FairTQ-Exact: Fair Top-k Query on Alpha-Fair (Technical Report)

ABSTRACT

The traditional top-k query was proposed to obtain a small subset from the database according to the user preference, which is explicitly expressed as a ranking scheme (i.e., utility function). However, a poorly-designed utility function may create discrimination, which in turn may cause harm to minority groups, e.g., women and ethnic minorities, and thus, fairness is becoming increasingly important in many situations, e.g., hiring and admission decisions, recently.

Motivated by this, our goal is to design a fair ranking scheme to alleviate discrimination. We propose a fairness model, called *α-fair*, to quantify the fairness of utility functions. An efficient algorithm called **FairTQ-Exact** is designed to find the fairest utility function with minimum modification penalty. We also offer an approximation algorithm **β-FairTQ-Appro** which returns an approximately fairest utility function with equal or even smaller modification compared with **FairTQ-Exact**. We conducted extensive experiments on both real and synthetic datasets to demonstrate our effectiveness and efficiency compared with the prior studies.

1 INTRODUCTION

It is often hard for users to find the tuples that interest them from a database system with millions of tuples for decision-making, e.g., hiring decision. We can use a traditional top-k query [14, 21, 30], where a user has to provide a preference function, called the *utility function*, and an output size, k. A larger weight on an attribute in the utility function means that this attribute is more important. Based on the utility function, the *utilities* of tuples can be computed as the weighted sum of attribute values. Then, the tuples in the database are ranked based on the utilities and the *top-k set* (i.e., the set of k tuples with the highest utilities) is returned. However, it is hard for users to specify an explicit utility function in most cases.

A poor *ranking scheme* may create *discrimination*, where discrimination means that people are treated in a different and especially worse way simply because of their membership in a group (e.g., women and African Americans) that is often discriminated against and denoted as the *protected group* defined by the *protected attribute* (e.g., gender and race). In ranking problems, discrimination means that people are ranked lower simply because of their membership in a protected group. For example, women have a lower success rate when applying for loans than men [33], so women are the protected group in this case. Men can also be the protected group in some situations. By DATA USA [1], in 2017, 11.5% of female applicants are accepted to MIT. In contrast, only 5.25% of males are accepted. Such unfairness is also reflected in other aspects, e.g., race [6]. This motivates us to address racism or sexism in society more actively. Failure of addressing these issues may lead to discrimination and conflicts, and a fairer ranking scheme that can help policymakers develop a fair policy to eliminate gender or race imbalance is of great importance. In this paper, we aim at helping users design a fair ranking scheme. In particular, we focus on the fairness in the top-k sets.

Consider an example with 9 applicants for a college in Table 1. The admission office needs a ranking scheme to give each applicant a score and then admits the top-7 applicants. The original utility function f_0 computes the score of each applicant t as $f_0(t) = 0.2 \times TOEFL + 0.1 \times GRE + 0.7 \times GPA$. The top-7 set is shown in column "f_0" of Table 1 (marked as 1), including 6 males but only 1 female (the admitted rates of males and females are 100% and 33.33%, respectively), which is unfair to females. Consider the top-7 set of another utility function $f_1(t) = 0.6 \times TOEFL + 0.1 \times GRE + 0.3 \times GPA$ in column "f_1". The admitted rates of males and females become 66.67% and 100%, so the gap is smaller, which makes f_1 a fairer utility function than f_0.

Table 1: Information of Applicants

ID	Name	Gender	TOEFL	GRE	GPA	f_0	f_1
1	Stanley	Male	110	335	3.5	1	1
2	Summer	Male	108	330	4.0	1	1
3	Rick	Male	115	330	3.2	1	1
4	Andy	Female	105	320	3.7	1	1
5	Morty	Male	87	310	3.9	1	0
6	Mabel	Female	88	315	2.8	0	1
7	Dipper	Male	87	310	4.0	1	1
8	Beth	Female	88	310	2.6	0	1
9	Jerry	Male	87	310	3.8	1	0

This example illustrates our problem. We want to help users design utility functions that produce fair top-k results for its increasing importance [12] in law, economics, and policy, etc. Moreover, when designing a new utility function, we do not want it to deviate too much from the original (user-specified) one, since a small modification penalty on the utility function is typically more preferable.

Taking a utility function f_0 as input, we return a utility function f_1 that is both fair and close to f_0. Unlike previous studies [5, 9] that simply set fairness as constraints, we set fairness as the main target, i.e., we aim to find the *fairest* top-k set. Note that it is not easy to select a suitable fairness constraint. A loose constraint may still create discrimination, while a hard constraint may not be satisfied by the dataset. In this case, the fairest result would be more preferred.

To achieve this, we propose a fairness model called *α-fair* which is a combination of the traditional fairness model and the privacy model to quantify the fairness of a top-k set. An efficient exact algorithm **FairTQ-Exact** (Fair Top-k Query) is proposed to help users find the desired top-k set efficiently. The main idea of **FairTQ-Exact** is to enumerate all possible top-k sets, and then check (a) their fairness in terms of α-fair and (b) the modification penalty (computed via quadratic programming). Several effective pruning methods are proposed to improve the efficiency of **FairTQ-Exact**.

We also propose an approximation algorithm **β-FairTQ-Appro** to satisfy users' different needs. Specifically, algorithm **β-FairTQ-Appro** returns an approximately fair top-k set with a guaranteed error bound on the fairness, while the modification penalty is no larger than that of the exact algorithm. We conducted experiments on real and synthetic datasets to show that our algorithms give fair ranking schemes efficiently.

The major contributions of this work are listed as follows.

- We design a fairness model called α-fair to measure the fairness of top-k sets, with techniques from privacy models;
- We design an efficient algorithm **FairTQ-Exact** to find the fairest utility function with minimum modification penalty;
- An approximation algorithm β-**FairTQ-Appro** is proposed to find fair enough solutions with even smaller modification penalty efficiently, to fit different needs; and
- Extensive experiments on real and synthetic datasets demonstrate our effectiveness and efficiency. Specifically, our algorithms outperform the baselines by one to two orders of magnitude in execution time and return fairer results. Our case study shows that our result provides fairer results compared with baselines.

The remainder of this paper is organized as follows. In Section 2, we give the problem definition. Section 3 presents our exact solution to the fairness problem. Section 4 describes our approximation algorithm. Section 5 reviews related work and Section 6 shows the results of our experiments. Finally, Section 7 gives our conclusion.

2 PROBLEM DEFINITION

We first give preliminaries about top-k sets in Section 2.1. We formalize the fairness model and the problem in Sections 2.2 and 2.3.

2.1 Preliminary: Top-k Set

Given a dataset D with n tuples, each tuple $t \in D$ has d scoring attributes, denoted by $t[1], \ldots, t[d]$, each of which is a non-negative real value. Each tuple t is associated with a protected attribute, which is a categorical value indicating the group of t. We divide tuples in D into disjoint partitions according to their groups. Let $\mathcal{P}$ and $D[j]$ be the set of all groups and the partition of D containing all tuples in group j, respectively. For example in Table 1, each tuple has 3 scoring attributes (i.e., TOEFL, GRE and GPA) and a protected attribute (i.e., gender). Given $\mathcal{P} = \{\text{Male, Female}\}$, we obtain 2 partitions, say $D[\text{Female}]$ and $D[\text{Male}]$, with 3 and 6 tuples, respectively.

Following typical settings in [21, 30], given a d-dimensional *utility vector* $w = (w[1], \ldots, w[d])$, the user preference on a tuple t is captured by a linear *utility function* w.r.t. w, denoted by f_w, where $f_w(t) = \sum_{i=1}^{d} w[i] \cdot t[i]$. For simplicity, we also denote f_w by w. We define the *utility* of tuple t w.r.t. w to be $f_w(t)$. In particular, we can rank all tuples in D in a descending order of their utilities (ties broken arbitrarily). The *top-k set* in D w.r.t. w, denoted by **top**-$k_w(D)$, is defined to be the set of k tuples in D with the highest utilities w.r.t. w. A top-k query w.r.t. w thus returns **top**-$k_w(D)$. When D and w are clear in the context, we also denote **top**-$k_w(D)$ by **top**-k_w or T for simplicity. Consider our running example in Table 1. Given $w = (0.2, 0.1, 0.7)$ and $k = 7$, the top-k query w.r.t. w returns a top-k set containing applicants of ID 1-5, 7 and 9, since they have the highest utilities w.r.t. w.

We assume w.l.o.g. that $\sum_{i=1}^{d} w[i] = 1$ where $w[i] \in [0, 1]$, since the norm of w does not affect the top-k results [23]. Note that $w[d] = 1 - \sum_{i=1}^{d-1} w[i]$. Thus, we could reduce the last dimension from the domain of w (i.e., from d-dimension to $(d-1)$-dimension) for more efficient processing [23]. We define the *utility space*, denoted by Ω, to be the $(d-1)$-dimensional subspace containing all utility functions after the dimension reduction. In our example in Table 1, the utility space is a subspace in a 2-dimensional space, as shown in the shaded triangular region of Figure 1(a), bounded by the two axis and the straight line, represented by $w[1] + w[2] = 1$.

Table 2: Clustering a Top-7 Set under Global Recoding

ID	Name	Gender	TOEFL	GRE	GPA
1	Stanley	Male	[100, 115]	[320, 335]	[2.6, 4.0]
2	Summer	Male	[100, 115]	[320, 335]	[2.6, 4.0]
3	Rick	Male	[100, 115]	[320, 335]	[2.6, 4.0]
4	Andy	Female	[100, 115]	[320, 335]	[2.6, 4.0]
5	Morty	Male	[87, 100)	[310, 320)	[2.6, 4.0]
7	Dipper	Male	[87, 100)	[310, 320)	[2.6, 4.0]
9	Jerry	Male	[87, 100)	[310, 320)	[2.6, 4.0]

2.2 Fairness Model

We formalize our fairness model. Following prior studies [5, 33], we focus on the *group fairness* model, which requires each group (e.g., male and female) is treated equally [21]. However, existing group fairness models require a "hard" constraint that is hard to be satisfied. Thus, we consider a "flexible" measurement under which group fairness is granted rather than having to satisfy a "hard" constraint.

Intuitively, our fairness measurement on a top-k set T follows two design principles. The first principle is based on the common criterion of group fairness. If the probability that tuples in each group appear in T is close among all groups, then T is fair. The second principle is to have a flexible measurement. To achieve this, instead of regarding T as a whole, we divide it into clusters and check the fairness of *each* cluster one-by-one. The overall fairness of T is *aggregated* from all clusters, rather than forcing every cluster to be fair.

In the following, we first consider how to divide a top-k set into clusters, and then formalize the fairness model for a top-k set.

Clustering in Top-k Set. In our fairness model, we divide a top-k set into multiple clusters such that each cluster contains similar tuples. In particular, to accurately measure the fairness in each cluster, we cluster without knowing the group information.

To this end, we adapt the idea of *global recoding* in privacy models [29, 31]. Specifically, global recoding solves a privacy-preserving problem to weaken the correlation between a sensitive attribute and other attributes, by "generalizing" (or "recoding") each non-sensitive attribute value with a "generalized" value (e.g., generalize a numerical value with a value range) using a global domain. Applying this idea, we define a *global domain* to generalize the scoring attributes in our model. In this way, global recoding is independent to the group information (defined in the protected attribute). After global recoding, tuples with the same generalized values are mapped to the same cluster, since their values fall in the same value range.

Let $min_{D,i}$ (resp. $max_{D,i}$) be the minimum (resp. maximum) value of the i-th scoring attribute in D, i.e., $min_{D,i} = \min_{t \in D} t[i]$ (resp. $max_{D,i} = \max_{t \in D} t[i]$). We define a *generalized domain* on the i-th scoring attribute to be a set of *disjoint* value ranges whose union equals $[min_{D,i}, max_{D,i}]$, e.g., $\{[min_{D,i}, max_{D,i}]\}$ is itself a generalized domain on the i-th scoring attribute. Given a tuple t in D and a generalized domain G_i on the i-th scoring attribute, we define the *recoding* of t under G_i, denoted by $G_i(t)$, to be the value range in G_i that $t[i]$ falls into. In our dataset D in Table 1, an example of generalized domain on attribute TOEFL can be $\{[87, 100), [100, 115]\}$

(note that the minimum and maximum value of attribute TOEFL in D is 87 and 115, respectively). Similarly, examples of generalized domains on attribute GRE and GPA can be $\{[310, 320), [320, 335]\}$ and $\{[2.6, 4.0]\}$, respectively. Then, the recodings of the tuple of ID 1 under the above generalized domains are $[100, 115]$, $[320, 335]$ and $[2.6, 4.0]$, on attribute TOEFL, GRE and GPA, respectively.

Consider two generalized domains on the (same) i-th scoring attribute, says G_i and G'_i. We say that G_i is *more general* than G'_i, denoted by $G_i \succeq G'_i$, if for any value range $x' \in G'_i$, there exists a value range $x \in G_i$ such that $x' \subseteq x$. To exemplify, a generalized domain $\{[2.6, 4.0]\}$ on attribute GPA is more general than another generalized domain $\{[2.6, 3.0), [3.0, 4.0]\}$ on attribute GPA.

With the recoding for *one* attribute, we extend it for *all* scoring attributes to formalize *global recoding*. We define a *global domain* on D to be a d-tuple whose i-th element is a generalized domain on the i-th scoring attribute for each $i \in [1, d]$. Given a global domain on D, says $\mathcal{G} = (G_1, \ldots, G_d)$, we define the *global recoding* of t under $\mathcal{G}$, denoted by $\mathcal{G}(t)$, to be a d-tuple whose i-th element is the recoding of t under G_i for each $i \in [1, d]$, i.e., $\mathcal{G}(t) = (G_1(t), \ldots, G_d(t))$. Continuing our example, a global domain on D can be $(\{[87, 100), [100, 115]\}, \{[310, 320), [320, 335]\}, \{[2.6, 4.0]\})$, under which the global recoding of the tuple in D of ID 1 is $([100, 115], [320, 335], [2.6, 4.0])$. Similarly, considering two global domains on D, says $\mathcal{G} = (G_1, \ldots, G_d)$ and $\mathcal{G}' = (G'_1, \ldots, G'_d)$, $\mathcal{G}$ is *more general* than $\mathcal{G}'$, denoted by $\mathcal{G} \succeq \mathcal{G}'$, if for each $i \in [1, d]$, G_i is *more general* than G'_i.

Given a global domain $\mathcal{G}$ on D and a top-k set T, we divide T into multiple clusters where each cluster contains tuples in T with the same global recoding under $\mathcal{G}$. To exemplify, consider the top-k set w.r.t. f_0 containing applicants of ID 1-5, 7 and 9 in Table 1. The clustering result under a global domain $\mathcal{G} = (\{[87, 100), [100, 115]\}, \{[310, 320), [320, 335]\}, \{[2.6, 4.0]\})$ is shown in Table 2. There are two clusters in the top-k set: one contains applicants of ID 1-4 and the other one contains applicants of ID 5, 7 and 9.

It is not hard to observe that with a *more general* global domain $\mathcal{G}$, fewer (or an equal number of) clusters will be formed since more tuples will be in a single cluster under a more general global domain. When $\mathcal{G}$ is the "most general" global domain on D (that is, each generalized domain G_i of $\mathcal{G}$ for $i \in [1, d]$ is exactly $\{[min_{D,i}, max_{D,i}]\}$), all tuples in any top-k set will be in one cluster.

The global recoding supports a *multi-level* generalization scheme [31], to allow flexible clustering results under different global domains. We can compute the best fairness (to be shown shortly) under the multi-level scheme, so that it is limited to any particular global domain. Specifically, for the i-th scoring attribute ($i \in [1, d]$), we take as input a list $\mathcal{L}_i = \{G_i^1, G_i^2, \ldots, G_i^{s_i}\}$ of s_i generalized domains, where s_i is a positive integer, $G_i^1 \succeq G_i^2 \succeq \cdots \succeq G_i^{s_i}$ and $G_i^1 = \{[min_{D,i}, max_{D,i}]\}$. Then, we define the *universal domain set*, denoted by $\mathcal{U}$, to be the set of *all* possible global domains such that for each $\mathcal{G} = (G_1, \ldots, G_d)$ in $\mathcal{U}$, $G_i \in \mathcal{L}_i$ ($i \in [1, d]$). That is, $\mathcal{U} = \mathcal{L}_1 \times \cdots \times \mathcal{L}_d$. In the following, we will use $\mathcal{U}$ as the multi-level generalization scheme to form different clustering results.

Fairness of Top-k Set. After introducing how to form clusters in a top-k set with the help of global recoding, we are ready to formalize our fairness measurement for a top-k set, say T. Given T and a global domain $\mathcal{G}$ on D, we denote $CL_{\mathcal{G}}(T)$ to be the set of all clusters in T under $\mathcal{G}$. Following our first design principle about the group fairness, we first define a criterion called *α-fair* for each cluster C in $CL_{\mathcal{G}}(T)$ based on a user-specified fairness parameter α.

Definition 2.1 (α-fair)**.** Given the set $\mathcal{P}$ of all groups in D and a non-negative real number $\alpha \in [0, \frac{1}{|\mathcal{P}|}]$, a cluster C in $CL_{\mathcal{G}}(T)$ is said to be *α-fair* if for each group $j \in \mathcal{P}$, $|C[j]| \geq \lceil \alpha \cdot |C| \rceil$, where $C[j]$ denotes the set of all tuples in C whose group is j.

To exemplify, consider the clustering result in Table 2 and $\alpha = 0.2$. The first cluster C_1 (with 4 applicants of ID 1-4) is α-fair since the numbers of males and females are both at least $\lceil \alpha \cdot |C_1| \rceil = 1$. However, the second cluster C_2 (with applicants of ID 5, 7 and 9) is not α-fair since the number of female applicants is 0 ($< \lceil \alpha \cdot |C_2| \rceil = 1$).

Note that the parameter α is at most $\frac{1}{|\mathcal{P}|}$ because it is impossible that the size of each $C[j]$ for $j \in \mathcal{P}$ is $\lceil \alpha \cdot |C| \rceil$ for $\alpha > \frac{1}{|\mathcal{P}|}$. Moreover, α controls the strength of fairness where a larger (resp. smaller) α implies a stronger (resp. looser) control. When α is (close to) $\frac{1}{|\mathcal{P}|}$, a cluster C cannot be α-fair unless each group has an almost equal number of members in C. In contrast, when α is 0, any cluster in $CL_{\mathcal{G}}(T)$ is α-fair since there is *no* restriction for the fairness.

After defining the α-fair criterion for each cluster in T, we formalize the *aggregated* fairness for all clusters in $CL_{\mathcal{G}}(T)$ based on our second design principle where we do *not* require that *all* clusters are α-fair. We first consider a group $j \in \mathcal{P}$. We define the *α-satisfied proportion* of T w.r.t. group j under global domain $\mathcal{G}$, denoted by $\alpha\text{-}SP_{\mathcal{G}}(T, j)$, to be the proportion of tuples of group j that are in the α-fair clusters in $CL_{\mathcal{G}}(T)$ among all tuples of group j in D. That is,

$$\alpha\text{-}SP_{\mathcal{G}}(T, j) = \frac{\sum_{C \in CL_{\mathcal{G}}(T) \text{ and } C \text{ is } \alpha\text{-fair}} |C[j]|}{|D[j]|} \quad (1)$$

Consider the female group in the top-7 set T in Table 2. The number of females in the first cluster (which is α-fair) is 1, while the number of females in the second cluster (which is *not* α-fair) is 0. Thus, $\alpha\text{-}SP_{\mathcal{G}}(T, \text{Female}) = (1+0)/3 = 0.333$. Similarly, $\alpha\text{-}SP_{\mathcal{G}}(T, \text{Male}) = (3+0)/6 = 0.5$. Note that α-satisfied proportion is a value in $[0, 1]$, and a larger value indicates that more tuples in a group are in α-fair clusters. In other words, a larger value of $\alpha\text{-}SP_{\mathcal{G}}(T, j)$ means that the fairness of group j is better granted. This is not a "hard" fairness constraint but instead, it gives a flexible measurement of fairness.

To ensure the fairness for *all* groups, we capture the *overall* fairness of a top-k set T by the *minimum* $\alpha\text{-}SP_{\mathcal{G}}(T, j)$ over all groups j in $\mathcal{P}$, after clustering T with global domain $\mathcal{G}$.

Definition 2.2. Given a top-k set T and a global domain $\mathcal{G}$, the *α-minimum satisfied proportion* of T under $\mathcal{G}$, denoted by $\alpha\text{-}MinSP_{\mathcal{G}}(T)$, is the minimum α-satisfied proportion of T under $\mathcal{G}$ w.r.t. each group $j \in \mathcal{P}$, i.e., $\alpha\text{-}MinSP_{\mathcal{G}}(T) = \min_{j \in \mathcal{P}} \alpha\text{-}SP_{\mathcal{G}}(T, j)$.

In our running example for the top-7 set T in Table 2, the α-minimum satisfied proportion of T under $\mathcal{G}$ (i.e., $\alpha\text{-}MinSP_{\mathcal{G}}(T)$) is equal to 0.333, which is the minimum between $\alpha\text{-}SP_{\mathcal{G}}(T, \text{Female})$ ($= 0.333$) and $\alpha\text{-}SP_{\mathcal{G}}(T, \text{Male})$ ($= 0.5$).

Recall that, following the multi-level generalization scheme of global recoding, we use the global domains in a universal domain set $\mathcal{U}$ to generate different clustering results for a given top-k set T. For each $\mathcal{G}$ in $\mathcal{U}$, we compute $\alpha\text{-}MinSP_{\mathcal{G}}(T)$. Let $\mathcal{G}^*$ be the *optimal* global domain in $\mathcal{U}$ with maximum $\alpha\text{-}MinSP_{\mathcal{G}^*}(T)$, i.e., $\mathcal{G}^* = \arg\max_{\mathcal{G} \in \mathcal{U}} \alpha\text{-}MinSP_{\mathcal{G}}(T)$. We define the *$\alpha$-fairness* of T under $\mathcal{U}$,

Table 3: Frequently Used Notations

Notation	Description
D and t	A dataset and a tuple in the dataset
$\mathcal{P}$	The set of groups
$D[j]$	The set of tuples of D in group j
w and $f_w(t)$	A utility function and the utility of t w.r.t. w
top-k_w / T	The top-k set (in D) w.r.t. w or T for simplicity
$\mathcal{G}$ / $\mathcal{U}$	A global domain / a universal domain set
α and β	The fairness parameters
α-$MinSP_{\mathcal{G}}(T)$	The α-minimum satisfied proportion of T under $\mathcal{G}$
α-$FN_{\mathcal{U}}(T)$	The α-fairness of T under $\mathcal{U}$
$m(w, w_0)$	The modification penalty of w from w_0
Ω and ρ	The utility space Ω and a region in Ω
$h_{t,t'}$	The hyperplane between t and t'
$\mathcal{H}$	A set of hyperplanes
$\mathcal{A}(\mathcal{H}, \Omega)$	An arrangement of $\mathcal{H}$ bounded by Ω
top-k_ρ	The unique top-k set (in D) w.r.t. any w in ρ
C_ρ and t_A	The candidate set for region ρ and an anchor in C_ρ
$\overline{C}$	The set of tuples in C_ρ dominating t_A
$\overline{C}^{omp}$	The set of dominating competitors of t_A

denoted by α-$FN_{\mathcal{U}}(T)$, to be the optimal α-$MinSP_{\mathcal{G}^*}(T)$ under $\mathcal{G}^*$.

Continue our running example for the top-7 set T in Table 2. We have already computed that α-$MinSP_{\mathcal{G}}(T) = 0.333$ for global domain $\mathcal{G} = (\{[87, 100), [100, 115]\}, \{[310, 320), [320, 335]\}, \{[2.6, 4.0]\})$. Consider the universal domain set $\mathcal{U}$ with 2 global domains, one is $\mathcal{G}$, and the other is $\mathcal{G}' = (\{[87, 115]\}, \{[310, 335]\}, \{[2.6, 4.0]\})$ (i.e., $\mathcal{U} = \{\mathcal{G}, \mathcal{G}'\}$). Note that according to the formal defintion of a universal domain set $\mathcal{U}$, there could be other global domains with different combinations (e.g., ({[87, 100), [100, 115]}, {[310, 335]}, {[2.6, 4.0]})), but here, we only use two global domains $\mathcal{G}$ and $\mathcal{G}'$ in $\mathcal{U}$ for easier illustration. It is easy to verify that α-$MinSP_{\mathcal{G}'}(T) = 0$ since all tuples in T form one cluster (containing 6 males and 1 female) which is not α-fair when $\alpha = 0.2$. Therefore, we obtain that the optimal global domain in $\mathcal{U}$ is $\mathcal{G}$ and the α-fairness of T under $\mathcal{U}$, denoted by α-$FN_{\mathcal{U}}(T)$, is 0.333.

Consider a universal domain set $\mathcal{U}$. A larger α-fairness value of a top-k set T under $\mathcal{U}$ indicates that this top-k set is fairer. This is because larger α-$FN_{\mathcal{U}}(T)$ implies larger α-$MinSP_{\mathcal{G}^*}(T)$, which in turn incurs larger α-$SP_{\mathcal{G}^*}(T, j)$ for each group j. This means that the fairness of each group j is better granted as we discussed previously (since more tuples from each group are in α-fair clusters).

In our example shown in Table 1, let T' be the top-7 set w.r.t. utility function f_1. That is, T' contains the applicants of ID 1-4 and 6-8 (with 4 males and 3 females) and thus T' is considered as a fairer set than the top-7 set T in Table 2. It is easy to verify that the α-fairness of T' under the same universal domain set $\mathcal{U}(= \{\mathcal{G}, \mathcal{G}'\})$ is 0.667, which is larger than α-$FN_{\mathcal{U}}(T) = 0.333$.

We summarize the frequently used notations in Table 3.

2.3 Fair Ranking Problem

In this work, we solve the fair ranking problem by designing a fair ranking scheme for the top-k query. Recall that the core of a ranking scheme is a utility function in the utility space Ω. Given a user-input utility function $w_0 \in \Omega$, we want to return a new utility function $w \in \Omega$ such that w does not deviate too much from w_0 and the set **top**-k_w is fair based on our fairness model. To quantify how far w deviates from w_0, we define the *modification penalty* of w from w_0, denoted by $m(w, w_0)$ to be the L_2 distance (a widely applied metric) between w and w_0, i.e., $m(w, w_0) = \sqrt{\sum_{i=1}^{d}(w[i] - w_0[i])^2}$.

We formalize our fair ranking problem as follows.

Problem 1 (Fair Ranking). *Given a dataset D, a real number α, a positive integer k, a universal domain set $\mathcal{U}$ and a utility function w_0, our goal is to find a utility function w^* in the utility space Ω such that the α-fairness of* **top**-k_{w^*} *under $\mathcal{U}$ (i.e., α-$FN_{\mathcal{U}}$(**top**-k_{w^*})) is maximized, while under the maximized α-$FN_{\mathcal{U}}$(**top**-k_{w^*}), the modification penalty of w^* from w_0 is minimized. Mathematically,*

$$w^* = \arg\min_{w' \in \Omega} m(w', w_0)$$
$$s.t. \quad \alpha\text{-}FN_{\mathcal{U}}(\textbf{top}\text{-}k_{w^*}) = \max_{w \in \Omega} \alpha\text{-}FN_{\mathcal{U}}(\textbf{top}\text{-}k_w).$$

In our running example, we are given a dataset D as shown in Table 1, $\alpha = 0.2$, $k = 7$, $\mathcal{U} = \{\mathcal{G}, \mathcal{G}'\}$ as shown previously and $w_0 = (0.2, 0.1, 0.7)$, we want to find a utility function w^* such that α-$FN_{\mathcal{U}}$(**top**-k_{w^*}) is maximized, while the modification penalty of w^* from w_0 is minimized. It can be obtained that the returned utility function w^* for this problem instance is $(0.21, 0.1, 0.69)$ which results in the fairest top-k set containing the applicants of ID 1-7 with the maximum α-fairness equal to 0.667 and the minimum modification penalty equal to 0.014, respectively.

3 EXACT ALGORITHM

In this section, we first present our exact algorithm called **FairTQ-Exact** for our fair ranking problem, which guarantees to find the top-k set in dataset D with the highest α-fairness and the corresponding utility function with the smallest modification penalty.

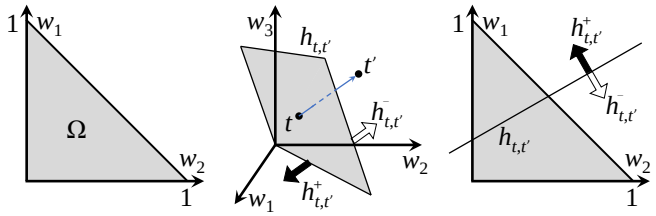
3.1 Overview

In this section, we give an overview of **FairTQ-Exact** first. Note that there are an infinite number of utility functions in the utility space Ω. Thus, it is infeasible to enumerate all utility functions to compute the top-k sets. Alternatively in this work, we adapt the hyperplane-based approach (a standard approach in the literature) to obtain the top-k sets [23, 30], which is much more efficient than the straightforward way of checking all $\binom{|D|}{k}$ subsets of D.

FairTQ-Exact works in the following three steps.

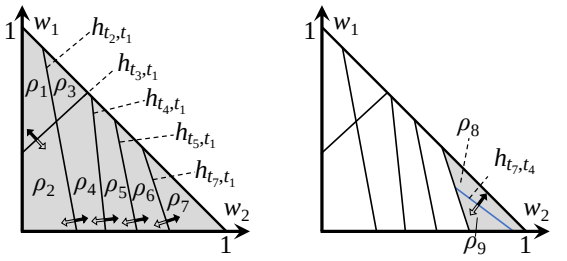
- **Step 1 (Feasible Top-k Sets Enumeration)**: We enumerate the set of all *feasible* top-k sets in D (which is much smaller than the set of all possible $\binom{|D|}{k}$ subsets of D) using an efficient hyperplane-based approach with effective optimizations.
- **Step 2 (α-fairness Computation)**: For each feasible top-k set T, we compute its α-fairness under the universal domain set $\mathcal{U}$ (i.e., α-$FN_{\mathcal{U}}(T)$) by finding the *optimal* global domain $\mathcal{G}^*$ in $\mathcal{U}$.
- **Step 3 (Best Utility Function Finding)**: For those top-k sets with the maximized α-fairness (possibly many), we find the *best* utility function w^* with the minimum modification penalty (among all those top-k sets) using *quadratic programming*.

Next, we present the details of the above three steps in Sections 3.2, 3.3 and 3.4, respectively. In Section 3.5 we provide two effective speedup techniques to accelerate the above steps.



(a) Utility Space (b) 3D Hyperplane (c) 2D Hyperplane

Figure 1: Examples of Utility Space and Hyperplane



(a) Initial Partitioning (b) Recursive Partitioning

Figure 2: Examples of Partition

3.2 Step 1 (Feasible Top-k Sets Enumeration)

We first present how to enumerate all top-k sets in D efficiently with a hyperplane-based approach. The intuition is to divide the utility space Ω into a number of small regions by the *arrangement* of a set $\mathcal{H}$ of hyperplanes, where each cell of the arrangement is a region bounded by some hyperplanes in $\mathcal{H}$ and it corresponds to a unique top-k set. Thus, to enumerate all top-k sets, we can enumerate all cells in this arrangement and find the top-k set for each cell.

We first formalize some geometric concepts. For each tuple t in D, it can be represented as a point in the d-dimensional space. Given two tuples t and t', we define the hyperplane between t and t', denoted by $h_{t,t'}$, to be the hyperplane in the d-dimensional space passing through the origin with $t'-t$ as its normal vector, i.e., $h_{t,t'} = \{w \in \mathbb{R}^d | (t'-t)\cdot w = 0\}$. In Figure 1(b), we give an example of hyperplane $h_{t,t'}$ (shown as a shaded plane) between tuple t and t' in the 3-dimensional space, where the normal of $h_{t,t'}$ is $t'-t$ (shown as a blue arrow). Recall that we reduced one dimension and consider the $(d-1)$-dimensional space for simplicity. We thus represent the hyperplane $h_{t,t'}$ in the $(d-1)$-dimensional space, by discarding the last dimension, e.g., in Figure 1(c), $h_{t,t'}$ is represented as a straight line in the (reduced) 2-dimensional space.

It can be observed that $h_{t,t'}$ divides the $(d-1)$-dimensional space into two halfspaces [30]. The halfspace containing all utility functions w such that $f_w(t) \geq f_w(t')$ (resp. $f_w(t) < f_w(t')$) is denoted by $h^+_{t,t'}$ (resp. $h^-_{t,t'}$), i.e., $h^+_{t,t'} = \{w \in \mathbb{R}^d | (t'-t)\cdot w \leq 0\}$ and $h^-_{t,t'} = \{w \in \mathbb{R}^d | (t'-t)\cdot w > 0\}$. In Figures 1(b) and (c), we indicate $h^+_{t,t'}$ and $h^-_{t,t'}$ with solid and hollow arrows, respectively, in both the (original) 3-dimensional space and the (reduced) 2-dimensional space.

A simple hyperplane-based approach can be implemented as follows. Firstly, for each pair of tuples t and t' in D, we create a hyperplane $h_{t,t'}$ and insert it to a set $\mathcal{H}$, which is initialized to an empty set. Secondly, we compute an arrangement in the $(d-1)$-dimensional space of all hyperplanes in $\mathcal{H}$, bounded by the utility space Ω. We denote this arrangement by $\mathcal{A}(\mathcal{H}, \Omega)$. Thirdly, for each cell of $\mathcal{A}(\mathcal{H}, \Omega)$ (i.e., a region in Ω, say ρ), we randomly pick a utility function w in ρ and compute the top-k set w.r.t. w (i.e., **top**-k_w) and insert it into the result set, which is initialized to an empty set. The correctness of this simple implementation is guaranteed by the following lemma, which says that the top-k set w.r.t. any utility function in ρ is the same; in this case (i.e., the top-k set is unique), we simply call it the top-k set w.r.t. ρ, denoted by **top**-k_ρ. For the sake of space, we give a proof sketch of each lemma/theorem in the paper, and the detailed proof can be found in Section C.

Lemma 3.1. *If w and w' are two utility functions in the same cell of arrangement $\mathcal{A}(\mathcal{H}, \Omega)$, then* **top**-$k_w$ = **top**-$k_{w'}$.

Proof sketch. If w and w' are in the same cell, the relative ranks of any two tuples in D are the same, since the halfspaces that contain w and w' are exactly the same. The same global ranking of all tuples in D clearly leads to the same top-k set w.r.t. w and w'. □

Unfortunately, this implementation is inefficient since there are $O(|D|^2)$ hyperplanes in $\mathcal{H}$ and computing the arrangement of $\mathcal{H}$ takes $O(|\mathcal{H}|^{d-1})$ time [23]. In the following, we present a more efficient hyperplane-based approach for computing the top-k sets. It is mainly inspired by [23], but we explore effective optimizations.

Efficient Hyperplane-based Approach. Our hyperplane-based approach mainly has two steps. The first step is a *filtering step*, which extracts a subset D' of D, such that $|D'|$ is significantly smaller than $|D|$ and only tuples in D' can appear in a top-k set. The second step is a *refinement step* which enumerates all feasible top-k sets in much smaller D' rather than D, which is very efficient.

The filtering step is handled by the k-*skyband* query [25]. Specifically, given tuples t and t', t is said to *dominate* t' if $t[i] \geq t'[i]$ for each $i \in [1, d]$ and there exists $i \in [1, d]$ such that $t[i] > t'[i]$. Clearly, if t dominates t', t has a higher utility and thus, a higher *rank* than t' w.r.t. any utility function in Ω. A tuple $t \in D$ is called a k-*skyband* [25] of D if t is dominated by fewer than k tuples in D. A k-skyband query returns the set D' of all k-skybands in D. It is easy to verify that if a tuple t is not a k-skyband of D (i.e., t is dominated by at least k tuples), then t cannot appear in any top-k set, since its rank is lower than at least k tuples in D w.r.t. any utility function in Ω. In the remaining steps, we merely focus on D' instead of D.

Example 3.1. Consider the dataset D in Table 1 where $k = 3$. For better illustration, we denote tuple t_i to be the applicant of ID i. Consider tuple t_9, which is dominated by 3 tuples in D (i.e., t_2, t_5 and t_7), and thus, t_9 is not a k-skyband of D. After other tuples are processed in a similar way, the set D' of all k-skybands of D is $\{t_1, \ldots, t_7\}$. □

In the refinement step, the core idea is to select an *anchor* from a set of *candidate* tuples (which is initially D'). Then, a region (which is initially set to Ω) is decomposed into sub-regions based on a "local" arrangement (i.e., an arrangement bounded by this region) of a set of hyperplanes each of which is a hyperplane between a *competitor* in the candidate set and the anchor. This above process is recursively executed for all sub-regions until we decompose the whole utility space Ω into desired regions where the top-k set w.r.t. each region is determined. Intuitively, by processing a region with an anchor, we can prune many candidates for sub-regions inside this region and reduce the number of hyperplanes to be processed.

The refinement step is recursively conducted via *partitioning*. Initially, the region we consider is the whole utility space Ω and

the *candidate* set, denoted by C_Ω, is initialized to be the set D' of all k-skybands of D. To perform partitioning, we choose a tuple in C_Ω, says t_A, to be the *anchor* (we will introduce the strategy of anchor selection for better efficiency later). Our goal is to determine the *rank* of the anchor among all tuples in C_Ω. We then perform the following steps to accomplish this goal. For the ease of understanding, we will explain each step using concrete examples.

Firstly, we find all *competitors* of anchor t_A from C_Ω, where a tuple $t_C \in C_\Omega$ is said to be a competitor of t_A if neither t_A is dominated by t_C nor t_A dominates t_C. Note that for a non-competitor of t_A, we already know whether it ranks higher/lower than t_A based on the dominance relation. Secondly, for each competitor t_C of t_A, we create the hyperplane h_{t_C,t_A} and insert it into a set $\mathcal{H}$ which is initialized to an empty set. We then compute the arrangement of all hyperplanes in $\mathcal{H}$ bounded by Ω, denoted by $\mathcal{A}(\mathcal{H}, \Omega)$. Thirdly, for each cell of $\mathcal{A}(\mathcal{H}, \Omega)$ (which is a region, say ρ), we find all the *dominating competitors* of t_A w.r.t. ρ where a competitor t_C is said to be a dominating competitor of t_A w.r.t. ρ if ρ is completely contained inside $h^+_{t_C,t_A}$. Intuitively, by the definition of $h^+_{t_C,t_A}$, $f_w(t_C) > f_w(t_A)$ for any utility function w in ρ. In other words, the rank of t_C is higher than t_A w.r.t. any utility function w in ρ and thus, t_C is a "dominating" competitor.

Example 3.2. Assume that we select t_1 in $D' = \{t_1, \ldots, t_7\}$ to be the anchor. The competitors of anchor t_1 are t_2, t_3, t_4, t_5 and t_7 (but not t_6 since t_1 dominates t_6). We insert 5 hyperplanes, namely h_{t_2,t_1}, h_{t_3,t_1}, h_{t_4,t_1}, h_{t_5,t_1} and h_{t_7,t_1}, to $\mathcal{H}$, which are shown as line segments in Figure 2(a). For each hyperplane, we also use the solid and hollow arrows to differentiate its two halfspaces (e.g., the solid arrow for h_{t_3,t_1} represents $h^+_{t_3,t_1}$). Clearly, we form an arrangement $\mathcal{A}(\mathcal{H}, \Omega)$ with 7 cells, namely regions $\rho_1, \ldots, \rho_7$. Here, t_3 is a dominating competitor of t_1 w.r.t. ρ_1, since ρ_1 lies completely inside $h^+_{t_3,t_1}$. Similarly, we can find the dominating competitors w.r.t. other regions. □

Then, for a given region ρ, we can obtain the rank of t_A w.r.t. *any* function w in ρ, denoted by k'. Intuitively, k' is equal to 1 plus the number of tuples that rank higher than t_A w.r.t. w. Let $\overline{C}$ be the set of tuples in C_Ω dominating t_A (and thus, they are *non-competitors*) and $\overline{C}^{omp}$ be the set of all dominating competitors of t_A w.r.t. ρ. Then, k' is computed to be $1+|\overline{C}|+|\overline{C}^{omp}|$, since only tuples in $\overline{C} \cup \overline{C}^{omp}$ can rank higher than t_A w.r.t. w. As a by-product, this also gives the top-k' set w.r.t. ρ in C_Ω to be $\textbf{top}\text{-}k'_\rho(C_\Omega) = \{t_A\} \cup \overline{C} \cup \overline{C}^{omp}$, where the k'-th rank tuple is exactly t_A.

Based on the relationship between k' and k, we can classify the region ρ into 3 different types, namely *equal-to* region, *less-than* region and *greater-than* region, which are defined to be a region (resulting from anchor t_A) in which the rank of t_A w.r.t. any function w (i.e., k') is equal to k, less than k and greater than k, respectively.

- *Equal-to region (i.e., $k' = k$):* If ρ is an *equal-to* region, the top-k set w.r.t. ρ (i.e., $\textbf{top}\text{-}k'_\rho(C_\Omega)$) is already determined, which is $\{t_A\} \cup \overline{C} \cup \overline{C}^{omp}$. Thus, we do not need further processing for ρ.
- *Less-than region (i.e., $k' < k$):* If ρ is a *less-than* region, the top-k' set w.r.t. ρ is determined similarly. Moreover, since $k' < k$, we only need to know the remaining $k-k'$ tuples to return the top-k sets. To achieve this, we recursively partition ρ in a similar way as Ω, with the following modifications: (a) the region now we consider is ρ rather than Ω (i.e., future arrangement is bounded "locally" by ρ); (b) since the top-k' w.r.t. ρ is determined, we exclude them and form a new candidate set C_ρ, i.e., $C_\rho = C_\Omega \setminus (\{t_A\} \cup \overline{C} \cup \overline{C}^{omp})$; and (c) instead of finding the top-k sets, we only need to find the top-$(k-k')$ sets w.r.t. the utility functions in ρ in C_ρ. Here, the number $k-k'$ is called the *rank quota* for this partitioning.
- *Greater-than region (i.e., $k' > k$):* If ρ is a *greater-than* region, we cannot determine the top-k sets in ρ. However, we know that t_A and all tuples ranked lower than t_A cannot be in any top-k set in ρ. Thus, we will do recursive partition of ρ where the candidate set C_ρ is set to be $\overline{C} \cup \overline{C}^{omp}$, with the same rank quota k.

Example 3.3. In Figure 2(a), ρ_3 and ρ_5 are two equal-to regions, and their corresponding top-k sets are computed to be $\{t_1, t_2, t_3\}$ and $\{t_1, t_2, t_4\}$, respectively, which can be directly returned.

Region ρ_1 is a less-than region with its top-2 set $\{t_1, t_3\}$ identified, where the rank of anchor t_1 is $k' = 2$. Then, ρ_1 will be further processed with rank quota 1 $(= 3-2)$ and the candidate set $C_{\rho_1} = \{t_2, t_4, t_5, t_6, t_7\}$. Note that t_1 and t_3 are not in C_{ρ_1}, since their ranks are already known. Our goal becomes finding the top-1 set in C_{ρ_1}. To do this, we select t_2 to be the new anchor, which dominates all other tuples in C_{ρ_1}, indicating that t_2 is exactly the top-1 set w.r.t. ρ_1 and ρ_1 will not be further partitioned. Combining $\{t_2\}$ with the previous top-2 set $\{t_1, t_3\}$, the final top-3 set returned for ρ_1 is $\{t_1, t_2, t_3\}$.

Region ρ_7 is a greater-than region since the rank k' of anchor t_1 is $5 > k(= 3)$, shown in the shaded region in Figure 2(b). Thus, ρ_7 is also recursively partitioned, with the same rank quota $k = 3$. The new candidate set C_{ρ_7} is $\{t_2, t_4, t_5, t_7\}$, with 3 tuples excluded: (a) anchor t_1 with rank $k' > k$, (b) t_6, which is dominated by t_1 and thus, has a lower rank than t_1, and (c) t_3, which has a lower rank than t_1 since ρ_7 lies in $h^-_{t_3,t_1}$ (i.e., t_3 is *not* a dominating competitor t_1 w.r.t. ρ_7). Assume that we select t_4 as the new anchor for ρ_7. Since t_4 is dominated by t_2 and t_4 dominates t_5, the only competitor of t_4 is t_7. We thus insert h_{t_7,t_4} (shown as a blue line segment in Figure 2(b)), resulting in two sub-regions, namely ρ_8 and ρ_9. By the definition of h_{t_7,t_4}, ρ_8 (resp. ρ_9) contains all utility functions w such that t_7 ranks higher (resp. lower) than t_4 w.r.t. w. As a result, ρ_8 is an equal-to region with the top-3 set $\{t_2, t_4, t_7\}$, while ρ_9 is a less-than region with the top-2 set $\{t_2, t_4\}$ and it will be further partitioned recursively. □

For each recursive partitioning of region ρ, we need to choose a new anchor from the candidate set C_ρ. To avoid the case where all sub-regions are less-than or greater-than regions (i.e., all of them need further processing), we use an efficient strategy to select the anchor for each partitioning, to ensure that there is at least one equal-to region. Specifically, we randomly pick a utility function w from ρ and find the top-k set w.r.t. w (i.e., $\textbf{top}\text{-}k_w(C_\rho)$). Then, we pick the k-th rank tuple in $\textbf{top}\text{-}k_w(C_\rho)$ as the new anchor.

Finally, the whole utility space Ω will be partitioned into multiple equal-to regions. For each region ρ, we can obtain the (unique) top-k set w.r.t. ρ, say $\textbf{top}\text{-}k_\rho$. As a result, all these top-k sets are returned.

Remark. We differ from [23] in the optimizations adopted below. (1) After obtaining the competitors of an anchor t_A for a region ρ, we process *all* competitors, by creating a hyperplane for *each* competitor. Thus, we can deduce the dominance relationships between t_A and all competitors. In [23], only *part* of competitors are processed and some remaining ones are explored in later partitioning. In this case, the type of some sub-regions cannot be determined im-

mediately. However, by processing *all* competitors, we can not only use the learned dominance relationships in future partitioning of all sub-regions of ρ, but also determine the type of all sub-regions in current partitioning. This allows us to apply our *speedup techniques* (Section 3.5) as early as possible. (2) When conducting recursive partitioning for a sub-region ρ' of region ρ, we consider not only the *static* relationships that hold globally in the dataset, but also the *dynamic* relationships (which varies for different sub-regions) that we deduce during previous partitioning, to reduce the candidates that we need to consider. In [23], only static relationships are used.

3.3 Step 2 (α-fairness Computation)

For each feasible top-k set T obtained, we then compute its α-fairness under a universal domain set $\mathcal{U}$ (i.e., $\alpha\text{-}FN_{\mathcal{U}}(T)$).

Since $\alpha\text{-}FN_{\mathcal{U}}(T) = \max_{\mathcal{G}\in\mathcal{U}} \alpha\text{-}MinSP_{\mathcal{G}}(T)$, a naive approach of computing the α-fairness is to enumerate all global domains in $\mathcal{U}$ and find the one with the maximum $\alpha\text{-}MinSP_{\mathcal{G}}(T)$. However, this is time-consuming. We thus propose a more efficient strategy, by computing an upper bound of $\alpha\text{-}MinSP_{\mathcal{G}}(T)$ for any $\mathcal{G} \in \mathcal{U}$.

Lemma 3.2. *Given a top-k set T in D and a universal domain set $\mathcal{U}$, $\alpha\text{-}MinSP_{\mathcal{G}}(T) \le \min_{j\in\mathcal{P}} \frac{|T[j]|}{|D[j]|}$ for any $\mathcal{G} \in \mathcal{U}$, where $T[j]$ denotes the set of all tuples in T whose group is j. Besides, $\alpha\text{-}MinSP_{\mathcal{G}}(T) = \min_{j\in\mathcal{P}} \frac{|T[j]|}{|D[j]|}$ if and only if all clusters in $CL_{\mathcal{G}}(T)$ are α-fair.*

Proof sketch. The first inequality clearly holds by Equation 1 and Definition 2.2. Similarly, if all clusters in $CL_{\mathcal{G}}(T)$ are α-fair, $\alpha\text{-}MinSP_{\mathcal{G}}(T) = \min_{j\in\mathcal{P}} \frac{|T[j]|}{|D[j]|}$ clearly holds. Reversely, assume that there exists a cluster in $CL_{\mathcal{G}}(T)$ that is not α-fair, it is easy to deduce a contradiction that $\alpha\text{-}MinSP_{\mathcal{G}}(T) < \min_{j\in\mathcal{P}} \frac{|T[j]|}{|D[j]|}$. □

Lemma 3.2 shows that if there exists a global domain $\mathcal{G}$ in $\mathcal{U}$ such that all clusters in $CL_{\mathcal{G}}(T)$ are α-fair, then the maximum possible $\alpha\text{-}MinSP_{\mathcal{G}}(T)$ ($= \min_{j\in\mathcal{P}} \frac{|T[j]|}{|D[j]|}$) is reached. Since other global domain in $\mathcal{U}$ cannot give a larger $\alpha\text{-}MinSP_{\mathcal{G}'}(T)$, we can immediately stop the enumeration of $\mathcal{U}$. In this case, $\mathcal{G}$ is exactly the *optimal* global domain in $\mathcal{U}$, which gives the desired maximum α-fairness. After computing the α-fairness of each feasible top-k set, we can easily obtain a set $\mathcal{T}_{\max}$ of top-k sets with the maximum α-fairness.

3.4 Step 3 (Best Utility Function Finding)

After obtaining a set $\mathcal{T}_{\max}$ of all top-k sets with the maximum α-fairness, we find the best utility function $w^* \in \Omega$ with the minimum modification penalty $m(w^*, w_0)$ from the given utility function w_0.

Recall that for each top-k set T in $\mathcal{T}_{\max}$, $T = \textbf{top-}k_\rho$ where ρ is a region in Ω such that the top-k set w.r.t. any utility function in ρ is the same. The problem is thus to find the nearest utility function of w_0 in ρ under the L_2 distance, which can be solved by *Quadratic Programming (QP)*. Denote by w_ρ the best utility function in region ρ. We formulate the following QP to compute the desired w_ρ:

$$
\begin{aligned}
w_\rho = \quad & \arg\min_{w} m(w, w_0) \\
\text{s.t.} \quad & (t' - t)\cdot w \le 0, \quad \forall h_{t,t'} \in \mathcal{H}^+ && (1)\\
& (t' - t)\cdot w > 0, \quad \forall h_{t,t'} \in \mathcal{H}^- && (2)\\
& \textstyle\sum_{i=1}^{d} w[i] = 1 \text{ and } 0 \le w[i] \le 1, \forall i \in [1, d] && (3)
\end{aligned}
$$

where $\mathcal{H}^+$ (resp. $\mathcal{H}^-$) is the set of hyperplanes such that for each $h_{t,t'} \in \mathcal{H}^+$ (resp. $\mathcal{H}^-$), ρ is bounded by $h_{t,t'}$ and ρ lies in $h^+_{t,t'}$ (resp. $h^-_{t,t'}$). Intuitively, Constraints (1) and (2) ensure that w is in targeted region ρ and Constraint (3) specifies the utility space Ω. To solve this QP, we adopt the interior-point algorithm QuadProg [22].

We solve the above QP for each top-k set in $\mathcal{T}_{\max}$. Finally, the best utility function w^* with the minimum modification penalty is returned and we obtain the optimal top-k set $T^* = \textbf{top-}k_{w^*}$.

3.5 Speedup Techniques

In this section, we introduce two effective speedup techniques which combine top-k set enumeration with minimum modification penalty and α-fairness, respectively. Recall that during partitioning in Section 3.2, we only terminate the processing for *equal-to* regions. Our speedup techniques complement this, by pruning more regions.

Pruning by modification penalty. The first speedup technique is based on an upper bound of $\alpha\text{-}FN_{\mathcal{U}}(T)$ for each top-k set T in D.

Lemma 3.3. *For any top-k set T in D, we have $\alpha\text{-}FN_{\mathcal{U}}(T) \le \frac{k}{|D|}$.*

Proof sketch. Assume there exists a top-k set T such that $\alpha\text{-}MinSP_{\mathcal{G}}(T) > \frac{k}{|D|}$. By Lemma 3.2, we have $\min_{j\in\mathcal{P}} \frac{|T[j]|}{|D[j]|} > \frac{k}{|D|}$. Then, $\forall j \in \mathcal{P}$, $\frac{|T[j]|}{k} > \frac{|D[j]|}{|D|}$. Summing up all groups, we could easily deduce a contradiction of $1 > 1$. □

Based on Lemma 3.3, since $\frac{k}{|D|}$ is the upper bound of α-fairness of all top-k sets in D, if we find a top-k set $T(= \textbf{top-}k_\rho)$ such that its α-fairness equals to $\frac{k}{|D|}$, then T is ensured to reach the maximum α-fairness and is thus ensured to be included in set $\mathcal{T}_{\max}$. Denote by w_ρ the utility function returned by the QP for this top-k set T. Recall that we want to find the best utility function w^* for the optimal top-k set $T^* \in \mathcal{T}_{\max}$ which has the minimum modification penalty. Then we know that $m(w^*, w_0) \le m(w_\rho, w_0)$. In other words, the penalty $m(w_\rho, w_0)$ is an upper bound of the optimal penalty. Therefore, we can directly terminate the further partitioning of a new region to be processed (regardless of its type) if there does not exist a utility function w in this region such that $m(w, w_0) \le m(w_\rho, w_0)$.

Pruning by α-fairness. The second speedup technique is particularly designed for less-than regions. Recall that in a less-than region, we determine the top-k' set, where $k' < k$. That is, any top-k set in this region must include those k' tuples whose group information is already known. Based on this, we can obtain an upper bound of the α-fairness of any top-k set in this region, even when the group information about the remaining $k - k'$ tuples is unknown.

Lemma 3.4. *Given a top-k set T of D and a top-k' set T' of D where $k' < k$, if $T' \subset T$, then $\alpha\text{-}FN_{\mathcal{U}}(T) \le \min_{j\in\mathcal{P}} \frac{|T'[j]|+k-k'}{|D[j]|}$, where $T'[j]$ denotes the set of all tuples in T' whose group is j.*

Proof sketch. Let j_{min} denote the group in $\mathcal{P}$ such that $\frac{|T[j]|}{|D[j]|}$ is minimized. By Lemma 3.2, $\alpha\text{-}FN_{\mathcal{U}}(T) \le \frac{|T[j_{min}]|}{|D[j_{min}]|}$. Let $\overline{T}$ denote $T \setminus T'$. Since $T' \subset T$, $|\overline{T}[j_{min}]| \le |\overline{T}| = k - k'$. Thus, $\frac{|T[j_{min}]|}{|D[j_{min}]|} = \frac{|T'[j_{min}]|+|\overline{T}[j_{min}]|}{|D[j_{min}]|} \le \frac{|T'[j_{min}]|+k-k'}{|D[j_{min}]|} \le \min_{j\in\mathcal{P}} \frac{|T'[j]|+k-k'}{|D[j]|}$. □

Based on Lemma 3.4, once we obtain a less-than region with a determined top-k' set (say T'), we can compute the upper

bound of the α-fairness of any top-k set T that contains T' to be $\min_{j\in\mathcal{P}} \frac{|T'[j]|+k-k'}{|D[j]|}$. If this upper bound is smaller than the best α-fairness observed so far, then we can terminate the processing of this less-than region, since all top-k sets in this region (which are ensured to contain T') cannot have a better α-fairness.

Correctness. Finally, we show the correctness of our **FairTQ-Exact** algorithm in the following theorem.

Theorem 3.4 (Correctness). *Given a dataset D, a real number α, a positive integer k, a universal domain set $\mathcal{U}$ and a utility function w_0, the **FairTQ-Exact** algorithm returns the utility function w^* such that the α-fairness, α-$FN_{\mathcal{U}}(\textbf{top}\text{-}k_{w^*})$, is maximized and meanwhile, $m(w^*, w_0)$ is minimized under the maximized α-fairness.*

Proof sketch. Step 1 ensures to enumerate all feasible top-k sets in D by [23], and thus the correctness is obviously guaranteed for Step 2 and Step 3. Based on Lemma 3.3 and 3.4, the speedup techniques only prunes top-k sets that cannot give better α-fairness or better utility function with smaller penalty. □

4 APPROXIMATION ALGORITHM

Although **FairTQ-Exact** adopts effective speedup techniques, we still have to compute the α-fairness of each feasible top-k set so that the fairest top-k sets can be obtained. When an *approximately* fairest top-k set is good enough, the *fairest* top-k sets might be unnecessary. Thus, we introduce our approximation algorithm called β-**FairTQ-Appro** to trade-off fairness, efficiency and modification penalty.

Consider a top-k set T. The major idea of β-**FairTQ-Appro** is to apply a *relaxed* fairness criterion such that verifying whether T satisfies this criterion is fast and if T satisfies this criterion, then T is approximately the fairest (i.e., the α-fairness of T is close to the maximum α-fairness). Once T is verified to satisfy this criterion, we can avoid the exhaustive search through a universal domain set $\mathcal{U}$ to compute α-$FN_{\mathcal{U}}(T)$ (in Section 3.3), and directly include T as a candidate top-k set in Step 3 for finding the best utility function. Since more candidate top-k sets are considered for finding the best utility function by the relaxed criterion, it is possible for us to obtain an even better utility function (i.e., a utility function with even smaller modification penalty than the exact solution).

We first formalize our relaxed fairness criterion called *β-relaxed-fair*. Recall that $T[j]$ (resp. $C[j]$) denotes the set of tuples in a top-k set T (resp. a cluster C in $CL_{\mathcal{G}}(T)$ where $\mathcal{G} \in \mathcal{U}$) whose group is j. Given a top-k set T and another fairness parameter $\beta \in [0, \frac{1}{|\mathcal{P}|}]$, we say that T is *β-relaxed-fair* if for each group $j \in \mathcal{P}$, $|T[j]| \geq \lceil \beta k \rceil$. It differs from α-fair (i.e., for each group $j \in \mathcal{P}$, $|C[j]| \geq \lceil \alpha \cdot |C| \rceil$, $\forall C \in CL_{\mathcal{G}}(T)$), by being defined on the top-k set T itself, rather than each cluster in T, and using a different parameter β to control the fairness (note that it is possible to set $\beta = \alpha$, which is the default setting in our experiments). Clearly, checking whether T is β-relaxed-fair is faster than computing the α-fairness of T (which finds the optimal global domain by searching through $\mathcal{U}$). More importantly, the following lemma shows that if $\beta \geq \alpha$ and T is β-relaxed-fair, its α-fairness is close to the maximized α-fairness.

Lemma 4.1. *Given the optimal top-k set T^* (i.e., α-$FN_{\mathcal{U}}(T^*)$ is maximized) and a β-relaxed-fair top-k set T in D, if $\beta \geq \alpha$, then*

$$\alpha\text{-}FN_{\mathcal{U}}(T) \geq \alpha\text{-}FN_{\mathcal{U}}(T^*) \cdot \beta.$$

Proof sketch. If T is β-relaxed-fair and $\beta \geq \alpha$, the one big cluster containing all tuples in T is α-fair. By Lemma 3.2, α-$FN_{\mathcal{U}}(T) = \min_{j\in\mathcal{P}} \frac{|T[j]|}{|D[j]|} \geq \frac{\lceil \beta k \rceil}{\max_{j\in\mathcal{P}} |D[j]|}$. By Lemma 3.3, α-$FN_{\mathcal{U}}(T^*) \leq \frac{k}{|D|}$. Thus, $\frac{\alpha\text{-}FN_{\mathcal{U}}(T)}{\alpha\text{-}FN_{\mathcal{U}}(T^*)} \geq \frac{\lceil \beta k \rceil}{\max_{j\in\mathcal{P}} |D[j]|} / \frac{k}{|D|} \geq \beta$. □

After defining the β-relaxed-fair criterion, based on Lemma 4.1, we can consider β-relaxed-fair top-k sets as approximate solutions and find the best utility function based on these top-k sets. However, care should be taken when implementing β-**FairTQ-Appro** since the fairest top-k set may not be β-relaxed-fair.

Specifically, β-**FairTQ-Appro** mainly follows the steps of the exact algorithm **FairTQ-Exact** but with some modifications. We still use a set $\mathcal{T}_{\max}$ to maintain all top-k sets with the highest α-fairness observed so far. However, we additionally maintain another set $\mathcal{T}_{\text{re}}$ to store all the β-relaxed-fair top-k sets.

We first use the same hyperplane-based approach (in Section 3.2) to enumerate all feasible top-k sets. For each feasible top-k set T obtained, we check whether T is β-relaxed-fair. If yes, we skip its exact α-fairness computation and directly insert T into $\mathcal{T}_{\text{re}}$. Otherwise, we compute the α-fairness of T as in **FairTQ-Exact** (in Section 3.3) and update $\mathcal{T}_{\max}$ if needed. After processing all feasible top-k sets, we obtain two sets, namely the set $\mathcal{T}_{\text{re}}$ of all the β-relaxed-fair top-k sets and the set $\mathcal{T}_{\max}$ of all top-k sets (that are not β-relaxed-fair). Note that different from **FairTQ-Exact**, the set $\mathcal{T}_{\max}$ may not contain the set T^* with the *(optimal) maximum* α-fairness. This is because if T^* is itself β-relaxed-fair, it is added to $\mathcal{T}_{\text{re}}$ directly, and the optimal α-fairness is not computed. In this case, $\mathcal{T}_{\max}$ only contains the top-k sets with the highest α-fairness *observed so far*, rather than the (optimal) maximum α-fairness .

Then we run the same Step 3 of finding the best utility function for both $\mathcal{T}_{\max}$ and $\mathcal{T}_{\text{re}}$ (in Section 3.4) and obtain the resulting best utility functions $w^*_{\max}$ and w^*_{re}, respectively. If only one of w^*_{re} and $w^*_{\max}$ is defined (e.g., $\mathcal{T}_{\text{re}}$ or $\mathcal{T}_{\max}$ is empty), we simply return the defined one. Otherwise, we compare the α-fairness between $T^*_{\text{re}} = \textbf{top}\text{-}k_{w^*_{\text{re}}}$ and $T^*_{\max} = \textbf{top}\text{-}k_{w^*_{\max}}$ to determine whether the set T^* with the maximum α-fairness is located in $\mathcal{T}_{\text{re}}$ or $\mathcal{T}_{\max}$.

- If T^*_{re} is fairer, we return w^*_{re} since $T^*_{\max}$ must not be the fairest (i.e., $T^* \notin \mathcal{T}_{\max}$). The fairest top-$k$ set is then in $\mathcal{T}_{\text{re}}$, which we have already considered when computing w^*_{re}.
- If $T^*_{\max}$ is fairer, we return the one from $\{w^*_{\text{re}}, w^*_{\max}\}$ with a smaller modification penalty. Here both the fairness of T^*_{re} and $T^*_{\max}$ approximate the optimum since T^*_{re} is β-relaxed-fair (by Lemma 4.1) and $T^*_{\max}$ is even fairer than T^*_{re}.

Theorem 4.1 ensures the correctness of β-**FairTQ-Appro** (when $\beta \geq \alpha$), i.e., the returned top-k set provides β-approximation on the fairness, with the modification penalty at most the optimum.

Theorem 4.1. *Given the optimal top-k set $T^* = \textbf{top}\text{-}k_{w^*}$ and the top-k set $T = \textbf{top}\text{-}k_{w}$ returned by β-**FairTQ-Appro**, if $\beta \geq \alpha$, then*

$$\alpha\text{-}FN_{\mathcal{U}}(T) \geq \alpha\text{-}FN_{\mathcal{U}}(T^*) \cdot \beta \text{ and } m(w, w_0) \leq m(w^*, w_0).$$

Proof sketch. Since the returned top-k set is either β-relaxed-fair or has larger α-fairness than a β-relaxed-fair top-k set, by Lemma 4.1, α-$FN_{\mathcal{U}}(T) \geq \alpha$-$FN_{\mathcal{U}}(T^*) \cdot \beta$. Since all the fairest top-k sets are included as candidates for finding the best utility function, the obtained utility function has at least as small penalty as w^*. □

5 RELATED WORK

We categorize the related work on fairness as follows.

Fairness Models. Recent fairness models [13, 15, 17] can be classified as follows. (1) *Group fairness* [15, 17], which requires that algorithms should *equally* treat different groups. Two commonly used group fairness models are *demographic parity* [17] and *equalized odds* [15], where the former requires that group members are equal likely to be picked for some decision-making tasks (e.g., being selected as the top-k) among all groups, while the latter requires similar group equality on specific metrics in classification. (2) *Individual fairness* [13], which requires that individuals with similar attributes should be treated similarly. While group fairness have been extensively studied, individual fairness receives less attention.

In comparison, our fairness model not only considers group fairness, by quantifying how equal treatment is given among groups, its decision-making is also driven by user preference (i.e., the utility function), which preserves individual fairness. This is because similar individuals will have similar utilities and thus they will be treated similarly (e.g., included in or excluded from the top-k set).

Fairness Problems and Algorithms. There are also studies for fairness problems and algorithms in decision making systems.

Designing Fair Ranking Scheme. [5] addresses a similar problem as ours, by designing a fair ranking scheme using a hard constraint in the fairness model (e.g., the proportion of the protected group in the returned top-k set must be at least a given threshold). However, it relies heavily on the choice of a good constraint, which could be difficult in many cases. The fairness constraint used in [5] only guarantees a minimum number on the protected group and may cause discrimination against non-protected members. We resolve this issue using a flexible fairness measurement instead.

To design a fair ranking scheme, [5] proposes an exact algorithm called *SATREGIONS*, which also adopts a hyperplane-based approach. Specifically, their hyperplanes are constructed in an angle coordinate system, which split the utility space into small regions. For each region, they check whether the corresponding top-k set satisfies a fairness constraint. Finally, the utility function in qualified regions that is closest to the input utility function is returned. Since there are many hyperplanes in *SATREGIONS*, whose geometric relationships is determined by time-consuming linear programming (LP), *SATREGIONS* is slow (e.g., inserting the first 1k hyperplanes takes > 5,000s [5]). *SATREGIONS* can be adapted to our problem by substituting their fairness model with our α-fair model to identify the desired top-k set and to accelerate. The adapted algorithm is called **SR-Adapt** as a baseline in our experiments for comparison. Compared to **FairTQ-Exact**, **SR-Adapt** is still inefficient due to high time cost for processing too many hyperplanes.

Fair Top-k Ranking. Another related problem is *fair top-k ranking* [9, 32, 33], which aims to directly return a fair top-k set in a dataset. [32] proposes a top-k ranking generator to systematically control the fairness in ranking which, however, only shows limited effectiveness. [33] proposes an algorithm called *FA*IR* to ensure the minimal proportion for a *single* protected group as in [32], by iteratively selecting candidates into the top-k set satisfying the fairness constraint and the "in-group" monotonicity of utility. However, FA*IR could discriminate unprotected groups and could not guarantee the monotonicity across groups (leading to *ranking violation*, see below). [9] selects the top-k set by maximizing the total "worthiness" (where a value is assigned to each tuple t and each rank r in a top-k set, representing the worthiness of the r-th rank tuple t), subject to a fairness constraint, which sets lower and upper bounds on the proportion of each group selected in the top-k set to grant fairness to all groups. However, both [33] and [9] could incur *ranking violation*, where a tuple t ranks higher than t' but t is dominated by t'. Clearly, if this case happens, no utility function can give such a top-k set.

We adopted FA*IR as a baseline (denoted by **FA*IR**), but in most cases, it incurs ranking violation. To avoid this, we adapt the idea of [9] to form a greedy competitor in our experiments, denoted by **Greedy**. It iteratively selects a tuple t from D such that t is not dominated by any remaining tuples in D (thus, no ranking violation) and meanwhile, t belongs to the group that contributes the least in the top-k set. A utility function with the minimum modification penalty can be returned by intersecting the halfspaces formed by every pair of tuple in the top-k set (if the intersection is non-empty).

Some recent studies explore fairness in other decision-making problems similar to ranking [7, 16, 34]. [34] studies fair happiness maximization query, which minimizes how regretful a user is if s/he obtains a subset from the dataset. [7, 16] study the fairness of preference aggregation, which finds a fair top-k result based on the preference of users. In this work, we focus on the traditional top-k queries.

Fairness in Machine Learning (ML). ML methods have been widely applied in decision-making systems, which gain increasing needs to ensure fairness. They can be classified as follows. (1) *Pre-processing approaches* [8, 26], which remove discrimination by modifying the dataset. (2) *In-processing approaches* [10, 11, 19, 20, 28], which redesign the machine learning algorithms to increase their fairness performance. (3) *Post-processing approaches* [15, 18], which directly modify the ML outputs to achieve fairness. However, these approaches need well-labeled data for training, while we capture the user preference via a utility function and thus, require no labels.

6 EXPERIMENT

We conducted extensive experiments on synthetic and real datasets to evaluate our effectiveness and efficiency. Experiments were conducted on a Linux PC with a 2.66 GHz CPU and 48 GB main memory.

Datasets. We used 4 real datasets, *XING*, *COMPAS*, *DOT* and *ALL-STAR*, and 2 synthetic datasets, *Independent* and *Anti-correlated*.

XING is a dataset collected by [33] which has 2,280 user profiles in a job finding website [2]. A fair ranking scheme is important to applicants for hiring decisions. We used two indicators *work_experience* and *education_experience* as scoring attributes and *gender* as the protected attribute to prevent unfair treatment of women in hiring.

COMPAS [4] has the criminal offense information of 6,889 individuals as an assessment for predicting recidivism. In this case, African Americans are often accessed to recidivate with a higher likelihood than other races. We thus want a fair ranking scheme to find the top-k individuals who are likely to recidivate, by setting *race* as the protected attribute. There are two groups, namely "African American" group and "others" group. Following [5], we used three scoring attributes, namely *c_days_from_compas, juv_other_count* and *start*.

DOT [24] is a flight on-time dataset with over 1 million flights conducted by 12 US carriers in the first three months of 2016. Fol-

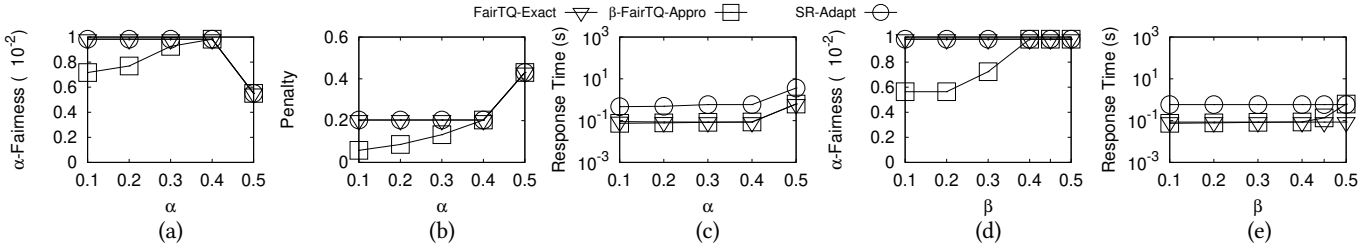


Figure 3: Effect of α and β on *XING*

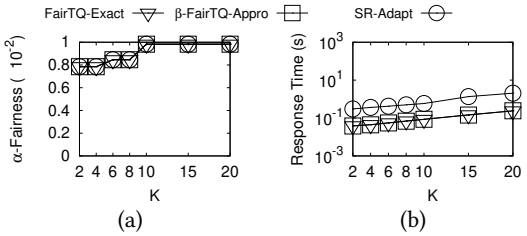


Figure 4: Effect of K on *XING*

lowing [5], we set *op_unique_carrier* as the protected attribute (with 12 groups where each group corresponds to a carrier) and set the three scoring attributes to be *dep_delay*, *taxi_in* and *arr_delay*.

ALLSTAR [3] has the statistics of 1,610 players on the National Basketball Association All-Star Game from 1950 to 2009. A fair ranking scheme finds players with the best performance, measured by four scoring attributes *pts*, *minutes*, *reb* and *asts*. For the protected attribute, we used *conference* (i.e., "eastern" or "western") such that eastern and western players are selected fairly in All-star games.

For synthetic datasets, values in *Independent* are independently generated, while in *Anti-correlated*, tuples with a large value in one attribute tend to have small values on other attributes. The protected attribute is randomly generated so that group sizes are almost equal.

For each dataset, we normalize the values of each scoring attribute into range $[0, 1]$ via the min-max normalization.

Algorithms. We evaluated our exact algorithm, **FairTQ-Exact**, and our approximation algorithm, β-**FairTQ-Appro**. The baseline algorithms (described in Section 5) are as follows. (1) **SR-Adapt**: the adapted algorithm of *SATREGIONS* [5]. (2) **Greedy**: the adapted algorithm of [9]. (3) **FA*IR**: the original algorithm of [33]. Algorithm **FA*IR** is implemented in Python (using its original implementation) and other algorithms are implemented in C++.

Parameter Settings. We study the following parameters: (1) the dimensionality d (i.e., the number of scoring attributes), (2) the dataset size $|D|$, (3) the number of groups $|\mathcal{P}|$, (4) the parameter k in the top-k query, (5) the fairness parameters α and β, and (6) the number of generalized domains K for each dimension in $\mathcal{U}$.

Our parameter settings mainly follow [5]. In real datasets, the default values of d, $|D|$ and $|\mathcal{P}|$ are those in the orginal dataset. In synthetic datasets, $d = 3$, $|D|$ =100k and $|\mathcal{P}|$=2 by default. The default values of k and α are 10 and 0.3, respectively, for all datasets.

To obtain the universal domain set $\mathcal{U}$ in our problem, for each dimension i, we generate a list $\mathcal{L}_i$ of K generalized domains where K is a positive integer (recall that $\mathcal{U}$ is formed by $\mathcal{U} = \mathcal{L}_1 \times \mathcal{L}_2 \times \cdots \times \mathcal{L}_d$, and thus the size of $\mathcal{U}$ is K^d). To obtain $\mathcal{L}_i = \{G_i^1, G_i^2, \ldots, G_i^K\}$ such that $G_i^1 \geq G_i^2 \geq \cdots \geq G_i^K$, where $G_i^1 = [0, 1]$ (by definition), we use an iterative approach. For each $j \in [2, K]$, we generate a random number c inside the largest value range in G_i^{j-1}, says $[a, b)$, and form the generalized domain G_i^j to be a set of all value ranges in G_i^{j-1} except $[a, b)$ and two newly-split ranges, $[a, c)$ and $[c, b)$.

Measurements. We adopted four measurements: (1) the response time, (2) the α-fairness of the top-k set, (3) the penalty of the returned utility function from the input utility function, (4) violation count: the sum of ranking violations occurred within the top-k set and between the top-k tuples and the remaining tuples (see Section 5). We ran each experiment 10 times, with 10 random utility functions, and report the average here. For the lack of space, we only report the results on some datasets; other results are consistent.

6.1 Results on Real Datasets

Effect of α and β. We first studied the effect of parameter α in our fairness model, by varying it from 0.1 to 0.5 on dataset *XING*; **Greedy** and **FA*IR** are excluded since they do not rely on α. We first set $\beta = \alpha$ for the approximation algorithm. When α is larger in Figure 3(a) (e.g., $\alpha > 0.4$), the α-fairness of the top-k set returned by exact algorithms has an obvious drop. The reason is that when α is larger (i.e., the user has a stricter fairness requirement), the criterion is harder to be satisfied, and thus the α-fairness of the returned top-k sets tends to be smaller. As such, the modification penalty increases (Figure 3(b)), since less utility functions satisfy the more restricted α-fairness requirement. Moreover, the upper bound of α-fairness (i.e., $\frac{k}{|D|}$) is also less likely to reach, resulting in slightly more time to process, since less regions could be pruned by the minimum penalty (Lemma 3.3), as shown in Figure 3(c). We set the default α as 0.4 on dataset *XING* to strike a balance between the fairness (note that small α indicates a loose criterion for fairness, which is not desired), penalty and the time cost. Other datasets are processed similarly.

Fixing α to 0.4, we varied β from 0.1 to 0.5 on dataset *XING*. As shown in Figure 3(d), when β is small (e.g., $\beta < \alpha = 0.4$), the top-k sets returned by the approximation algorithm are of poor fairness, since the α-fairness cannot be granted. The fairness is improved when $\beta \geq 0.4$, where the observed α-fairness of the approximation algorithm are close to the optimal α-fairness found by exact algorithms (guaranteed by Lemma 4.1). However, when β is set to the largest (i.e., 0.5), the β-relaxed-fair criterion is strict where few top-k sets satisfy. Then, the approximation algorithm need even more time (Figure 3(e)) since it degrades to an exact algorithms with extra maintenance cost. Thus, we set $\beta = \alpha$ in the rest experiments, which gives fair enough results and meanwhile keeps the algorithms fast.

Effect of K. To study the effect of different levels of global recoding, we varied K from 2 to 15 on dataset *XING*. When K is smaller,

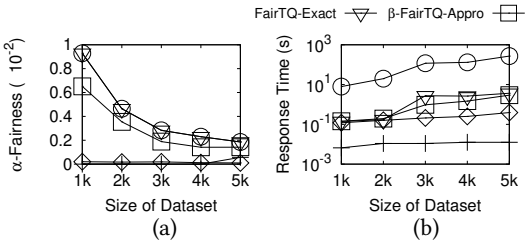


Figure 5: Effect of $|D|$ on *COMPAS*

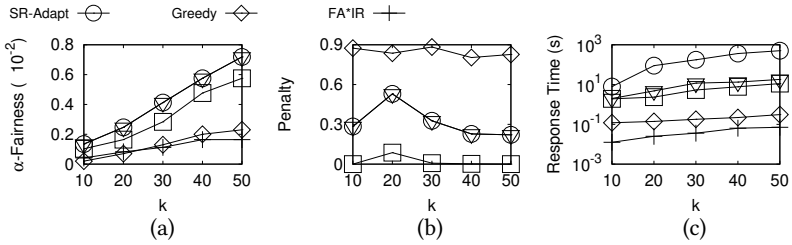


Figure 6: Effect of k on *COMPAS*

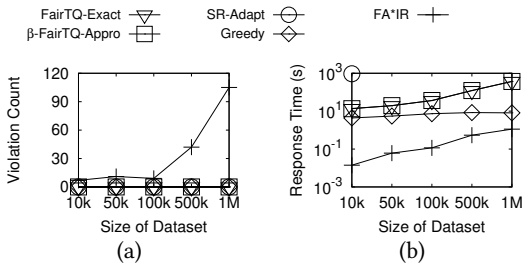


Figure 7: Effect of $|D|$ on *DOT*

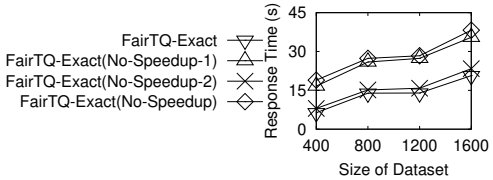


Figure 8: Ablation Studies of Speedup on *ALLSTAR*

$\mathcal{U}$ contains less fine-grained global domains for identifying good clusters and thus, the α-fairness is smaller (Figure 4(a)). As expected, it takes more time to process more global domains for all algorithms. We set K to be 10 on all datasets for its good performance.

Effect of $|D|$. We studied the effect of dataset size $|D|$ by randomly selecting a subset of 1k to 5k tuples (resp. 10k to 1M tuples) from dataset *COMPAS* (resp. dataset *DOT*) in Figure 5 (resp. Figure 7). As shown in Figures 5(b) and 7(b), when $|D|$ increases, the response time for all algorithms increases, as expected. Nevertheless, **FairTQ-Exact** outperforms the exact baseline **SR-Adapt** by 1-2 orders of magnitude, while it generates the same output as **SR-Adapt**. Compared with **FairTQ-Exact**, **β-FairTQ-Appro** can have up to 20%-50% improvement of response time in many cases, and it can still find the sufficiently fair top-k set (Figure 5(a)). Although baseline **Greedy** has shorter response time than our algorithms, the top-k sets it returns are poor in fairness (e.g., nearly 0 fairness in Figure 5(a)). Baseline **FA*IR** is also fast, but it incurs a number of ranking violations in Figure 7(a). In other words, **FA*IR** cannot return a valid utility function, and its response time does not include the time for finding such utility function. Finally, we observe that the α-fairness of all algorithms decreases when $|D|$ increases (with a fixed k). This is because α-fairness is computed based on the proportion of desired tuples in a top-k set over $|D|$ by definition. This drop also conforms with our upper bound of α-fairness (i.e., $\frac{k}{|D|}$).

Effect of k. When k is varied from 10 to 50 on dataset *COMPAS*, both the response time and the α-fairness of all algorithms increase, as shown in Figures 6(a) and (c), since more top-k sets have to be enumerated and the fairness of each group can be better granted with a large k. Our algorithms consistently beat the baselines in superior time efficiency and fair output. Moreover, it can be observed that the approximate results (returned by **β-FairTQ-Appro**) have smaller modification penalty than the exact result (returned by **FairTQ-Exact**) in Figures 6(b). Since they only need to find approximate solutions, they have more feasible regions to search for a closer utility function (to the input utility function). Baseline **Greedy** has undesirably large penalty and poor fairness although its response time is small. Similarly, the fastest baseline **FA*IR** cannot return a utility function due to ranking violation.

Ablation Studies of Speedup Techniques. We conducted ablation studies of the speedup techniques (proposed in Section 3.5) for our exact algorithm **FairTQ-Exact**. We denote **FairTQ-Exact** without the first speedup technique (i.e., pruning by modification penalty) and without the second speedup technique (i.e., pruning by α-fairness) by **FairTQ-Exact(No-Speedup-1)** and **FairTQ-Exact(No-Speedup-2)**, respectively. We include another baseline of **FairTQ-Exact** without using any speedup technique, denoted by **FairTQ-Exact(No-Speedup)**. As shown in Figure 8, when the first speedup technique is disabled, the efficiency degrades significant, which verifies the usefulness of pruning by modification penalty. **FairTQ-Exact(No-Speedup-2)** is also slower than **FairTQ-Exact**, but the gap is not large, which indicates that the effectiveness of the second speedup technique is less obvious than the first one. This is because the second speedup technique only works on equal-to regions, while the first one prunes any type of regions as long as the maximum α-fairness has been reached and the region being processed cannot contain a better utility function. Note that **FairTQ-Exact** (with both speedup techniques) takes only around half time to response compared to **FairTQ-Exact(No-Speedup)**.

Case Studies of Fairness. We also conducted case studies on the fairness results of our algorithm compared with baselines on real datasets. One typical result on dataset *COMPAS* (where African Americans account for 34% among all races) under default settings is shown as follows. Baseline **Greedy** returns a top-k set (people with higher chance of recidivation) where 80% of them are African Americans, indicating its incapability of achieving fairness. Baseline **FA*IR** returns a top-k set where 90% of them are non-African Americans, since **FA*IR** controls the fairness by adding more non-African Americans into the top-k set. This, however, creates obvious discrimination to other races while the African Americans are protected. Our algorithms output a top-k set containing 40% African Americans, a fair result for both the protected and the unprotected groups.

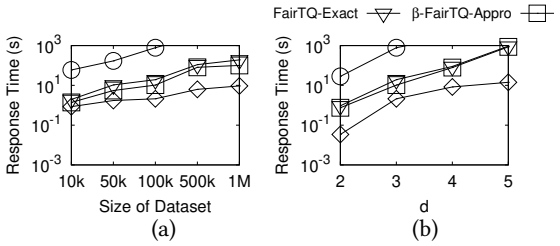


Figure 9: Effect of $|D|$ and d on *Independent*

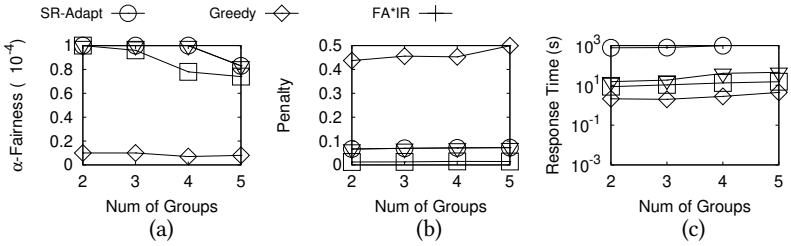


Figure 10: Effect of $|\mathcal{P}|$ on *Anti-correlated*

6.2 Results on Synthetic Datasets

Effect of $|D|$ and d. We also tested the effect of $|D|$ for the response time in synthetic dataset *Independent*, varying $|D|$ from 10k to 1M. As shown in Figure 9(a), the trends and the superior time efficiency of our algorithms are consistent as before. In particular, **FairTQ-Exact** is two orders of magnitude faster than **SR-Adapt**. To process 100k tuples, **SR-Adapt** needs around 1000s, but **FairTQ-Exact** only needs around 20s. On the largest dataset of 1M tuples, **FairTQ-Exact** (resp. β-**FairTQ-Appro**) returns exact (resp. approximate) results reasonably in 190s (resp. 100s), while **SR-Adapt** cannot process more than 100k tuples in reasonable time (i.e., $< 1000s$).

We evaluated the effect of d in Figure 9(b). Due to the nature of computational geometry, the complexity of our problem rises with larger d, and the time took by all algorithms grows obviously. Nonetheless, **FairTQ-Exact** can run in more practical time than **SR-Adapt** (which fails to handle more than 3 dimensions in 1000s).

Effect of $|\mathcal{P}|$. We varied the number of groups (i.e., $|\mathcal{P}|$) from 2 to 5 in Figure 10 on dataset *Anti-correlated*. Based on our fairness model, given a top-k set T on the synthetic dataset, if all group contributes more equally in T, T tends to be fairer. This condition is harder to satisfy with more groups. Thus, when $|\mathcal{P}|$ increases, the α-fairness tends to decrease (Figure 10(a)), while the penalty increases slightly (Figure 10(b)), since fewer top-k sets are the fairest top-k sets, leading to fewer regions on which we can search for the best utility functions. This harder condition also makes us more difficult to find a top-k set with the optimal fairness, so that our speedup techniques can be applied (Lemma 3.3). Thus, the response time slightly increases (Figure 10(c)). Still, our algorithms (esp. β-**FairTQ-Appro**) is consistently faster than **SR-Adapt** and scales well w.r.t. $|\mathcal{P}|$.

6.3 User Study

To demonstrate the effectiveness of **FairTQ-Exact** in real scenarios, we conducted a user study on dataset *GermanCredit*. We used three scoring attributes (*credit history, credit amount* and *age*) to evaluate the credit of applicants applying for the loan and used *Gender* as the protected attribute. We conducted a survey on the platform "Amazon Mechanical Turk" by inviting participants to indicate which set is fairer among the top-10 sets (10 applicants who are most likely to get a loan) returned by **FairTQ-Exact**, **SATREGIONS** (the original algorithm of our adapted baseline **SR-Adapt** with its original fairness model) and **FA*IR**. Participants are required to compare the fairness based on gender ratio, relative ranking between males and females, or their own perspective about fairness. We asked participants to select the fairer one among every two algorithms.

Table 4: Survey Results

Alg1	Alg2	Alg1 is fairer	Alg2 is fairer
FairTQ-Exact	**FA*IR**	86%	9%
FA*IR	**SATREGIONS**	31%	59%
FairTQ-Exact	**SATREGIONS**	72%	21%

We received 298 responses in total and obtained 98 valid responses after *cross-checking* (see Section A for details). The results of survey are listed in Table 4. Note that the sum of each row is not 100% because some participants can not judge which algorithm is fairer. As for the fairest algorithm, about 66% of responses select **FairTQ-Exact**, 13% of responses select **SATREGIONS** and 11% select **FA*IR**, which shows that **FairTQ-Exact** achieves the fairest results among all algorithms.

6.4 Summary

The experiments on real and synthetic datasets demonstrated the superiority of our algorithms. Specifically, our exact algorithm (which guarantees to return the fairest top-k set) achieves one to two orders of magnitude shorter response time compared with the exact baseline. When the dataset size scales to 1M, it has reasonable response time (i.e., around 190s), while the response time of the exact baseline is unacceptable (e.g., > 1000s). Our approximation algorithms can find the approximately fairest top-k sets with superior time efficiency and even smaller penalty than exact algorithms. With our user study, we also demonstrate the effectiveness of our fairness model to give fair top-k sets compared with other fairness models.

7 CONCLUSION

In this paper, we propose a fairness model called α-fair, which flexibly quantifies the fairness of top-k results, and design an efficient algorithm **FairTQ-Exact** to help users find a fair utility function. Speedup techniques are used to improve the efficiency. We also propose an approximation algorithm to help users find good enough solutions with theoretical bounds efficiently. We conducted experiments on real and synthetic datasets to demonstrate the effectiveness and efficiency of our algorithms. As for future research, we consider expanding our fairness problem to different scenarios to help users solve more complicated problems about fairness. For example, we can combine Ashudeep's work [27] by adding the fairness about relative ranking between tuples into our fairness model. It is also interesting to consider our problem for more than one protected attribute leveraging the idea of [7] (which handles multiple protected attributes for a different problem of preference aggregation).

REFERENCES

[1] 2023. Data USA: Massachusetts Institute of Technology. Retrieved April, 2023 from https://datausa.io/profile/university/massachusetts-institute-of-technology#admissions

[2] 2023. Find the right job for you. Retrieved April, 2023 from https://www.xing.com/

[3] 2023. NBA Players stats since 1950. Retrieved April, 2023 from https://www.kaggle.com/datasets/drgilermo/nba-players-stats

[4] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. 2023. *Machine bias: Risk assessments in criminal sentencing.* ProPublica. Retrieved April, 2023 from https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing

[5] Abolfazl Asudeh, H. V. Jagadish, Julia Stoyanovich, and Gautam Das. 2019. Designing Fair Ranking Schemes. In *Proceedings of the 2019 International Conference on Management of Data*. Association for Computing Machinery, 1259–1276.

[6] Sara N. Bleich, Mary G. Findling, Logan S. Casey, Robert J. Blendon, John M. Benson, Gillian K. SteelFisher, Justin M. Sayde, and Carolyn Miller. 2019. Discrimination in the United States: Experiences of black Americans. *Health Services Research* 54, S2 (2019), 1399–1408.

[7] Kathleen Cachel, Elke Rundensteiner, and Lane Harrison. 2022. Mani-rank: Multiple attribute and intersectional group fairness for consensus ranking. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 1124–1137.

[8] Flavio Calmon, Dennis Wei, Bhanukiran Vinzamuri, Karthikeyan Natesan Ramamurthy, and Kush R Varshney. 2017. Optimized Pre-Processing for Discrimination Prevention. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc., 3992–4001.

[9] L. Elisa Celis, Damian Straszak, and Nisheeth K. Vishnoi. 2018. Ranking with Fairness Constraints. In *45th International Colloquium on Automata, Languages, and Programming*, Vol. 107. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 28:1–28:15.

[10] Evgenii Chzhen, Christophe Denis, Mohamed Hebiri, Luca Oneto, and Massimiliano Pontil. 2019. Leveraging Labeled and Unlabeled Data for Consistent Fair Binary Classification. (2019). arXiv:1906.05082

[11] Andrew Cotter, Maya R. Gupta, Heinrich Jiang, Nathan Srebro, Karthik Sridharan, Serena Wang, Blake E. Woodworth, and Seungil You. 2018. Training Well-Generalizing Classifiers for Fairness Metrics and Other Data-Dependent Constraints. (2018). arXiv:1807.00028

[12] Rachel Courtland. 2018. Bias detectives: the researchers striving to make algorithms fair. *Nature* 558 (2018), 357–360.

[13] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. 2012. Fairness through Awareness. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. Association for Computing Machinery, 214–226.

[14] Marlene Goncalves and María-Esther Vidal. 2005. Top-k Skyline: A Unified Approach. In *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*. Springer Berlin Heidelberg, 790–799.

[15] Moritz Hardt, Eric Price, and Nathan Srebro. 2016. Equality of Opportunity in Supervised Learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*. Curran Associates Inc., 3323–3331.

[16] Md Mouinul Islam, Dong Wei, Baruch Schieber, and Senjuti Basu Roy. 2022. Satisfying complex top-k fairness constraints by preference substitutions. *Proceedings of the VLDB Endowment* 16, 2 (2022), 317–329.

[17] Faisal Kamiran and Toon Calders. 2011. Data preprocessing techniques for classification without discrimination. In *Knowledge and Information Systems*, Vol. 33. 1–33.

[18] Faisal Kamiran, Toon Calders, and Mykola Pechenizkiy. 2010. Discrimination Aware Decision Tree Learning. In *Proceedings of the 2010 IEEE International Conference on Data Mining*. IEEE Computer Society, 869–874.

[19] Chen Karako and Putra Manggala. 2018. Using Image Fairness Representations in Diversity-Based Re-Ranking for Recommendations. In *Adjunct Publication of the 26th Conference on User Modeling, Adaptation and Personalization*. Association for Computing Machinery, 23–28.

[20] Preethi Lahoti, Krishna P. Gummadi, and Gerhard Weikum. 2019. Operationalizing Individual Fairness with Pairwise Fair Representations. *Proceedings of the VLDB Endowment* 13, 4 (2019), 506–518.

[21] Jongwuk Lee, Gae-won You, and Seung-won Hwang. 2009. Personalized top-k skyline queries in high-dimensional space. *Information Systems* 34 (2009), 45–61.

[22] Renato D.C. Monteiro and Ilan Adler. 1989. Interior Path Following Primal-Dual Algorithms. Part II: Convex Quadratic Programming. *Math. Program.* 44, 1 (1989), 43–66.

[23] Kyriakos Mouratidis and Bo Tang. 2018. Exact Processing of Uncertain Top-k Queries in Multi-Criteria Settings. *Proceedings of the VLDB Endowment* 11, 8 (April 2018), 866–879.

[24] United States Department of Transportation. 2023. *Bureau of transportation statistics.* Retrieved April, 2023 from https://www.transtats.bts.gov/Tables.asp?QO_VQ=EFD&QO_anzr=Nv4yv0r%20b0-gvzr%20cr4s14zn0pr%20Qn6n0

[25] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005. Progressive Skyline Computation in Database Systems. *ACM Trans. Database Syst.* 30, 1 (March 2005), 41–82.

[26] Babak Salimi, Luke Rodriguez, Bill Howe, and Dan Suciu. 2019. Interventional Fairness: Causal Database Repair for Algorithmic Fairness. In *Proceedings of the 2019 International Conference on Management of Data*. Association for Computing Machinery, 793–810.

[27] Ashudeep Singh and Thorsten Joachims. 2018. Fairness of Exposure in Rankings. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2219–2228.

[28] Ashudeep Singh and Thorsten Joachims. 2019. Policy Learning for Fairness in Ranking. *Advances in Neural Information Processing Systems* 32 (2019), 5427–5437.

[29] Manolis Terrovitis, Nikos Mamoulis, and Panos Kalnis. 2011. Local and global recoding methods for anonymizing set-valued data. *The VLDB Journal* 20, 1 (2011), 83–106.

[30] Weicheng Wang, Raymond Chi-Wing Wong, and Min Xie. 2021. Interactive Search for One of the Top-k. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, 13.

[31] Raymond Chi-Wing Wong, Jiuyong Li, Ada Wai-Chee Fu, and Ke Wang. 2006. (α, k)-Anonymity: An Enhanced k-Anonymity Model for Privacy Preserving Data Publishing. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 754–759.

[32] Ke Yang and Julia Stoyanovich. 2017. Measuring Fairness in Ranked Outputs. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*. Association for Computing Machinery.

[33] Meike Zehlike, Francesco Bonchi, Carlos Castillo, Sara Hajian, Mohamed Megahed, and Ricardo Baeza-Yates. 2017. FA*IR: A Fair Top-k Ranking Algorithm. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. Association for Computing Machinery, 1569–1578.

[34] Jiping Zheng, Yuan Ma, Wei Ma, Yanhao Wang, and Xiaoyang Wang. 2022. Happiness maximizing sets under group fairness constraints. *Proceedings of the VLDB Endowment* 16, 2 (2022), 291–303.

A CROSS-CHECKING IN USER STUDY

To avoid the influence of participants' random choices on the final results, we used the method of cross-checking by creating two tables. In the first table (containing the original dataset *GermanCredit*), each algorithm selects the 10 applicants who are most likely to get a loan. Each tuple in the second table has opposite attribute values of the tuple in the first table. Then, each algorithm selects the 10 applicants from the second table who are most likely to fail their applications. Since the 2 tables are related, only when the participant's evaluations of the 3 algorithms are the same for the two tables, the response of this participant is valid.

B PSEUDO-CODE OF OUR ALGORITHMS

B.1 Algorithm FairTQ-Exact

We show the pseudo-code of our exact algorithm **FairTQ-Exact** in Algorithm 1 which is proposed in Section 3.

Line 1 is the filtering step, which computes the set D' of k-skybands of D. The refinement step (Line 3) is recursively conducted via procedure Partition (Line 6-37, see below), whose initial call considers the region $\rho = \Omega$ and the candidate set $C_\rho = D'$ with rank quota $r = k$. Partition also takes a top-$(k-r)$ set T_ρ as input, which stores $(k-r)$ top-ranked tuples w.r.t. ρ that have been determined in previous partitioning; they will be combined with the r tuples found in the following partitioning to form the final top-k set. In the initial call of Partition, T_ρ is empty since no tuples are identified yet. Finally, when the desired utility function w^* is identified, it is returned in Line 4.

We maintain the following global variables in **FairTQ-Exact** (Line 2): (a) a value F^* (initialized to $-\infty$), recording the best-known α-fairness; (b) the best utility function w^* (initialized to w_0) obtained so far; and (c) a value m^* (initialized to ∞) to record

Algorithm 1 FairTQ-Exact

Input: Dataset D, Universal Domain Set $\mathcal{U}$, Initial Utility Function w_0, Utility Space Ω, Real Number α, Positive Integer k
Output: The optimal utility function w^*

1: $D' \leftarrow$ the set of all k-skybands of D ▷ filtering step
2: $F^* \leftarrow -\infty$, $m^* \leftarrow \infty$, $w^* \leftarrow w_0$ ▷ global variables
3: Partition(Ω, k, D', $\emptyset$) ▷ refinement step
4: **return** w^*
5:
6: **procedure** Partition(Region ρ, Rank Quota r, Candidate Set C_ρ, The Top-$(k-r)$ Set T_ρ w.r.t. ρ)
7: $t_A \leftarrow$ an anchor chosen from C_ρ
8: $\overline{C} \leftarrow$ the set of tuples in C_ρ that dominate t_A
9: $\underline{C} \leftarrow$ the set of tuples in C_ρ that are dominated by t_A
10: $C^{omp} \leftarrow C_\rho \setminus (\overline{C} \cup \underline{C} \cup \{t_A\})$ ▷ *competitors* of the anchor
11: $\mathcal{H} \leftarrow \emptyset$
12: **for** $t_C \in C^{omp}$ **do**
13: $\mathcal{H} \leftarrow \mathcal{H} \cup \{h_{t_C,t_A}\}$
14: $\mathcal{A}(\mathcal{H}, \rho) \leftarrow$ an arrangement of $\mathcal{H}$ bounded by ρ
15: **for** each sub-region ρ' in $\mathcal{A}(\mathcal{H}, \rho)$ **do**
16: **if** $m^* \neq \infty$ and $\forall w \in \rho', m(w, w_0) > m^*$ **then**
17: **continue** ▷ prune ρ' by Lemma 3.3
18: $\overline{C}^{omp} \leftarrow$ the set of *dominating competitors* of t_A in C^{omp}
19: $r' \leftarrow |\overline{C}^{omp}| + |\overline{C}| + 1$ ▷ rank of anchor t_A w.r.t. ρ'
20: $T_{r'} \leftarrow \overline{C}^{omp} \cup \overline{C} \cup \{t_A\}$ ▷ $T_{r'} =$ **top**-$r'_{\rho'}(C_\rho)$
21: **if** $r' > r$ **then** ▷ *greater-than* region
22: Partition(ρ', r, $\overline{C} \cup \overline{C}^{omp}$, T_ρ)
23: **else if** $r' < r$ **then** ▷ *less-than* region
24: $T_{\rho'} \leftarrow T_\rho \cup T_{r'}$
25: **if** $\min_{j \in \mathcal{P}} \frac{|T_{\rho'}[j]|+r-r'}{|D[j]|} < F^*$ **then**
26: **continue** ▷ prune ρ' by Lemma 3.4
27: Partition(ρ', $r - r'$, $C_\rho \setminus T_{r'}$, $T_{\rho'}$)
28: **else** ▷ *equal-to* region
29: $T \leftarrow T_\rho \cup T_{r'}$, $F \leftarrow \alpha\text{-}FN_{\mathcal{U}}(T)$ ▷ Section 3.3
30: **if** $F \geq F^*$ **then**
31: $w_{\rho'} \leftarrow$ best utility function in ρ' ▷ Section 3.4
32: **if** $F = F^*$ and $m(w_{\rho'}, w_0) < m(w^*, w_0)$ **then**
33: $w^* \leftarrow w_{\rho'}$
34: **else if** $F > F^*$ **then**
35: $F^* \leftarrow F$, $w^* \leftarrow w_{\rho'}$
36: **if** $F = \frac{k}{|D|}$ **then**
37: $m^* \leftarrow m(w^*, w_0)$

the best-known penalty. Note that unless we have found a top-k set whose α-fairness is the optimal (i.e., $\frac{k}{|D|}$, Lemma 3.3), we will not record m^* (Line 36-37) and Lemma 3.3 not be triggered.

Procedure Partition. In Partition, we first pick an anchor t_A from the candidate set C_ρ (Line 7), based on which we find the competitors and build the corresponding hyperplanes in $\mathcal{H}$ (Line 8-14). After obtaining the arrangement $\mathcal{A}(\mathcal{H}, \rho)$, we process each cell, i.e., a sub-region ρ', in $\mathcal{A}(\mathcal{H}, \rho)$ as follows (Line 15-37).

Firstly, we check whether ρ' can be pruned by Lemma 3.3, i.e., if the optimal α-fairness $\frac{k}{|D|}$ is already found (i.e., $m^* \neq \infty$) and utility functions in ρ' cannot give smaller modification penalty than the best m^*, we terminate the processing of ρ' immediately (Line 16-17). Otherwise, we find the set $\overline{C}^{omp}$ of dominating competitors of t_A (Line 18), based on which we determine the rank of anchor t_A w.r.t. ρ', denoted by r', and the top-r' set w.r.t. ρ' in C_ρ, denoted by $T_{r'}$ (= **top**-$r'_{\rho'}(C_\rho)$) (Line 19-20). According to the type of region ρ', we do the following: (a) If ρ' is a greater-than region (Line 21-22), we recursively partition it by restricting the candidate set to be $\overline{C} \cup \overline{C}^{omp}$, where $\overline{C}$ is the set of tuples in C_ρ dominating t_A. (b) If ρ' is a less-than region (Line 23-27), $T_{r'}$ contains r' more top-ranked tuples w.r.t. ρ'. Recall that the first $k - r$ top-ranked tuples w.r.t. ρ' are already determined (in the input T_ρ) and thus in total, we have identified a set of $k - r + r'$ top ranked tuples w.r.t. ρ', denoted by $T_{\rho'}$ (Line 24). We only need to compute the remaining $r - r'$ tuples to return the final top-k set T w.r.t. ρ'. Before performing the exact computation, we first obtain an upper bound of the α-fairness of T (Line 25). If the upper bound is not smaller than the best F^* observed so far, we recursively partition ρ' by setting the rank quota to be $r - r'$ for the remaining tuples in C_ρ (i.e., $C_\rho \setminus T_{r'}$) (Line 27). Otherwise, we terminate its further partitioning by Lemma 3.4 (Line 26). (c) If ρ' is an equal-to region (Line 28-37), the complete top-k set T w.r.t. ρ' (i.e., $T =$ **top**-$k_{\rho'}$) is determined. We then update the global variables accordingly and terminate the processing of ρ'.

B.2 Algorithm β-FairTQ-Appro

We show the pseudo-code of our approximation algorithm β-**FairTQ-Appro** in Algorithm 2 which is proposed in Section 4.

Similar to **FairTQ-Exact**, we maintain the variables for the best α-fairness (among all the non-β-relaxed-fair top-k sets) (denoting F^*_{non}), the best utility function (among all the non-β-relaxed-fair top-k sets) (denoting w^*_{non}) and the best modification penalty when the upper bound of α-fairness (i.e., $\frac{k}{|D|}$) has been reached. Additionally, we maintain a set Γ to store the regions for all the β-relaxed-fair top-k sets. In the procedure Partition, we follow most of the steps in **FairTQ-Exact**. Specifically, we execute Line 7-14 of Algorithm 1 to form an arrangement $\mathcal{A}(\mathcal{H}, \rho)$ from a selected anchor t_A from C_ρ For each sub-region ρ' in the arrangement $\mathcal{A}(\mathcal{H}, \rho)$, we execute Line 16-20 of Algorithm 1 to obtain the rank r' of t_A w.r.t. region ρ' and the top-r' set $T_{r'}$. Based on the relationship of r' and the current rank quota r, region ρ' is similarly classified into 3 categories, namely greater-than region, less-than region and equal-to region (as defined in Section 3.2).

- *Greater-than region (i.e., $k' > k$):* Following **FairTQ-Exact**, we execute Line 22 of Algorithm 1 for recursive partitioning of greater-than region ρ'.
- *Less-than region (i.e., $k' < k$):* We execute Line 24-27 of Algorithm 1 for recursive partitioning of less-than region ρ'. Note that we replace F^* with F^*_{non}, since we use F^*_{non} for the currently best α-fairness.
- *Equal-to region (i.e., $k' = k$):* We obtain the determined the top-k set w.r.t. a utility function in ρ' (i.e., $T = T_\rho \cup T_{r'}$). We first check whether T is β-relaxed-fair. If yes, we directly add ρ' into set Γ. Otherwise, we still compute the α-fairness of T, assigned to F. Then, we execute Line 30-37 of Algorithm 1 (replacing F^* with F^*_{non} and replacing w^* with w^*_{non}) to update F^*_{non}, w^*_{non} and m^*.

Algorithm 2 β-FairTQ-Appro

Input: Dataset D, Universal Domain Set $\mathcal{U}$, Initial Utility Function w_0, Utility Space Ω, Real Number α and β, Positive Integer k
Output: A β-approximated utility function w

1: $D' \leftarrow$ the set of all k-skybands of D ▷ filtering step
2: $F^*_{non} \leftarrow -\infty$, $m^* \leftarrow \infty$, $w^*_{non} \leftarrow w_0$, $\Gamma \leftarrow \emptyset$ ▷ global variables
3: Partition(Ω, k, D', $\emptyset$) ▷ refinement step
4: **if** $\Gamma = \emptyset$ **then**
5: **return** w^*_{non}
6: **else**
7: $w^*_{re} \leftarrow$ the utility function in Γ with the minimum penalty
8: **if** $F^*_{non} = -\infty$ **then**
9: **return** w^*_{re}
10: $T^*_{re} \leftarrow$ **top**-$k_{w^*_{re}}(D)$
11: **if** α-$FN_{\mathcal{U}}(T^*_{re}) > F^*_{non}$ **then**
12: **return** w^*_{re}
13: **else**
14: **return** the one from $\{w^*_{re}, w^*_{non}\}$ with smaller penalty
15:
16: **procedure** Partition(Region ρ, Rank Quota r, Candidate Set C_ρ, The Top-$(k-r)$ Set T_ρ w.r.t. ρ)
17: execute Line 7-14 of Algorithm 1 to form an arrangement $\mathcal{A}(\mathcal{H}, \rho)$ from a selected anchor t_A from C_ρ
18: **for** each sub-region ρ' in $\mathcal{A}(\mathcal{H}, \rho)$ **do**
19: execute Line 16-20 of Algorithm 1 to obtain the rank r' of t_A w.r.t. ρ' and the top-r' set $T_{r'}$
20: **if** $r' > r$ **then** ▷ *greater-than* region
21: execute Line 22 of Algorithm 1 for recursive partitioning of greater-than region ρ'
22: **else if** $r' < r$ **then** ▷ *less-than* region
23: execute Line 24-27 of Algorithm 1 for recursive partitioning of less-than region ρ' (replacing F^* with F^*_{non})
24: **else** ▷ *equal-to* region
25: $T \leftarrow T_\rho \cup T_{r'}$ ▷ Top-k set wrt a utility function in ρ'
26: **if** T is β-relaxed-fair **then**
27: $\Gamma \leftarrow \Gamma \cup \rho'$
28: **else**
29: $F \leftarrow \alpha$-$FN_{\mathcal{U}}(T)$
30: execute Line 30-37 of Algorithm 1 (replacing F^* with F^*_{non} and replacing w^* with w^*_{non}) to update F^*_{non}, w^*_{non} and m^*

After all sub-regions have been processed, we determine the final returned utility function as follows, which implements the major idea introduced in Section 4. We first check whether Γ is empty (Line 4). If yes (i.e., there exists no β-relaxed-fair top-k set), we directly return w^*_{non} leading to the exactly fairest top-k set (Line 5). Otherwise, we find the best utility function in Γ with the minimum penalty, assigned to w^*_{re} (Line 7). At this point, we also check whether w^*_{non} is defined by checking whether F^*_{non} is still the initial value (i.e., $-\infty$) (Line 8). If $F^*_{non} = -\infty$, we directly return w^*_{re} (Line 9). Otherwise, we compare which is fairer between the top-k set w.r.t. w^*_{re} and the top-k set w.r.t. w^*_{non} (Line 11). If the top-k set w.r.t. w^*_{re} is fairer, we return w^*_{re} (Line 12). If the top-k set w.r.t. w^*_{non} is fairer, we return the one with smaller penalty between w^*_{re} and w^*_{non} (Line 14).

C PROOF

Proof of Lemma 3.1. For two utility functions w and w' in the same cell of arrangement $\mathcal{A}(\mathcal{H}, \Omega)$, it is clear that for each hyperplane h in $\mathcal{H}$, w and w' lie in the same halfspace of h. Consider a hyperplane between any two tuple t and t', clearly, the utility relationship is the same for w and w'. Thus, w and w' give the exactly same list of tuples in D sorted by their utility, which implies **top**-$k_w(D)$ = **top**-$k_{w'}(D)$. □

Proof of Lemma 3.2. According to Equation 1,

$$\alpha\text{-}SP_{\mathcal{G}}(T, j) = \frac{\sum_{C \in CL_{\mathcal{G}}(T) \text{ and } C \text{ is } \alpha\text{-fair}} |C[j]|}{|D[j]|}.$$

Clearly, $\sum_{C \in CL_{\mathcal{G}}(T) \text{ and } C \text{ is } \alpha\text{-fair}} |C[j]| \leq \sum_{C \in CL_{\mathcal{G}}(T)} |C[j]| = |T[j]|$. Thus, according to Definition 2.2,

$$\alpha\text{-}MinSP_{\mathcal{G}}(T) = \min_{j \in \mathcal{P}} \alpha\text{-}SP_{\mathcal{G}}(T, j) \leq \min_{j \in \mathcal{P}} \frac{|T[j]|}{|C[j]|}$$

Next, we show that α-$MinSP_{\mathcal{G}}(T) = \min_{j \in \mathcal{P}} \frac{|T[j]|}{|D[j]|}$ if and only if all clusters in $CL_{\mathcal{G}}(T)$ are α-fair. If all clusters in $CL_{\mathcal{G}}(T)$ are α-fair, α-$MinSP_{\mathcal{G}}(T) = \min_{j \in \mathcal{P}} \frac{|T[j]|}{|D[j]|}$ clearly holds since

$$\alpha\text{-}SP_{\mathcal{G}}(T, j) = \frac{\sum_{C \in CL_{\mathcal{G}}(T) \text{ and } C \text{ is } \alpha\text{-fair}} |C[j]|}{|D[j]|} = \frac{|T[j]|}{|D[j]|}.$$

Let j_{min} denote the group in $|\mathcal{P}|$ such that $\frac{|T[j]|}{|D[j]|}$ is minimized. If α-$MinSP_{\mathcal{G}}(T) = \min_{j \in \mathcal{P}} \frac{|T[j]|}{|D[j]|}$, then α-$MinSP_{\mathcal{G}}(T) = \frac{|T[j_{min}]|}{|D[j_{min}]|}$, Assume that there exists a cluster in $CL_{\mathcal{G}}(T)$ that is not α-fair. Then, for group j_{min},

$$\alpha\text{-}SP_{\mathcal{G}}(T, j_{min}) = \frac{\sum_{C \in CL_{\mathcal{G}}(T) \text{ and } C \text{ is } \alpha\text{-fair}} |C[j_{min}]|}{|D[j_{min}]|} < \frac{|T[j_{min}]|}{|D[j_{min}]|}.$$

The above indicates that there exists a group in $|\mathcal{P}|$ (e.g., j_{min}) such that the α-satisfied proportion of T w.r.t. this group under $\mathcal{G}$ is smaller than $\frac{|T[j_{min}]|}{|D[j_{min}]|}$. Then, α-$MinSP_{\mathcal{G}}(T)$ (which is the minimized α-satisfied proportion) is also smaller than $\frac{|T[j_{min}]|}{|D[j_{min}]|}$, which contradicts our assumption. Therefore, if α-$MinSP_{\mathcal{G}}(T) = \min_{j \in \mathcal{P}} \frac{|T[j]|}{|D[j]|}$, then all clusters in $CL_{\mathcal{G}}(T)$ are α-fair. □

Proof of Lemma 3.3. Assume that there exists a top-k set T in D such that α-$FN_{\mathcal{U}}(T) > \frac{k}{|D|}$. Then, by definition, there exists a global domain $\mathcal{G} \in \mathcal{U}$, such that α-$MinSP_{\mathcal{G}}(T) > \frac{k}{|D|}$. According to Lemma 3.2, α-$MinSP_{\mathcal{G}}(T) \leq \min_{j \in \mathcal{P}} \frac{|T[j]|}{|D[j]|}$. Then, we have $\min_{j \in \mathcal{P}} \frac{|T[j]|}{|D[j]|} > \frac{k}{|D|}$. This indicates that, $\forall j \in \mathcal{P}$, $\frac{|T[j]|}{|D[j]|} > \frac{k}{|D|}$. Then, $\forall j \in \mathcal{P}$, $\frac{|T[j]|}{k} > \frac{|D[j]|}{|D|}$. Summing up all groups, we have $\sum_{j \in \mathcal{P}} \frac{|T[j]|}{k} > \sum_{j \in \mathcal{P}} \frac{|D[j]|}{|D|}$, which leads to $1 > 1$. Thus, we prove that for any top-k set T in D, α-$FN_{\mathcal{U}}(T) \leq \frac{k}{|D|}$. □

Proof of Lemma 3.4. According to Lemma 3.2 and the definition of α-$FN_{\mathcal{U}}(T)$, there exists $\mathcal{G} \in \mathcal{U}$ such that α-$FN_{\mathcal{U}}(T) = \alpha$-$MinSP_{\mathcal{G}}(T) \leq \min_{j \in \mathcal{P}} \frac{|T[j]|}{|D[j]|}$. Let j_{min} denote the group in $\mathcal{P}$ such that $\frac{|T[j]|}{|D[j]|}$ is minimized. Let $\overline{T}$ denote $T \setminus T'$, and let $\overline{T}[j_{min}]$

denote the set of tuples in $\overline{T}$ that belong to group j_{min}. Since $T' \subset T$, $|\overline{T}[j_{min}]| \leq |\overline{T}| = |T \setminus T'| = k - k'$. Thus,

$$\begin{aligned}\alpha\text{-}FN_{\mathcal{U}}(T) &\leq \frac{|T[j_{min}]|}{|D[j_{min}]|} \\ &= \frac{|T'[j_{min}]| + |\overline{T}[j_{min}]|}{|D[j_{min}]|} \\ &\leq \frac{|T'[j_{min}]| + k - k'}{|D[j_{min}]|} \\ &\leq \min_{j \in \mathcal{P}} \frac{|T'[j]| + k - k'}{|D[j]|}\end{aligned}$$

□

Proof of Theorem 3.4. First, following [23], if we skip the two pruning strategies in Line 16-17 (i.e., pruning a sub-region ρ' by Lemma 3.3) and Line 25-26 (i.e., pruning a sub-region ρ' by Lemma 3.4), respectively, then Algorithm 1 will enumerate all top-k sets in D by processing all the *equal-to* regions (Line 29).

Then, for each top-k set T, we compute its α-fairness F. If F is smaller than the currently maximum α-fairness F^*, then we can safely discard this top-k set (i.e., discard this equal-to region). Otherwise (i.e., if F is at least F^*), we first compute the best utility function $w_{\rho'}$ in this region that has the minimum penalty to w_0 (Line 31). If $F = F^*$, then we update the currently best utility function w^* only when the best utility function $w_{\rho'}$ in this region is closer to w_0 than the currently best utility function w^*. This ensures that $m(w^*, w_0)$ is minimized under the maximized α-fairness F^*. Otherwise (i.e., $F > F^*$), we find a larger α-fairness, and thus F^* will be updated to this larger α-fairness. Accordingly, the currently best utility function w^* (which only works for the previous α-fairness) is cleared and reset to the best utility function $w_{\rho'}$ in this region.

Finally, we show that after applying the two pruning strategies in Line 16-17 and Line 25-26, respectively, the correctness is still ensured. Firstly, Line 16-17 prune a sub-region only when $m^* \neq \infty$ (note that m^* is initialized to ∞), which means that an α-fairness equal to $\frac{k}{|D|}$ has been found and Line 37 has been executed. According to Lemma 3.3, $\frac{k}{|D|}$ is an upper bound of the α-fairness of any top-k set in D. Thus, in this case (i.e., if an α-fairness equal to $\frac{k}{|D|}$ has been found), there will not be a larger α-fairness value (Line 35 will not be executed any more). Then, w^* will be updated only when the same α-fairness is found in a sub-region which contains a utility function with smaller penalty (Line 32-33). Therefore, we can safely prune a region when $m^* \neq \infty$ and this sub-region cannot contain any better utility function (Line 16). Secondly, Line 25-26 prune a *small-than* region ρ' when $\min_{j \in \mathcal{P}} \frac{|T_{\rho'}[j]| + r - r'}{|D[j]|} < F^*$. According to Lemma 3.4, we know that any top-k set T inside region ρ' (which is a super-set of $T_{\rho'}$) will have an α-fairness at most $\min_{j \in \mathcal{P}} \frac{|T_{\rho'}[j]| + r - r'}{|D[j]|}$. Thus, for any top-$k$ set T inside region ρ', $\alpha\text{-}FN_{\mathcal{U}}(T) < F^*$, which means that this region can be safely pruned. □

Proof of Lemma 4.1. Recall that in $\mathcal{U}$, we have the "most general" global domain, which is denoted by $\mathcal{G}_0$. Under $\mathcal{G}_0$, all tuples in top-k set T are placed into one big cluster. Thus, if T is β-relaxed-fair and $\beta \geq \alpha$, one can see that this big cluster is also α-fair. In that case, according to Lemma 3.3, $\alpha\text{-}MinSP_{\mathcal{G}_0}(T)$ reaches an upper bound of $\alpha\text{-}MinSP_{\mathcal{G}_0}(T)$ for any $\mathcal{G} \in \mathcal{U}$ (which equals to $\min_{j \in \mathcal{P}} \frac{|T[j]|}{|D[j]|}$). $\mathcal{G}_0$ is thus the optimal global domain in $\mathcal{U}$. Therefore,

$$\begin{aligned}\alpha\text{-}FN_{\mathcal{U}}(T) &= \min_{j \in \mathcal{P}} \frac{|T[j]|}{|D[j]|} \\ &\geq \min_{j \in \mathcal{P}} \frac{\lceil \beta k \rceil}{|D[j]|} \\ &\geq \frac{\lceil \beta k \rceil}{\max_{j \in \mathcal{P}} |D[j]|}\end{aligned}$$

Based on Lemma 3.3, we have $\alpha\text{-}FN_{\mathcal{U}}(T^*) \leq \frac{k}{|D|}$. Thus,

$$\begin{aligned}\frac{\alpha\text{-}FN_{\mathcal{U}}(T)}{\alpha\text{-}FN_{\mathcal{U}}(T^*)} &\geq \frac{\lceil \beta k \rceil}{\max_{j \in \mathcal{P}} |D[j]|} \Big/ \frac{k}{|D|} \\ &\geq \frac{\lceil \beta k \rceil}{\max_{j \in \mathcal{P}} |D[j]|} \cdot \frac{|D|}{k} \\ &\geq \beta \cdot \frac{|D|}{\max_{j \in \mathcal{P}} |D[j]|} \geq \beta\end{aligned}$$

□

Proof of Theorem 4.1. Since the most steps of algorithm β-**FairTQ-Appro** (illustrated in Algorithm 2) follow our exact algorithm, we know that all the equal-to regions will give all the top-k sets in D. Clearly, all the top-k sets that are β-relaxed-fair will be included in Γ, while the remaining top-k sets (i.e., the non-β-relaxed-fair) will be processed in the same way as in the exact algorithm. Therefore, after the recursive call of procedure Partition (Line 3), the utility function w^*_{non} will be the one such that the α-fairness $F^*_{non} = \alpha\text{-}FN_{\mathcal{U}}(T^*_{non})$ where $T^*_{non} = \textbf{top}\text{-}k_{w^*_{non}}(D)$, is maximized among all the non-β-relaxed-fair top-k sets and meanwhile, $m(w^*_{non}, w_0)$ is minimized under the maximized α-fairness (based on Theorem 3.4).

Next, we show the claims in Theorem 4.1 in different cases.

Case (1). When $\Gamma = \emptyset$ which means that no top-k set is β-relaxed-fair, all the steps exactly follow the exact algorithm, and thus the returned result w^*_{non} exactly gives the optimal top-k set T^* which guarantees that $\alpha\text{-}FN_{\mathcal{U}}(T^*_{non}) = \alpha\text{-}FN_{\mathcal{U}}(T^*) \geq \alpha\text{-}FN_{\mathcal{U}}(T^*) \cdot \beta$. and $m(w^*_{non}, w_0) \leq m(w^*, w_0)$ based on Theorem 3.4.

Case (2). When $\Gamma \neq \emptyset$ and $F^*_{non} = -\infty$, the variable F^*_{non} has never been updated, which indicates that all top-k sets in D is β-relaxed-fair. In this case, we return the utility function w^*_{re} in Γ which guarantees that the top-k set w.r.t. w^*_{re} (i.e., (T^*_{re}) is β-relaxed-fair (and thus $\alpha\text{-}FN_{\mathcal{U}}(T^*_{re}) \geq \alpha\text{-}FN_{\mathcal{U}}(T^*) \cdot \beta$ by Lemma 4.1). Since Γ includes all regions in the utility space and we return the utility function in Γ (i.e., w^*_{re}) with the minimum penalty, we will obviously have $m(w^*_{re}, w_0) \leq m(w^*, w_0)$.

Case (3). When $\Gamma \neq \emptyset$ and $\alpha\text{-}FN_{\mathcal{U}}(T^*_{re}) > F^*_{non} = \alpha\text{-}FN_{\mathcal{U}}(T^*_{non})$, the optimal top-$k$ set T^* must be β-relaxed-fair, since otherwise it will be assigned to $T^*_{non} = \textbf{top}\text{-}k_{w^*_{non}}(D)$ and then we cannot have $\alpha\text{-}FN_{\mathcal{U}}(T^*_{re}) > \alpha\text{-}FN_{\mathcal{U}}(T^*_{non})$. According to Lemma 4.1, for the returned top-k set $T = T^*_{re} = \textbf{top}\text{-}k_{w^*_{re}}(D)$, we have $\alpha\text{-}FN_{\mathcal{U}}(T) \geq \alpha\text{-}FN_{\mathcal{U}}(T^*) \cdot \beta$. Since the region leading to T^* must be included in Γ (because T^* is β-relaxed-fair according to our discussion above) and w^*_{re} is the utility function in Γ with the minimum penalty, we have $m(w^*_{re}, w_0) \leq m(w^*, w_0)$.

Case (4). When $\Gamma \neq \emptyset$ and $\alpha\text{-}FN_{\mathcal{U}}(T^*_{re}) \leq F^*_{non} = \alpha\text{-}FN_{\mathcal{U}}(T^*_{non})$ we return a utility function w which is either w^*_{re} or w^*_{non}. For w^*_{re}, clearly, by Lemma 4.1 and since $T^*_{re} = \textbf{top}\text{-}k_{w^*_{re}}(D)$ is β-relaxed-fair, $\alpha\text{-}FN_{\mathcal{U}}(T^*_{re}) \geq \alpha\text{-}FN_{\mathcal{U}}(T^*) \cdot \beta$. For w^*_{non}, similarly, $\alpha\text{-}FN_{\mathcal{U}}(T^*_{non}) \geq \alpha\text{-}FN_{\mathcal{U}}(T^*_{re}) \geq \alpha\text{-}FN_{\mathcal{U}}(T^*) \cdot \beta$. Since the optimal top-$k$ set T^* could be either β-relaxed-fair or $\textbf{top}\text{-}k_{w^*_{non}}(D)$, the returned utility function w will satisfy $m(w, w_0) \leq m(w^*, w_0)$ following similar discussion in Case (3). □