wise constraints, and transfer weights from the attributes on the left side to the right in the corresponding complete bipartite graph. As the resulted adjustment is produced in this one-step manner, this kind of adjustment is called a *one-step adjustment*. However, an adjustment on w can be produced by more than one step. We call it as a *multiple-step adjustment*, which is obtained by performing multiple one-step weight transfers. We use Example 1 to show the superiority of the multiple-step adjustments over the one-step counterpart.

Example 1 Consider the two houses x, y and the preference vector w shown in Table 2; clearly, $w \cdot x - w \cdot y = 0.08$ and hence, the user prefers x to y. Suppose that we want to reverse their ranks by adjusting w under the pair-wise weight transition constraints given in Fig. 1a. There is only one feasible one-step adjustment: transferring at least 0.2 weight from *Price* → *Dist*. The resulting preference vector w has $0.1 - 0.2 = -0.1$ weight on *Price* and $0.1 + 0.2 = 0.3$ weight on *Dist*. On the other hand, the adjustment, obtained by first transferring 0.2 weight from *SRank* to *Price*, and then transferring 0.2 weight from *Price* to *Dist*, is a multiple-

step (more specifically, two-step) adjustment. Therefore, the adjusted w has 0.2 weight on *SRank* and 0.3 weight on *Dist*, and interestingly, the weight on the *intermediate* attribute *Price* is unchanged.

Although both adjustments can be used to reverse the ranks of x and y and have just two dimensions changed, the two-step adjustment looks more reasonable than the one-step adjustment. This is because the latter has a negative weight on *Price* which seems less common as it indicates that the user prefers higher prices. In contrast, the two-step adjustment is more explainable — it indicates the user prefers shorter distance to public transport than having a higher school rank, implying that the user is possibly an elder person with no school child.

In general, multi-step adjustments can catch more possibilities and produce more reasonable preference vectors. This capability is of great importance to those systems that aim to provide explainable results to users, like HomeSeeker. Astute readers may have noticed that the process of producing multiple-step adjustments is reminiscent of network flows. Indeed, for multiple-step adjustments, the pair-wise constraints in a complete bipartite graph are essentially equivalent to the *flow-like constraints* as shown in Fig. 1b in a flow network. In particular, multiple-step adjustment in Example 1 is a flow from *SRank* $\rightarrow$ *Price* $\rightarrow$ *Dist* with a flow value of 0.2.

Motivated by the above, we aim to produce multi-step adjustments and adopt the flow-like weight transition constraints. The corresponding flow network is called a *Weight Transition Graph*, as formally defined in Sect. 3.

2.1 A case study

In this section, we use real estate data to present a case study, explaining (1) how MDA can help the user incrementally shape her preference especially when exploring data she is unfamiliar with, and (2) user's (degree of) preference between a certain pair of dimensions might also change w.r.t. the results obtained after every adjustment, reflecting her personalized tolerance on the importance of the dimensions.

Setup To avoid overwhelming the readers, we pick 10 representative houses (i.e., the property type is house) from the 713 houses sold in a suburb of Melbourne in 2016, out of the total 69, 862 ones traded Australia-wise in the same year. Again, we focus on the aforementioned six attributes of the house. The *raw* attribute values of these 10 houses are shown in Table 3. The corresponding normalized values are shown in Table 4, which are obtained by dividing the maximum value among the 713 houses in the corresponding dimension (shown in the first row of Table 3) and assigning signs (either positive or negative) to ensure the larger values are the better for the users.

Table 3 The 10 representative houses (raw data) ranked by the initial w

ID	Price	Bed	Size	SRank	Dist	Tran
Max Value	1700	11	1600	128	4000	72
1	1090	4	728	40	516	40
2	800	4	600	80	500	35
3	1080	4	769	80	400	50
4	**1430**	**4**	**1020**	**75**	**520**	**50**
5	1460	3	780	72	1100	30
6	1320	3	600	80	1000	20
7	**1480**	**3**	**1010**	**76**	**1400**	**27**
8	1680	4	722	60	2000	22
9	1230	3	500	72	1200	40
10	1120	4	678	60	1252	55
Initial w	0.5	0.5	0.5	0.5	0.5	0.5

Table 4 The 10 representative houses (normalized data) ranked by the initial w

ID	Price	Bed	Size	SRank	Dist	Tran
1	−0.641	0.364	0.455	−0.313	−0.129	−0.556
2	−0.471	0.364	0.375	−0.625	−0.125	−0.486
3	−0.635	0.364	0.481	−0.625	−0.1	−0.694
4	−0.841	0.364	0.638	−0.586	−0.13	−0.694
5	−0.859	0.273	0.488	−0.563	−0.275	−0.417
6	−0.776	0.273	0.375	−0.625	−0.25	−0.278
7	−0.871	0.364	0.631	−0.594	−0.35	−0.375
8	−0.988	0.364	0.451	−0.469	−0.5	−0.306
9	−0.724	0.273	0.313	−0.563	−0.3	−0.556
10	−0.659	0.364	0.424	−0.469	−0.313	−0.764

The initial weight in the preference vector w is set as uniform, i.e., w $= \{0.5, 0.5, 0.5, 0.5, 0.5, 0.5\}$. The weight transition graph G, constituted by the flow-like weight transition constraints, is set by our domain knowledge and is shown in Fig. 2.

In a weight transition graph G, the value associated with each directed edge indicates the cap of weight transition (i.e., capacity) via one dimension to another. For example, the capacity of link *Price* $\rightarrow$ *Dist* is 0.12, indicating in a *rank-reversal* operation, it can transfer at most 0.12 weight from any other dimensions via *Price* to *Dist*. If a cheaper house is favored than a house with many bedrooms, there will be a weight transition process from *Bed* to *Price*. The change of the weight value in the preference vector w indicates to which extent one dimension could be sacrificed for another. Moreover, some edges for two dimensions are not balanced, such as *Bed* and *SRank*. The weight from *Bed* to *SRank* (0.15) is larger than that from *SRank* to *Bed* (0.06). It reflects a commonly adopted principle in practice: it is com-

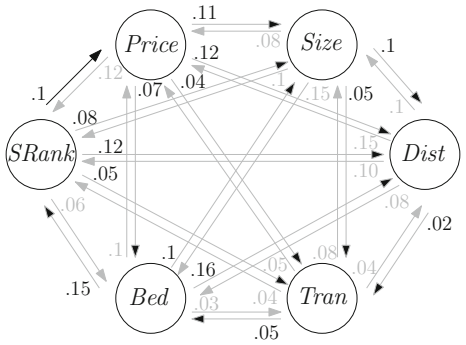


Fig. 2 The weight transition graph G in the case study: The capacity of each edge is shown near to its starting node with the same colour

Table 5 Ranking after each step (R1 means Rank 1, etc.)

Step	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10
1	1	2	6	5	7	**3**	4	8	**10**	9
2	1	2	8	7	6	**5**	10	**4**	3	9
3	1	7	2	8	**4**	5	**6**	10	3	9
4	1	2	7	**3**	6	**4**	5	10	8	9
5	1	2	7	4	6	3	5	8	10	9

mon that the number of bedrooms can be sacrificed for better education quality, but it is rare the education quality can be sacrificed to gain more bedrooms.

So long as within the cap specified in the weight transition graph G, the weight transferred from one to another somehow reflects the extent of user's (relative) tolerance. For example, if 0.07 is transferred from *Price* to *Bed*, it means the user would prefer a house with more bedrooms even it is more expensive; if only 0.02 is transferred from *Price* to *Bed*, then the user may not have a strong preference over a house with many bedrooms.

Next we show a sequence of steps, each of which indicates the user's interaction with the system. In each step, two houses are selected for *rank-reversal* and accordingly, how the preference vector is adjusted. We record the weight transitions on each dimension and present the updated ranked list as a result of each interaction.

Upon the results obtained from the initial preference vector in Table 3, the user called Alice will invoke the *rank-reversal* operation in the following steps.

Step 1: Alice selects House 4 and House 7 for *rank-reversal*. The resulted rank under the adjusted preference vector w and the detailed adjustment Δw on w are shown in rows of Step 1 in Tables 5 and 6, respectively. After the adjustment, the weight on *Bed* decreases 0.05, on *Dist* decreases 0.05 and the weight on *Tran* increases 0.1. It means Alice prefers houses with fewer bedrooms and longer distance to train station, but shorter travel time to city center. Now her selected House 7 is ranked at the 5^{th} position.

Table 6 Adjusted preference vector w after each step

Step		Price	Bed	Size	SRank	Dist	Tran
Initial w		0.5	0.5	0.5	0.5	0.5	0.5
1	Δw		−0.05			−0.05	+0.1
	w	0.5	0.45	0.5	0.5	0.45	0.6
2	Δw				+0.2	−0.15	−0.05
	w	0.5	0.45	0.5	0.7	0.3	0.55
3	Δw		−0.1	+0.2	−0.1		
	w	0.5	0.35	0.7	0.6	0.3	0.55
4	Δw	+0.15		−0.1	−0.05		
	w	0.65	0.35	0.6	0.55	0.3	0.55
5	Δw	−0.08		+0.08			
	w	0.57	0.35	0.68	0.55	0.3	0.55

Step 2: Alice further specifies a *rank-reversal* operation between House 3 and House 10. Afterwards, the House 10's rank lifts from 9^{th} to 7^{th} (see Step 2 in Tables 5 and 6), and the preference vector is adjusted to w = {0.5, 0.45, 0.5, 0.7, 0.3, 0.55}. That tells Alice can sacrifice the distance to train station and the travel time to city for gaining a better education zone.

Step 3: Alice selects House 5 and House 4 and performs a *rank-reversal* operation. See the results in Step 3 in Tables 5 and 6. The updated preference vector is w = {0.5, 0.35, 0.7, 0.6, 0.3, 0.55}. This means the land size gains more preference, while the education and bedroom number become less important to Alice.

Step 4: Alice selects House 4 and House 6 for *rank-reversal*, the result after which is in Step 4 in Tables 5 and 6. We can see that the House 6 previously ranked 7^{th} becomes 5^{th}, and the preference vector is updated to w = {0.65, 0.35, 0.6, 0.55, 0.3, 0.55}. From this step, it indicates that Alice somehow prefers houses with a cheaper price, and she can sacrifice the land size and/or the school rank to gain so.

Step 5: However, Alice's such tolerance (in Step 4) is just to a limited extent. Thus, instead of blindly choosing a cheaper and cheaper house, Alice wants a further adjustment with reversing the ranks of House 3 and House 4. After the *rank-reversal* operation, the results are in Step 5 in Tables 5 and 6. Now the weight is w = {0.57, 0.35, 0.68, 0.55, 0.3, 0.55}, and we can see a small weight dropdown on the price (-0.08), indicating her tolerance on lower price is only valid within a certain extent.

This whole process (Step 1 – Step 5) reveals Alice's preferences adjustment on houses while she may not have a clear idea at the beginning, or her preference can also vary depending on the results obtained or new insights gained from the updated results.

3 Problem formulation

We first introduce some useful notations (see Table 1) and then formally define the Rank-Reversal problem and the Minimum Dimension Adjustment (MDA) problem.

Notation For an integer n, we use $[n]$ to denote the set of integers in the range of $[1, n]$. Consider an n-dimensional vector w; we use $w[i]$ to denote the ith coordinate value of w by for $i \in [n]$. Without explicit statements, in this paper, the index variables i, j are understood to be integers in $[n]$. Given a vector w, the l_p-norm of w is computed as $\|\mathrm{w}\|_p = (\sum_{i=1}^{n} |w[i]|^p)^{\frac{1}{p}}$, where $p \in [0, \infty)$. Specifically, $\|\mathrm{w}\|_0$ is the number of dimensions with nonzero coordinates. Furthermore, since each dimension corresponds to an attribute, we use the term "dimension" and "attribute" interchangeably.

Problem definition Consider a recommender system running with a vector embedding learned by a specified preference learning model on a set of users and a set of items. More specifically, in the embedding, each item has n attributes and is represented by an n-dimensional vector in $[-1, 1]^n$. Each user has an n-dimensional *preference vector* $\mathrm{w} \in [-1, 1]^n$, where $w[i]$ is called the *weight* on the i-th dimension. Define the *preference score* of a user (with a preference vector w) on an item x as the dot-product between w and x, i.e., $\mathrm{w} \cdot \mathrm{x}$, which represents the extent of the user being fond of the item.

To provide recommendations to a user, the system returns a recommended list of items ordered by the preference scores under the user's preference vector w. Meanwhile, the system supports a type of *rank-reversal* operations for the user, where the system can be asked to *reverse* the relative order of a pair of items x and y in the recommended list having $\mathrm{w} \cdot \mathrm{x} > \mathrm{w} \cdot \mathrm{y}$, if such a pair does not meet the user's *true subjective preference* (i.e., the user's true personal preference). The system performs the rank-reversal operation by adjusting w to a new preference vector w′ satisfying $\mathrm{w}' \cdot \mathrm{x} \leq \mathrm{w}' \cdot \mathrm{y}$ with a goal to minimize the l_p-norm of $\Delta\mathrm{w} = \mathrm{w}' - \mathrm{w}$, i.e., $\|\Delta\mathrm{w}\|_p$, for some $p \in [0, \infty)$.

In order to keep the weights stable in the preference vector and avoid short-term interest drifts [12,28], purely aiming to minimize $\|\Delta\mathrm{w}\|_p$ is not sufficient and the system may not wish to allow arbitrary Δw's. Rather, it may limit the amount (i.e., how many) of weights that can be transferred from one attribute to another. For this purpose, the system maintains a set of *global constraints on the weight transitions between attributes*. These constraints constitute a *Weight Transition Graph* defined as below.

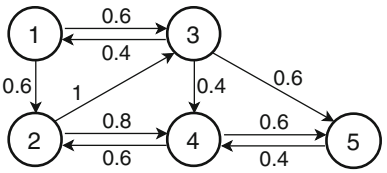


Fig. 3 A weight transition graph $G = \langle V, E \rangle$. Each node in V represents a distinct dimension in a 5-dimensional space. The capacities are shown aside each edge in E specifying the maximum amount of weights that can be transferred from its left endpoint to the right

Definition 1 *(Weight Transition Graph)* The weight transition graph is defined as a weighted directed graph $G = \langle V, E \rangle$, where:

(i) $V = [n]$ and a node $i \in V$ corresponds to the i-th dimension (attribute);
(ii) each directed edge $(i, j) \in E$ is associated with a non-negative capacity $c_{i,j}$ which specifies the maximum amount of weight that can be transferred from Dimension-i to Dimension-j *through the edge* (i, j).

□

As shown in Fig. 3, the capacity of $(4, 5)$ is 0.6 indicating that Dimension-4 can transfer weight with an amount at most 0.6 to Dimension-5 via this edge. Likewise, the capacity of $(5, 4)$ imposes a constraint that the weight transferred from Dimension-5 to Dimension-4 via the edge $(5, 4)$ is with an amount at most 0.4.

Definition 2 *(Weight Transition)* A weight transition in a weight transition graph $G = \langle V, E \rangle$ is a *value assignment* $\{e_{i,j}\}_{(i,j)\in E}$ to the edges of E such that the *capacity constraint* holds: $0 \leq e_{i,j} \leq c_{i,j}$ for all $(i, j) \in E$. □

It is worth pointing out that the directed edge (i, j) is not the only way for Dimension-i to affect Dimension-j. Instead, Dimension-i can transfer weights to Dimension-j through other dimensions. For example, in Fig. 3, in addition to the edge $(4, 5)$, Dimension-4 can transfer weights to Dimension-5 via a *path* consisting of the edges: $(4, 2)$, $(2, 3)$, $(3, 5)$.

In fact, the weight transition graphs and the weight transitions are reminiscent of the *flow networks* and *flows* in graph theory. As we will see in the next section, there is indeed a subtle connection between these concepts.

For simplicity, in the rest of this paper, when the context is clear, we may simplify the following notations:

- $G = \langle V, E \rangle$ by G,
- Dimension-i by i, and
- $\{e_{i,j}\}_{(i,j)\in E}$ by $\{e_{i,j}\}$.

Definition 3 *(Legal Weight Adjustment)* A weight adjustment Δw on w is *legal* if it satisfies:

(i) $\mathrm{w} + \Delta\mathrm{w} \in [-1, 1]^n$,
(ii) Δw can be represented by a weight transition $\{e_{i,j}\}$ in G, where:

$$\Delta w[j] = \sum_{(i,j)\in E} e_{i,j} - \sum_{(j,k)\in E} e_{j,k} \text{ for all } j \in [n].$$

$\Delta w[j]$ is the difference between the sum of all "incoming" weights transferred to j and the sum of all "outgoing" weights transferred from j in the weight transition $\{e_{i,j}\}$. □

The Rank-Reversal problem is defined as follows:

Definition 4 *(*Rank-Reversal *Problem)* Given a weight transition graph $G = \langle V, E\rangle$, a preference vector w, a pair of two items x, y satisfying $\mathrm{w}\cdot\mathrm{x} - \mathrm{w}\cdot\mathrm{y} > 0$ and a specified value of $p \in [0, \infty)$, the goal of the *rank-reversal* problem is to compute a legal weight adjustment Δw on w such that:

(i) $(\mathrm{w} + \Delta\mathrm{w})\cdot(\mathrm{x} - \mathrm{y}) \le 0$, and
(ii) $\|\Delta\mathrm{w}\|_p$ is minimized.

If such a Δw does not exist, then return NOT EXISTS. □

The *minimum dimension adjustment* (MDA) Rank-Reversal ***problem*** In order to make the system comprehensible and avoid confusing the users with an excessive number of attributes changed, a desirable goal is to minimize the total number of dimensions whose weights are changed in the adjustment. For this purpose, we set $p = 0$ in the Rank-Reversal problem, and we call it as the MDA problem for short. In other words, the aim of the MDA problem is to minimize the l_0 norm of Δw.

The MDA problem can be formulated as a minimization programming as follows:

$$\text{minimize } \|\Delta\mathrm{w}\|_0$$

subject to

$$\Delta w[j] = \sum_{(i,j)\in E} e_{i,j} - \sum_{(j,k)\in E} e_{j,k} \quad \forall j \in [n] \tag{1}$$

$$-1 \le w[i] + \Delta w[i] \le 1 \quad \forall i \in [n] \tag{2}$$

$$0 \le e_{i,j} \le c_{i,j} \quad \forall (i,j) \in E \tag{3}$$

$$\Delta\mathrm{w}\cdot(\mathrm{x} - \mathrm{y}) \le -\mathrm{w}\cdot(\mathrm{x} - \mathrm{y}) \tag{4}$$

It is easy to verify that Constraints (1)(2)(3) ensure Δw to be a legal weight adjustment, and Constraint (4) guarantees that the rank order of x and y is reversed.

In the MDA problem, the effect of preserving some of user's previous feedback is implicitly achieved by minimizing the number of dimensions changed when performing a *rank-reversal* operation. This is because from our observations in the motivated application, HomeSeeker in real estate scenario, users' preference would not change frequently in a short term, but they do over a long period. This is often due to the change of their circumstance. For example, they can afford a more expensive house thanks to the increase of their income, or they prefer a house with more bedrooms as the growth of their family. Therefore, it is important to have the flexibility of supporting such kind of long-term interest drifts, while at the same time, the system should also be able to largely retain users' recent preference. For this purpose, the MDA problem minimizes the number of dimensions changed, instead of explicitly imposing restrictive constraints in the rank reversal.

Remark It is worth mentioning that the case that a legal adjustment does not exist is not an issue, yet it is of great importance to the system. The potential causes to this can be: (i) the user is an adversary and trying to mislead the system to rank a much better item lower than the worse one; (ii) the underlying weight transition graph is too restrictive. In either case, such information can help the system to resist attacks and improve its model.

4 Preliminaries and a connection to flows

We first introduce some preliminaries about flows and its related notions, e.g., maximum flows. Then we show a subtle relation between legal weight adjustments and flows.

4.1 Flow preliminaries

Flows A *flow network* (for short, a *network*) is defined as a directed weighted graph $\mathcal{G} = \langle V \cup \{s, t\}, E\rangle$ where: (i) s and t are two distinguished nodes called the *source* and the *sink*, respectively; and (ii) each directed edge $(u, v) \in E$ is associated with a *non-negative capacity*, denoted by $c_{u,v}$. A *flow* in $\mathcal{G} = \langle V \cup \{s, t\}, E\rangle$, is a *value assignment* to the edges of E, denoted by $f = \{e_{u,v}\}_{(u,v)\in E}$. A flow f is *valid* if it satisfies two constraints:

(i) **Capacity Constraint:** The value assigned to an edge cannot exceed the capacity of the edge, i.e., $0 \le e_{u,v} \le c_{u,v}, \forall (u, v) \in E$.
(ii) **Conservation Constraint:** Except the distinguished nodes s and t, for any node $v \in V$, the sum of the values assigned to v's in-edges must be equal to the sum of the values assigned to v's out-edges, namely, $\sum_{(u,v)\in E} e_{u,v} = \sum_{(v,k)\in E} e_{v,k}, \forall v \in V$.

For a valid flow f, by the conservation constraint, equality $\sum_{(s,u)\in E} e_{s,u} - \sum_{(v,s)\in E} e_{v,s} = \sum_{(v,t)\in E} e_{v,t} - \sum_{(t,u)\in E} e_{t,u}$

always holds, and the value of the either side of this equality is defined as the *flow value of* f. In a flow network, a valid flow with the maximum value is called a *maximum flow*.

Consider two nodes $a, b \in V$, *a flow from a to b*, denoted by $f_{a\to b}$, is a flow where s can pass flow values via the edge (s, a) only, and t can receive flow values via the edge (b, t) only. That is, $f_{a\to b}$ is a value assignment $\{e_{u,v}\}_{(u,v)\in E}$ such that: (i) $e_{s,k} = 0$ for all $(s, k) \in E$ with $k \neq a$, and (ii) $e_{k,t} = 0$ for all $(k, t) \in E$ with $k \neq b$. Furthermore, we extend this notion for a node a and a set S of nodes. For a node $a \in V$ and $S \subseteq V$ with $a \notin S$, $f_{a\to S}$ is a flow from a to S, where s can pass flows via a only and t can receive flows via the nodes in S only. Analogously, $f_{S\to a}$ is a flow from S to a.

Consider two valid flows $f = \{e_{u,v}\}_{(u,v)\in E}$ and $f' = \{e'_{u,v}\}_{(u,v)\in E}$ in $\mathcal{G}$; the flow f' is a *sub-flow* of f if the value assignment obtained by subtracting f' from f, computed as $\{e_{u,v} - e'_{u,v}\}_{(u,v)\in E}$ and denoted by $f - f'$, is a valid flow. Moreover, if a sub-flow f' of f is also a flow from a to b, then we say f' is a sub-flow from a to b in f, and $f - f'$ is the resulted flow by removing the sub-flow f' from a to b in f. Again, the notion of sub-flow also applies to $f_{a\to S}$ and $f_{S\to a}$. Finally, in addition to the capacity, if each edge $(i, j) \in E$ in a flow network $\mathcal{G} = \langle V \cup \{s, t\}, E\rangle$, is also associated with a *cost* $\alpha_{i,j}$. The *cost* of a valid flow $f = \{e_{i,j}\}_{(i,j)\in E}$, denoted by cost$(f)$ is computed as: cost$(f) = \sum_{(i,j)\in E} \alpha_{i,j} \cdot e_{i,j}$.

4.2 A connection between MDA and flows

Weight transitions in G (and hence, legal weight adjustments) are reminiscent of the notion of flows. Our crucial observation is that, legal weight adjustments Δw on w are essentially equivalent to flows in the graph (flow network) defined as follows.

Definition 5 *(Extended Weight Transition Graph)* Given a weight transition graph G, a preference vector w and two sets of dimensions $S, T \subseteq [n]$, the extended weight transition graph is defined as a flow network, denoted by $G_{\text{ext}}(S, T, \text{w}) = \langle V \cup \{s, t\}, E_{\text{ext}}\rangle$, constructed by adding to G:

- a source node s and a sink node t,
- for each $i \in S$, a directed edge (s, i) with capacity of $1 + w[i]$,
- for each $i \in T$, a directed edge (i, t) with capacity of $1 - w[i]$. □

See Fig. 4 for an example, where the capacities of $(s, 1)$ and $(s, 2)$ are $1+w[1] = 1.6$ and $1+w[2] = 1.2$ respectively, and the capacities of $(3, t)$ and $(5, t)$ are $1 - w[3] = 1.6$ and $1 - w[5] = 0.6$ respectively.

Algorithm 1: Construction from Δw to a flow

Input: A weight transition graph $G = \langle V, E\rangle$ and a legal weight adjustment Δw on w

Output: A flow $flow(\Delta\text{w})$ on $G_{\text{ext}}(S(\Delta\text{w}), T(\Delta\text{w}), \text{w})$

1 Let $\{e_{i,j}\}_{(i,j)\in E}$ be the weight transition representing Δw. By Definition 3, such a weight transition must exist;
2 For each $j \in S(\Delta\text{w})$, set $e_{s,j} = -\Delta\text{w}[j]$;
3 For each $j \in T(\Delta\text{w})$, set $e_{j,t} = \Delta\text{w}[j]$;
4 Return $flow(\Delta\text{w}) = \{e_{i,j}\}_{(i,j)\in E_{\text{ext}}} = \{e_{i,j}\}_{(i,j)\in E} \cup \{e_{s,j}\}_{j\in S(\Delta\text{w})} \cup \{e_{j,t}\}_{j\in T(\Delta\text{w})}$;

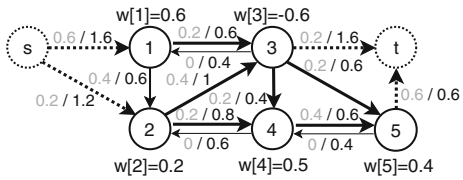


Fig. 4 An extended weight transition graph $G_{\text{ext}}(S, T, \text{w})$ of G with respect to w $= (0.6, 0.2, -0.6, 0.5, 0.4)$ with $S = \{1, 2\}$ and $T = \{3, 5\}$. In addition to the capacity, a flow value through each edge is shown in grey and in the form of "flow value / capacity". The flow in the figure represents a legal weight adjustment Δw $= (-0.6, -0.2, 0.2, 0, 0.6)$ on w

For a legal weight adjustment Δw, we define $S(\Delta\text{w}) = \{i \mid \Delta w[i] < 0\}$ as the set of dimensions whose weights have been decreased by Δw, and $T(\Delta\text{w}) = \{i \mid \Delta w[i] > 0\}$ as the set of dimensions whose weights have been increased by Δw.

Observation 1 *The flow, $flow(\Delta w)$, constructed by Algorithm 1 is valid on $G_{\text{ext}}(S(\Delta w), T(\Delta w), w)$.*

Proof It suffices to show that $flow(\Delta\text{w})$ satisfies both the Capacity Constraint and the Conservation Constraint.

Capacity Constraint. First, since Δw is legal, by Definition 3, $0 \le e_{i,j} \le c_{i,j}$ for all $(i, j) \in E$. Second, for $j \in S(\Delta\text{w})$, we have: (i) $\Delta\text{w}[j] < 0$, and (ii) $-1 \le w[j] + \Delta\text{w}[j] \le 1$ because Δw is legal. Thus, $0 \le e_{s,j} = -\Delta\text{w}[j] \le 1 + w[j] = c_{s,j}$ satisfying the Capacity Constraint. Analogously, we can show that $0 \le e_{j,t} \le c_{j,t}$ for all $j \in T(\Delta\text{w})$. Therefore, $flow(\Delta\text{w}) = \{e_{i,j}\}_{(i,j)\in E_{\text{ext}}}$ satisfies the Capacity Constraint for all $(i, j) \in E_{\text{ext}}$.

Conservation Constraint. For $j \notin S(\Delta\text{w}) \cup T(\Delta\text{w})$, we have $\Delta\text{w}[j] = 0$. Furthermore, by Definition 3 and since j has no edge with either s or t, $\Delta\text{w}[j] = \sum_{(i,j)\in E} e_{i,j} - \sum_{(j,k)\in E} e_{j,k} = \sum_{(i,j)\in E_{\text{ext}}} e_{i,j} - \sum_{(j,k)\in E_{\text{ext}}} e_{j,k} = 0$. Thus, the Conservation Constraint is satisfied for all the nodes j in this case.

For $j \in S(\Delta\text{w})$, in $\{e_{i,j}\}_{(i,j)\in E_{\text{ext}}}$, we have: $\sum_{(i,j)\in E_{\text{ext}}} e_{i,j} - \sum_{(j,k)\in E_{\text{ext}}} e_{j,k} = (\sum_{(i,j)\in E} e_{i,j} - \sum_{(j,k)\in E} e_{j,k}) + e_{s,j} = \Delta\text{w}[j] - \Delta\text{w}[j] = 0$. Therefore, the Conservation Constraint is met for all $j \in S(\Delta\text{w})$. Again, by symmetry, we can show that for all $j \in T(\Delta\text{w})$, j satisfies the Conservation Constraint.

Algorithm 2: Construction from a flow to Δw

Input: A valid flow $f = \{e_{i,j}\}_{(i,j)\in E_{\text{ext}}}$ on $G_{\text{ext}}(S, T, \text{w})$
Output: A weight adjustment Δw with $S(\Delta\text{w}) \subseteq S$ and $T(\Delta\text{w}) \subseteq T$

1 For each $i \in S$, set $\Delta\text{w}[i] = -e_{s,i}$;
2 For each $i \in T$, set $\Delta\text{w}[i] = e_{i,t}$;
3 For each $i \notin S \cup T$, set $\Delta w[i] = 0$;
4 Return Δw;

Putting everything together, $flow(\Delta\text{w})$ is valid on $G_{\text{ext}}(S(\Delta\text{w}), T(\Delta\text{w}), \text{w})$. □

Observation 2 *The weight adjustment Δw constructed by Algorithm 2 is legal on G with respect to w, and satisfies $S(\Delta w) \subseteq S$ and $T(\Delta w) \subseteq T$.*

Proof Since $f = \{e_{i,j}\}_{(i,j)\in E_{\text{ext}}}$ is a valid flow on $G_{\text{ext}}(S(\Delta\text{w}), T(\Delta\text{w}), \text{w})$, $e_{i,j} \geq 0$ for all $(i, j) \in E_{\text{ext}}$. Thus, $S(\Delta\text{w}) \subseteq S$ and $T(\Delta\text{w}) \subseteq T$ hold. Furthermore, as f satisfies the Capacity Constraint, the assignment $\{e_{i,j}\}_{(i,j)\in E}$ is a weight transition in G. Moreover, since $0 \leq e_{s,i} = -\Delta\text{w}[i] \leq c_{s,i} = 1 + w[i]$ for $i \in S$, we have $-1 \leq w[i] + \Delta\text{w}[i] \leq 1$. Similarly, $-1 \leq w[i] + \Delta\text{w}[i] \leq 1$ for all $i \in T$. Therefore, $\text{w} + \Delta\text{w} \in [-1, 1]^n$. Finally, as f satisfies the Conservation Constraint, $\Delta\text{w}[j] = \sum_{(i,j)\in E} e_{i,j} - \sum_{(j,k)\in E} e_{j,k}$ for all $j \in [n]$. Thus by Definition 3, Δw is a legal weight adjustment. □

From Observations 1 and 2, we know that a legal weight Δw and a valid flow in $G_{\text{ext}}(S(\Delta\text{w}), T(\Delta\text{w}), \text{w})$ are essentially convertible from each other.

For example, in Fig. 4, the weight adjustment $\Delta\text{w} = (-0.6, -0.2, 0.2, 0, 0.6)$ on $\text{w} = (0.6, 0.2, -0.6, 0.5, 0.4)$ is legal. This is because: (i) $\text{w} + \Delta\text{w} = (0, 0, -0.4, 0.5, 1)$ is a preference vector, i.e., in $[-1, 1]^n$, and (ii) it can be interpreted as a weight transition in G by the flow values in grey assigned to the edges of E. With $S(\Delta\text{w}) = \{1, 2\}$ and $T(\Delta\text{w}) = \{3, 5\}$, Δw can be further interpreted as a flow in $G_{\text{ext}}(S(\Delta\text{w}), T(\Delta\text{w}), \text{w})$ (as constructed in the proof of Observation 1), which is the value assignment to the edges of E_{ext} as shown in grey in Fig. 4.

Our next lemma further reveals a subtle connection between the MDA problem and the cost of flows.

Definition 6 *(Cost Associated $G_{\text{ext}}(S, T, \text{w})$)* Given $G_{\text{ext}}(S, T, \text{w})$ and two items x, y, the cost associated extended weight transition graph, denoted by $G^{\text{cost}}_{\text{ext}}(S, T, \text{w})$ (with respect to x and y), is constructed by associating a cost to each edge by the following rules:

- $\alpha_{s,i} = y[i] - x[i]$ for all $i \in S$,
- $\alpha_{i,t} = x[i] - y[i]$ for all $i \in T$,
- $\alpha_{i,j} = 0$ for all other edges of E_{ext}. □

Lemma 1 *Consider a valid flow $f = \{e_{i,j}\}_{(i,j)\in E_{\text{ext}}}$ in $G^{\text{cost}}_{\text{ext}}(S, T, w)$ and the legal weight adjustment Δw constructed from f by Algorithm 2. Then we have:*

$$cost(f) = \Delta w \cdot (x - y).$$

Proof

$$\begin{aligned}
\text{cost}(f) &= \sum_{(i,j)\in E_{\text{ext}}} \alpha_{i,j} \cdot e_{i,j} \\
&= \sum_{i\in S(\Delta\text{w})} \alpha_{s,i} \cdot e_{s,i} + \sum_{i\in T(\Delta\text{w})} \alpha_{i,t} \cdot e_{i,t} \\
&= \sum_{i\in S(\Delta\text{w})} (y[i] - x[i]) \cdot (-\Delta w[i]) \\
&\quad + \sum_{i\in T(\Delta\text{w})} (x[i] - y[i]) \cdot \Delta w[i] \\
&= \Delta\text{w} \cdot (\text{x} - \text{y}). \qquad (5)
\end{aligned}$$

The second equality follows from the fact that $\alpha_{i,j} = 0$ for all edges neither $s = i$ nor $t = j$ by the cost assignment in Definition 6. The third equality holds by the cost assignment and the construction of Δw from f by Algorithm 2. The last equality is established by the fact that $\Delta\text{w}[i] = 0$ for all $i \notin S(\Delta\text{w}) \cup T(\Delta\text{w})$. Therefore, the lemma follows. □

Remark The connection between the MDA problem and the cost of valid flows on $G^{\text{cost}}_{\text{ext}}(S, T, \text{w})$ is based on an assumption that both S and T are provided; they respectively specify the dimensions that *will* be decreased and increased when computing a legal weight adjustment Δw. Specifically, if one can specify S and T such that: (i) a feasible legal Δw exists for the MDA problem (w.r.t., w, x and y) conditioned on S and T, i.e., only the dimensions in S (resp., T) are decreased (resp., increased); and (ii) $|S| + |T|$ is minimized, then one can construct an optimal Δw by finding a valid flow with minimum cost on $G^{\text{cost}}_{\text{ext}}(S, T, \text{w})$. Interestingly, as we show in the next section, finding such "optimal" S and T actually makes the problem NP-hard.

5 NP-hardness

In this section, we give a proof sketch to show the NP-hardness of the MDA problem. For this purpose, we define the following problems.

Definition 7 *(MC Decision)* Given a collection of m sets $\mathcal{S} = \{S_1, \ldots, S_m\}$ with a universe $U = \cup_{S_i\in\mathcal{S}} S_i$, two integers $k \leq m$ and $\tau \leq |U|$, the MC decision problem is to determine whether there exists a collection $\mathcal{S}' \subseteq \mathcal{S}$ such that $|\mathcal{S}'| \leq k$ and $|\cup_{S_i\in\mathcal{S}'} S_i| \geq \tau$. □

It is known that the MC decision problem is NP-complete.

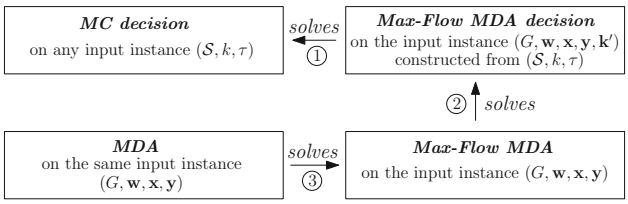


Fig. 5 A roadmap for the proof of Theorem 1. As the MC decision problem is known to be NP-complete, Arrow (1) implies the Max-Flow MDA problem is NP-complete as well. Arrow (2) indicates that the Max-Flow MDA problem is NP-hard. Finally, the NP-hardness of the MDA problem follows from Arrow (3)

Consider two n-dimensional vectors x and y. Define the set of *x-dominant dimensions* (w.r.t., y) as $xDom = \{i \mid x[i] > y[i], \forall i \in [n]\}$; and similarly, the set of *y-dominant dimensions* (w.r.t., x) as $yDom = \{i \mid y[i] > x[i], \forall i \in [n]\}$. Moreover, define the set of *equal dimensions* between x and y as $eqDom = [n] \setminus (xDom \cup yDom)$.

Definition 8 *(Max-Flow MDA)* Given a weight transition graph G, a preference vector w and two items x, y, the goal of the Max-Flow MDA problem is to find two sets $S \subseteq xDom$ and $T \subseteq yDom$ such that: (i) the maximum flow value in the extended weight transition graph $G_{\text{ext}}(S, T, \text{w})$ of G is $\geq 1/2 \cdot \text{w} \cdot (\text{x} - \text{y})$ and (ii) $|S| + |T|$ is minimized . If such a solution (S, T) does not exist, return NOT EXISTS. □

Definition 9 *(Max-Flow MDA Decision)* The Max-Flow MDA decision problem is to decide whether there exists a feasible solution (S, T) to the Max-Flow MDA problem such that $|S| + |T| \leq k'$ for an additional input integer k'. □

Theorem 1 *The MDA problem is NP-hard.*

Proof (Sketch) Below we give a proof sketch for Theorem 1, for which the detailed proof can be found in the supplemental materials. A roadmap of our proof is as shown in Fig. 5. Since Arrow (2) in Fig. 5 is obvious, our proof mainly focuses on proving Arrow (1) and Arrow (3), and hence, there are two main steps:

(i) Showing arrow (1) Given an input instance $(\mathcal{S}, k, \tau)$ to the MC decision problem, we construct a *specific input instance* $(G, \text{w}, \text{x}, \text{y}, k')$ to the Max-Flow MDA decision problem. We show that the answer to the Max-Flow MDA decision problem on $(G, \text{w}, \text{x}, \text{y}, k')$ is YES *if and only if* the answer to MC decision problem on $(\mathcal{S}, k, \tau)$ is YES. [Shown in Lemma 1 in the Supplementary Material]

(ii) Showing arrow (3) Consider the input instance $(G, \text{w}, \text{x}, \text{y}, k')$ to the Max-Flow MDA decision problem constructed in Step (i). Clearly, such a decision problem can be answered by the Max-Flow MDA problem. Let $(G, \text{w}, \text{x}, \text{y})$ be the corresponding input instance (ignoring k') to the Max-Flow MDA problem. On this specific input instance $(G, \text{w}, \text{x}, \text{y})$, an optimal Max-Flow MDA solution can be constructed in polynomial time by any optimal solution to the MDA problem on the same input instance $(G, \text{w}, \text{x}, \text{y})$. [Shown in Lemmas 2 and 3 in the Supplementary Materil]

Step (i) implies that the Max-Flow MDA decision problem is NP-complete and hence, the Max-Flow MDA problem is NP-hard. Step (ii) shows that the MDA problem is "no easier" than the Max-Flow MDA problem, because the former can answer the hard instance (i.e., the input instance that makes the problem NP-hard) of the latter. Therefore, the NP-hardness of the MDA problem follows. □

6 A 2.17-approximate algorithm

Theorem 1 eliminates the hope of seeking a polynomial-time algorithm for solving the MDA problem. Even worse, as the objective function in the MDA problem is discontinuous (and hence, not differentiable), most existing methods for solving optimization problems such as the Gradient Descent [3] and the Lagrange Multiplier Method [4] are not applicable to obtain "local optima". To remedy this, in this section, we design a new problem called the F-problem, in which the objective function is continuous, differentiable and monotone. Moreover, we show that, under a mild assumption, with an optimal solution to the F-problem, we can construct a 2.17-approximate solution to the MDA problem in polynomial time. Although solving the F-problem is still NP-hard due to the concavity of its objective [27], existing optimization techniques are sufficiently efficient to obtain a "fairly good" result.

6.1 A mild assumption

Assumption 1 (Discretizable Assumption). There exists a sufficiently large fixed value $M \geq 1$, whose value is known a priori and by multiplying which all the coordinate values in w, x, y as well as the edge capacities in G can be scaled to integers.

It should be noted that Assumption 1 holds in most of the applications in practice. For one thing, the values in an input instance $(G, \text{w}, \text{x}, \text{y})$ are often represented in the form of rational numbers. For the other thing, the input numbers usually have limited precision. In either case, the input values can be scaled to integers by a proper fixed value M. Moreover, Assumption 1 is equivalent to assuming that the coordinates in w, x, y as well as the edge capacities in G are integer multiples of $\frac{1}{M}$. Finally, a real value is said to be $\frac{1}{M}$*-multiple* if and only if it is an integer multiple of $\frac{1}{M}$.

Remark The MDA problem is still NP-hard under Assumption 1, as our proofs apply to this case as well.

6.2 The F-problem

Our basic idea is to define another objective that can closely approximate the behaviours of $\|\Delta \mathrm{w}\|_0$, while it is continuous, differentiable and monotone.

Consider a $\frac{1}{M}$-multiple input instance $(G, \mathrm{w}, \mathrm{x}, \mathrm{y})$; let $L = \max_{i\in[n]} |x[i] - y[i]|$ be the maximum absolute difference between the coordinates of x and y on the same dimension. We define the *minimum quantity unit* as: $\delta = \min\{\frac{1}{2n\cdot M^2\cdot L}, \frac{1}{M^2}\}$. Furthermore, we define a function $z(u)$ on $u \in (-\infty, \infty)$, $z(u) = (\frac{u}{\delta})^2$, and a function $g(z(u))$ on $z(u) \in [0, \infty)$, $g(z(u)) = \gamma \cdot \frac{1-e^{-z(u)}}{1+e^{-z(u)}}$, where $\gamma = \frac{1+e^{-1}}{1-e^{-1}}$.

Observation 3 *The function $g(z(u))$ is monotonically increasing with $z(u) \in [0, \infty)$ and has the following characteristics:*

- $g(0) = 0$ *and* $g(1) = 1$*;*
- $g(z(u)) \geq 1$*, for all* $z(u) \geq 1$*, that is,* $|u| \geq \delta$*;*
- $g(z(u)) \leq \gamma$*, for all* $z(u) \in [0, \infty)$.

Proof All these characteristics can be verified easily. □

Finally, for an n-dimensional vector $\mathrm{u} \in \mathbb{R}^n$, define

$$F(\mathrm{u}) = \sum_{i=1}^{n} g(z(u[i])).$$

Definition 10 *(The F-problem)* Consider the same input instance $(G, \mathrm{w}, \mathrm{x}, \mathrm{y})$ to the MDA problem, the F-problem aims to find a legal weight adjustment $\Delta\mathrm{w}$ minimizing $F(\Delta\mathrm{w})$ subject to *exactly the same* constraints in the MDA.

Return NOT EXISTS if a feasible $\Delta\mathrm{w}$ does not exist. □

Hardness of the F-problem The F-problem aims to minimize a concave function $F(\Delta\mathrm{w})$ over a set of convex constraints, equivalently, to maximize a convex objective. Unfortunately, this optimization falls into the class of problems called *Sigmoidal Programming* which is known to be not only NP-hard, but also NP-hard to approximate to *an arbitrary precision* [27]. In other words, in general, there is no constant approximate algorithm for the Sigmoidal Programming unless $NP = P$. Nonetheless, as shown in [27], there exists a polynomial algorithm for Sigmoidal Programming with an *additive error* depending only on the number of *active constraints* and the *non-convexity* of the sigmoid functions in the objective. As the discussion on this result is out of the scope of this paper, interested readers may refer to [27] for more details.

Moreover, local optima to the F-problem can be efficiently obtained by various existing optimization techniques such as the Gradient Descent [3] and the Lagrange Multiplier Method [4] implemented in a number of popular publicly available frameworks like Tensorflow [1] and PyTorch [7], which have significantly reduced the effort required to tackle the problem.

Fact 1 [27] *The F-problem is NP-hard.*

6.3 The approximate algorithm

An overview of the algorithm Recall that, as mentioned in Sect. 4.2, deciding optimally which dimensions should have their weight increased and which should have their weight decreased makes the MDA problem NP-hard. To circumvent this issue, we seek guidance from the optimal solutions to the F-problem on the same input instance $(G, \mathrm{w}, \mathrm{x}, \mathrm{y})$. In particular, given an optimal solution $\Delta\mathrm{w}$ to the F-problem, we first "remove" the adjustments on the dimensions whose absolute values are less than the minimum quantity unit δ, i.e., those Dimension-i with $|\Delta\mathrm{w}[i]| < \delta$. The rationale here is that the changes on those dimensions are small enough to be discarded, without which we should still be able to find a feasible solution. As such, the next step in our algorithm is to construct a specific feasible solution to the MDA problem by only adjusting the dimensions with $|\Delta\mathrm{w}[i]| \geq \delta$.

We extend the notions of $S(\Delta\mathrm{w})$ and $T(\Delta\mathrm{w})$, and define $S_\delta(\Delta\mathrm{w}) = \{i \mid i \in [n] \wedge \Delta w[i] \leq -\delta\}$ and $T_\delta(\Delta\mathrm{w}) = \{i \mid i \in [n] \wedge \Delta w[i] \geq \delta\}$. As a result, $|\Delta\mathrm{w}[i]| \geq \delta$ for all $i \in S(\Delta\mathrm{w}) \cup T(\Delta\mathrm{w})$. The crucial idea in our approximate algorithm is to construct a specific feasible solution $\Delta\mathrm{w}^*$ to the MDA problem such that $S(\Delta\mathrm{w}^*) \subseteq S_\delta(\Delta\mathrm{w})$ and $T(\Delta\mathrm{w}^*) \subseteq T_\delta(\Delta\mathrm{w})$.

The algorithm Our 2.17-approximate algorithm is as simple as follows:

- **Step (1).** Solve the F-problem. If the F-problem is not feasible, return NOT EXISTS. Otherwise, let $\Delta\mathrm{w}$ be an optimal solution.
  - If $|\Delta\mathrm{w}[i]| \geq \delta$ for all $\Delta\mathrm{w}[i] \neq 0$, then return $\Delta\mathrm{w}^* = \Delta\mathrm{w}$ as a 2.17-approximate solution to the MDA problem.
  - Otherwise, go to Step (2).
- **Step (2).** Construct $G^{\mathrm{cost}}_{\mathrm{ext}}(S_\delta(\Delta\mathrm{w}), T_\delta(\Delta\mathrm{w}), \mathrm{w})$ according to the cost assignment rules in Definition 6.
- **Step (3).** Compute a valid flow $f^* = \{e^*_{i,j}\}_{(i,j)\in E_{\mathrm{ext}}}$ on $G^{\mathrm{cost}}_{\mathrm{ext}}(S_\delta(\Delta\mathrm{w}), T_\delta(\Delta\mathrm{w}), \mathrm{w})$ with $\mathrm{cost}(f^*)$ minimized.
- **Step (4).** Construct the weight adjustment $\Delta\mathrm{w}^*$ from f^* by Algorithm 2.
- **Step (5).** Return $\Delta\mathrm{w}^*$ as a 2.17-approximate solution to the MDA problem.

Correctness The correctness of the above algorithm follows from the three claims below:

- **Claim (1):** The F-problem is infeasible if and only if the MDA problem returns NOT EXISTS.
- **Claim (2):** $\Delta \mathrm{w}^*$ is a feasible solution to the MDA problem.
- **Claim (3):** $\|\Delta \mathrm{w}^*\|_0 \leq 2.17 \cdot \|\Delta \mathrm{w}_{opt}\|_0$, where $\Delta \mathrm{w}_{opt}$ is an optimal MDA solution.

Proof of Claim (1) It follows immediately from the fact that the F-problem shares exactly the same constraints with the MDA problem.

Proof of Claim (2) There can be only two possibilities that the approximate algorithm returns a solution $\Delta \mathrm{w}^*$. The first happens at Step (1), and the second occurs at Step (5).

For the first case, as $\Delta \mathrm{w}^*$ is a feasible solution to the F-problem, it is also a feasible solution to the MDA problem because the problems share exactly the same constraints.

For the second case, as $\Delta \mathrm{w}^*$ is constructed from a valid flow $f^* = \{e^*_{i,j}\}_{(i,j)\in E_{\text{ext}}}$ in $G^{\text{cost}}_{\text{ext}}(S_\delta(\Delta \mathrm{w}), T_\delta(\Delta \mathrm{w}), \mathrm{w})$ by Algorithm 2, according to Observation 2, $\Delta \mathrm{w}^*$ is a legal weight adjustment on w, and hence, it satisfies Constraints (1) (2) and (3). Next, we show that $\Delta \mathrm{w}^*$ also satisfies Constraint (4), namely, $\Delta \mathrm{w}^* \cdot (\mathrm{x} - \mathrm{y}) \leq -\mathrm{w} \cdot (\mathrm{x} - \mathrm{y})$. Furthermore, since by Lemma 1, we know that $\text{cost}(f^*) = \Delta \mathrm{w}^* \cdot (\mathrm{x} - \mathrm{y})$, it thus suffices to show:

Lemma 2 $\text{cost}(f^*) \leq -w \cdot (x - y)$.

Proof In the following, we construct a valid flow f on $G^{\text{cost}}_{\text{ext}}(S_\delta(\Delta \mathrm{w}), T_\delta(\Delta \mathrm{w}), \mathrm{w})$ such that $\text{cost}(f) \leq -\mathrm{w} \cdot (\mathrm{x} - \mathrm{y})$. By the fact that $\text{cost}(f^*)$ is minimized, we thus have $\text{cost}(f^*) \leq \text{cost}(f)$ and the lemma follows.

There are three main steps to construct such a valid flow f on $G^{\text{cost}}_{\text{ext}}(S_\delta(\Delta \mathrm{w}), T_\delta(\Delta \mathrm{w}), \mathrm{w})$:

- Step (i). Construct a valid flow $f(\Delta \mathrm{w}) = \{e_{i,j}\}_{(i,j)\in E_{\text{ext}}}$ on $G_{\text{ext}}(S(\Delta \mathrm{w}), T(\Delta \mathrm{w}), \mathrm{w})$ from $\Delta \mathrm{w}$ by Algorithm 1, where $\Delta \mathrm{w}$ is the optimal solution to the F-problem obtained in Step (1) in the 2.17-approximate algorithm.
- Step (ii). Utilise $f(\Delta \mathrm{w})$ to construct a valid flow f' on $G^{\text{cost}}_{\text{ext}}(S_\delta(\Delta \mathrm{w}), T_\delta(\Delta \mathrm{w}), \mathrm{w})$. In particular, define $S_{rm} = S(\Delta \mathrm{w}) \setminus S_\delta(\Delta \mathrm{w})$, that is, S_{rm} is the set of all the nodes i such that $\Delta \mathrm{w}[i] < 0$ and $\Delta \mathrm{w}[i] > -\delta$. Likewise, $T_{rm} = T(\Delta \mathrm{w}) \setminus T_\delta(\Delta \mathrm{w})$ is the set of all the nodes i such that $\Delta \mathrm{w}[i] > 0$ and $\Delta \mathrm{w}[i] < \delta$. Then $f' = \{e'_{i,j}\}_{(i,j)\in E_{\text{ext}}}$ is obtained by subtracting all the valid sub-flows passing through the nodes in $S_{rm} \cup T_{rm}$ from $f(\Delta \mathrm{w})$. Since f' is still a valid flow in $G_{\text{ext}}(S(\Delta \mathrm{w}), T(\Delta \mathrm{w}), \mathrm{w})$ and it has no flows passing through the nodes in $S_{rm} \cup T_{rm}$, f' is also valid in $G^{\text{cost}}_{\text{ext}}(S_\delta(\Delta \mathrm{w}), T_\delta(\Delta \mathrm{w}), \mathrm{w})$.
- Step (iii). Round the flow f' to a flow $f = \{e_{i,j}\}_{(i,j)\in E_{\text{ext}}}$ such that: (a) f is a valid flow in $G^{\text{cost}}_{\text{ext}}(S_\delta(\Delta \mathrm{w}), T_\delta(\Delta \mathrm{w}), \mathrm{w})$, (b) all the values $e_{i,j}$ in f are multiples of $\frac{1}{M}$, and (c) $\text{cost}(f) \leq \text{cost}(f')$. As shown in [17], under Assumption 1, such a flow f must exist.

Next we claim that $\text{cost}(f') < -\mathrm{w} \cdot (\mathrm{x} - \mathrm{y}) + \frac{1}{M^2}$. Before we prove this argument, suppose that it is true. By Property (c) of the flow f obtained in Step (iii), we have $\text{cost}(f) \leq \text{cost}(f') < -\mathrm{w} \cdot (\mathrm{x} - \mathrm{y}) + \frac{1}{M^2}$, equivalently, $\text{cost}(f) + \mathrm{w} \cdot (\mathrm{x} - \mathrm{y}) < \frac{1}{M^2}$. Moreover, by Assumption 1 and Property (b) of f, $\text{cost}(f) + \mathrm{w} \cdot (\mathrm{x} - \mathrm{y})$ is a multiple of $\frac{1}{M}$. Therefore, $M^2 \cdot (\text{cost}(f) + \mathrm{w} \cdot (\mathrm{x} - \mathrm{y}))$ is an integer. As a result, $M^2 \cdot (\text{cost}(f) + \mathrm{w} \cdot (\mathrm{x} - \mathrm{y})) < 1$ is equivalent to $M^2 \cdot (\text{cost}(f) + \mathrm{w} \cdot (\mathrm{x} - \mathrm{y})) \leq 0$. Therefore, $\text{cost}(f) \leq \mathrm{w} \cdot (\mathrm{x} - \mathrm{y})$ and the lemma follows.

Now we prove the last missing piece: $\text{cost}(f') < -\mathrm{w} \cdot (\mathrm{x} - \mathrm{y}) + \frac{1}{M^2}$, and then complete the whole proof of Lemma 2.

Recall that f' is obtained by subtracting the sub-flows passing through all the nodes in $S_{rm} \cup T_{rm}$ from $f(\Delta \mathrm{w})$. During the subtraction, observe that for each time, the sub-flow $f_{i \to T(\Delta \mathrm{w})}$ passing through a node $i \in S_{rm}$ is removed, that is, $e_{s,i}$ becomes 0, $\Delta \mathrm{w}[i]$ is increased by less than δ. At the same time, some of $e_{j,t}$ for $j \in T(\Delta \mathrm{w})$ are decreased, but the sum of them is also less than δ, because $f_{i \to T(\Delta \mathrm{w})}$ is valid and hence satisfies the Conservation Constraint. Therefore, removing $f_{i \to T(\Delta \mathrm{w})}$ can increase the value of $\Delta \mathrm{w} \cdot (\mathrm{x} - \mathrm{y})$ by less than $2 \cdot \delta \cdot L$, where $L = \max_{i \in [n]} |x[i] - y[i]|$. By an analogous argument, removing a sub-flow $f_{S(\Delta \mathrm{w}) \to j}$ for a node $j \in T(\Delta \mathrm{w})$ can increase the value of $\Delta \mathrm{w} \cdot (\mathrm{x} - \mathrm{y})$ by less than $2 \cdot \delta \cdot L$.

Furthermore, since $|S_{rm} \cup T_{rm}| \leq n$, there can be at most n such sub-flows removed. Thus, the total increment on the value of $\Delta \mathrm{w} \cdot (\mathrm{x} - \mathrm{y})$ for obtaining the flow f' is less than $2 \cdot n \cdot \delta \cdot L$. In other words, $\Delta \mathrm{w}' \cdot (\mathrm{x} - \mathrm{y}) - \Delta \mathrm{w} \cdot (\mathrm{x} - \mathrm{y}) < 2 \cdot n \cdot \delta \cdot L \leq \frac{1}{M^2}$, where $\Delta \mathrm{w}'$ is the weight adjustment constructed from f' by Algorithm 2. Therefore, by Lemma 1, $\text{cost}(f') = \Delta \mathrm{w}' \cdot (\mathrm{x} - \mathrm{y}) < -\mathrm{w} \cdot (\mathrm{x} - \mathrm{y}) + \frac{1}{M^2}$, and the proof is thus complete. □

Proof of Claim (3) Define a function $h(v)$ on $v \in \mathbb{R}$ such that $h(v) = 0$ for $v = 0$ and $h(v) = 1$ for $v \neq 0$. Note that if $\Delta \mathrm{w}^*$ is returned at Step (1) in the algorithm, then $\Delta \mathrm{w}^* = \Delta \mathrm{w}$ and $\|\Delta \mathrm{w}^*\|_0 = |S_\delta(\Delta \mathrm{w}) \cup T_\delta(\Delta \mathrm{w})|$. Otherwise, $\Delta \mathrm{w}^*$ is constructed from the flow f^* in $G^{\text{cost}}_{\text{ext}}(S_\delta(\Delta \mathrm{w}), T_\delta(\Delta \mathrm{w}), \mathrm{w})$, and thus, only the dimensions in $S_\delta(\Delta \mathrm{w}) \cup T_\delta(\Delta \mathrm{w})$ of $\Delta \mathrm{w}^*$ can have nonzero values. Therefore, in either case, we have:

$$\|\Delta \mathrm{w}^*\|_0 \leq |S_\delta(\Delta \mathrm{w}) \cup T_\delta(\Delta \mathrm{w})|. \tag{6}$$

By the definition of $S_\delta(\Delta \mathrm{w})$ and $T_\delta(\Delta \mathrm{w})$, we know that $|\Delta w[i]| \geq \delta$ for all $i \in S_\delta(\Delta \mathrm{w}) \cup T_\delta(\Delta \mathrm{w})$. Thus, by the second bullet in Observation 3,

$$|S_\delta(\Delta \mathrm{w}) \cup T_\delta(\Delta \mathrm{w})| \leq \sum_{i=1}^{n} F(\Delta w[i]). \tag{7}$$

Furthermore, since $\Delta \mathrm{w}_{\mathrm{opt}}$ is also a feasible solution to the F-problem, by the optimality of $\Delta \mathrm{w}$ (with respect to the F-problem) and the third bullet in Observation 3, we have:

$$\sum_{i=1}^{n} F(\Delta w[i]) \leq \sum_{i=1}^{n} F(\Delta w_{\mathrm{opt}}[i]) \leq \gamma \cdot \sum_{i=1}^{n} h(\Delta w_{\mathrm{opt}}[i]). \tag{8}$$

Combining Inequalities (6) (7) and (8), we have:

$$\|\Delta \mathrm{w}^*\|_0 \leq \gamma \cdot \sum_{i=1}^{n} h(w_{\mathrm{opt}}[i]) = \gamma \cdot \|\Delta \mathrm{w}_{opt}\|_0.$$

The lemma follows from the fact that $\gamma = \frac{1+e^{-1}}{1-e^{-1}} < 2.17$. □

Therefore, by Claims (1), (2) and (3), we have:

Theorem 2 (Correctness) *Our algorithm is a correct* 2.17*-approximate algorithm for solving the MDA problem.*

Here, it is worth noting that a crucial step (i.e., Step 1) in our 2.17-approximate algorithm is to compute an *optimal* solution to the F-problem. Unfortunately, as aforementioned, the F-problem itself is NP-hard. Except for Step 1, each step in our algorithm can be performed in polynomial time. We thus have the last theorem as below.

Theorem 3 (Efficiency) *Given an optimal solution to the F-problem, we can obtain a* 2.17*-approximate solution to the MDA problem in polynomial time.*

7 Efficient heuristic algorithms

Though there are plenty of handy tools to find local optima for the F-problem, those optimization techniques are efficient in the sense of polynomial time versus NP-hardness. However, they may not be efficient enough when aiming to support online queries in practice. Motivated by this, in this section, we proposed two simple, efficient, yet effective heuristic algorithms for solving the MDA problem. The first algorithm is called *Iterative Linear Programming* (ItrLP), and the second one is a variant of the known *Successive Shortest Path* (SSP) algorithm [9], called *Iterative Successive Shortest Path* (ItrSSP) algorithm.

7.1 Iterative linear programming algorithm

Linear programming (LP) Perhaps one of the most straightforward heuristic algorithms is to relax the optimization problem from minimizing $\|\Delta \mathrm{w}\|_0$ to minimizing $\|\Delta \mathrm{w}\|_1$ subject to exactly the same constraints. Specifically, as a heuristic, we can consider:

$$\text{minimize} \quad \sum_{i=1}^{n} |\Delta \mathrm{w}[i]|$$

subject to Constraints (1) (2) (3) and (4).

In order to eliminate the absolute value operator in the objective function, we introduce n variables q_i and $2 \cdot n$ constraints: $\Delta w[i] \leq q_i$ and $-\Delta w[i] \leq q_i$ for $i \in [n]$. As a result, the optimization problem is written as a linear programming:

$$\text{minimize} \quad \sum_{i=1}^{n} q_i$$

subject to

Constraints (1) (2) (3) (4)

$\Delta w[i] \leq q_i \quad i \in [n]$

$-\Delta w[i] \leq q_i \quad i \in [n]$

Clearly, the last two constraints enforce q_i to be non-negative and equal to $|\Delta \mathrm{w}[i]|$ for all $i \in [n]$. Thus, $\sum_{i=1}^{n} q_i = \|\Delta \mathrm{w}\|_1$. Moreover, as $\Delta \mathrm{w}$ satisfies Constraints (1) (2) (3) and (4), $\Delta \mathrm{w}$ is a feasible solution to the MDA problem. Further, if the above LP is infeasible, neither is the MDA problem.

Interestingly, though the idea of linear programming is simple, as we show next, it can achieve an approximation guarantee depending on the minimum nonzero weight adjustment on some dimension in an optimal LP solution.

Consider an input instance $(G, \mathrm{w}, \mathrm{x}, \mathrm{y})$ to the MDA problem (and hence, its LP variant); and let $\Delta \mathrm{w}$ be a feasible MDA solution. We define:

$$\sigma(\mathrm{w}) = 1 + \max_{i \in [n]} |\mathrm{w}[i]| \text{ and } \gamma(\Delta \mathrm{w}) = \min_{i:\Delta \mathrm{w}[i] \neq 0} |\Delta \mathrm{w}[i]|.$$

Since both w and $\Delta \mathrm{w} + \mathrm{w}$ are in $[-1, 1]^n$, we have:

$$\max_{i \in [n]} |\Delta \mathrm{w}[i]| \leq 1 + \max_{i \in [n]} |\mathrm{w}[i]| = \sigma(\mathrm{w}) \leq 2.$$

Observation 4 *Let Δw be any feasible solution to the MDA problem. Then we have:*

$$\gamma(\Delta w) \cdot \|\Delta w\|_0 \leq \|\Delta w\|_1 \leq \sigma(w) \cdot \|\Delta w\|_0.$$

Proof Recall that the function $h(v) = 1$ for $v \neq 0$, and $h(v) = 0$ for $v = 0$. Thus, $\|\Delta \mathrm{w}\|_0 = \sum_{i \in [n]} h(|\Delta \mathrm{w}[i]|)$. Moreover, by the definitions of $\sigma(\mathrm{w})$ and $\gamma(\Delta \mathrm{w})$, it is easy to verify that:

$$\gamma(\Delta \mathrm{w}) \cdot h(|\Delta \mathrm{w}[i]|) \leq |\Delta \mathrm{w}[i]| \leq \sigma(\mathrm{w}) \cdot h(|\Delta \mathrm{w}[i]|)$$

holds for all $i \in [n]$. The observation follows by summing up the above inequality for all $i \in [n]$. □

Theorem 4 *Consider an input instance (G, w, x, y) to both the MDA problem and its LP variant. Let Δw^*_{lp} be an optimal LP solution and Δw^*_{mda} an optimal MDA solution. We have:*

$$\|\Delta w^*_{lp}\|_0 \leq \frac{\sigma(w)}{\gamma(\Delta w^*_{lp})} \cdot \|\Delta w^*_{mda}\|_0.$$

Proof Since both $\Delta \mathrm{w}^*_{lp}$ and $\Delta \mathrm{w}^*_{mda}$ are feasible solutions to the MDA problem and its LP variant, by Observation 4, we have:

$$\gamma(\Delta \mathrm{w}^*_{lp}) \cdot \|\Delta \mathrm{w}^*_{lp}\|_0 \leq \|\Delta \mathrm{w}^*_{lp}\|_1,$$

and

$$\|\Delta \mathrm{w}^*_{mda}\|_1 \leq \sigma(\mathrm{w}) \cdot \|\Delta \mathrm{w}^*_{mda}\|_0.$$

Moreover, combining the fact that $\|\Delta \mathrm{w}^*_{lp}\|_1 \leq \|\Delta \mathrm{w}^*_{mda}\|_1$, we have:

$$\gamma(\Delta \mathrm{w}^*_{lp}) \cdot \|\Delta \mathrm{w}^*_{lp}\|_0 \leq \sigma(\mathrm{w}) \cdot \|\Delta \mathrm{w}^*_{mda}\|_0.$$

Therefore, the theorem follows. □

From Theorem 4, while $\sigma(\mathrm{w})$ is independent to $\Delta \mathrm{w}^*_{lp}$ and can be upper bounded by 2, the approximation ratio of an optimal LP solution $\Delta \mathrm{w}^*_{lp}$ actually depends on its smallest absolute nonzero coordinate $\gamma(\Delta \mathrm{w}^*_{lp})$. When $\gamma(\Delta \mathrm{w}^*_{lp})$ is large, e.g., $\gamma(\Delta \mathrm{w}^*_{lp}) \geq 1$, then the approximation ratio of $\Delta \mathrm{w}^*_{lp}$ is no more than $\sigma(\mathrm{w}) \leq 2$. Unfortunately, $\gamma(\Delta \mathrm{w}^*_{lp})$ can be small in general, making $\Delta \mathrm{w}^*_{lp}$ as an MDA solution not satisfying. To remedy this issue, next we propose a method called *Iterative Linear Programming* to further refine $\Delta \mathrm{w}^*_{lp}$.

Iterative linear programming (ItrLP) Given a feasible MDA solution $\Delta \mathrm{w}$, let $P(\Delta \mathrm{w}) = \{i \mid \Delta \mathrm{w}[i] = 0\}$ be the set of dimensions with zero adjustment in $\Delta \mathrm{w}$, and $\bar{P}(\Delta \mathrm{w}) = [n] \setminus P(\Delta \mathrm{w})$, i.e., the set of dimensions with nonzero adjustment in $\Delta \mathrm{w}$.

The basic idea of ItrLP is as follows. When $\Delta \mathrm{w} = \Delta \mathrm{w}^*_{lp}$ is obtained, we try to "remove" the *smallest nonzero absolute* adjustment on Dimension-i, by: (i) keeping all the dimensions in $P(\Delta \mathrm{w})$ unchanged, and (ii) just utilizing the dimensions in $\bar{P}(\Delta \mathrm{w}) \setminus \{i\}$ to obtain a new feasible LP solution $\Delta \mathrm{w}'$. If $\Delta \mathrm{w}'$ exists, then we have $\|\Delta \mathrm{w}'\|_0 = \|\Delta \mathrm{w}\|_0 - 1$ because the adjustment on Dimension-i is saved. Repeat this process on $\Delta \mathrm{w}'$ until no feasible solution can be obtained, and then return the last feasible LP solution. As a result, a refined solution based on $\Delta \mathrm{w}^*_{lp}$ is obtained.

Although the ItrLP can refine an optimal LP solution $\Delta \mathrm{w}^*_{lp}$ gradually, the computational cost of repeatedly solving the LP's may be expensive. Moreover, if $\|\Delta \mathrm{w}^*_{lp}\|_0$ is very large, the refinement quality may be limited. In the next subsection, we propose another algorithm which, as shown in the experiment, is more efficient than the ItrLP and often can achieve a better MDA solution.

7.2 Iterative successive shortest path algorithm

In the following, we propose an algorithm, called *Iterative Successive Shortest Path* (ItrSSP) algorithm, which is an extension of the known *Successive Shortest Path* (SSP) algorithm [9] for solving the Min-Cost Flow problem. Specifically, the input to the Min-Cost Flow problem consists of:

- a cost associated flow-network $\mathcal{G} = \langle V \cup \{s, t\}, E\rangle$, where each edge $(i, j) \in E$ has capacity $c_{i,j} \geq 0$ and is associated with a cost $\alpha_{i,j}$,
- a specified flow value K.

The goal of the problem is to find a valid flow $f = \{e_{i,j}\}_{(i,j)\in E}$ such that: (i) the flow value of f, denoted by value(f), is K, and (ii) the cost of f, denoted by cost$(f) = \sum_{(i,j)\in E} e_{i,j} \cdot \alpha_{i,j}$, is minimized; or decide such a flow f with value$(f) = K$ does not exist.

7.2.1 Preliminaries: successive shortest path algorithm

As a first step, we introduce the following notions:

Pre-flows Consider a flow $f = \{e_{i,j}\}_{(i,j)\in E}$. If f satisfies the Capacity Constraint, then f is called a *pre-flow*. Note that a pre-flow is not necessarily a valid flow as it is allowed to violate the Conservation Constraint.

Residual networks Without loss of generality, we assume that $\mathcal{G} = \langle V \cup \{s, t\}, E\rangle$ has no parallel edges with reversed directions, because otherwise, we can avoid the parallel edges by adding intermediate vertices. Given a pre-flow $f = \{e_{i,j}\}_{(i,j)\in E}$ on a cost associated flow-network $\mathcal{G}$, the residual network of $\mathcal{G}$ with respect to f, denoted by $\mathcal{G}_f = \langle V_f, E_f\rangle$, is constructed as follows:

- $V_f = V \cup \{s, t\}$,
- Initialize $E_f = \emptyset$; and for each edge $(i, j) \in E$,
  - if $e_{i,j} < c_{i,j}$, add an edge (i, j) to E_f with (residual) capacity $r_{i,j} = c_{i,j} - e_{i,j}$ and cost $\bar{\alpha}_{i,j} = \alpha_{i,j}$;
  - if $e_{i,j} > 0$, add a *reversed* edge (j, i) to E_f with (residual) capacity $r_{j,i} = e_{i,j}$ and cost $\bar{\alpha}_{j,i} = -\alpha_{i,j}$.

Potential assignment to vertices A potential assignment to the vertices is a value assignment $\pi = \{\pi_i\}$ to each vertex $i \in V \cup \{s, t\}$.

Reduced costs Given a pre-flow f and a potential assignment π, the reduced cost of each edge $(i, j) \in E_f$ with respect to f and π, is defined as $\bar{\alpha}^{\pi}_{i,j} = \bar{\alpha}_{i,j} - \pi_i + \pi_j$. A potential

assignment π is *valid with respect to* f if $\bar{\alpha}^{\pi}_{i,j} \geq 0$ for all $(i, j) \in E_f$. The detailed steps of the Successive Shortest Path algorithm are shown in Algorithm 3:

Algorithm 3: Successive Shortest Path Algorithm

Input: $\mathcal{G} = \langle V \cup \{s, t\}, E\rangle$ and a flow value K
Output: A valid min-cost flow f_{out} with flow value equal to K, or f_{out} being a min-cost maximum flow.

1 $f_{\text{out}} \leftarrow \{e_{i,j} = 0\}_{(i,j)\in E}$;
2 $\pi \leftarrow \{\pi_i = 0\}_{i\in V\cup\{s,t\}}$;
3 $\mathcal{G}_{f_{\text{out}}} \leftarrow$ the residual network of $\mathcal{G}$ w.r.t. f_{out};
4 **while** *value*$(f_{\text{out}}) < K$ **do**
5 Compute the shortest distance $dist(s, v)$ from s to each vertex v in $\mathcal{G}_{f_{\text{out}}}$ with the reduced costs $\bar{\alpha}^{\pi} = \{\bar{\alpha}^{\pi}_{i,j}\}$ being the edge weights;
6 $Q \leftarrow$ the shortest path from s to t computed;
7 **if** Q *does not exist* **then**
8 Return f_{out} as a min-cost maximum flow;
9 **else**
10 Increase the values in f_{out} assigned to the edges on Q with value

$$\min\{c_{f_{\text{out}}}(Q), K - \text{value}(f_{\text{out}})\},$$

where $c_{f_{\text{out}}}(Q)$ is the smallest (residual) capacity among the edges on Q;
11 $\pi_i \leftarrow \pi_i - dist(s, i)$ for all $i \in V_{f_{\text{out}}}$ (and thus, $\bar{\alpha}^{\pi}$ is updated);
12 $\mathcal{G}_{f_{\text{out}}} \leftarrow$ the (new) residual network of $\mathcal{G}$ with respect to (the new) f_{out};
13 **end**
14 **end**
15 Return f_{out} as a min-cost flow with flow value equal to K;

The rationale of Algorithm 3 is to "push" a flow from s to t with the *most cost-effective* path Q and repeat this process until a flow can be returned. Consider the flow f_{out} returned by Algorithm 3 with input $\mathcal{G}$, K. If value$(f_{\text{out}}) = K$, then f_{out} is a min-cost flow with flow value equal to K. Otherwise, f_{out} is a min-cost maximum flow. In other words, by setting $K = \infty$, we can always obtain a min-cost maximum flow by Algorithm 3.

Complexity analysis The running time of Algorithm 3 is dominated by the overall cost within the While-Loop from Line 3–12. The cost of each iteration in the While-Loop is bounded by the cost of computing the shortest distances from s to each other vertex in the residual network. By applying the Dijkstra algorithm with a Fibonacci heap, this cost is bounded by $O(|E| + |V| \cdot \log |V|)$.

It remains to bound the number of iterations in the While-Loop. When $c_{i,j}$ and $\alpha_{i,j}$ for all $(i, j) \in E$ are integers, then in each iteration, the value of the flow gets increased by at least 1 unit. Therefore, the total number of iterations is at most value(f_{out}), and the overall cost of Algorithm 3 is $O(\text{value}(f_{\text{out}}) \cdot (|E| + |V| \cdot \log |V|))$.

Known properties Below we give three known *invariants* maintained in Algorithm 3 without proofs as they can be derived from the standard correctness proof of the SSP algorithm [9]. They will be useful when analysing our algorithm proposed next.

- **Invariant 1.** At the end of each iteration in the While-Loop of Algorithm 3, after updating f_{out} and π by the shortest path Q in the residual network, π is always valid with respect to f_{out}. Namely, $\bar{\alpha}^{\pi}_{i,j} \geq 0$ for all $(i, j) \in E_{f_{\text{out}}}$.
- **Invariant 2.** At the end of each iteration in the While-Loop of Algorithm 3, f_{out} is valid and the cost of f_{out} is the smallest among all the valid flows with flow value value(f_{out}).
- **Invariant 3.** The shortest path from any $u \in V_{f_{\text{out}}}$ to any other $v \in V_{f_{\text{out}}}$ in the residual network $\mathcal{G}_{f_{\text{out}}}$ with the reduced costs being edge weights is always the *most cost-effective* when pushing flows from u to v. That is, pushing flow from u to v along this shortest path can incur the least cost of the flow.

Remark 1 In fact, there is a heuristic way to improve the efficiency in each iteration in the While-Loop of Algorithm 3. That is, in each iteration, we can stop the Dijkstra algorithm as soon as the shortest path from s to t is found. It is known that the potential assignment π can be updated to ensure Invariant 1 holds.

For the sake of simplicity, we skip the discussion about it as the running time bound is the same anyway.

Remark 2 While Algorithm 3 is a general algorithm for solving the Min-cost Flow problem, interestingly, as we will see shortly, for the cost associated network in our application, we even don't need to maintain the potential assignment π and can stop the Dijkstra algorithm as soon as the shortest path from s to t is found in each iteration. As a result, the efficiency is considerably improved.

7.2.2 The ItrSSP algorithm

For two items x and y, recall that $xDom = \{i \mid x[i] > y[i]\}$, $yDom = \{i \mid y[i] > x[i]\}$ and $eqDom = [n] \setminus (xDom \cup yDom)$. Let $S = xDom \cup eqDom$ and $T = yDom \cup eqDom$. Consider an input instance ($G = \langle V, E\rangle$, w, x, y) to the MDA problem. We construct a cost associated extended graph, denoted by $\tilde{G}^{\text{cost}}_{\text{ext}} = \langle V \cup \{s, t\}, \tilde{E}_{\text{ext}}\rangle$, by adding to G:

- a source node s and a sink node t;
- an edge (s, i) with capacity $c_{s,i} = 1 + w[i]$ and cost $\alpha_{s,i} = y[i] - x[i]$ for each $i \in S$;

– an edge (i, t) with capacity $c_{i,t} = 1 - w[i]$ and cost $\alpha_{i,t} = x[i] - y[i]$ for each $i \in T$;
– $\alpha_{i,j} = 0$, for all $(i, j) \in E$ with neither $i = s$ nor $j = t$.

By the definitions of S and T, we have $\alpha_{i,j} \leq 0$ for all $(i, j) \in \tilde{E}_{\text{ext}}$. Thus, the flow with the minimum cost among all valid flows in $\tilde{G}^{\text{cost}}_{\text{ext}}$ must be a maximum flow. This is because otherwise, one can always decrease the cost by increasing the flow value.

By Lemma 1, the cost of any valid flow $f(\Delta\text{w})$ in $\tilde{G}^{\text{cost}}_{\text{ext}}$ is equal to $\Delta\text{w}\cdot(\text{x}-\text{y})$, where Δw is the legal weight adjustment constructed from $f(\Delta\text{w})$ by Algorithm 2. Moreover, for the same reason as in the proof of Lemma 2, removing any valid sub-flow with value q from $f(\Delta\text{w})$ will increase the value of $\Delta\text{w} \cdot (\text{x} - \text{y})$ by at most $2 \cdot q \cdot L$.

The basic idea of our algorithm is that we start with a min-cost maximum flow $f(\Delta\text{w})$ in $\tilde{G}^{\text{cost}}_{\text{ext}}$, and try to subtract some sub-flows from $f(\Delta\text{w})$ to reduce the value of $\|\Delta\text{w}\|_0$ yet ensuring the condition $\Delta\text{w} \cdot (\text{x} - \text{y}) \leq -\text{w} \cdot (\text{x} - \text{y})$ still holds. For this purpose, following a similar idea of the ItrLP algorithm, we try to "remove" the changes on the dimensions with small absolute nonzero adjustments.

Below are the detailed steps of the ItrSSP algorithm:

– **Step (1).** Let $\mathcal{G} = \tilde{G}^{\text{cost}}_{\text{ext}}$ and set $K = \infty$. Run a simplified variant[2] of Algorithm 3 with input $\mathcal{G}$ and K. Let f_{out} be the returned min-cost maximum flow and π the valid potential assignment w.r.t. f_{out} when the algorithm terminates.
  – If $\text{cost}(f_{\text{out}}) > -\text{w} \cdot (\text{x} - \text{y})$, return NOT APPLICABLE. In this case, only decreasing (resp. increasing) the dimensions in S (resp. T) is not sufficient to obtain a feasible MDA solution. Thus, the algorithm is not applicable here.
– **Step (2).** Let $f = f_{\text{out}}$ and Δw be the legal weight adjustment corresponding to f. Delete from $\mathcal{G}$, the edges (s, i) for all $i \in S \setminus S(\Delta\text{w})$, and the edges (i, t) for all $i \in T \setminus T(\Delta\text{w})$. Observe that all these deleted edges are "unused" in the flow f. Namely, the values in f assigned to these edges are 0. Let $\tilde{G}$ be the resulted graph after these edge removals. Note that π is still a valid potential assignment w.r.t. f, as removing edges will not violate the requirement $\bar{\alpha}^{\pi}_{i,j} \geq 0$ for all $(i, j) \in \tilde{G}$. Therefore, f is a valid min-cost maximum flow in $\tilde{G}$.

[2] As mentioned earlier in Remark 2 in the previous subsection, in our application, only the edges adjacent to either s or t in $\mathcal{G}$ have nonzero costs. Therefore, it is sufficient to set $\pi_s = \pi_t = \min_{i=s\vee j=t} |\alpha_{i,j}|$ and $\pi_i = 0$ for all i other than s and t. It can be verified that π is always valid for any valid flow in $\mathcal{G}$, and hence, there is no need to update it at the end of each iteration in the While-Loop of Algorithm 3. As a result, it suffices to stop the Dijkstra algorithm as soon as the shortest path from s to t is found and skip the maintenance of π in each iteration.

Algorithm 4: Sub-flow Removal Procedure

Input: $\tilde{G}$, a min-cost maximum flow f in $\tilde{G}$, two vertices a and b in $\tilde{G}$, and a flow value K to remove
Output: A min-cost flow f' with flow value $\text{value}(f) - K$.

$f' \leftarrow f$;
$\mathcal{G}'_f \leftarrow$ the residual network of $\tilde{G}$ w.r.t. f';
$\text{cost}_{min} \leftarrow \min_{i=s\vee j=t} |\alpha_{i,j}|$;
$\pi_s \leftarrow \text{cost}_{min}, \pi_t \leftarrow \text{cost}_{min}$;
$\pi_i = 0$ for all $i \in V_f \setminus \{s, t\}$;
$\bar{\alpha}^{\pi}_{i,j} = \bar{\alpha}_{i,j} - \pi_i + \pi_j$ for all $(i, j) \in E_f$;
while *TRUE* **do**
  $excess(a) \leftarrow K$;
  Compute the shortest path from a to b in $\mathcal{G}_{f'}$, denoted by Q;
  Increase the values in f' assigned to the edges on Q with value
  $$\Delta = \min\{c_{f'}(Q), excess(a)\},$$
  where $c_{f'}(Q)$ is the smallest (residual) capacity among the edges on Q;
  $\mathcal{G}_{f'} \leftarrow$ the (new) residual network of $\tilde{G}$ with respect to (the new) f';
  $excess(a) \leftarrow excess(a) - \Delta$;
  if $excess(a) = 0$ **then**
    Return f';

– **Step (3).** Consider $i = \arg\min_{i:\Delta\text{w}[i]\neq 0} |\Delta\text{w}[i]|$, that is, Dimension-$i$ is the dimension with the smallest absolute nonzero weight adjustment in Δw.
  – If $i \in S(\Delta\text{w})$, remove the edge (s, i) from $\tilde{G}$ and subtract from f the sub-flow $f_{i\to T(\Delta\text{w})}$ with flow value $e_{s,i}$ by calling Algorithm 4 with input: $\tilde{G}$, f, $a = t$, $b = i$ and $K = e_{s,i}$.
  – If $i \in T(\Delta\text{w})$, remove the edge (i, t) from $\tilde{G}$ and subtract from f the sub-flow $f_{S(\Delta\text{w})\to i}$ with flow value $e_{i,t}$ by calling Algorithm 4 with input: $\tilde{G}$, f, $a = i$, $b = s$ and $K = e_{i,t}$.

  Let f' be the min-cost flow returned. (The correctness of Algorithm 4 follows from the facts that (i) the input flow f is a min-cost flow; and (ii) Algorithm 4 maintains exactly the same invariant as Invariant 3 of Algorithm 3.)
– **Step (4).** Run the While-Loop in (the simplified variant of) Algorithm 3 with $\tilde{G}$ and $K = \infty$ starting with the state of f' being f_{out}. Thus, when the While-Loop terminates, f_{out} is a min-cost maximum flow in $\tilde{G}$. (This follows from the fact that both Invariants 1 and 2 are maintained in during this process.)
– **Step (5).** If $\text{cost}(f_{\text{out}}) > -\text{w} \cdot (\text{x} - \text{y})$, return Δw as an MDA solution. Otherwise, let $f = f_{\text{out}}$ and update Δw to be the legal weight adjustment corresponding to f and then repeat from Step (3).

Correctness of the overall ItrSSP algorithm When the ItrSSP algorithm is applicable, the correctness follows from the facts that: (i) f is valid at the beginning; (ii) every sub-

flow subtracted from f in Algorithm 4 is valid; (iii) every flow pushed in the While-Loop of Algorithm 3 is also valid; and (iv) $\text{cost}(f) = \Delta\text{w} \cdot (\text{x} - \text{y}) \leq -\text{w} \cdot (\text{x} - \text{y})$ always holds. As a result, it ensures that the legal weight adjustment Δw corresponding to f satisfies Constraints (1) (2) (3) and (4), and thus, Δw is a feasible solution to the MDA problem.

Running time complexity analysis Recall that by Assumption 1, all the input values are $\frac{1}{M}$-multiple. Let C be the running cost of the Dijkstra algorithm on the residual network $\mathcal{G}_f$. It is easy to know that $C = O(|E| + |V| \cdot \log |V|)$, because $|V_f| = |V|$ and $|E_f| = O(|E|)$. Since in the ItrSSP algorithm, the cost of each iteration in the While-Loop of Algorithm 3 and Algorithm 4 is bounded by $O(C)$, it is sufficient to bound the total number of iterations of the overall ItrSSP algorithm.

First, since in Step (1), we run Algorithm 3 to obtain a min-cost maximum flow f_{out}, the cost is bounded by $O(M \cdot \text{value}(f_{\text{out}}) \cdot C) = O(M \cdot n \cdot C)$. Second, Step (2) takes at most $O(|E|)$ time. Third, in Step (3), we run Algorithm 4 to remove a sub-flow with value $|\Delta\text{w}[i]|$. Notice that each iteration in Algorithm 4 removes a sub-flow with value at least $\frac{1}{M}$. There can be at most $O(M \cdot |\Delta\text{w}[i]|) = O(M)$ iterations, since $|\Delta\text{w}[i]| \leq 2$. Observe that at the beginning of Step (3), f is a maximum flow on $\tilde{G}$, and f_{out} obtained in Step (4) is a flow on $\tilde{G}$ after removing an edge. Thus, $\text{value}(f_{\text{out}}) \leq \text{value}(f)$. Moreover, after the sub-flow removals in Step (3), we have $\text{value}(f') = \text{value}(f) - |\Delta\text{w}[i]|$.

Therefore, in Step (4), $\text{value}(f_{\text{out}}) - \text{value}(f') \leq |\Delta\text{w}[i]|$. As a result, there can be at most $O(M \cdot |\Delta\text{w}[i]|) = O(M)$ iterations. Finally, Step (5) can only repeat Step (3) and Step (4) at most $2n$ times, since each time it removes an edge adjacent to either s or t and there are at most $2n$ such edges. Therefore, the total number of iterations is bounded by $O(M \cdot n)$.

Combining the cost of the Dijkstra algorithm, the overall running time complexity of our ItrSSP is bounded by $O(M \cdot n \cdot (|E| + |V| \cdot \log |V|))$.

Remark 1 Although the ItrSSP needs to perform multiple iterations, unlike the ItrLP which has a blow-up of the number of iterations in the time complexity, the running time complexity of the ItrSSP algorithm remains the same as the bound of the SSP algorithm. In other words, the algorithm only incurs a constant factor overhead when refining the min-cost maximum flow for the MDA problem.

Remark 2 In practice, the value M of the input is often small, and at each time, the algorithm can push a flow value far larger than the worst-case amount $\frac{1}{M}$. As a result, both Algorithms 3 and 4 can stop with a small number of iterations in the While-Loop. Moreover, since each repeat of Step (3) and Step (4) decreases $\|\Delta\text{w}\|_0$ by 1, in general, the total number of repeats is far smaller than n. Therefore, in practice, the ItrSSP algorithm can run very fast. This is also observed in our experiment (see Fig. 10).

8 The weight transition graph

The weight transition graph G is an important input in the *rank-reversal* problem. However, in general, it may not be easy to obtain it directly due to limited or even no access to commercial real-world personalized ranking systems, not to mention that G varies from one domain to another. To remedy this, we first describe an alternative algorithm to construct G *with no domain knowledge required*. While this algorithm is just one of many possible ways to construct a weight transition graph G, it may hint some insight of the construction rationale. Next, we further study how the feasibility of the problem is under different instances of G (under different settings of key parameters in the algorithm), which depends on the capability and the density of the edges in G specified by different parameters.

Remark Although different G can affect the *rank-reversal* results, the focus of this paper is to propose effective algorithms for incremental preference adjustment when G is given; and our methods are indeed orthogonal to the problem of how G is obtained.

8.1 An alternative construction algorithm of *G*

An overview of the algorithm The basic idea of the construction algorithm is to perform a *certain* sequence of *rank-reversal* operations by solving linear programmings, i.e., the LP algorithm, on a weight transition graph G' which is a *complete* graph and of which the capacity of each edge is 2, the maximum capacity. In other words, G' imposes no flow-like constraints on the reversals, and thus, the LP is always feasible under G'.

Recall that each weight adjustment Δw on w is essentially a flow-value assignment, $\{e_{i,j}\}_{i \neq j}$, to the edge (i, j) in G'. Let $\mu_{i,j}$ and $\sigma_{i,j}$ be the *mean* and *standard deviation* of the flow values assigned to edge (i, j) in G' over all the *rank-reversal* operations in the sequence, respectively. The rationale of constructing a weight transition graph G (not the G') is that if the capacity $c_{i,j}$, for each edge (i, j) in G, is set to $\mu_{i,j} + \beta \cdot \sigma_{i,j}$ for some β (say $\beta = 3$), then the flow-like constraints (i.e., $c_{i,j}$'s) imposed by G are likely to admit feasible solutions for the *rank-reversal* operations. In particular, the parameter β controls the "strictness" of the constraints imposed by G.

Another natural way to control the strictness of the constraints is to control the number of edges in G. If only n^ρ edges are randomly picked and kept in G from the $n \cdot (n-1)$

Table 7 The percentages of feasible operations versus different parameter settings for G

	β (with $\rho = 1.5$)						ρ (with $\beta = 0.5$)				
Datasets	0	0.1	0.2	**0.5**	0.8	1.0	1.0	1.25	**1.5**	1.75	2.0
Amz	55%	73%	81%	89%	93%	100%	61%	70%	89%	95%	99%
Fsq	45%	65%	84%	90%	95%	99%	65%	73%	90%	96%	100%
Nfl	40%	66%	80%	85%	92%	100%	55%	68%	85%	91%	97%
Mvl	44%	72%	80%	90%	100%	100%	60%	72%	90%	95%	99%
Rtb	20%	34%	60%	81%	90%	98%	51%	68%	81%	88%	95%

possible edges, the larger ρ, the more likely of G to admit feasible solutions for the *rank-reversal* operations.

Finally, the sequence of *rank-reversal* operations are generated with the given training data and the vector embedding of the items learnt by a specified preference learning model (see details about the models in Sect. 9.1.2). More specifically, the training data is a list of user log records, each of which is a user-item pair with *chronological order preserved*, e.g., the purchase history of the users on Amazon, the movie rating history of the users on Netfix. With the vector embedding of all the items learnt by the model, a list $\mathcal{L}$ of all the user-item pairs can be extracted from the training data, with chronological order preserved. Each pair in the list $\mathcal{L}$, denoted by (u, y), represents the record that user u has purchased item y in the corresponding time stamp. Furthermore, the preference vector w_u of each user u is initialized as a random vector in $[-1, 1]^n$.

For a pair (u, y) in $\mathcal{L}$, consider the ranking list of all the items obtained by w_u. If there is an item x such that x is ranked higher than y but the pair (u, x) does not occur before (u, y) in $\mathcal{L}$, then the rank order of x and y should be reversed. This is because such a rank order contradicts with the fact that the user u has bought y at this time stamp while u has not chosen x before. Based on this idea, a *rank-reversal* operation is generated by randomly choosing such an item x (if multiple x's exist) for each $(u, \mathrm{y}) \in \mathcal{L}$.

The algorithm For a specified preference learning model and a given training set, we construct G as follows.

- *Step 1*: Extract the list $\mathcal{L}$ in the aforementioned manner.
- *Step 2*: Initialize w_u as a random vector in $[-1, 1]^n$ for each user u.
- *Step 3*: For $l = 1, 2, \ldots, |\mathcal{L}|$, process the lth user-item pair (u, y) in $\mathcal{L}$ by the following procedure:
  - Sort all the items with w_u.
  - Randomly pick an item x which is ranked higher than y (i.e., $\mathrm{w}_u \cdot \mathrm{x} > \mathrm{w}_u \cdot \mathrm{y}$) and whose corresponding pair (u, x) does not occur before (u, y) in $\mathcal{L}$. If no such x exists, skip and process the next pair.
  - Perform a *rank-reversal* operation with the complete graph G', w_u, x and y by the algorithm LP.
  - Let $\{e_{i,j}\}_{i \neq j}$ be the flow value assignment of the legal weight adjustment of the optimal LP solution. Record $\tilde{e}^l_{i,j} = e_{i,j}$ for each edge (i, j).
- *Step 4*: Compute the mean $\mu_{i,j}$ and the standard deviation $\sigma_{i,j}$ over all the $|\mathcal{L}|$ recorded values $\tilde{e}^l_{i,j}$ for each edge (i, j), where $l = 1, \ldots, |\mathcal{L}|$.
- *Step 5*: The desired weight transition graph G is obtained from G' with two parameters: (i) the edge density ρ and (ii) the standard deviation factor β in the edge capacities. The detailed steps are as follows:
  - The edge set E of G is generated by picking n^ρ edges uniformly at random from the edges in G'.
  - The capacity $c_{i,j}$ for each $(i, j) \in E$ is computed by $\mu_{i,j} + \beta \cdot \sigma_{i,j}$.

  As a result, with different combinations of ρ and β, we can control the "strictness" of the constraints on the *rank-reversal* operations imposed by G.

8.2 Feasibility of G with different ρ and β

As mentioned earlier, the problem of performing a *rank-reversal* operation can be infeasible if G is too restrictive. The causes for this can be either G being too sparse or the edge capacities being too small. We define the *feasibility* of G as the percentage of feasible *rank-reversal* operations under the constraints imposed by G with a specified preference learning model and a user with a preference vector initialized by a random vector. We examine the feasibility of G with the following steps:

- Select WARP (detailed descriptions can be found in Sect. 9.1.2) as the reference preference learning model.
- Consider the user with the richest information in the training data, namely, the user occurs the most in $\mathcal{L}$ in the corresponding specified dataset.
- Initialize w as a random vector in $[-1, 1]^n$.
- Construct G with different parameter settings: $\rho = 1.0, 1.25, 1.5, 1.75, 2.0$ and $\beta = 0, 0.1, 0.2, 0.5, 0.8, 1.0$.

– Simulate 100 actions from the user with G by using LP to perform 100 *rank-reversal* operations one by one (generated by the method as described in Sect. 9.1.6).
– The feasibility of G is measured by the percentage of feasible *rank-reversal* operations out of the 100 user actions.

Two sets of settings are shown in Table 7: $\rho = 1.5$ with varying β and $\beta = 0.5$ with varying ρ. With a fixed ρ, the percentage of feasible operations gets increased w.r.t. β. Similarly, with a fixed β, as the number of edges in G grows, the feasible percentage increases. This is because more edges can be used to find a feasible solution. By default, we set $\mathbf{æ} = \mathbf{1.5}$ and $\mathbf{fi} = \mathbf{0.5}$ in our experiments.

9 Experiments

In this section, we aim to evaluate the effectiveness and efficiency of our incremental preference adjustment algorithms, as per user's action(s) to the ranking result initially by a certain reference preference learning model. We highlight the following key questions as well as the section number used to clarify or answer it.

– Q1: What reference preference learning models shall we use that are widely adopted for personalized ranking? (see Sect. 9.1.2)
– Q2: In our MDA problem setting, how to:
  – obtain the *ground truth* of the ranking results (w.r.t. a certain reference preference learning model) that is used to measure the effectiveness of our incremental adjustment techniques (see Sect. 9.1.2).
  – initialize the preference vector, w, of the user (see Sect. 9.1.4),
  – construct the weight transition graph G without domain knowledge (see Sect. 8).
– Q3: How do we select users and simulate their action(s) when interacting with the ranking system? (see Sect. 9.1.6)
– Q4: Given the ground truth, what is an appropriate choice of measurement on effectiveness? (see Sects. 9.1.5 and 9.2)
– Q5: What are the baseline methods to be used for comparison purpose with our approaches? (see Sect. 9.1.3)

9.1 Experiment setup

9.1.1 Datasets and experiment environment

Datasets We conduct experiments on five real datasets that have been widely used in personalized ranking [28].

– *Amazon*[3] (Amz) is a dataset that contains 584,000 reviews of 427,000 users on 24,000 Amazon instant videos.
– *FourSquare*[4] (Fsq) is constructed based on 16 million ratings on 9,185 places that 348,000 users have visited.
– *Netflix*[5] (Nfl) contains over 22 million ratings from 363,000 users on 14,000 movies.
– *MovieLens*[6] (Mvl) contains 900,000 rating records of 6,011 users on 3,678 movies.
– *RateBeer*[7] (Rtb) contains 2,592 users and 2,732 products (i.e., beers), with in total 125,000 user ratings.

Experiment environment All experiments are conducted on a Windows 10 machine equipped with a 2.4 GHz Intel CPU and 8 GB RAM, and all our algorithms are implemented by C++ and compiled by MinGW (a variant of gcc for Windows) with O3 flag turned on.

9.1.2 Ground truth formation and reference models

Ground-truth formation

In the context of neural networks, embeddings are low-dimensional, learned continuous vector representations of discrete variables. In our case, embeddings offer us accurate information of user preferences, as they are trained over long-term user-item interaction data. Thus, for a specified reference preference learning model and a specified user, embedding (i.e., the preference vector) learned by the model is considered as the *true* preference of the user. Hence, we treat such a preference vector as the "*ground-truth preference*" of the user (under the preference learning model), which is used to measure the effectiveness of the preference vector adjusted by performing *rank-reversal* operations with our proposed methods. More details are in Sect. 9.1.5.

Reference preference learning models We adopt the following three representative models:

– *Bayesian Personalized Ranking* (BPR) [22].
– *Weighted Approximate-Rank Pairwise Loss* (WARP) [29].
– *Adversarial Personalized Ranking for Recommendation* (APR) [14].

Among the three, BPR and WARP are widely used in the community [22,29], while APR [14] is a recent method.

[3] http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/ratings_Amazon_Instant_Video.csv.
[4] https://sites.google.com/site/yangdingqi/home/foursquare-dataset.
[5] http://academictorrents.com/browse.php?search=Netflix.
[6] https://grouplens.org/datasets/movielens/.
[7] https://snap.stanford.edu/data/web-RateBeer.html.

Model training Each of the above three models embeds each user into a preference vector (hence, it is the *ground-truth preference* for the user) and each item into a vector; all the vectors are of n dimensions. These embeddings are trained in the same way and on the same datasets as in their original papers. Specifically, LightFM[8] is used to train BPR and WARP embeddings, and APR is trained using the code provided by the original author.[9] Moreover, we consider $n = 100$ in all embeddings.

9.1.3 Methods for comparison

We compare the performance of six algorithms:

- *LP*: The algorithm solving the MDA problem by linear programming with l_1-norm rather than l_0-norm as the objective while subject to the same conditions. The returned results are feasible solutions to the MDA.
- *ItrLP*: This is the algorithm which refines optimal LP solutions by solving LPs iteratively.
- *ItrSSP*: The Iterative Successive Shortest Paths algorithm (proposed in Sect. 7).
- *Approx*: A heuristic implementation of our approximate algorithm (proposed in Sect. 6) for solving the MDA problem, where an optimal LP solution is treated as an optimal solution to the F-problem.
- *Hybrid*: This is a hybrid algorithm combining the ItrSSP and LP algorithm. The basic idea is to further reduce the number of dimensions changed via removing small changes on certain dimensions from the optimal solution returned from LP. That is to say, hybrid takes the solution of LP as a starting point to run ItrSSP.
- *Greedy*: This method performs *rank-reversal* operations *without* the constraints imposed by the weight transition graph G. As a result, it always greedily chooses the "most effective" dimensions to adjust the preference vector to reverse the item pair. *Greedy* is only used in the experiments of effectiveness evaluation, to illustrate the importance of the role played by G in the *rank-reversal* operations.

9.1.4 Initializing the preference vectors

The preference vector w for each user is initialized by a *Random Initialization*. Specifically, each coordinate of w is chosen uniformly and independently from the range $[-1, 1]$. We note that the random initialization makes no assumption on the users' preference vectors, which nicely shows how a preference adjustment method can effectively support new (i.e., cold-start) users.

[8] https://github.com/lyst/lightfm.

[9] https://github.com/hexiangnan/adversarial_personalized_ranking.

9.1.5 Measurement of effectiveness

Consider a preference learning model, a preference adjustment method and a specified user. Let w be the initial preference vector of the user. Afterward, w is adjusted by performing a certain number of *rank-reversal* operations with the specified method. Let T_k be the list of top-k recommended items obtained with the adjusted w (after the operations), and T_k^* the list of the top-k recommended items obtained by the ground-truth preference w* of the user (under the selected model). The *effectiveness* of w (and hence, the effectiveness of the specified method) is measured by the *overlap ratio* of T_k and T_k^*, which is defined as $\frac{|T_k \cap T_k^*|}{k}$. In all our experiments, the default value of k is 100 unless stated otherwise.

9.1.6 User behaviour simulation

Given a selected preference learning model and a user with a preference vector w as an input, we simulate the action(s) of the user with requests for *rank-reversal* operations, where the items x and y are generated as follows.

As defined earlier, T_k^* is the top-k list under the ground-truth preference w* of the user, while T_k is the top-k list of the input preference vector w. The item x is picked uniformly at random from $T_k \setminus T_k^*$, and the item y is chosen in the same way from $T_k^* \setminus T_k$.

The rationale here is that the ground-truth preference w* indicates that the user prefers y more than x, while w ranked x better than y. Therefore, a *rank-reversal* operation for x and y is needed.

9.2 Measurement on the numbers of dimensions changed

Next, we measure the numbers of dimensions changed when performing *rank-reversal* operations for each competitor (described in Sect. 9.1.3). The experiment setup is as follows:

- Consider a preference learning model, a dataset and their corresponding G constructed with default parameters.
- Fix the user with the richest information in the training data, and initialize w by a random vector.
- Perform *rank-reversal* operations (which are generated by the method in Sect. 9.1.6) one by one with a specified algorithm until 100 *feasible* operations have been performed.
- Output the average number of dimensions changed over these 100 feasible operations.

The results of the methods on the BPR, WARP and APR model over all datasets are shown in Fig. 6. Overall, Hybrid is

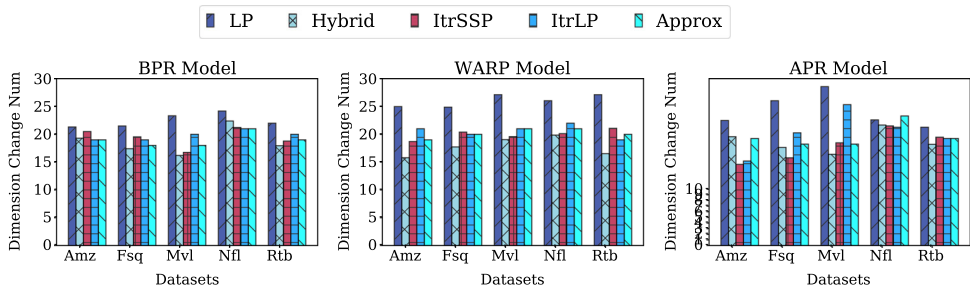


Fig. 6 The average number of dimensions changed on five datasets

the best on minimizing the numbers of dimensions changed. Detailed observations are as follows. (1) LP performs the worst because it aims to minimize the l_1-norm instead of l_0-norm. (2) The performance of ItrSSP, Approx and ItrLP are similar, because they all try to refine a feasible solution to reduce the l_0-norm. (3) ItrLP is slightly better than LP, because the iterative process can further reduce the number of dimensions changed. (4) Hybrid performs better than ItrSSP over most datasets, which indicates that choosing a proper start point for ItrSSP method may lead to better results. Moreover, Hybrid is always better than LP as it refines based on an LP optimal solution.

9.3 Measurement on the effectiveness of adjustments

Next, we study the effectiveness score (defined in Sect. 9.1.5). In the following, for each dataset, we consider a representative group of users who have the richest user-item interaction history (i.e., the training data): *top-5 most active users*. The portions of the logs of the top-5 active users over the total user logs are: Amazon (0.17%), Netflix (0.05%), FourSquare (0.07%), Movielens (0.92%), RateBeer (5.16%). Since the group of users have relatively rich information in the training data, their ground-truth preference vectors are considerably accurate to users' *real-world* preference. As a result, these ground-truth preference vectors serve as good indicators for measuring our methods' effectiveness. These experiments are conducted as follows:

- For each of the top-5 users, perform 100 feasible *rank-reversal* operations with the same experiment setup as in Sect. 9.2, except for the last bullet point, where we output the average effectiveness (defined in Sect. 9.1.5) for every 10 feasible *rank-reversal* operations. Denote the average effectiveness of the ith batch of 10 feasible operations of the jth user by $\tau_{i,j}$, where $i = 1, \ldots, 10$ and $j = 1, \ldots, 5$.
- Output the overall average of the effectiveness $\tau_{i,j}$ over all the 5 users on the corresponding ith batch of 10 feasible *rank-reversal* operations, i.e., $\frac{1}{5}\sum_{j=1}^{5}\tau_{i,j}$.

The results of the top-5 users over BPR, WARP, and APR are shown in Figs. 7, 8, and 9 respectively. Note that we take the average among the top-5 most active users when reporting the result. We report our observations from different angles as follows.

Orthogonality to the preference learning models As shown in Figs. 7, 8, and 9, the Adjustment Effectiveness of our methods are all monotonically increasing with respect to the number of rounds over all three different embedding models, across all five datasets. The adjusted w consistently tends to the ground-truth preference w^* in the corresponding model quickly from a random initialization. It shows that our proposed methods can cooperate with different reference preference learning models and provide effective incremental adjustment to each of them. Moreover, the effectiveness of the adjustments by our methods dramatically increases after only the first one or two rounds, that is, only 10 or 20 *rank-reversal* operations are processed.

Overall observations across datasets and models We first provide a summary of the overall observations on all methods: (1) Over all datasets at the end of 10 rounds, ItrSSP has the best performance, followed by Hybrid, ItrLP , and Approx. For example over the BPR model and the MovieLens dataset, ItrSSP, Hybrid, ItrLP, Approx, LP, and Greedy reach 0.6, 0.48, 0.46, 0.45, 0.4, and 0.08 respectively at the end of the last round. ItrSSP performs the best, because it iterates and runs Min-Cost-Max-Flow many times. (2) ItrLP and LP have similar results where ItrLP is slightly better. It is because they both start with LP, but ItrLP further iterates this process. (3) Approx is slightly better than LP, because it uses the results from LP as input. (4) As the number of rounds increases, all methods except for Greedy show an increasing trend on the effectiveness, indicating that these methods indeed help incrementally adjust the preference vector to what the user's real intention. The highest effectiveness of Greedy is 0.3, which is far from the average effectiveness (0.42) of the second worst LP. The reason is: greedy changes significant weights in one single *rank-reversal* operation, while the rest adjust the weights smoothly to cater for users' preferences.

Over the BPR model (Fig. 7), it is almost consistent that ItrSSP outperforms Hybrid which in turn outperforms

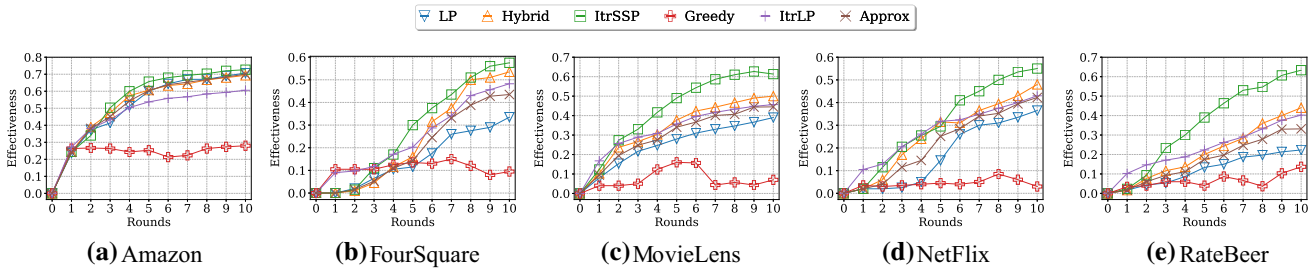


Fig. 7 Effectiveness test of incremental adjustment over BPR (Top-5 active users)

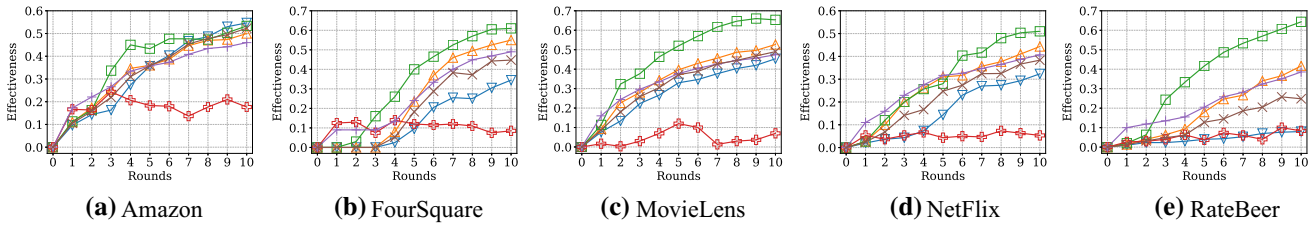


Fig. 8 Effectiveness test of incremental adjustment over WARP (Top-5 active users)

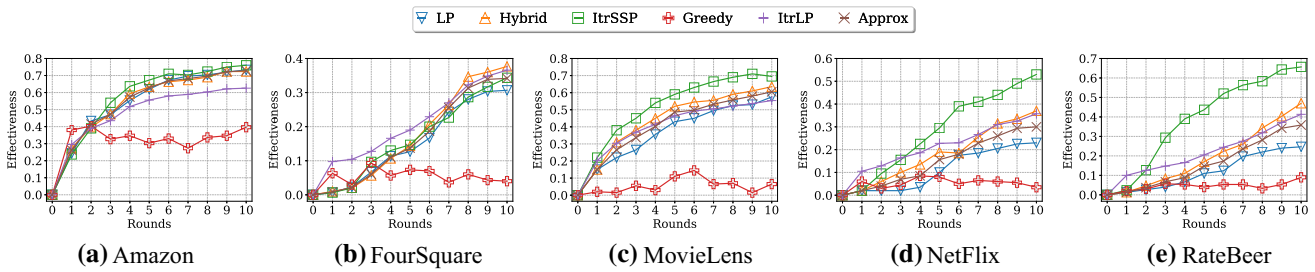


Fig. 9 Effectiveness test of incremental adjustment over APR (Top-5 active users)

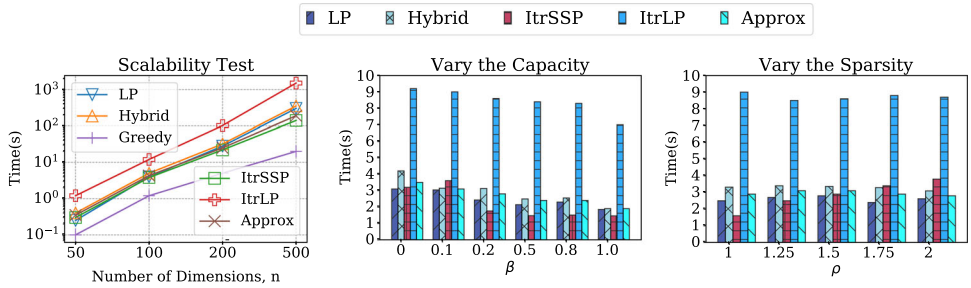
Approx. This is because iterations in ItrSSP can improve the effectiveness. Moreover, by comparing ItrSSP and Hybrid, we find that different start points of the iterations in ItrSSP can affect the effectiveness: the min-cost maximum flow (as the start point in ItrSSP) works better than the solution of LP (adopted as the start point in Hybrid) here.

Over the WARP model (Fig. 8), the effectiveness from high to low is ItrSSP, Hybrid, Approx, LP, and the Greedy. This is because both Hybrid and Approx adopt the LP as the start point, and hence improve the performance of LP. Hybrid adopts an iterative strategy, making its performance better than Approx. Similarly, over the APR model (Fig. 9), the order of these algorithms' effectiveness remains for the same reason as described over the BPR model.

Importance of G From Figs. 7, 8, and 9, we can also find that Greedy has the worst performance across all datasets and reference preference learning models; in most cases its effectiveness score is about 0.1 only. As aforementioned in Sect. 9.1.3, *Greedy* works without the weight transition graph G. In contrast, the remaining methods which enforce the constraints imposed by G have a much better performance. Therefore, it indicates that the role played by G in the MDA problem is important and significant.

Observations across datasets (1) All methods except for Greedy (0.28) and ItrLP (0.6) have similar performance (0.7) on Amazon. (2) ItrSSP, Approx and Hybrid have significant differences on the remaining datasets. For example, on RateBeers over APR, ItrSSP outperforms Hybrid by 55%, which in turn outperforms ItrLP by 10%, which in turn outperforms Approx by 12%, which in turn outperforms LP by 40%, which in turn outperforms Greedy by 150%. (3) Approx and ItrLP have similar performance, because they both use the results of LP as the start point of their methods. (4) The performance of LP sometimes is as bad as Greedy (e.g., on the RateBeer dataset). That is because LP sometimes may change many dimensions and totally replace the top-k results, while ItrLP performs better as it adopts an iterative strategy to avoid such cases.

Observations across the preference learning models Here, we compute the average performance gap over all reference preference learning models. On Netflix, ItrSSP outperforms the second best method Hybrid by 21%, which in turn outperforms ItrLP by 11%, which in turn beats Approx by



(a) Efficiency w.r.t. dimension n **(b)** Efficiency w.r.t. Capacity **(c)** Efficiency w.r.t. Number of Edges

Fig. 10 Efficiency test

6%, which in turn beats LP by 13% . On the Movielens dataset, ItrSSP outperforms Hybrid by 17%, which in turn outperforms ItrLP by 24%. The performance curves of the competitors in the same dataset are similar, probably because the distribution of the item groups has no significant change when the items are embedded.

9.4 Efficiency evaluation

At last, we test the computation efficiency of the five methods with respect to the problem size. The five of them are in the scalability test, while the Greedy is omitted in the Capacity and Sparsity test as it is not constrained by the weight transition graph. Particularly, we conduct the experiment on the Amazon dataset with APR being the reference preference learning model, because the dataset does not impact the efficiency much. We vary the parameter n with the values in {50, **100**, 200, 500}, β in {0, 0.1, 0.2, **0.5**, 0.8, 1.0}, and ρ in {1, 1.25, **1.5**, 1.75, 2.0}, where the default values for each parameter have been highlighted in bold. When varying one parameter, the other two are set to default values.

From Fig. 10a, we have several observations. (1) The running time of all methods increases sub-linearly when the number of dimensions increases, and Greedy is the most efficient one. (2) The ItrLP is the worst among all of the efficiency tests, because it takes multiple rounds of LP. (3) Approx is slightly slower than LP, but slightly faster than Hybrid. (4) When increasing the capacities of G by varying β (Fig. 10b), ItrSSP is the most efficient one followed by LP and Hybrid over all cases except for β=0.5. This is because that with the increase of edge capacities, the item pairs can be reversed more easily, and thus, less processing time is required. However, as Hybrid and LP need to solve linear programming, their efficiency is affected. (5) When the number of edges increases (Fig. 10c), the performance of ItrSSP is affected the most, while the performance of LP, Approx, and ItrLP is not impacted too much. This is because it takes more time for ItrSSP to find the flows when the number of edges increases.

Remark Overall, ItrSSP is recommended, because it achieves the best effectiveness without losing too much efficiency and dimension change in most cases, over all the five real datasets and under all the three preference learning models, as compared to all other competitors.

10 Related work

The most related work is preference learning [8,13–16,22,28,35], which is the core part of all personalized systems and aims to learn users' preferences over different objects. This process often requires a large amount of user feedback data, either explicit [23] or implicit [13,14,22,28], and it maps a user to a latent feature space, modeling users' preferences. Existing literature can be broadly divided into three categories: (i) offline preference learning [5,10,13,14,21–23], (ii) preference adjustment [12,15,16,28,31,32], and (iii) interactive preference elicitation [8,30,35]. Preference elicitation aims to address the cold start problem, which is a special case of preference adjustment. Thereby we will describe the first two in more details.

Offline preference learning The idea of using pair-wise ranking optimization for personalized searching and recommendation is first proposed by Rendle et al. [22], and the method BPR (Bayesian Personalized Ranking) is still one of the most widely used models in the area of preference learning. The idea of BPR is to learn users' pair-wise preference from historical implicit feedback. APR (Adversarial personalized ranking for recommendation) [14] applies an adversarial regularizer to BPR, and thereby largely improves the model's robustness. Recent advance in deep learning motivates researchers to model users' preferences directly from a latent space, which shows promising performance [13]. However, all these methods require a large training set available, for the purpose of modelling users' preferences effectively.

Preference adjustment As the offline trained model often lacks the flexibility to be adapted into a real-time recommendation system, recent studies concentrate on the dynamic

adaptation of users' preferences [12,28,31]. These studies often model users' short-term preferences based on users' real-time interaction data, which are of high velocity. Therefore, usually a window-based stream model is used, in which way the model only updates users' preferences at certain time points. It is challenging to learn users' preferences in real time, in the sense that short-term interest can be unstable and leads to interest drift. In order to improve the robustness, researchers [12,28] often consider users' offline learned preferences in addition to short-term interest. A different stream-based model is proposed [15], which incrementally updates users' preferences. As the sparse feedback may be an issue in an online learning setting, the authors propose to use a popularity-based preference, and also design an efficient Matrix Factorization updating strategy. While stream-based preference learning algorithms achieve a reasonable trade-off between effectiveness and efficiency, they often update users' short-term interests after a fixed time window. Model performance may be affected if the window size is too large or too small. Instead, we consider the incremental learning approach, which updates users' preference model interactively.

Learning users' preferences in an interactive manner can also be addressed using the deep reinforcement learning (DRL) methods [6,33,34]. DRL-based preference learning methods mainly benefit from the "trial-and-error" style. Zhao et al. [34] propose the DEER framework that captures users' short-term interests based on the click information. DeepPage [33] can perform a whole-page optimization according to users' real-time interactions. However, all these methods require a large number of user interaction logs in the offline training process in order to achieve a high performance. If only limited interaction data is available, it is difficult for the DRL framework to learn an optimal policy.

To summarize, compared to existing methods, our technique in this paper is orthogonal to the choice of preference learning models, and can incrementally adjust every individual user's preference on-the-go.

11 Conclusions

In this paper, we formalized the *rank-reversal* operation into the Minimum Dimension Adjustment (MDA) problem. We first proved that the MDA problem is NP-hard, and then we showed that a 2.17-approximate solution can be obtained in polynomial time, provided that an optimal solution to a carefully designed problem is given. Moreover, we developed two heuristic algorithms: (i) Iterative Linear Programming (ItrLP) and (ii) Iterative Successive Shortest Path (ItrSSP). While the ItrLP can achieve an approximation guarantee, the ItrSSP is proven to be efficient. Finally, we conducted extensive experiments to study the efficiency and effectiveness of our solutions.

Acknowledgements This work is supported in part by ARC under Grants DP200102611, DP180102050, DE190101118, and a Google Faculty Award.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: a system for large-scale machine learning. In: OSDI, vol. 16 (2016)
2. Ageev, A.A., Sviridenko, M.I.: Approximation algorithms for maximum coverage and max cut with given sizes of parts. In: IPCO (1999)
3. Barzilai, J., Borwein, J.M.: Two-point step size gradient methods. IMA J. Numer. Anal. **8**(1), 141–148 (1988)
4. Bertsekas, D.P.: Constrained Optimization and Lagrange Multiplier Methods. Academic Press, Cambridge (2014)
5. Bhargava, A., Ganti, R., Nowak, R.: Active algorithms for preference learning problems with multiple populations. CoRR arXiv:1603.04118 (2016)
6. Chen, S.Y., Yu, Y., Da, Q., Tan, J., Huang, H.K., Tang, H.H.: Stabilizing reinforcement learning in dynamic environment with application to online recommendation. In: SIGKDD (2018)
7. Chintala, S.: An overview of deep learning frameworks and an introduction to pytorch (2017)
8. Das, M., Morales, G.D.F., Gionis, A., Weber, I.: Learning to question: leveraging user preferences for shopping advice. In: SIGKDD (2013)
9. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. J. ACM **19**(2), 248–264 (1972)
10. Ganti, R., Rao, N.S., Balzano, L., Willett, R., Nowak, R.D.: On learning high dimensional structured single index models. In: AAAI (2017)
11. Gardner, W.A.: Learning characteristics of stochastic-gradient-descent algorithms: a general study, analysis, and critique. IEEE Trans. Signal Process. **6**(2), 113–133 (1984)
12. Grbovic, M., Cheng, H.: Real-time personalization using embeddings for search ranking at airbnb. In: SIGKDD (2018)
13. He, X., Chua, T.S.: Neural factorization machines for sparse predictive analytics. In: SIGIR (2017)
14. He, X., He, Z., Du, X., Chua, T.S.: Adversarial personalized ranking for recommendation. In: SIGIR (2018)
15. He, X., Zhang, H., Kan, M.Y., Chua, T.S.: Fast matrix factorization for online recommendation with implicit feedback. In: SIGIR (2016)
16. Huang, Y., Cui, B., Zhang, W., Jiang, J., Xu, Y.: TencentRec: real-time stream recommendation in practice. In: SIGMOD (2015)
17. Kang, D., Payor, J.: Flow rounding. arXiv:1507.08139 (2015)
18. Kempe, D., Kleinberg, J., Tardos, É.: Maximizing the spread of influence through a social network. In: SIGKDD. ACM (2003)
19. Li, M., Bao, Z., Sellis, T., Yan, S., Zhang, R.: Homeseeker: a visual analytics system of real estate data. J. Vis. Lang. Comput. **45**, 1–16 (2018)
20. Li, Y., Fan, J., Wang, Y., Tan, K.L.: Influence maximization on social graphs: a survey. IEEE Trans. Knowl. Data Eng. **30**(10), 1852–1872 (2018)
21. Qian, L., Gao, J., Jagadish, H.V.: Learning user preferences by adaptive pairwise comparison. Proc. VLDB Endow. **8**(11), 1322–1333 (2015)

22. Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-Thieme, L.: BPR: bayesian personalized ranking from implicit feedback. In: UAI (2009)
23. Salakhutdinov, R., Mnih, A., Hinton, G.: Restricted boltzmann machines for collaborative filtering. In: ICDM (2007)
24. Tang, Y., Shi, Y., Xiao, X.: Influence maximization in near-linear time: a martingale approach. In: SIGMOD. ACM (2015)
25. Tang, Y., Xiao, X., Shi, Y.: Influence maximization: near-optimal time complexity meets practical efficiency. In: SIGMOD. ACM (2014)
26. Teevan, J., Dumais, S.T., Horvitz, E.: Potential for personalization. ACM Trans. Computer-Human Interact. (TOCHI) **17**(1), 4 (2010)
27. Udell, M., Boyd, S.: Maximizing a sum of sigmoids. Optim. Eng (2013)
28. Wang, W., Yin, H., Huang, Z., Wang, Q., Du, X., Nguyen, Q.V.H.: Streaming ranking based recommender systems. In: SIGIR (2018)
29. Weston, J., Bengio, S., Usunier, N.: Wsabie: scaling up to large vocabulary image annotation. In: IJCAI (2011)
30. Yang, L., Hsieh, C.K., Yang, H., Pollak, J.P., Dell, N., Belongie, S., Cole, C., Estrin, D.: Yum-me: a personalized nutrient-based meal recommender system. ACM Trans. Inf. Syst. (TOIS) **36**(1), 7 (2017)
31. Yin, H., Cui, B., Chen, L., Hu, Z., Zhou, X.: Dynamic user modeling in social media systems. ACM Trans. Inf. Syst. (TOIS) **33**(3), 10 (2015)
32. Yin, H., Zhou, X., Cui, B., Wang, H., Zheng, K., Nguyen, Q.V.H.: Adapting to user interest drift for POI recommendation. IEEE Trans. Knowl. Data Eng. **28**(10), 2566–2581 (2016)
33. Zhao, X., Xia, L., Zhang, L., Ding, Z., Yin, D., Tang, J.: Deep reinforcement learning for page-wise recommendations. In: RecSys (2018)
34. Zhao, X., Zhang, L., Ding, Z., Xia, L., Tang, J., Yin, D.: Recommendations with negative feedback via pairwise deep reinforcement learning. In: SIGKDD (2018)
35. Zhou, K., Yang, S.H., Zha, H.: Functional matrix factorizations for cold-start recommendation. In: SIGIR (2011)