# A Light-speed Linear Program Solver for Personalized Recommendation with Diversity Constraints

**author names withheld**

**Under Review for OPT 2022**


## Abstract

We study a structured linear program (LP) that emerges in the need of ranking candidates or items in personalized recommender systems. Since the candidate set is only known in real time, the LP also needs to be solved in real time. Latency and user experience are major considerations, requiring the LP to be solved within just a few milliseconds. Although typical instances of the problem are not very large in size, this stringent time limit is beyond the capability of most existing commercial LP solvers, which can take 20 milliseconds or more to find a solution. Thus, reliable methods that address the real-world complication of latency become necessary. In this paper, we propose a fast specialized LP solver for a structured problem with diversity constraints. Our method solves the dual problem, making use of the piece-wise affine structure of the dual objective function, with an additional screening technique that helps reduce the dimensionality of the problem as the algorithm progresses. Experiments reveal that our method can solve the problem within roughly 1 millisecond, yielding a 20x improvement in speed over the most performant standard LP solvers. This speed-up can help improve the quality of recommendations without affecting user experience, highlighting how optimization can provide solid orthogonal value to machine-learned recommender systems.


## 1. Introduction

Linear programs (LPs) arise naturally in several applications across various scientific and industrial disciplines. Examples include scheduling [16], matching entities [7, 11, 22], network flow [4, 9], recommender systems [1–3, 18] and many others [13, 14]. Despite the significant development of LP solvers after decades of research, modern applications bring new challenges in solving LP problems, either with extreme-scale datasets [8], or with strict restrictions on time budget (e.g. in an online setting where LPs need to be solved in real-time). These challenges have sparked recent explorations for more efficient algorithms to solve LPs in both academia and industry, see e.g. [5, 6, 8, 17, 19, 21].

In this paper, we focus on a framework for re-ranking candidates in recommender systems under diversity or grouping constraints [12]. Consider a social media platform that makes personalized recommendations for its users. Suppose there are $m$ candidates (which could be items [3] or people [22]) for generating recommendations for user $u$; each candidate $i$ has a utility score $c_i \in \mathbb{R}$ that is generated by a machine learning model, representing the probability that $u$ may interact with this candidate. Suppose there are $n$ slots to present the recommendations, with weight $w_j > 0$ representing the importance of the slot $j$. In particular, we assume $w_1 > w_2 > \cdots > w_n$, i.e., slots in the 'front' are more important. We also make the realistic assumption that $n$ is much smaller

© .

than $m$, for example, $n$ equals $10 \sim 20$, while $m$ can range from hundreds to thousands. The recommendations and scores are specific to a given user $u$, and vary across users.

The primary goal of the system is to find an assignment of the $m$ candidates into the $n$ slots so as to maximize the total weighted utility score. We formulate the LP problem per user $u$, and solve the LPs for all different users in parallel. Let matrix $X^u \in \mathbb{R}^{m \times n}$ represent a randomized[1] assignment for each user $u$'s recommendations, where each $X^u_{ij} \in [0, 1]$ stands for the probability that candidate $i$ is assigned to slot $j$. For simplicity, we remove the superscript of $u$ in all the following notations. Then for each user $u$, the recommendation problem can be formulated as:

$$\max_{X \in \mathbb{R}^{m \times n}} \; c^\top X w \quad \text{s.t. } X \in \mathcal{S}_{m,n} \tag{1}$$

where $c = (c_1, c_2, ..., c_m)^\top \in \mathbb{R}^m$, $w = (w_1, w_2, ..., w_n)^\top \in \mathbb{R}^n$, and the set $\mathcal{S}_{m,n}$ is defined by

$$\mathcal{S}_{m,n} := \{X \in \mathbb{R}^{m \times n} \mid X 1_n \leq 1_m,\ 1_m^\top X = 1_n^\top,\ X \geq 0\} \tag{2}$$

where $1_k$ is the vector in $\mathbb{R}^k$ with all coordinates being 1 (for any $k \geq 1$). The constraint $X 1_n \leq 1_m$ implies that for each candidate $i$, the probability that $i$ is assigned to some slot is at most 1. The constraint $1_m^\top X = 1_n^\top$ implies that each slot must be assigned a candidate. Note that (1) has a simple closed-form solution: let $\sigma^*$ be a permutation on $[m]$ such that $c_{\sigma^*(1)} \geq c_{\sigma^*(2)} \geq \cdots \geq c_{\sigma^*(m)}$, then matrix $X^*$ with $X^*_{\sigma^*(j),j} = 1$ for all $j \in [n]$ and all other entries being 0, is an optimal solution of (1). In other words, the candidates with the top $n$ scores are placed in the top $n$ slots, in descending order of the scores.

In the system described above, only utility scores are used to make assignments. Recommendations obtained this way often lead to selection bias and can favor a particular group in terms of the overall ranking. For example, different kinds of people, different activity levels across the platform, different types of content etc. may get a very different exposure or placement in the rankings. Average ranks of one group may be much higher than those of another group when the system is solely optimized for overall utility which increases homogeneity over time.

To diversify the recommendations, we introduce an additional *diversity constraint* to this problem. Let $a_i \in \mathbb{R}$ denote the diversity feature of the candidate $i$. The following linear program restricts the range of the weighted summation of the diversity features:

$$\max_{X \in \mathbb{R}^{m \times n}} \; c^\top X w \quad \text{s.t. } X \in \mathcal{S}_{m,n},\ b_1 \leq a^\top X w \leq b_2. \tag{3}$$

where $a = (a_1, a_2, ..., a_m)^\top \in \mathbb{R}^m$, and $b_1$ and $b_2$ are chosen lower and upper bounds on the total weighted diversity. This allows us to inject any diversity restriction we want with appropriate choices on $a, b_1$ and $b_2$. For example, the constraint can be the impression gap between certain groups of people from different industries, or exposure gap between different genres of content. A few concrete examples of real-world use cases which can benefit from our specialized solver are discussed in appendix (Section E).

Unlike (1), the LP in (3) does not have a closed-form solution. For problem (3) with small or medium size, existing commercial solvers like Gurobi [15] designed for generic LPs can solve the problem in roughly 20 milli-seconds (on an example with $m = 100$, $n = 10$). However, for an online application where data $c_i$ (and the set of candidates themselves) are frequently updated, the

1. We use the term randomized to refer to the fact that the entries of $X$ can be fractional

personalized LP problem (3) needs to be solved in an "on-click" manner: when a user clicks a button to load the recommendations, the LP problem under the hood needs to be solved in an instant, i.e. ideally within $3 \sim 5$ milli-seconds. To the best of our knowledge, existing algorithms are unable to solve this problem within this stringent time limit.

In this paper, we introduce a specialized algorithm for problem (3) that can solve an instance with $m = 100$ and $n = 10$ within one milli-second, which is a 20x speedup compared to commercial LP solvers. Our method introduces a dual variable for the diversity constraint in (3) and solves the one-dimensional dual problem using a bisection method; when the iteration is close enough to the optimal dual solution, we trace the piece-wise linear structure of the dual objective function to locate the exact value of the dual optimal solution. The primal optimal solution can then be recovered from the dual optimal solution. We also introduce a screening technique that can detect and drop candidates that would not be assigned to any slots in the optimal solution. This screening technique can significantly accelerate the algorithm when the number of candidates $m$ is large.

The remainder of the paper is organized as follows. In Section 2, we simplify the program (3) and introduce our main algorithmic framework for solving it. In Section 3 we introduce the screening technique to further accelerate the algorithm. In Section 4, we present numerical results of our algorithm. We finally wrap up with a discussion in Section 5. Proofs of all results are placed in the appendix for easier reading of the main paper.

## 2. Dual problem and our solution approach

**Reduction to a one-sided diversity constraint:** In problem (3), the diversity constraint introduces both lower bound $b_1$ and upper bound $b_2$ on $a^\top X w$. However, to find the optimal solution of (3), either the lower bound or the upper bound can be dropped, as shown by the following lemma. Define $\mathcal{X}_0$ to be the set of optimal solution without the diversity constraint, i.e., $\mathcal{X}_0 := \operatorname{argmax}\{c^\top X w \mid X \in \mathcal{S}_{m,n}\}$.

**Lemma 1** *(i) If* $\min_{X \in \mathcal{X}_0}\{a^\top X w\} > b_2$*, then the set of optimal solutions of* (3) *is the same as*

$$\max_{X \in \mathbb{R}^{m \times n}} \; c^\top X w \quad \text{s.t.} \quad X \in \mathcal{S}_{m,n},\; a^\top X w \le b_2. \tag{4}$$

*(ii) If* $\max_{X \in \mathcal{X}_0}\{a^\top X w\} < b_1$*, then the set of optimal solutions of* (3) *is the same as*

$$\max_{X \in \mathbb{R}^{m \times n}} \; c^\top X w \quad \text{s.t.} \quad X \in \mathcal{S}_{m,n},\; a^\top X w \ge b_1. \tag{5}$$

*(iii) If there exists* $\bar{X} \in \mathcal{X}_0$ *such that* $b_1 \le a^\top \bar{X} w \le b_2$*, then* $\bar{X}$ *is an optimal solution of* (3)*.*

Lemma 1 provides a procedure to drop one side of the diversity constraint without changing the optimal solution. To use it in practice, we need to evaluate $\mathcal{X}_0$, which can be efficiently computed via a ranking of the coordinates of $c$. When there is no tie in the ranking of the coordinates of $c$, the optimal set $\mathcal{X}_0$ will only contain one point, say $\mathcal{X}_0 = \{\bar{X}\}$, and Lemma 1 removes redundant constraints.

**Properties of the dual objective function:** Having made use of the reduction result of Lemma 1, we now discuss how to solve problem (4). Note that problem (5) can also be written into a similar form as (4), with $a$ replaced by $-a$ and $b_2$ replaced by $-b_1$. To solve problem (4), we introduce a dual variable $\lambda \ge 0$ for the constraint $a^\top X^* w \le b_2$, and define the dual objective function:

$$g(\lambda) := \max_{X \in \mathcal{S}_{m,n}} c^\top X w - \lambda(a^\top X w - b_2) = \max_{X \in \mathcal{S}_{m,n}} (c - \lambda a)^\top X w + b_2 \lambda \tag{6}$$

then $g$ is a piece-wise linear convex function, and the dual problem of (4) is

$$\min_{\lambda \geq 0} \quad g(\lambda) \tag{7}$$

For each fixed $\lambda \geq 0$, the value $g(\lambda)$ can be evaluated by solving the maximization in (6), which can be computed in closed form via a sorting of the coordinates of $c - \lambda a$. Let $\mathcal{X}_\lambda$ be the set of optimal solutions of (6). Then the extreme points of $\mathcal{X}_\lambda$ are given by:

$$\begin{aligned}\operatorname{ext}(\mathcal{X}_\lambda) = \{X^{(\sigma)} \mid \sigma \text{ is a permutation on } [m] \text{ with} \\ (c - \lambda a)_{\sigma(1)} \geq (c - \lambda a)_{\sigma(2)} \geq \cdots \geq (c - \lambda a)_{\sigma(n)}\}\end{aligned}$$

where $X^{(\sigma)} \in \mathcal{S}_{m,n}$ is the matrix with $X_{\sigma(i),i} = 1$ for $i \in [n]$ and all other entries being 0. If there are no ties among the top $n$ values of $c - \lambda a$, the set $\operatorname{ext}(\mathcal{X}_\lambda)$ (and also $\mathcal{X}_\lambda$) contains only one element, i.e., (6) has a unique solution. Once $\mathcal{X}_\lambda$ is computed, the derivatives of $g$ can also be evaluated, via a standard application of Danskin's theorem [10].

**Lemma 2** *For any $\lambda > 0$, the left derivative and right derivative of $g$ are given by*

$$g'_-(\lambda) = b_2 - \max_{X \in \mathcal{X}_\lambda} \{a^\top X w\}, \quad \textit{and} \quad g'_+(\lambda) = b_2 - \min_{X \in \mathcal{X}_\lambda} \{a^\top X w\}. \tag{8}$$

Note that $g$ is a piece-wise linear function. For $\lambda$ in the interior of a linear block, the left and right derivatives are the same: $g'_-(\lambda) = g'_+(\lambda)$. For $\lambda$ at the intersection of two linear blocks (called a *kink* of $g$), it holds $g'_-(\lambda) < g'_+(\lambda)$. In particular, if at some $\lambda > 0$, the top $n$ coordinates of $c - \lambda a$ is unique, then $\mathcal{X}_\lambda$ contains only one point, and by (8), we know that $\lambda$ must lie in the interior of a linear block. Furthermore, the top $n$ values of $c - \lambda a$ are the same for all other $\lambda$ in the interior of the same linear block. This motivates a method to trace the piece-wise linear structure of $g$ via observing the top $n$ coordinates of $c - \lambda a$ as $\lambda$ changes.

For $\lambda > 0$, let $K_R(\lambda)$ be the smallest kink of $g$ that is larger than $\lambda$, and $K_L(\lambda)$ be the largest kink of $g$ that is smaller than $\lambda$. For any vector $y \in \mathbb{R}^m$, let $\mathcal{T}_n(y)$ be the set of top $n$ coordinates of $y$ (including ties), that is,

$$\mathcal{T}_n(y) := \Big\{ i \in [m] \;\Big|\; \#\{j \in [m] | y_j < y_i\} \leq n - 1 \Big\}. \tag{9}$$

For a given $\lambda > 0$, letting $z := c - \lambda a$, then $K_R(\lambda)$ and $K_L(\lambda)$ can be computed in closed form:

$$K_R(\lambda) = \lambda + \min_{(i,j)} \Big\{ \frac{z_i - z_j}{a_i - a_j} \;\Big|\; j \in \mathcal{T}_n(z),\ i \in [m],\ a_i - a_j > 0,\ z_i - z_j > 0 \Big\} \tag{10}$$

$$K_L(\lambda) = \lambda - \min_{(i,j)} \Big\{ \frac{z_i - z_j}{a_j - a_i} \;\Big|\; j \in \mathcal{T}_n(z),\ i \in [m],\ a_j - a_i > 0,\ z_i - z_j > 0 \Big\} \tag{11}$$

As shown in (11) and (10), one can trace the exact value of the kinks of $g$ starting from an arbitrary $\lambda > 0$. As long as $\#\mathcal{T}_n(y) = O(n)$, the computation cost of $K_R(\lambda)$ and $K_L(\lambda)$ is $O(mn)$ operations.

**Bisection and tracing for the dual problem:** To solve problem (4), we solve the one-dimensional dual problem (7) using a bisection method, which maintains and updates an interval $[\lambda_{\min}, \lambda_{\max}]$ that contains the dual optimal solution $\lambda^*$ (for simplicity, assume it is unique). In each iteration, the algorithm computes the value and derivatives of $g$ at the midpoint $\lambda = (\lambda_{\min} + \lambda_{\max})/2$ of this

interval; if $g'_+(\lambda) < 0$, then we know $\lambda < \lambda^*$, and update the interval to $[\lambda, \lambda_{\max}]$; if $g'_-(\lambda) > 0$, then we know $\lambda > \lambda^*$, and update the interval to $[\lambda_{\min}, \lambda]$; otherwise it holds $g'_-(\lambda) \leq 0 \leq g'_+(\lambda)$, which indicates that $\lambda = \lambda^*$.

In the above, the bisection method shrinks the interval by *a* half in each iteration, so the value of *the* dual optimal solution can be quickly located in a small interval after a few iterations. When the interval is small enough, the algorithm computes a nearby kink following (11) or (10), and checks if this kink is the exact optimal dual solution (using the signs of left and right derivatives). If the kink is the optimal solution, the algorithm is terminated immediately; otherwise we continue on the bisection iterations. The details of the above mentioned algorithm are summarized in Algorithm 1 in the appendix. Once the dual problem is solved, the primal solution can be recovered in closed form – see Section D in the appendix for details.

Note that in this algorithm, the major algebraic operations are conducted in the computations of $\mathcal{X}_\lambda$ and $K_L(\lambda)$ (or $K_R(\lambda)$). In each iteration of Algorithm 1, to compute $\mathcal{X}_\lambda$, one needs to sort the vector $c - \lambda a$, which takes $m \log(m)$ operations. When $\lambda_{\max} - \lambda_{\min} < \Delta$, an additional tracing step is needed to compute $K_L(\lambda)$ or $K_R(\lambda)$, which takes $O(mn)$ operations. Therefore, the cost of Algorithm 1 scales (at least) linearly with respect to $m$ which can get expensive when $m$ is very large. In Section 3 we discuss how to accelerate the algorithm in such a case.

## 3. A screening step when $m$ is large

When the number of candidates $m$ is large, we propose a screening technique that is used in the process of Algorithm 1, which can quickly detect and discard candidates that will not be in the optimal solution. Note that the candidates that are present in the optimal solution must be the top $n$ coordinates of the vector $c - \lambda^* a$, where $\lambda^*$ is the dual optimal solution. Although $\lambda^*$ is not available at the beginning, after a few iterations of Algorithm 1, we have an interval $[\lambda_{\min}, \lambda_{\max}]$ that can approximately locate the value of $\lambda^*$. If at some point $\lambda$ that is close to $\lambda^*$, we observe that some candidate $i$ has a very low rank in the sorting of $c - \lambda a$ (far from being top $n$), it is a strong evidence that maybe $i$ is also not in the top $n$ of vector $c - \lambda^* a$. The following lemma provides a precise statement underlying this intuition.

**Lemma 3** *Let $\lambda^*$ be the solution of (7), and let $\lambda_{\min}$, $\lambda_{\max}$ satisfy $\lambda_{\min} \leq \lambda^* \leq \lambda_{\max}$. For any $i \in [m]$, if there exist indices $i_1, ...., i_n \in [m]$ such that*

$$(c - \lambda_{\min} a)_i < \min_{k \in [n]} \{(c - \lambda_{\min} a)_{i_k}\}, \quad \textit{and} \quad (c - \lambda_{\max} a)_i < \min_{k \in [n]} \{(c - \lambda_{\max} a)_{i_k}\} \tag{12}$$

*Then the $i$-th row of the solution of (4) is zero.*

According to Lemma 3, to prove that candidate $i \in [m]$ is not selected in the optimal assignment, it suffices to find a set of "good" indices $i_1, ...., i_n$ such that (12) holds true. Different choices of $i_1, ...., i_n$ may lead to different quality of screening. When the interval $[\lambda_{\min}, \lambda_{\max}]$ is small enough, if we take $i_1, ...., i_n$ to be the top $n$ coordinates of $c - \lambda a$ for some $\lambda \in [\lambda_{\min}, \lambda_{\max}]$, then they may also have a high rank in the sortings of both $c - \lambda_{\min} a$ and $c - \lambda_{\max} a$, and can be used as good candidates for the screening in (12).

We apply a screening step in each iteration of Algorithm 1. In particular, we take $i_1, ...., i_n$ as the top $n$ coordinates of $c - \bar{\lambda} a$ for $\lambda = (\lambda_{\min} + \lambda_{\max})/2$, and check the inequalities in (12) for

all other $i \in [m]$. If for some $i$ the inequalities (12) hold true, then $i$ can be discarded, and the dimension of the problem is reduced. This is summarized in Algorithm 2 in appendix.

## 4. Numerical experiments

We conduct experiments on synthetic and real datasets to show the efficiency of our algorithm.
**Speedup by the screening procedure**. To show the advantage of the screening steps introduced in Section 3, we compare the runtimes of Algorithms 1 and 2 on synthetic examples with $n = 10$ and different values of $m$. See Section E for data generation procedure.

| | $n = 10$ | | $n = 30$ | |
|---|---|---|---|---|
| | Alg 2 | Gurobi | Alg 2 | Gurobi |
| $m = 100$ | 0.8 | 14.8 | 1.7 | 25.4 |
| $m = 300$ | 1.2 | 20.3 | 2.1 | 60.1 |
| $m = 1000$ | 2.5 | 52.9 | 3.7 | 166.6 |
| $m = 3000$ | 5.6 | 115.8 | 7.3 | 526.9 |
| $m = 10000$ | 14.3 | 336.5 | 19.1 | 1765.3 |

Table 1: Runtime comparison of Algorithm 2 with Gurobi. Reported runtimes are in milli-seconds.

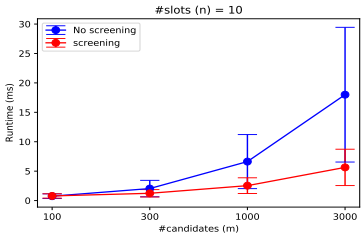


Figure 1: Runtime comparison of Algorithm 1 (no screening) and Algorithm 2 (screening).

Figure 1 shows runtime comparison of Algorithms 1 and 2 with the number of slots $n = 10$ and number of candidates $m \in \{100, 300, 1000, 3000\}$. The results are the average of 20 independent replications (with standard errors). As shown in Figure 1, when $m = 100$, the runtimes of Algorithm 1 and Algorithm 2 are close (both are roughly one ms). For larger values of $m$, Algorithm 2 is faster than Algorithm 1, and the gap widens as $m$ increases. In particular, when $m = 3000$, Algorithm 2 achieves a $3 \sim 4$ times acceleration over Algorithm 1.
**Comparison with Gurobi**. We compare the performance of Algorithm 2 with the commercial LP solver Gurobi [15] on the synthetic data generated in the same way as described in the last subsection. Gurobi is used under the default settings. Table 1 compares the runtimes of Algorithm 2 and Gurobi on instances of problem (3) with $m \in \{100, 300, 1000, 3000, 10000\}$ and $n \in \{10, 30\}$. As shown by Table 1, when $n = 10$, Algorithm 2 is roughly 20 times faster than Gurobi, and the acceleration is more significant for larger values of $m$. For $n = 30$, the gap between Algorithm 2 and Gurobi is even larger; in particular, for the largest example with $m = 10000$ and $n = 30$, Algorithm 2 achieves more than 90 times acceleration over Gurobi.

## 5. Conclusion

We develop a highly efficient specialized linear program solver for recommender system with diversity constraints, where recommendations are made in real time with time budget in milli-seconds. Our solver is efficient in both runtime and memory requirement, with an acceleration of more than 20 times over Gurobi. Although our primary application appears in personalized recommendation, this algorithm can be applied in other ranking applications where the problem can be formulated in the form (3).

## References


[1] Deepak Agarwal, Bee-Chung Chen, Pradheep Elango, and Xuanhui Wang. Personalized click shaping through lagrangian duality for online recommendation. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 485–494, 2012.

[2] Deepak Agarwal, Bee-Chung Chen, Rupesh Gupta, Joshua Hartman, Qi He, Anand Iyer, Sumanth Kolar, Yiming Ma, Pannagadatta Shivaswamy, Ajit Singh, et al. Activity ranking in linkedin feed. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1603–1612, 2014.

[3] Deepak Agarwal, Bee-Chung Chen, Qi He, Zhenhao Hua, Guy Lebanon, Yiming Ma, Pannagadatta Shivaswamy, Hsiao-Ping Tseng, Jaewon Yang, and Liang Zhang. Personalizing linkedin feed. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1651–1660, 2015.

[4] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows. 1988.

[5] David Applegate, Mateo Díaz, Oliver Hinder, Haihao Lu, Miles Lubin, Brendan O'Donoghue, and Warren Schudy. Practical large-scale linear programming using primal-dual hybrid gradient. *Advances in Neural Information Processing Systems*, 34:20243–20257, 2021.

[6] David Applegate, Oliver Hinder, Haihao Lu, and Miles Lubin. Faster first-order primal-dual methods for linear programming using restarts and sharpness. *arXiv preprint arXiv:2105.12715*, 2021.

[7] Eduardo M Azevedo and E Glen Weyl. Matching markets in the digital age. *Science*, 352 (6289):1056–1057, 2016.

[8] Kinjal Basu, Amol Ghoting, Rahul Mazumder, and Yao Pan. Eclipse: An extreme-scale linear program solver for web-applications. In *International Conference on Machine Learning*, pages 704–714. PMLR, 2020.

[9] Mokhtar S Bazaraa, John J Jarvis, and Hanis D Sherali. *Linear programming and network flows*. John Wiley & Sons, 2008.

[10] J Frédéric Bonnans and Alexander Shapiro. Optimization problems with perturbations: A guided tour. *SIAM review*, 40(2):228–264, 1998.

[11] Fedor Borisyuk, Liang Zhang, and Krishnaram Kenthapadi. Lijar: A system for job application redistribution towards efficient career marketplace. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1397–1406, 2017.

[12] Jaime Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336, 1998.

[13] Rupesh Gupta, Guanfeng Liang, Hsiao-Ping Tseng, Ravi Kiran Holur Vijay, Xiaoyu Chen, and Rómer Rosales. Email volume optimization at linkedin. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106, 2016.

[14] Rupesh Gupta, Guanfeng Liang, and Rómer Rosales. Optimizing email volume for sitewide engagement. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1947–1955, 2017.

[15] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL https://www.gurobi.com.

[16] Fred Hanssmann and Sidney W Hess. A linear programming approach to production and employment scheduling. *Management science*, (1):46–51, 1960.

[17] Tianyi Lin, Shiqian Ma, Yinyu Ye, and Shuzhong Zhang. An admm-based interior-point method for large-scale linear programming. *Optimization Methods and Software*, 36(2-3): 389–424, 2021.

[18] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

[19] Rohan Ramanath, S. Sathiya Keerthi, Yao Pan, Konstantin Salomatin, and Kinjal Basu. Efficient algorithms for global inference in internet marketplaces. *CoRR*, abs/2103.05277, 2021. URL https://arxiv.org/abs/2103.05277.

[20] Ashudeep Singh and Thorsten Joachims. Fairness of exposure in rankings. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2219–2228, 2018.

[21] Chaobing Song, Cheuk Yin Lin, Stephen J Wright, and Jelena Diakonikolas. Coordinate linear variance reduction for generalized linear programming. *arXiv preprint arXiv:2111.01842*, 2021.

[22] Huanyang Zheng and Jie Wu. Online to offline business: urban taxi dispatching with passenger-driver matching stability. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 816–825. IEEE, 2017.

## Appendix A. Proof of Lemma 1

(i) Actually we can prove that both (3) and (4) have the same set of optimal solutions as the following problem:

$$\max_{X \in \mathbb{R}^{m \times n}} \quad c^\top X w \quad \text{s.t. } X \in \mathcal{S}_{m,n},\ a^\top X w = b_2. \tag{13}$$

Note that for any $X \in \mathcal{S}_{m,n}$ with $a^\top X w < b_2$, one can write $\tilde{X} \in \mathcal{S}_{m,n}$ as a convex combination of $X$ and some $X^0 \in \mathcal{X}_0$, such that $a^\top \tilde{X} w = b_2$ and $c^\top \tilde{X} w > c^\top X w$.

(ii) can be proved via a similar argument as the proof of (i);

(iii) holds true trivially.

## Appendix B. Proof of Lemma 3

Note that there exists $\alpha \in [0, 1]$ such that $\lambda^* = (1-\alpha)\lambda_{\min} + \alpha\lambda_{\max}$. Hence,

$$\begin{aligned}(c - \lambda^* a)_i &= (1-\alpha)(c - \lambda_{\min} a)_i + \alpha(c - \lambda_{\max} a)_i \\ &< (1-\alpha)(c - \lambda_{\min} a)_{i_k} + \alpha(c - \lambda_{\max} a)_{i_k} = (c - \lambda^* a)_{i_k}\end{aligned}$$

for all $k \in [n]$, where the inequality is by (12). As a result, we know that $i$ is not the top $n$ coordinates of $c - \lambda^* a$, and hence the $i$-th row of the solution of (4) is zero.

## Appendix C. Algorithm Details and Discussions

**Algorithm 1** Bisection + tracing to solve (7)

**Input**: Tolerances $\Delta > \delta \geq 0$.
Initially set $\lambda_{\min} = 0$, $\lambda_{\max} = \infty$ and $\lambda = 1$.
**while** $\lambda_{\max} - \lambda_{\min} > \delta$:
  Solve (6) and get $\mathcal{X}_\lambda$. Compute $g'_+(\lambda)$ and $g'_-(\lambda)$ using (8).
  **If** $g'_+(\lambda) > 0$:
    If $\lambda_{\max} - \lambda_{\min} < \Delta$:
      Compute $K_R(\lambda)$ and $g'_+(K_R(\lambda))$ and $g'_-(K_R(\lambda))$.
      If $g'_+(K_R(\lambda)) \leq 0 \leq g'_-(K_R(\lambda))$, **break** and **output** $\lambda^* = K_R(\lambda)$.
    If $\lambda_{\max} < \infty$, then set $\lambda_{\min} = \lambda$, and $\lambda = (\lambda_{\max} + \lambda_{\min})/2$;
    Else, $\lambda_{\max} = \infty$, then set $\lambda_{\min} = \lambda$ and $\lambda = 2\lambda$.
  **If** $g'_-(\lambda) < 0$:
    If $\lambda_{\max} - \lambda_{\min} < \Delta$:
      Compute $K_L(\lambda)$ and $g'_+(K_L(\lambda))$ and $g'_-(K_L(\lambda))$.
      If $g'_+(K_L(\lambda)) \leq 0 \leq g'_-(K_L(\lambda))$, **break** and **output** $\lambda^* = K_L(\lambda)$.
    Set $\lambda_{\max} = \lambda$ and $\lambda = (\lambda_{\max} + \lambda_{\min})/2$.
  **If** $g'_+(\lambda) \leq 0 \leq g'_-(\lambda)$:
    **output** $\lambda^* = \lambda$ as an optimal solution of (7).
**end while**
**Output** The interval $[\lambda_{\min}, \lambda_{\max}]$.

**Algorithm 2** Bisection + tracing + screening to solve (7)

**Input**: Tolerances $\Delta > \delta > 0$.
Initially set $\lambda_{\min} = 0$, $\lambda_{\max} = \infty$ and $\lambda = 1$. Set $\mathcal{I} = [m]$.
**while** $\lambda_{\max} - \lambda_{\min} > \delta$:
  Solve (6) and get $\mathcal{X}_\lambda$ (with $\mathcal{I}$ in place of $[m]$).
  Compute $g'_+(\lambda)$ and $g'_-(\lambda)$ using (8).
  **If** $g'_+(\lambda) > 0$:
    If $\lambda_{\max} - \lambda_{\min} < \Delta$:
      Compute $K_R(\lambda)$ and $g'_+(K_R(\lambda))$ and $g'_-(K_R(\lambda))$ (with $\mathcal{I}$ in place of $[m]$).
      If $g'_+(K_R(\lambda)) \le 0 \le g'_-(K_R(\lambda))$, **break** and **output** $\lambda^* = K_R(\lambda)$.
    If $\lambda_{\max} < \infty$, then set $\lambda_{\min} = \lambda$, and $\lambda = (\lambda_{\max} + \lambda_{\min})/2$;
    Else, $\lambda_{\max} = \infty$, then set $\lambda_{\min} = \lambda$ and $\lambda = 2\lambda$.
  **If** $g'_-(\lambda) < 0$:
    If $\lambda_{\max} - \lambda_{\min} < \Delta$:
      Compute $K_L(\lambda)$ and $g'_+(K_L(\lambda))$ and $g'_-(K_L(\lambda))$ (with $\mathcal{I}$ in place of $[m]$).
      If $g'_+(K_L(\lambda)) \le 0 \le g'_-(K_L(\lambda))$, **break** and **output** $\lambda^* = K_L(\lambda)$.
    Set $\lambda_{\max} = \lambda$ and $\lambda = (\lambda_{\max} + \lambda_{\min})/2$.
  **If** $g'_+(\lambda) \le 0 \le g'_-(\lambda)$:
    **output** $\lambda^* = \lambda$ as an optimal solution of (7).
  (Screening to shrink $\mathcal{I}$)
  **If** $\lambda_{\max} < \infty$ :
    1. Let $i_1, ..., i_n$ be the indices in $\mathcal{I}$ corresponding to top $n$ elements of $(c - \lambda a)$.
    (Break ties arbitrarily, if any).
    2. Find out

$$\mathcal{I}_{drop} := \left\{ i \in \mathcal{I} \;\middle|\; \text{Inequalities in (12) are satisfied} \right\}$$

    and update $\mathcal{I} = \mathcal{I} \setminus \mathcal{I}_{drop}$.
**end while**
**Output** The interval $[\lambda_{\min}, \lambda_{\max}]$.

Note that in Algorithm 2, a set of active indices $\mathcal{I}$ is maintained and updated in each iteration. For the computations of $\mathcal{X}_\lambda$ and $K_L(\lambda)$ (or $K_R(\lambda)$), only indices in $\mathcal{I}$ are considered, which will have computational costs $O(|\mathcal{I}| \log(|\mathcal{I}|))$ and $|\mathcal{I}|n$ respectively. If the screening step can discard a large set of candidates in the first few iterations of the Algorithm 2, then the overall cost of Algorithm 2 is significantly lower than that of Algorithm 1.

## Appendix D. Recovering primal solutions

If Algorithm 1 finds the exact dual optimal solution $\lambda^*$, the (primal) optimal solution of (4) can be recovered directly. Note that Algorithm 1 also makes $\mathcal{X}_{\lambda^*}$ available. Since $\lambda^*$ is the dual optimal solution, $g'_-(\lambda^*) \le 0$ and $g'_+(\lambda^*) \ge 0$, implying $\min_{X \in \mathcal{X}_{\lambda^*}} \{a^\top X w\} \le b_2 \le \max_{X \in \mathcal{X}_{\lambda^*}} \{a^\top X w\}$. Let $X^1$ and $X^2$ be two extreme points of $\mathcal{X}_{\lambda^*}$ such that $a^\top X^1 w \le b_2$ and $a^\top X^2 w \ge b_2$. Define $\rho := (b_2 - a^\top X^2 w)/(a^\top X^1 w - a^\top X^2 w)$ and $X^* := \rho X^1 + (1-\rho) X^2$, then $X^* \in \mathcal{S}_{m,n}$ and $a^\top X^* w = b_2$. By the KKT condition, $X^*$ is an optimal solution of problem (4).

## Appendix E. Data generation

We set the weights $w$ as $w_j = 1/\log(1+j)$ for all $j \in [n]$ as commonly used in the Discounted Cumulative Gain (DCG) measure [20]. We generate vector pairs $(a, c) \in \mathbb{R}^{m\times 2}$ such that $\{(a_i, c_i)\}_{i=1}^n$ are i.i.d. bi-variate Gaussian random variable with mean $\mathbb{E}(a_i) = \mathbb{E}(c_i) = 0$, variance $\mathrm{var}(a_i) = \mathrm{var}(c_i) = 1$ and covariance $\mathrm{cov}(a_i, c_i) = \alpha$ for some chosen $\alpha \in (0, 1)$. Note that $\alpha$ controls the angle between the vectors $a$ and $c$; for a larger value of $\alpha$, the angle between $a$ and $c$ are smaller (with high probability). We set $\alpha = 0.5$. Finally, to generate $b_1$ and $b_2$, we compute $\bar{X} \in \mathrm{argmax}_{X\in\mathcal{S}_{m,n}}\{c^\top Xw\}$, and evaluate $a^\top \bar{X}w$; if $a^\top \bar{X}w > 0$ (which happens with high probability if $\alpha$ is not too small), take $b_2 = -b_1 = 0.8(a^\top \bar{X}w)$; otherwise drop this instance and regenerate one with $a^\top \bar{X}w > 0$. With these values of $b_1$ and $b_2$, the constraint $b_1 \le a^\top Xw \le b_2$ must be active at the optimal solution of (3).

## Appendix F. Performance on Real Datasets

We evaluate the performance of Algorithm 2 on proprietary real-world datasets that arise in a recommender system of people recommendations to generate connections. To improve the diversity across different groups of the recommended people, we encode the group a candidate $i$ belongs to as $a_i \in \{-1, 0, 1\}$. Here $a_i = 1$ represents group 1, $a_i = -1$ stands for group 2, and $a_i = 0$ denotes other groups. Information about the specific feature according to which such groups are formed are withheld here due to privacy considerations. We choose some $b > 0$ and set $b_2 = -b_1 = b$ in problem (3).

We focus on two instances of the problem with $n = 10$ slots and $m = 500$ or $1000$ candidates respectively. To evaluate the robustness of Algorithm 2, we created 20 noised replications of problem (3) with noised scores $c_i$ ($i \in [m]$). More precisely, we set $c = c^0 + 0.2\frac{\|c^0\|_2}{\|\epsilon\|_2}\epsilon$, where $c^0 \in \mathbb{R}^m$ is the original utility scores, and $\epsilon \in \mathbb{R}^m$ is a random vector with i.i.d. $N(0, 1)$ coordinates. Other data ($a$ and $w$) assume the same values as the original data.

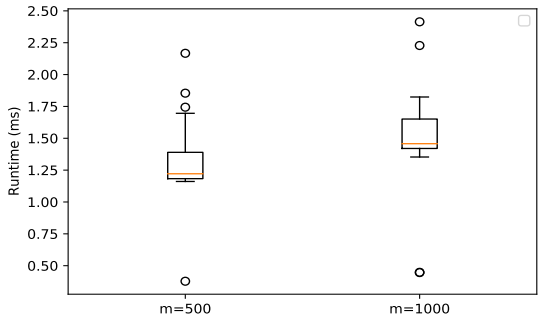


Figure 2: Performance of Algorithm 2 on real datasets

Figure 2 presents the box plot of Algorithm 2 runtimes for these 20 replications. For $m = 500$, most replications have a runtime between 1 ms and 2.25 ms; there are one or two replications that have a runtime below 0.5 ms – these replications correspond to the case when the diversity constraint $b_1 \le a^\top Xw \le b_2$ is not active at the optimal solution, and Algorithm 2 terminates earlier. A similar

argument applies to the examples with $m = 1000$. As a conclusion, Algorithm 2 is quite robust across different runs with noised data.

## Appendix G. Real-world Use Cases

Below, we present a few concrete real-world use cases which can potentially leverage and benefit from our specialized solver.

One might be interested in providing more diverse exposure to different types of content creators on a social network platform. For example, the platform might want to show diverse video recommendations from creators from different countries or states.

Introducing systematic diversity in movie recommendations can lead to better exploration and user retention. For instance, the recommendations can span across genres such as "romance", "action", "sci-fi", "comedy", etc., to better exploit the interests of the users.

Our algorithms can also find applications related to ride-sharing apps. Experienced and frequent drivers on such platforms usually have enough exposure to riders. However, more balanced exposure to new or less frequent drivers may help ride-sharing platforms achieve more sustainable growth. Of course, the same strategy may also find relevance in other two-sided marketplace platforms, such as house sharing, restaurant recommendations, etc.