Optimizing Long-term Social Welfare in Recommender Systems (Supplemental)

Martin Mladenov [1] **Elliot Creager** [2 3 *] **Omer Ben-Porat** [4 †] **Kevin Swersky** [1] **Richard Zemel** [2 3] **Craig Boutilier** [1]

A. Algorithms and Proofs

A.1. Greedy Optimization and Theorem 1

A.1.1. Preliminaries

Before we begin the proof, we make a few notational modifications to significantly simplify it. Let $\mathcal{C}$ be a set of content providers (hereinafter providers), $\mathcal{U}$ a set of users, and $A \in \mathbb{R}^{|\mathcal{U}|\times|\mathcal{C}|}$ a utility matrix. Furthermore, let $D : \mathcal{U} \to \mathbb{N}$ be a user demand function (specifying how many queries a user u submits to the system) and ν_c for $c \in \mathcal{C}$ be the provider survival threshold, indicating how many queries the provider needs to receive in order to be viable. For this section, we will make the following simplifying assumptions:

- every user has exactly one unique query during the epoch, and
- every user's view contributes exactly one unit towards a provider's viability.

Under these assumptions, the set of queries becomes identical to the set of users ($\mathcal{U} = \mathcal{Q}$), and $\bar{Q}(q_u) = 1$. We proceed to prove the submodularity of user welfare as a function of the provider set subject to the above restrictions. After that, we discuss the reduction of Problem 4 to this restricted case.

The fairness dynamics problem is then to find a matching such that

$$\begin{aligned} X^* = \arg\max_{X,Y} & \sum_{u\in\mathcal{U}} \left(\sum_{t=1}^{D(u)} \sum_{c\in\mathcal{C}} A_{uc} X_{uct} \right) \\ \text{subject to } & \sum_{c\in\mathcal{C}} X_{uct} = 1 \quad \forall u \in \mathcal{U}, t \in \{1,\dots,D(u)\} \\ & X_{uct} \le Y_c \quad \forall u \in \mathcal{U}, c \in \mathcal{C} \\ & \sum_{u\in\mathcal{U}} \sum_{t=1}^{D(u)} X_{uct} \ge \nu_c Y_c, \quad \forall c \in \mathcal{C} \\ & X_{uct}, Y_c \in \{0,1\}, \quad \forall u \in \mathcal{U}, c \in \mathcal{C}, t \in [1,\dots,D(u)], \end{aligned} \tag{6}$$

The problem (6) is a hard combinatorial problem, so the question is if we can derive good heuristics for solving it. Of

[*]Work done as a student researcher at Google Research [†]Work partially done as an intern at Google Research [1]Google Research [2]University of Toronto [3]Vector Institute [4]Technion. Correspondence to: Martin Mladenov <mmladenov@google.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

particular interest is the following greedy heuristic: let $C \subseteq \mathcal{C}$ and define $g : 2^{\mathcal{C}} \to \mathbb{R}$ as

$$\begin{aligned} g(C) \mapsto \max_{X} & \sum_{u \in \mathcal{U}} \left(\sum_{t=1}^{D(u)} \sum_{c \in C} A_{uc} X_{uct} \right) \\ \text{subject to } & \sum_{c \in C} X_{uct} = 1 \quad \forall u \in \mathcal{U}, t \in [1, \ldots, D(u)] \\ & \sum_{u \in \mathcal{U}} \sum_{t=1}^{D(u)} X_{uct} \geq \nu_c \quad \forall c \in C \\ & X_{uct} \in \{0, 1\}, \quad \forall u \in \mathcal{U}, c \in \mathcal{C}, t \in [1, \ldots, D(u)]. \end{aligned} \tag{7}$$

That is, $g(C)$ is the best matching if the provider set $C \subset \mathcal{C}$ is fixed externally. Despite that (7) has binary constraints on X_{uct}, its constraint matrix is Totally Unimodular; hence, we are guaranteed that (7) is integral. The goal is then to start with $C = \emptyset$ and greedily add providers while $g(C)$ keeps improving. In order for this to work well, g would need to be sub-modular, which is precisely what we prove next.

Theorem 1. *For every two providers $c_0, c_1 \in \mathcal{C}$ and $C \subseteq \mathcal{C} \setminus \{c_0, c_1\}$, it holds that*

$$g(C \cup \{c_0, c_1\}) - g(C \cup \{c_1\}) \leq g(C \cup \{c_0\}) - g(C). \tag{8}$$

A.1.2. Proof of Theorem 1 in the Unit Case

Let us make a simplification: a user with $D(u)$ queries is equivalent to $D(u)$ independent users; thus, we will just work with an extended user set. We now present the terminology used in this proof. A matching $X : \mathcal{U} \to \mathcal{C}$, is a function from users to providers. We denote by $\mathcal{C}(X)$ the serving providers under X, i.e., $\mathcal{C}(X) = \{c \mid \exists u \in \mathcal{U}, X(u) = c\}$. Further, we say that a matching X is *feasible* if every provider in $\mathcal{C}(X)$ meets her threshold under X, namely, if for every $c \in \mathcal{C}(X)$ it holds that $|\{u \in \mathcal{U} \mid X(u) = c\}| \geq \nu_c$. We denote by $F(X)$ the value obtained for a feasible matching X in Problem (7) (note that X may not be optimal w.r.t.. $\mathcal{C}(X)$). In the rest of the proof, we rely on optimal matchings for $C, C \cup \{c_1\}$ and $C \cup \{c_0, c_1\}$ to construct a new matching, X^0. The active providers under X^0 are $C \cup \{c_0\}$ and, as we shall show, X^0 satisfies

$$g(C \cup \{c_0, c_1\}) - g(C \cup \{c_1\}) \leq F(X^0) - g(C). \tag{9}$$

The latter immediately implies Inequality (8), since by definition of g,

$$F(C \cup \{c_0, c_1\}) \leq \max_{X : \mathcal{C}(X) = C \cup \{c_0\}} F(X) = g(C \cup \{c_0\}).$$

We are now ready to develop the tools required for the proof. The next notion assists to succinctly quantify the difference in user utility between two matchings.

Definition 1. *Let X be and Y be two feasible matchings. We call a triplet (c, c', u) a relocation triplet w.r.t. X, Y if $X(u) = c$, $Y(u) = c'$ and $c \neq c'$.*

Importantly, two matchings define a unique set of (ordered) relocation triplets. conversely, a source matching and relocation triplets uniquely define the target matching.

Let X and X^1 denote (any) optimal matching induced by $g(C)$ and $g(C \cup \{c_1\})$ in Problem (7), respectively. We now construct a graph whose nodes are the providers and its edges correspond to relocation triplets w.r.t. X, X^1. Formally, let $G^1 = (\mathcal{C}, E^1, w)$ denote a directed multi-graph, where the set of nodes is $\mathcal{C}$; E^1 is the set of all relocation triplets w.r.t. X, X^1, where every triplet (c, c', u) forms a directed edge from $c = X(u)$ to $c' = X^1(u)$ with an ID of u; and the weight function w is defined by $w(c, c', u) = A_{uc'} - A_{uc}$. Observe that the number of users each provider c (a node in the graph) obtains under X^1 equals

$$|\{u \in \mathcal{U} \mid X(u) = c\}| + \deg^+(c) - \deg^-(c), \tag{10}$$

where $\deg^+(c)$ denotes the indegree of c and its outdegree is denoted by $\deg^-(c)$. Moreover, the sum of weights is precisely the difference in utility between X and X^1, i.e.,

$$F(X^1) - F(X) = g(C \cup \{c_1\}) - g(C) = \sum_{e \in E^1} w(e).$$

In the next proposition, we use the fact that X, X^1 are optimal w.r.t. their provider sets to characterize properties of G^1.

Proposition 1. *It holds that:*

(1) G^1 does not contain directed cycles.

(2) The only sink in G^1 is c_1.

The proof of Proposition 1 appears below. Proposition 1 suggests that G^1 is a DAG with flow conservation, so we can decompose its edges into a set of independent paths (for any arbitrary partition into paths) between a source, i.e. a provider with an excess of users under the matching X, and the sink c_1.

Next, we introduce a second graph, $G^{0,1}$, with relocation triplets from X^1 to $X^{0,1}$, the optimal matching for $g(C \cup \{c_0, c_1\})$. Formally, $G^{0,1} = (\mathcal{C}, E^{0,1}, w)$ is a directed multi-graph, with the same set of nodes and the same weight function w. $E^{0,1}$ is composed of all relocation triplets from X^1 to $X^{0,1}$. By mirroring the proof of Proposition 1, we conclude that $G^{0,1}$ contains no cycles and that c_0 is the unique sink of every directed path in it. This graph is of special interest because its sum of weights is the left hand side of Inequality (8). Namely, $\sum_{e \in E^{0,1}} w(e) = F(X^{0,1}) - F(X^1) = g(C \cup \{c_0, c_1\}) - g(C \cup \{c_1\})$. It also describes how to optimally relocate users from $C \cup \{c_1\}$ to $C \cup \{c_0, c_1\}$.

After understanding the structural properties of G^1 and $G^{0,1}$, we are ready to construct the promised matching X^0 (recall Inequality (9)). Let $G = (\mathcal{C}, E^1 \cup E^{0,1}, w)$ be the graph on the same set of nodes $\mathcal{C}$, with all the edges from both E^1 and $E^{0,1}$ (notice that the same edge cannot appear in both). For simplicity, we refer to paths in E^1 as *blue* and to paths in $E^{0,1}$ as *red*, for some arbitrary partition into paths. Our goal is to select a subset E of edges from $E^1 \cup E^{0,1}$, which, when applied to X, will induce the matching X^0. To that end, we devise an iterative process to construct the set E, by adding one path at the time. The key property of this process, which we formalize via Algorithm 1, is that there exists a mapping from every red path to a new path, composed of red and (potentially) blue edges, with a less or equal weight than that red path.

To illustrate why this process is necessary, observe that not every subset of $E^1 \cup E^{0,1}$ can be applied to X in order to obtain a new valid matching. In particular, recall that $E^{0,1}$ is the difference between X^1 and $X^{0,1}$; thus, a red path may involve the relocation (c, c', u), where u might have been matched to c' due some blue relocation (c'', c, u). To ensure that the subset we pick will result in a valid matching, we make the following distinction: a subset E such that $E \subseteq E^1 \cup E^{0,1}$ is called *consistent* if for any relocation triplet $(c, c', u) \in E$ either $X(u) = c$ or there exists another relocation triplet $(c'', c, u) \in E$. Informally, E is consistent if every user u that was relocated to c' from c was either matched to c in X, or was relocated to c from another provider. Consistency of the relocation triplets is a necessary, but not a sufficient condition for the resulting matching to be feasible.

Another useful notion is that of a *junction node*. We say that a node $c \in \mathcal{C}$ is a junction w.r.t. $E^1, E^{0,1}$ if there exists a blue edge $(c'', c, u) \in E^1$ and a red edge $(c, c', u) \in E^{0,1}$ for some $c', c'' \in \mathcal{C}$ and $u \in \mathcal{U}$. See Fig. 5 for illustration.

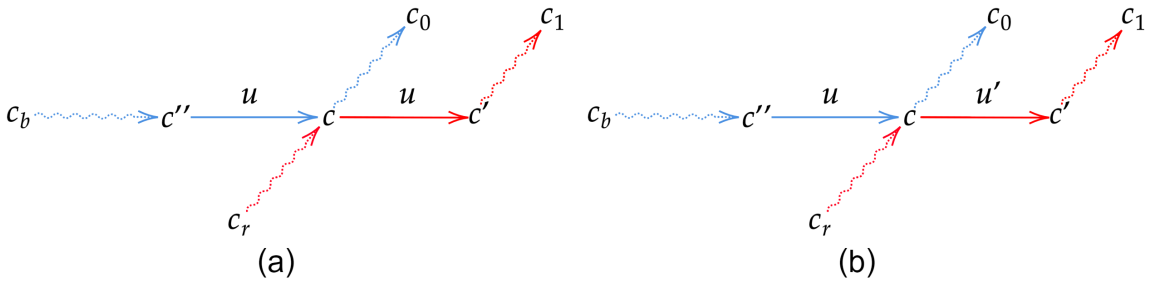


Figure 5. Exemplifying the definition of a junction node. In (a), the node c is a junction between the red path that starts at c_r (and ends at c_0) and the blue path that starts at c_b (and ends at c_1). The reason is that it receives the user u from c'' along a blue edge, and passes u along a red edge. In (b), however, c is not a junction, since the user it passes onward along the red path is u', which is not the user c receives along the blue path.

Next, we employ Algorithm 1 on the blue and red paths in $E^1 \cup E^{0,1}$. We show that

Lemma 1. *The output E of Algorithm 1 satisfies the following:*

1. *E is consistent.*

2. *When applied to X, the resulting matching X^0 is feasible.*
3. $\sum_{e\in E} w(e) \geq \sum_{e\in E^{0,1}} w(e)$.

The proof of Lemma 1 appears below. As elaborated above, the matching X and the relocation triplets in E uniquely define the matching X^0. By the second part of Lemma 1, X^0 is feasible. Moreover, by the third part of Lemma 1 and the definition of relocation triplets, we have

$$g(C \cup \{c_0, c_1\}) - g(C \cup \{c_1\}) = \sum_{e\in E^{0,1}} w(e) \leq \sum_{e\in E} w(e) = F(X^0) - g(X).$$

This completes the proof of the theorem.

Algorithm 1 Flow Construction for X^0

let B be the set of blue paths and R be the set of red paths.
let $E \leftarrow \emptyset$ be the set of new paths.
while $R \neq \emptyset$ **do**
  if there is a junction node w.r.t. B, R **then**
    let c be a junction node, and b, r denote the paths whose edges $(c'', c, u) \in b$, $(c, c', u) \in r$ form the junction such that c is the closest junction node to the sink of b.
    add to E the edge (c'', c, u) and all the directed edges that precede it in b, and (c, c', u) and all subsequent directed edges in r.
    remove r from R, b from B.
    continue
  else
    add all the edges in R to E, set $R \leftarrow \emptyset$.
  end if
end while
return E

A.1.3. Proofs of Proposition 1 and Lemma 1

Proof of Proposition 1. For (1), assume by contradiction that a simple cycle $e_1, e_2, \dots e_k$ exists for some $k \in \mathbb{N}$. Since $c_1 \notin \mathcal{C}(X)$, there is no relocation triplet with c_1 in the first entry, and hence c_1 does not participate in the cycle. We proceed by analyzing the weight of the cycle, $\sum_{i=1}^{k} w(e_i)$. [1]

- If $\sum_{i=1}^{k} w(e^i) = 0$, we can remove the cycle from the graph and obtain a new graph $\tilde{G^1}$ and a corresponding matching $\tilde{X^1}$. Observe that the number of users every provider gets is the same as in X (see Equation (10)), and hence not only $\mathcal{C}(\tilde{X^1}) = \mathcal{C}(X^1) = C \cup \{c_1\}$ but also every provider in that set meets her threshold. Further, we did not change the sum of weights, and $F(X^1) - F(X) = F(\tilde{X^1}) - F(X)$ implies $F(X^1) = F(\tilde{X^1})$; hence, $\tilde{X^1}$ is also optimal and we can assume w.l.o.g. that X^1 does not contain such cycles.
- If $\sum_{i=1}^{k} w(e^i) > 0$, we denote by $\tilde{X}$ a matching such that

$$\tilde{X}(u) = \begin{cases} c' & \text{if the edge } (c, c', u) \text{ belongs to the cycle} \\ X(u) & \text{otherwise} \end{cases}.$$

  Since the number of users each provider in $C = \mathcal{C}(X) = \mathcal{C}(\tilde{X})$ gets under $\tilde{X}$ is the same as under X, $\tilde{X}$ is feasible. Moreover, $F(\tilde{X}) - F(X) > 0$; hence, we obtain a contradiction to the optimality of X.
- If $\sum_{i=1}^{k} w(e^i) < 0$, we can use an argument similar to the previous case to claim sub-optimality of X^1.

[1] In general, a set of relocation triplets can contain cycles with positive/negative weights, if providers pass different users along the cycle. However, as we prove, this cannot happen in G^1 due to the optimality of X^1.

For (2), assume by contradiction that a node $v \in \mathcal{C}, v \neq c_1$ is a sink, and observe that we must have $v \in C$ since $\mathcal{C}(X^1) = C \cup \{c_1\}$. Let $v_1, \ldots, v_k, v$ denote the shortest path ending at v. Because $c_1 \notin \mathcal{C}(X)$, we know that c_1 cannot participate in this path. Further, X is feasible and hence v gets at least ν_v users under X^1. The analysis identically to the first part of the proposition, arguing that the contradiction assumption entails the existence of a path with positive/negative weights, in contrast to the optimality of X and X^1. □

Proof of Lemma 1. Assume by contradiction that the output E is not consistent. By definition of consistency, there exists an edge $e = (c, c', u)$ such that

1. $X(u) \neq c$, and
2. $(c'', c', u) \notin E$ for every $c \in \mathcal{C}$.

Notice that $e \in E \subseteq E^1 \cup E^{0,1}$; hence, e is either blue or red. If e is blue, let $b(e)$ denote the path e is part of. Since $e \in E$ and is blue, the only way it could have been added to E is via Line 1. This means that either e is the first edge in $b(e)$, in which case $X(u) = c$ since E^1 is consistent; or e is an intermediate edge in $b(e)$, in which case there exists another edge $e' = (c'', c', u) \in b(e)$ that precedes it, again because E^1 is consistent. In both cases, we obtain contradiction.

Otherwise e is red. Let r denote the path that contains e. We have two cases:

- If c is a junction w.r.t. the initial B, R. In this case, there exists a blue path $b \in B$ that contains an edge (c'', c, u), by the definition of a junction node. Moreover, at some point in the execution e was added, so b must have been identified as a path containing an edge that forms a junction node v (not necessarily c) in Line 1. Recall that in Line 1 we assume that v is the closest junction node to the sink of b, which is c_1; hence, all edges of b that precedes the outgoing edge from v are added to E too, including (c'', c, u). This implies a contradiction.
- Else, c is not a junction. If e is the first edge in the red path r, then $X^1(u) = c$, and since c is not a junction, $X(u) = c$ as well. This holds because both X, X^1 are feasible. Otherwise, if e is an intermediate edge in r, then there must exists a red edge (c'', c, u) for some $c'' \in C$, because $E^1 \cup E^{0,1}$ is consistent. Since red edges like e are inserted to E in Lines 1 and 1, the preceding edges in their red path, including (c'', c, u), are added as well. In both cases, we reach a contradiction.

Second part Denote the matching obtained by applying the relocation triplets of E to X by X^0. To show that X^0 is feasible, we need to show that for every $c \in C \cup \{c_0\}$, it holds that $\left|\{u : X^0(u) = c\}\right| \geq \nu_c$. To do so, we rely on the feasibility of X^1 and $X^{1,0}$, whose relocation edges were used to construct X^0. We divide the analysis into three parts:

- If $c = c_0$. Since $X^{1,0}$ is feasible, we know that the $\deg^+(c_0)$ in $G^{0,1}$ is at least ν_{c_0} (Recall the quantification of the number of matched users in Equation (10)). Since E contains the final edge of every red path, the indegree of c_0 in $G^0 = (\mathcal{C}, E, w)$ is the same as in $G^{0,1}$.
- Else, if c is the source of at least one path in E. In this case, it must have been the source of some paths in E^1 (blue) and $E^0, 1$ (red). Recall that the sink of every blue path is c_1, and the sink of every red path is c_0. Moreover, if c participates in other red/blue paths, it must be an intermediate node; thus, we can analyze its loss of users due to the paths in which c is the source solely. Since $E \subseteq E^1 \cup E^{0,1}$, c is the source of less paths in $G^0 = (\mathcal{C}, E, w)$ than in $G^{0,1}$; therefore, its indegree in G^0 is greater or equal to its indegree in $G^{0,1}$, which implies that X^0 matched c with at least as many users as $X^{0,1}$.
- Finally, for any other c, X^0 matches c with the same number of users as $X^{0,1}$, since its difference between the indegree and the outdegree in $G^0 = (\mathcal{C}, E, w)$ remains as in $G^{0,1}$.

Third part The proof of this part is based on the following observation:

Observation 1. *Let b be a blue path with source c_b and sink c_1, r be a red path with source c_r and sink c_0, and let c be a junction w.r.t. b and r, with edges $(c'', c, u) \in b$ and $(c, c', u) \in r$. Denote by p the path that starts from c_b, takes the edge (c'', c, u) and the edges that precedes in b, and then takes (c, c', u) and its subsequent edges in r, ending at c_0. Then, $\sum_{e \in p} w(e) \leq \sum_{e \in E^{0,1}} w(e)$.*

To see why Observation 1 holds, recall that the prefix of p from its source to (c'', c, u) inclusive, all blue edges, must have higher weight than the prefix of r from its source to (c, c', u), exclusive. This is true since otherwise we could find a heavier blue path to replace b in E^0. However, this cannot be true as X^1, which accounts for the blue edges in G^1, is an optimal matching for $g(C \cup \{c_1\})$. Finally, Algorithm 1 adds red paths either in their entirety (Line 1) or by modifying them to be heavier according to Observation 1; hence, $\sum_{e \in E} w(e) \geq \sum_{e \in E^{0,1}} w(e)$.

To complete the picture, it remains to argue that problems in which user queries contribute non-unit amounts to provider viability can be reduced to the unit case analyzed in the previous sections. That is, constraints of the type $\sum_{u \in \mathcal{U}} w_{uc} X_{uc} \geq \nu_c Y_c$, where the weights w are non-negative rational numbers, can be converted into constraints of the type $\sum_{u \in \mathcal{U}} X_{uc} \geq \nu_c Y_c$. Assuming that the weights are rational numbers, the linear program can be replaced with an equivalent one with integer coefficients, that is, w_{uc} are integer. Each of these integer counts then can be replaced by a fictional user that contributes a unitary count towards the provider's viability. □

A.2. Non-linear Optimization via Column Generation

A.2.1. Formulation

As discussed in Sec. 3.2, it is desirable to have a procedure that can optimize social welfare under non-linear utility models. To this end, we extend the mixed-integer linear program in Problem (4) to handle non-linear utilities. Let $C \in \mathcal{C}^k$ be a k-tuple of providers. A pair $(q_u, C) \in \mathcal{Q} \times \mathcal{C}^k$ represents a possible answer to user u's k queries identical to q_u by the provider tuple C. We call such a tuple a *star* $q_u C$. For each star, we use a variable $\pi_{q_u C}$ to represent the policy's match to q_u.

$$\begin{aligned} \max_{\pi,\, y} \quad & \sum_{u \in \mathcal{U}} \sum_{q_u \in \mathcal{Q}} \sum_{C \in \mathcal{C}^k} \pi_{q_u,C} \bar{\sigma}(q_u, C) \\ \text{s.t.} \quad & \sum_{C \in \mathcal{C}^k} \pi_{q_u,C} \leq 1 && u \in \mathcal{U}, \\ & \sum_{\{C \in \mathcal{C}^k | c \in C\}} \pi_{q_u,C} \leq y_c && u \in \mathcal{U}, c \in \mathcal{C}, \\ & \sum_{u \in \mathcal{U}} \sum_{C \in \mathcal{C}^k} \#[q_u C, c] \overline{Q}(q_u) \pi_{q_u,C} \geq \nu_c y_c, && c \in \mathcal{C}, \end{aligned} \tag{11}$$

where $\#[uC, c]$ is the number of times provider c appears in star $q_u C$, and $\bar{\sigma}(q_u, C) = \rho(u) P_u(q_u) \sigma(q_u, C)$. We rely on the linear relaxation of the integrality constraints to approximate the solution of (11) efficiently. It is not obvious if and how this problem can be approximated via discrete algorithmic techniques, so we resort to relaxing the integrality constraints and solving the problem as a linear program. Even under the linear relaxation, the problem size still grows proportionally to $\mathcal{C}^k$ due to the number of variables introduced by linearization. The redeeming property of this problem, however, is that the number of constraints grows proportionally to $\mathcal{U} \times \mathcal{Q} \times \mathcal{C}$ and not $\mathcal{C}^k$. Hence, it is feasible to approach the problem from a column generation perspective.

A.2.2. Column Generation

A standard column generation approach for solving a large linear program is a two-step iterative algorithm in which the LP is initially constructed using a small subset of its variables to obtained a reduced-size (master) problem. The dual of the master problem is solved to obtain a dual optimal solution, which is then used to find a (as of yet not generated) variable with maximal reduced cost. That variable is added to the master problem. The method iterates until no variable with positive reduced cost can be found, or some convergence tolerance is reached.

When the set of primal variables is large, the problem of finding a variable with maximal reduced cost (also called a column generation oracle) is still a hard combinatorial optimization problem (typically some flavor of knapsack). However, these problems tend to be massively decomposable so the running time does not scale exponentially in practice.

We now proceed to derive a column generation oracle for Problem (11). Let $\mathcal{A} = (A, b, c)$ denote an LP in inequality form, denoting the optimization problem $x^* = \arg\max_{x: Ax \leq b, x \geq 0} c^T x$. Let y^* be an optimal dual solution to $\mathcal{A}$. The reduced cost problem is thus $\hat{c} = c - A^T y^*$. The column generation oracle thus solves the problem $i^* = \arg\max_i \hat{c}$, which corresponds to the index of the primal variable with highest reduced cost. We now discuss solving the column generation problem given the specific form of (11).

We adopt the following convention for naming the dual variables corresponding to constraints in (11):

$$\begin{aligned}
\beta_u &: \sum_{C \in \mathcal{C}^k} \pi_{q_u,C} \leq 1 && u \in \mathcal{U}, \\
\gamma_{uc} &: \sum_{\{C \in \mathcal{C}^k | c \in C\}} \pi_{q_u,C} \leq y_c && u \in \mathcal{U}, c \in \mathcal{C}, \\
\alpha_c &: \sum_{u \in \mathcal{U}} \sum_{C \in \mathcal{C}^k} \#[q_u C, c] \overline{Q}(q_u) \pi_{q_u,C} \geq \nu_c y_c && c \in \mathcal{C}.
\end{aligned}$$

The column generation problem (derived by computing the dual and maximizing the reduced cost) then becomes:

$$uC^* = \arg \max_{u \in \mathcal{U}, C \in \mathcal{C}^k} \quad \bar{\sigma}(q_u, C) - \left(\beta_u + \sum_{c \in C} \gamma_{uc} - \sum_{c \in C} \#[q_u C, c] \overline{Q}(q_u) \alpha_c \right).$$

Let us now discuss how the above maximization can be solved. First, observe that the problem decomposes in the user variable u. That is for each $u \in \mathcal{U}$, we can independently solve the maximization over C. This can be done in parallel for each user and the maximum over u can be computed by enumeration. Supposing u is fixed, we still have to solve a series of non-linear integer optimization problems due to the non-linear nature of $\bar{\sigma}$. We can covert the non-linear problems to linear in two steps. First, we convert the tuple maximization problem to a binary-variable one as by introducing slot indicator variables for each of the elements of the tuple C. That is:

$$\max_x \quad \bar{\sigma} \left(\sum_{t \in 1:k} \sum_c x_{ct} A_{uc} \right) - \left(\beta_u + \sum_{t \in 1:k} \sum_{c \in \mathcal{C}} x_{ct} \gamma_{uc} - \sum_{c \in \mathcal{C}} \left(\sum_t x_{ct} \right) \alpha_c \right) \text{ s.t. } \sum_c x_{ct} = 1 \quad \forall t \in 1:k \,.$$

$$\begin{aligned}
\max_x \quad & \bar{\sigma}'(m_i) \cdot \left(\sum_{t \in 1:k} \sum_c x_{ct} A_{uc} \right) - \left(\beta_u + \sum_{t \in 1:k} \sum_{c \in \mathcal{C}} x_{ct} \gamma_{uc} - \sum_{c \in \mathcal{C}} \left(\sum_t x_{ct} \right) \alpha_c \right) \\
& \text{s.t. } \sum_c x_{ct} = 1 \quad \forall t \in 1:k \,, \quad l_i \leq \sum_{t \in 1:k} \sum_c x_{ct} A_{uc} \leq u_i,
\end{aligned}$$

where $\bar{\sigma}'(m_i)$ is the derivative of $\bar{\sigma}$ at m_i. Under smoothness assumptions on $\bar{\sigma}$, this linearization provides a bounded approximation to the original problem. Again, these interval sub-problems can be solved in parallel.

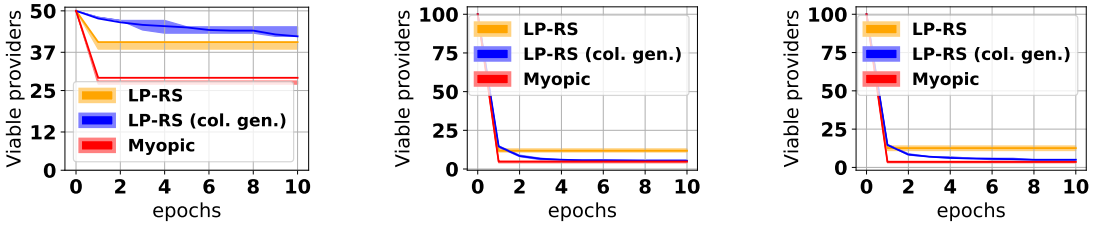
A.2.3. Illustrative Evaluation of Column Generation

Here we describe some preliminary experiments using the column generation strategy described above. We find that column generation is capable of keeping more providers viable than the myopic baseline at early steps. However the performance was less reliable than the LP-RS approach that was evaluated in Section 4. In some settings, the column generation approach fails to maintain a consistent matching in successive iterations, resulting in a slowly declining number of viable providers over time. We hypothesize that this is due to rounding errors in the procedure, or early stopping before convergence (our implementation used 300 iterations of column generation rather than running exhaustively until convergence). Therefore we expect that improvements can be made by fine-tuning this approach, but leave this to future work.

Figure 6 shows the results of the experiments. While we use the same embeddings data as in the main body of the paper, but we scale down the problem size to compensate for the slower runtime of the column generation approach; this explains the differing number of viable providers at equilibrium compared with the $LP - RS$ approach presented in Section 4. In the synthetic setting we used 50 providers, about 260 users and viability threshold of $\nu = 5$. In the other two datasets we used a competitive (from the provider perspective) setting of 100 providers, 100 users, and viability threshold of $\nu = 8$. We used slate size of 1 for all datasets.

B. Training Details

Here we provide details of training for the embeddings described in Section 4.2.



(a) Synthetic embeddings. (b) Movielens embeddings. (c) SNAP embeddings.

Figure 6. Simulations that evaluate the column generation matching strategy (LP-RS col. gen.) on smaller problems. While LP-RS col. gen. is capable of finding a good matching at a given step of simulation, its solutions are relatively inconsistent compared with LP-RS in the limit of several time steps.

Movielens We trained a non-negative matrix factorization embedding space using the Movielens dataset (Harper & Konstan, 2015). We use the distribution of this dataset containing about $100,000$ ratings of about $9,000$ movies by about 600 users. The dataset comprises a sparse ratings matrix Given the sparse ratings matrix $R \in \mathbb{R}_{\geq 0}^{N_{\text{users}} \times N_{\text{movies}}}$. We use the binarized engagement matrix $E \in \{0,1\}^{N_{\text{users}} \times N_{\text{providers}}}$ with $E_{i,j} = \mathbb{1}(R_{i,j})$. The embeddings are produced by finding low-rank non-negative factors of the engagement matrix $E \in \{0,1\}^{N_{\text{users}} \times N_{\text{providers}}}$, by solving the optimization problem

$$\min_{U,V} ||(E - UV^T)||_F^2 + \lambda_U ||U||_F^2 + \lambda_V ||V||_F^2 \tag{12}$$

which yields factors $U \in \mathbb{R}_{\geq 0}^{N_{\text{users}} \times N_{\text{topic}}}$ and $U \in \mathbb{R}_{\geq 0}^{N_{\text{providers}} \times N_{\text{topic}}}$.

The factors $U \in \mathbb{R}_{\geq 0}^{N_{\text{users}} \times N_{\text{topic}}}$ and $C \in \mathbb{R}_{\geq 0}^{N_{\text{providers}} \times N_{\text{topic}}}$ yield row and column vectors that are treated as the embedding vectors; in this case, a single content provider is equivalent to a single movie from the dataset.

The rows of these factor matrices were used to sample user and provider vectors in the RS ecosystem. Note that the value of the ratings were not used, so the "affinity" between user and movie in this embedding space is a measure of how likely the user is to *watch* the movie, rather than rate it highly. The randomly initialized factors U, V are alternatively updated via Weighted alternating least squares (Hu et al., 2008) for 100 iterations. We used embedding rank $N_{\text{topic}} = 20$, and set $\lambda_U = 1$ and $\lambda_v = 1$..

SNAP The dataset consists of a large list of (followee, follower) pairs, where each user is given a unique node ID label. We turn this dataset into a set of providers and users as follows. First, we randomly subsample 100k of the 41 million users. We designate followees as providers. For every provider, we then remove their follow edges, so that they do not follow anyone else. This makes the graph bipartite, where users follow providers. We then choose the top 500 providers in terms of follower count, and remove any users that do not follow at least one of them. This leaves a total of 500 providers, and 59,394 users.

for each user i, we learn a 24-dimensional vector $\mathbf{u}_i$, and for each provider j, a 24-dimensional vector $\mathbf{v}_j$. We train these embeddings by cross-entropy to predict whether there is an edge A_{ij} between user i and provider j, where the probability is given by,

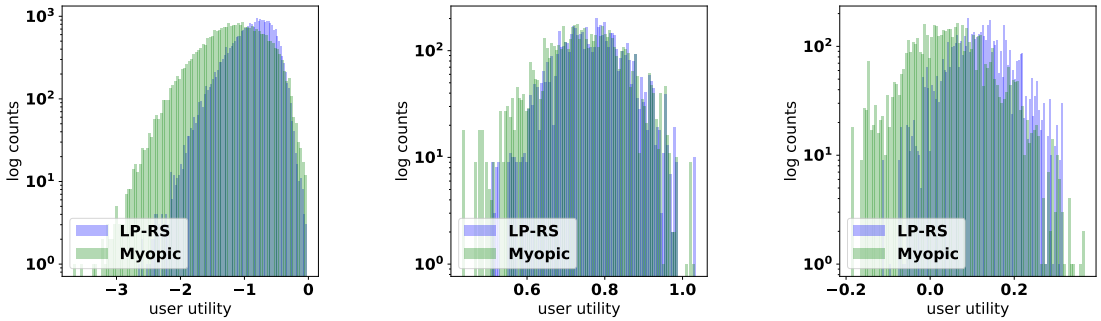
$$P(A_{ij} = 1) = \sigma(\mathbf{u}_i^\top \mathbf{v}_j) \tag{13}$$

Where $\sigma(\cdot)$ is the sigmoid function. A_{ij} is 1 if user i follows provider j. We add a small amount of weight decay to ensure that the embeddings are well behaved.

C. Simulation Details

This section contains details to reproduce the simulations described in Section 4.3.

Exploring Embedding Type We generate 50 provider vectors and a varying number of user vectors (between 4412 and 4672 per run). Provider and user vectors are sampled in a 10-dimensional topic space, with provider vectors sampled



(a) Synthetic embeddings. (b) Movielens embeddings. (c) SNAP embeddings.

Figure 7. Histogram of user utility.

normally with variance 50. These provider vectors serve as cluster means for the mixture-of-Gaussians that generates user vectors. The prior over cluster assignments depends on the variant (*uniform* vs. *skewed* described in the text). User variance was set to 0.1 in the *uniform* variant, and user variance scaled inversely with popularity in the *skewed* variant. The slate size was $s = 1$, with viability threshold set to $\nu = 80$. We run for 10 epochs using 5 seeds for each method/data type pair, and report average values plus or minus one standard deviation in Table 1.

Tradeoffs in regret and welfare We generate synthetic embeddings of the *skewed* variant, with 50 providers and around 900 users. We use slate size $s = 4$ with viability threshold $\nu = 9$. Other settings are carried over from the previous experiment. We run steps of simulation until the policies converge then measure the welfare and max regret metrics.

Synthetic simulation We generate synthetic embeddings of the *skewed*. The parameters are similar to those described above, with 50 providers (distributed normally with $\sigma^2 = 5$) and about $10,000$ users. We simulate the RS for ten epochs with slate size $s = 1$ and viability threshold $\nu = 78.5$.

Movielens simulation Starting with the learned low-rank factors, we subsample 250 movie column (which serve as providers) and $1,000$ user columns. We simulate the RS for ten epochs with slate size $s = 1$ and viability threshold $\nu = 10$.

SNAP simulation Starting with the learned embedding, we subsample 300 providers and 566 users. We simulate the RS for ten epochs with slate size $s = 1$ and viability threshold $\nu = 10$.

D. Additional Histograms

Figure 7 shows user utility histograms for all simulations.

E. Extended Welfare-Regret Tradeoff Results

Table 3 extends the result from Table 2 by including the Myopic baseline recommender.

References

Harper, F. M. and Konstan, J. A. The Movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5(4):1–19, 2015.

Hu, Y., Koren, Y., and Volinsky, C. Collaborative filtering for implicit feedback datasets. In *2008 Eighth IEEE International Conference on Data Mining*, pp. 263–272. Ieee, 2008.

		Avg. Welfare	Max Regret	Viable Providers
0.1	LP-RS	18.02 ± 1.05	7.24 ± 0.77	47.20 ± 1.72
	Myopic	13.49 ± 1.26	10.17 ± 0.20	11.80 ± 1.47
0.18	LP-RS	19.79 ± 1.16	7.97 ± 0.84	47.20 ± 1.72
	Myopic	14.83 ± 1.39	11.18 ± 0.22	11.80 ± 1.47
0.26	LP-RS	21.87 ± 1.28	8.84 ± 0.91	46.60 ± 2.15
	Myopic	16.42 ± 1.54	12.37 ± 0.25	11.80 ± 1.47
0.35	LP-RS	24.30 ± 1.43	10.14 ± 1.18	45.20 ± 3.06
	Myopic	18.29 ± 1.71	13.79 ± 0.27	11.80 ± 1.47
0.43	LP-RS	27.17 ± 1.58	11.65 ± 0.91	44.00 ± 3.16
	Myopic	20.50 ± 1.92	15.45 ± 0.31	11.80 ± 1.47
0.51	LP-RS	30.44 ± 1.79	14.19 ± 1.46	44.80 ± 0.98
	Myopic	23.08 ± 2.16	17.40 ± 0.35	11.80 ± 1.47
0.59	LP-RS	34.30 ± 1.95	16.50 ± 1.73	43.00 ± 1.90
	Myopic	26.07 ± 2.44	19.65 ± 0.39	11.80 ± 1.47
0.67	LP-RS	38.67 ± 2.29	20.53 ± 1.46	42.60 ± 1.50
	Myopic	29.51 ± 2.76	22.24 ± 0.44	11.80 ± 1.47
0.75	LP-RS	43.69 ± 2.61	24.61 ± 1.31	41.80 ± 1.72
	Myopic	33.44 ± 3.13	25.21 ± 0.50	11.80 ± 1.47
0.84	LP-RS	49.51 ± 2.90	28.95 ± 1.11	39.80 ± 3.06
	Myopic	37.91 ± 3.55	28.57 ± 0.57	11.80 ± 1.47
0.92	LP-RS	56.15 ± 3.26	33.05 ± 1.16	36.80 ± 3.97
	Myopic	42.94 ± 4.02	32.36 ± 0.64	11.80 ± 1.47
1.0	LP-RS	63.62 ± 3.58	37.89 ± 1.39	34.20 ± 5.08
	Myopic	48.59 ± 4.55	36.62 ± 0.73	11.80 ± 1.47

Table 3. The discounting factor γ allows LP-RS to trade off between average user welfare and max user regret.