2023

COMPACT REPRESENTATIVES OF DATABASES AND RESPONSIBLE DATA MANAGEMENT

Suraj Shetiya

COMPACT REPRESENTATIVES OF DATABASES
AND
RESPONSIBLE DATA MANAGEMENT

by

SURAJ SHETIYA

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON
August 2023

To my maternal grandmother Ratnaprabha R. Javali, and grand father figure (late) Ganapathi Prabhu who guided me and nurtured me.

To my parents Chandrakala and Suresh Shetiya who have always supported me and my brother Suhas who has been an inspiring role model.

ACKNOWLEDGEMENTS

I would like to express heartfelt appreciation to my supervising professor, Dr. Gautam Das for his constant support and guidance. Numerous intellectual discussions with him have been a constant part of my doctorate journey and were the biggest learning opportunities for me. He has also guided me through the process of growing as a researcher. I am also very grateful for the emphasis on *quality of research process* which has given me great learning opportunities.

I would like to thank Dr. Abolfazl Asudeh who has mentored me through the years. He has guided me during the various papers and helped me grasp the *art* of paper writing. He has also been an inspiration and a role model for the way he approaches a problem. I have also loved the various intellectual discussions with him during these years.

I would also like to express my gratitude to my esteemed committee members Dr. Chengkai Li, Dr. Vassilis Athitsos, and Dr. Shirin Nilizadeh for taking their time to serve on the dissertation committee and provide their invaluable feedback and guidance through out my doctorate. I am very grateful to late Dr. Ramez Elmasri for his guidance and invaluable feedback. Some of his narrated interactions with Don Knuth have been very inspirational.

Through out my doctorate journey, I have been very fortunate to have found many good friends and colleagues. First of all, I would like to express my gratitude to Azade Nova, with whom I started on the first research project. Interactions with her have been very inspiring and encouraging. I would like to thank Dr. Saravanan Thirumuruganathan for his invaluable inputs during many of my interactions. I am grateful to Dr. Jees Augustine for his interest in my research and invaluable advice in times of need. I wish to thank Dr.

Sona Hassani for her support and encouragement during various occasions along my PhD journey. I would like to thank Sadia Ahmed, Dr. Shohedul Hasan and Dr. Abhishek Santra who have helped me during numerous occasions.

I am grateful to all the teachers who taught me during the years I spent in university, first in India and finally in the United States. I would especially like to thank my teacher and project guide Dr. Suneetha K R for encouraging and inspiring me to pursue graduate studies. For arousing my interest in Computer Science, I would like to thank professor LMR from Vijaya Junior College, Bangalore. I would like to thank my close friends from BIT -Radhesh Anand, Manasa Vegulla, Suhas Shamasundara and Suhas T S and from NetApp - Kapil Bagrecha, Pavan Karkun and Swapnil Srivastava for being there during thick and thin.

On my personal front, I am very grateful to my maternal grandmother Ratnaprabha R. Javli and my late uncle Ganapathi Puttu Prabhu for always being guiding figures during the hard times of my life. I would like to thank my parents Chandrakala Shetiya and Suresh Shetiya for their sacrifice, encouragement and patience. Without their encouragement and endurance, this work would not have been possible. I would like to express my deep gratitude to my brother Suhas Shetiya who has encouraged and inspired me during the journey. I would like to thank my wife Bhavana Prabhu for the fun times and for inspiring me to keep going during the hard ones.

August 8, 2023

ABSTRACT

COMPACT REPRESENTATIVES OF DATABASES

AND

RESPONSIBLE DATA MANAGEMENT

Suraj Shetiya, Ph.D.

The University of Texas at Arlington, 2023

Supervising Professor: Gautam Das

With the advent of advanced computational models, we are being constantly judged by AI systems, complex algorithmic systems based on data that has been collected about us. These analysis are critical as they span many wide spread areas of our lives. For instance, these systems have been shown to find effective ways to fight back and make informed decisions during the COVID-19 pandemic.

The wide spread use of these models naturally give rise to a few questions regarding explaining decisions that these systems have made, fairness questions in various parts of these systems. In this dissertation, we present three important problems pertaining to interpret-ability and explain-ability in interactive black-box systems.

User preference is often complex to express and quantify. In our first work in the dissertation, we present the problem of finding a small set of items from a data-set. These set of items minimize the user's *regret* of missing out on the top item from the whole data-set. Such a model also presents an opportunity for interactively understanding the user's

preference and providing the user with the top item in the data-set. Our work presents the novel framework to deal with a class of regret measures.

Use of data which is riddled with representative biases leads to unfair outcomes. In our second word, we present the problem of integrating fairness into range queries. More importantly, our goal is to eliminate representative biases from the output data which often is consumed by complex systems which can make critical decisions. In our work, we present a innovative index structure to deal with the single predicate range queries and a neighborhood based search to deal with multi-predicate queries.

In our third work in this dissertation, we present the problem of explain-ability in black box matching systems. Our approach relies on Shapley values based methods for explanations. To speed the slow nature of Shapley algorithm, we present a sampling based approximate approach with theoretical guarantees. We present a few common problems that arise in these bipartite black-box matching systems. For each of these problems, we illustrate the design of the Shapley value function.

For each of these problems, we discuss our approach to the problem, analyze its behavior and empirically evaluate our procedure. Our extensive experimental results show the efficacy and efficiency of our techniques.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

Introduction

Data mining is a field of computer science that deals with the extraction of knowledge from data. Data mining algorithms are often used to solve a variety of problems. However, one of the challenges with data mining is that it can be used to generate biased results. This is because data mining algorithms are often trained on datasets that are not representative of the population as a whole. As a result, the results of data mining algorithms can be biased against certain groups of people.

In this dissertation, we present three works which deal with explain-ability and interpret-ability domain. First, we cover an interesting problem of understanding and interpreting user preferences in a multi-criteria decision making systems. Second, we present the work of integrating fairness constraints into database queries. Lastly, our work on explaining matching queries in black-box bipartite systems is presented.

Compact representatives of databases Given a database with numeric attributes, it is often of interest to rank the tuples according to linear scoring functions. For a scoring function and a subset of tuples, the regret of the subset is defined as the (relative) difference in scores between the top-1 tuple of the subset and the top-1 tuple of the entire database. Finding the regret ratio minimizing set (RRMS), i.e., the subset of a required size k that minimizes the maximum regret-ratio across all possible ranking functions, has been a well-studied problem in recent years. This problem is known to be NP-complete and there are several approximation algorithms for it. Other NP-complete variants have also been investigated, e.g., finding the set of size k that minimizes the average regret ratio over all linear functions. Prior work [1, 2] has designed customized algorithms for different

variants of the problem, and are unlikely to easily generalize to other variants. In this work we take a different path towards tackling these problems. In contrast to the prior works, we propose a unified algorithm for solving different problem variants. Unification is done by localizing the customization to the design of variant-specific subroutines or *"oracles"* that are called by our algorithm. Our unified algorithm takes inspiration from the seemingly unrelated problem of clustering from data mining, and the corresponding $K - MEDOID$ algorithm. We design our algorithm with innovations ranging from linear programming, edge sampling in graphs, and volume estimation of multi-dimensional convex polytopes. We provide rigorous theoretical analysis, as well as substantial experimental evaluations over real and synthetic data sets to demonstrate the practical feasibility of our approach. The details of our work on compact representatives of databases is presented in Chapter 2.

Responsible data management

This dissertation covers two published papers that fall under the umbrella of responsible data management.

Fairness in range queries: We are being constantly judged by automated decision systems that have been widely criticized for being discriminatory and unfair. Since an algorithm is only as good as the data it works with, biases in the data can significantly amplify unfairness issues. In this paper, we take initial steps towards integrating fairness conditions into database query processing and data management systems. Specifically, we focus on selection bias in range queries. We formally define the problem of fairness-aware range queries as obtaining a fair query which is most similar to the user's query. We propose a sub-linear time algorithm for single-predicate range queries and efficient algorithms for multi-predicate range queries. Our empirical evaluation on real and synthetic data-sets confirms the effectiveness and efficiency of our proposal. Chapter 3 presents all the details of our work on integrating fairness into range queries.

Shapley values based explanations to bipartite matching scenarios: In this work, we initiate research in explaining matchings. In particular, we consider the large-scale two-sided matching applications where preferences of the users are specified as (ranking) functions over a set of attributes and matching recommendations are derived as top-k. We consider multiple natural explanation questions, concerning the users of these systems. Observing the competitive nature of these environments, we propose multiple Shapley-based approaches for explanation. Besides exact algorithms, we propose a sampling-based approximation algorithm with provable guarantees to overcome the combinatorial complexity of the exact Shapley computation. Our extensive experiments on real-world and synthetic data sets validate the usefulness of our proposal and confirm the efficiency and accuracy of our algorithms. The details of this work is present in Chapter 4.

CHAPTER 2

A Unified Optimisation Algorithm For Solving "Regret-Minimising Representative" Problems

Given a database with numeric attributes, it is often of interest to rank the tuples according to linear scoring functions. For a scoring function and a subset of tuples, the regret of the subset is defined as the (relative) difference in scores between the top-1 tuple of the subset and the top-1 tuple of the entire database. Finding the regretratio minimizing set (RRMS), i.e., the subset of a required size k that minimizes the maximum regret-ratio across all possible ranking functions, has been a well-studied problem in recent years. This problem is known to be NP-complete and there are several approximation algorithms for it. Other NP-complete variants have also been investigated, e.g., finding the set of size k that minimizes the average regret ratio over all linear functions. Prior work have designed customized algorithms for different variants of the problem, and are unlikely to easily generalize to other variants. In this paper we take a different path towards tackling these problems. In contrast to the prior, we propose a unified algorithm for solving different problem variants. Unification is done by localizing the customization to the design of variant-specific subroutines or "oracles" that are called by our algorithm. Our unified algorithm takes inspiration from the seemingly unrelated problem of clustering from data mining, and the corresponding K-MEDOID algorithm. We make several innovative contributions in designing our algorithm, including various techniques such as linear programming, edge sampling in graphs, volume estimation of multi-dimensional convex polytopes, and several others. We provide rigorous theoretical analysis, as well as substantial experimental evaluations over real and synthetic data sets to demonstrate the practical feasibility of our approach.

2.1 Introduction

Data-driven decision making is challenging when there are multiple criteria to be considered. Consider a database of n tuples with d numeric attributes. In certain cases, "experts" can come up with a (usually linear) function to combine the criteria into a "goodness score" that reflects their preference for the tuples. This function can then be used for ranking and evaluating the tuples [3–6]. However, devising such a function is challenging [7, 8], hence not always a reasonable option, especially for ordinary non-expert users [9]. For instance consider a user who wants to book a hotel in Miami, FL. She wants to find a hotel that is affordable, is close to a beach, and has a good rating. It is not reasonable to expect her to come up with a ranking function, even though she may roughly know what she is looking for. Therefore, she will probably start exploring different options and may end up spending several confusing and frustrating hours before she can finalize her decision. Alternatively, one could remove the set of "dominated" tuples [10], returning a Pareto-optimal [11] (a.k.a. *skyline* [9, 10]) set, which is the smallest set guaranteed to contain the "best" choice of the user, assuming that her preference is monotonic [10]. In the case where user preferences are further restricted to linear ranking functions, only the *convex hull* of the dataset needs to be returned.

The problem with the skyline or convex hull is that they can be very large themselves, sometimes being a significant portion of the data [12, 13], hence they lose their appeal as a small representative set for facilitating decision making. Nanongkai et al. [12] came up with the elegant idea of finding a small set that may not contain the absolute "best" for any possible user (ranking function), but guarantees to contain a *satisfactory* choice for each possible function. To do so, they defined the notion of "*regret-ratio*" of a representative subset of the dataset for any given ranking function as follows: it is the relative score difference between the best tuple in the database and the best tuple in the representative set. Given $k < n$, the task is to find the *regret-ratio minimising set* (RRMS), i.e., a subset

of size k that minimises the maximum regret-ratio across all possible ranking functions. This problem is shown to be NP-complete, even for a constant (larger than two) number of criteria (attributes) [14]. Other researchers have also considered different versions of the problem formulation. For instance Chester et. al. [1] generalise the notion of regret from the comparison of the the actual top-1 of database to the top-k. More recently, in [2, 15] the goal was to compute the representative set that minimises the *average regret-ratio* across all possible functions, instead of minimising the max regret-ratio. All these variants have been shown to be NP-complete.

Given their intractable nature, there has been significant effort in designing efficient heuristics and approximation algorithms for these problems. The RRMS problem has been investigated in several papers [13, 14, 16], and several approximation algorithms have been designed; the algorithms in [13, 14] run in polynomial time and can approximate the max-regret ratio within any user-specified accuracy threshold. The average regret-ratio problem has been investigated in [15], and a different greedy approach has been proposed, which achieves a constant (though not any user-specified) approximation factor with high probability.

2.1.1 Technical Highlights

In this chapter, we make the following observations about the previous works: (a) the proposed algorithms are dependent on the specific problem formulation, and do not seem to generalise to different variants, and (b) the focus has been on designing approximation algorithms, and not on optimal algorithms. Consequently, we take a different route towards solving these problems, and our work makes two important contributions:

Firstly, we develop a *unified algorithm* that works across different formulations of the problem, including max [12] and average [15] regret minimising sets. The unified algorithm makes calls to a subroutine (we refer to it as an "oracle"), and it is this subrou-

tine/oracle that needs to be customised for each problem variant. Thus the customisation is localised to the design of the oracle.

Secondly, we make a connection between the various regret minimising problems and the seemingly unrelated classical data mining problem of *clustering* and the well-known K-MEDOID [17] algorithm. Most variants of clustering are NP-hard, yet the K-MEDOID algorithm is extremely popular in practice and is based on a hill-climbing approach to find a local optima. One of the main technical highlights of our contributions is to take inspiration from, and design our unified algorithm based on the K-MEDOID algorithm, even though the regret minimization problems seemingly appear quite different from clustering problems. One of the consequences of our approach is that the well-known advantages of the K-MEDOID algorithm transfer over to our unified algorithm. For example, the K-MEDOID algorithm has the *any-time* property; given more time it can be repeatedly restarted from different random starting configurations, which gives it the ability to improve the local optima that it has discovered thus far. Our unified algorithm also has this property, which can be useful in time-sensitive applications, including a query answering system.

To achieve our two contributions, several novel and challenging technical problems had to be solved. The K-MEDOID algorithm provided inspiration, but was not really easily adaptable for our case. Instead, we had to carefully model our problem as that of optimally partitioning the space of all ranking functions into k convex geometric regions, such that for each region exactly one tuple from the database is the "representative", i.e., it has the highest (max, or average, depending on the problem variant being solved) score for any function in that region. At a high level, our algorithm first chooses k tuples randomly as our initial representative set. Then it only examines these k tuples, and partitions the function space into k convex regions such that the tuple associated with each region outscores the remaining $k-1$ tuples for any ranking function within its region. Then, the database is

examined to update the best representative for each region, and the process iterates until a local optima is reached.

The process of examining the database for updating the best representative for any region required us to develop variant-specific subroutines/oracles. For the case of the max regret-ratio, we propose different innovative strategies for designing an efficient oracle. For large regions, we design a threshold-based algorithm based on function-space discretization. For narrow regions, we model the problem as an instance of edge sampling from a weighted graph where edge weights are determined by solving (constant-sized) linear programs.

For the average regret-ratio case, designing an oracle was challenging. As another of our innovative technical highlights, we show that it is reducible to the classical problem of computing the volume of polytopes defined by the intersection of $O(n)$ half-spaces. Unfortunately, this approach has a time complexity of $\Omega(n^d)$ [18]. Even if we assume that the dimension of the database is fixed, this high complexity makes this approach only of theoretical interest. Consequently, we propose an alternative approach based on Monte-Carlo sampling of the function space.

We provide proof of convergence of our unified algorithm, as well as provide detailed theoretical analyses of our oracles. We also conduct extensive empirical experiments on both real and synthesis datasets that show the efficiency and effectiveness of our proposal.

2.1.2 Summary of Contributions

In summary, our contributions are as follows:

- We develop a unified algorithm for solving different variants of the regret-ratio representative set problem. This is in contrast to prior works that developed custom algorithms for each variant; our customisation is localised to the design of variant-specific "oracles".

- Our unified algorithm is inspired by the seemingly unrelated K-MEDOID clustering algorithm. Thus it is designed to find locally optimal solutions, in contrast to prior works that focused on approximation algorithms.
- Our unified algorithm is based on showing that the optimal representative set of k tuples induces a partitioning of the function space into k convex regions, each identified by exactly $(k-1)$ hyperplanes (independent of the number of dimensions), where within every region one of the tuples of the set is the "best" in terms of the optimisation goal.
- We provide innovative approaches to designing custom oracles for the different problem variants, including graph transformation, a threshold-based algorithm, and Monte-Carlo estimation.
- Our approaches are principled, supported by theoretical analysis and guarantees.
- We conduct extensive experiments on both real and synthesis datasets to demonstrate the efficiency and effectiveness of our proposal.

The rest of this chapter is organised as follows. In section § 2.2, we introduce the terms and formally define the problem. § 2.3 makes the connection between the *URM* algorithm and the K-MEDOID algorithm, provides the proof of convergence, and ends with a running example. In § 2.4 and § 2.5, we present the oracles for max and average regret-ratio, respectively. Experiment results and related work are provided in § 2.6 and § 2.7, respectively.

2.2 Preliminaries

Data Model: We consider a database $\mathbb{D}$ in the form of n tuples t_1 to t_n, defined over d numeric attributes A_1 to A_d. We use the notation $t_i[j]$ to show the value of t_i on attribute A_j. Without loss of generality, we assume that attribute values are normalised and standardised

as the non-negative real numbers, $\mathbb{R}^+$. The numeric attributes are used for scoring and ranking the tuples. Additionally, the dataset may also include non-ordinal attributes that may be used in filtering, but not in ranking. Finally, for each attribute A_i, we assume that the larger attribute values are preferred. The values x of an attribute A_i with smaller-preferred nature require a straight forward transformation such as $(\max(A_i) - x)/(\max(A_i) - \min(A_i))$.

Example 1. *As a running example in this chapter, consider a dataset of* 10 *tuples, defined over the attributes* A_1 *to* A_3*, as shown below. The values of the attributes are normalised in range [0,100] and for all attributes the higher values are preferred. In this example* $t_4[3]$ *refers to the value of tuple* t_4 *on attribute* d_3 *which is equal to* 75*.*

tuple	A_1	A_2	A_3
t_1	*60*	*80*	*77*
t_2	*55*	*75*	*63*
t_3	*75*	*60*	*59*
t_4	*68*	*70*	*75*
t_5	*80*	*75*	*73*
t_6	*56*	*65*	*91*
t_7	*61*	*78*	*80*
t_8	*90*	*60*	*58*
t_9	*86*	*68*	*74*
t_{10}	*77*	*67*	*82*

Ranking Model: The database tuples are ranked using the scores assigned by a *ranking function*. For every tuple $t \in \mathbb{D}$, a ranking function $f : \mathbb{R}^d \rightarrow \mathbb{R}^+$ assigns a non-negative score to t. A tuple t_i outranks t_j based on f if its score is larger than the one of t_j. Following the literature in regret-minimising context [1,12–14], we consider the class of linear ranking

functions in this chapter[1]. The score of a tuple t based on a linear function f with a weight vector $\vec{w} = \{w_1, w_2, \cdots, w_d\}$ is computed as:

$$f_{\vec{w}}(t) = \sum_{i=1}^{d} w_i \cdot t[i] \tag{2.1}$$

In the rest of chapter, we simplify $f_{\vec{w}}(t)$ to $f(t)$ when $\vec{w}$ is clear in the context. As an example, let us consider Example 1, while choosing $\vec{w} = \langle .25, .5, .25 \rangle$ for the ranking function. Using this function, $f(t_1) = .25 \times 60 + .5 \times 80 + .25 \times 77 = 74.25$. Computing the score of other tuples, the ordering of the tuples in $\mathbb{D}$ based this function is $\{t_5, t_1, t_7, t_9, t_{10}, t_4, t_6, t_2, t_8, t_3\}$.

In a d-dimensional space that every tuple is presented as a point, every linear function can be modelled as a origin-starting ray that passes through the point specified by its weight vector. For example, the function f with the weight vector $\vec{w} = \langle .25, .5, .25 \rangle$ is modelled as the ray that starts from the origin and passes through the point $\langle .25, .5, .25 \rangle$. The ordering of tuples based on f is specified by the ordering their projections on the ray of f (please refer to [7] for further details). As a result, the universe of origin-starting ray in the first quadrant of the d dimensional space shows the universe of linear functions. We call this the function space.

A tuple t_i of a database is said to dominate tuple t_j if each of the attribute values for t_i is not smaller than that of t_j's while there exists an attribute A_k where $t_i[k] > t_j[k]$. For instance, in Example 1, $t_1 : \langle 60, 80, 77 \rangle$ dominates $t_2 : \langle 55, 75, 63 \rangle$. The set of tuples from the database which are not dominated tuples in the database is known as the *skyline* (or Pareto-optimal) [9–11].

A maxima representative, or simply a *representative*, is a subset $S \subseteq \mathbb{D}$ that is used for finding the maximum of $\mathbb{D}$ for any arbitrary ranking function f. Skyline is the minimal representative of $\mathbb{D}$ that guarantees the containment of the maximum of any function in the

[1]Note that a large class of non-linear functions can fit this model after a straight-forward linearization [19].

class of monotonic ranking functions. That is the reason it is popular for multi-criteria decision making (in the absence of a ranking function). Similarly, *convex-hull* is the minimal representative that contains the maximum for the class of linear ranking functions.

However, the skyline or convex-hull may contain a large portion of the database, which diminishes their applicability as a small set for decision making [12, 13]. For instance, the skyline of Example 1 is $\{t_1, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\}$, which includes 80% of the tuples.

2.2.1 Regret optimisation measures

Regret ratio is a measure of dissatisfaction of a user when she sees results returned from the subset instead of the entire database. Let f be a ranking function, then regret ratio is defined as the ratio of the difference in ranking function scores between the top database tuple and the top tuple from the representative set to the score of the top database tuple:

Definition 2.2.1 (Regret Ratio)**.** *Given a function f, let tuple t be $argmax_{\forall t \in \mathbb{D}} f(t)$. The regret ratio of set $S \subseteq \mathbb{D}$ for a ranking function f can be computed as*

$$rr(f, S) = \min_{t' \in S} \frac{f(t) - f(t')}{f(t)} \tag{2.2}$$

For instance, in Example 1, consider the set $S = \{t_2, t_3, t_4\}$ and the function f with the weight vector $\vec{w} = \langle .25, .5, .25 \rangle$. t_5 is the max for this function ($f(t_5) = 75.75$) while t_4 is the tuple with the max score ($f(t_4) = 70.75$) in S. Therefore, $rr(f, S)$ in this example is $(75.75 - 70.75)/75.75 \simeq .066$.

For a universe of ranking functions, an aggregate over the regret ratio of each ranking function is considered as the regret ratio of the representative set for the universe.

In this chapter, we provide *a unified* model that can handle a wide variety of aggregates (any ℓ^p norm):

Definition 2.2.2. *ℓ^p norm regret measure : Given a database $\mathbb{D}$, a representative $S \subseteq \mathbb{D}$, a real number $p \geq 1$, and a set of ranking functions F, the ℓ^p norm aggregate measure Agg^p is defined as:*

$$RR_F(S) = Agg^p_{\forall f \in F} rr(f, S)$$

While different aggregates can be used here, so far the literature has considered *maximum* and *average* regret ratio minimising sets. Without limiting our proposal to a specific value of norm, we especially show the adaptation of our framework for the existing aggregates, i.e., (i) ℓ_∞: maximum regret ratio and (ii) ℓ_1: average regret ratio. We will lay out the formal definitions of these measures in the remainder of this section.

Definition 2.2.3. *Maximum (ℓ^∞ norm) Regret Ratio: Given a database $\mathbb{D}$ and a set of functions F, the maximum regret ratio [12] of a set $S \subseteq \mathbb{D}$ is the maximum value of regret ratio for the set S over the set of all possible ranking functions F. That is,*

$$RR_F(S) = \sup_{f \in F} rr(f, S) \tag{2.3}$$

While the maximum regret ratio looks at the worst case regret ratio for a set of functions, a different measure for user dissatisfaction is the average regret ratio.

Definition 2.2.4. *Average (ℓ^1 norm) Regret Ratio: Given a database $\mathbb{D}$, a set of functions F, and a probability distribution $\eta(.)$ where $\eta(f)$ is the probability of each function $f \in F$, the average regret ratio [15] of a set $S \subseteq \mathbb{D}$ is defined as*

$$ARR_F(S) = \int_{f \in F} \eta(f)\, rr(f, S)\, df \tag{2.4}$$

Even though the regret ratio notions are defined for general classes of functions, the majority of the existing work consider F as the class of linear ranking functions (Equation 2.1) [1, 12–15, 20]. Also, for average regret ratio, the uniform distribution is considered as the probability distribution of ranking functions $\eta(.)$ [15]. We follow the literature

on these. In the rest of the chapter, we simplify the notations $RR_F(S)$ and $ARR_F(S)$ to $RR(S)$ and $ARR(S)$ for the class of linear ranking functions ($\mathcal{L}$).

2.2.2 Problem Definition

In this chapter, we consider the problem of finding a compact representative of size k from a database such that the regret optimisation measure is minimised. Formally:

> **ℓ^p REGRET RATIO REPRESENTATIVE PROBLEM:**
>
> *Given a dataset $\mathbb{D}$, a set of ranking functions $\mathcal{F}$, and a value k, find the representative S of $\mathbb{D}$ for $\mathcal{F}$ such that $|S| = k$ and ℓ^p norm regret measure (Definition 2.2.2) of S is minimised.*

Specifically, the problem for the max. (resp. avg.) regret ratio is to find a set $S \subseteq \mathbb{D}$ of size k that minimises the max. (resp. avg.) regret ratio. For any constant number of dimensions larger than 2, the problem for max. regret ratio is shown to NP-complete [14]. Similarly, the problem for average regret ratio is proven to be NP-complete [15].

The two problems we consider in this chapter are the max and average regret ratio optimising set problems. In section 3, we describe the Regret Minimising Framework algorithm, a general framework to solve the regret ratio class of problems, followed by sections 4 and 5 which discuss the details of the oracles for the Maximum and Average regret ratio problems respectively. We show empirical results for both these problems in section 6.

2.3 Unified Regret Minimisation

2.3.1 Overview

In this section, we propose our *Unified Regret Minimizer (URM)* algorithm which is inspired from the K-MEDOID algorithm.

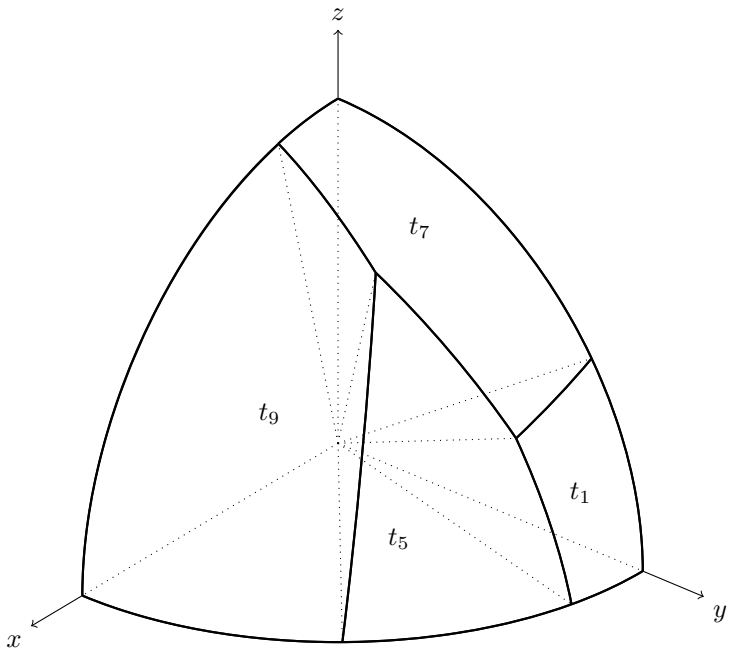


Figure 2.1: In Example 1, the tuples in a representative $\{t_1, t_5, t_7, t_9\}$ partition the entire function space into convex regions. In general, for a representative of size k, each of the regions are formed as the intersection of $(k-1)$ half spaces. Each tuple can be thought of as being in charge of its own region.

Before providing the details of the algorithm, it is necessary to explain two central ideas in this chapter: (i) representative and (ii) region of a tuple t, defined in definition 2.3.1 and definition 2.3.2 respectively.

Definition 2.3.1. *Given a function f and a set of tuples $S \subseteq \mathbb{D}$, a tuple $t \in S$ is a representative for f if it has the maximum score, based on f among elements of S. Formally:*

$$\rho(f, S) = \mathit{argmax}_{t \in S} f(t)$$

Definition 2.3.2 (Region of The Tuple t)**.** *Given a set S and a tuple $t \in S$, the region of t is the set of functions for which t is the representative. That is,*

$$R_t(S) = \{f \in \mathcal{L} \mid \rho(f, S) = t\}$$

For example, Figure 2.1 shows the regions of the tuples $\{t_1, t_5, t_7, t_9\}$ in Example 1.

Having the necessary definitions in place, we now provide a brief overview of the K-MEDOID and then show the transformation of our problem to it. Recall that given a database $\mathbb{D}$, the output size k and a regret measure, the *URM* algorithm finds a set of k representative tuples from $\mathbb{D}$, such that the regret measure is minimised.

K-MEDOID *algorithm:* This is an iterative algorithm that partitions a set of n objects into k clusters, such that the distance between objects belonging to a cluster and their cluster centre are minimised. While many clustering algorithms (such as *kNN* [21]) have cluster centres that may not belong to the set of n objects, the K-MEDOID algorithm sets itself apart by choosing the cluster centres from the set of n objects. More details about the algorithm is provided in [17].

Although the clustering problem and the K-MEDOID algorithm seems far removed and disconnected from our regret minimisation problems, the notion of *region of a tuple*, provided in Definition 2.3.2 is the key in the problem transformation. As we shall show in the following, we see the problem as a partitioning of the function space into k convex regions while the tuples are the centroids.

Before providing further technical details in § 2.3.2, we would like to highlight a key difference between the nature of the problems: While the K-MEDOID algorithm deals with countable set of objects n, the *URM* algorithm deals with an infinite set of objects (functions). The K-MEDOID algorithm clusters n objects (n being a discrete and finite number) into k clusters, each of which contains finite number of objects. In contrast, as we shall elaborate next, the *URM* algorithm clusters the continuous space of ranking functions, which contains an infinite number of functions into k convex regions of ranking functions.

The *URM* algorithm forms the basis for a unified framework to find compact representatives for a variety of regret measures. It does this by abstracting the various notions of regret measures into an *oracle*. At a high level, *URM* operates as follows. It is initialised

with a set of k tuples (tuples), which form the compact representatives for the first iteration of the algorithm. These k tuples are used to partition the entire function space into k convex regions, such that for each region the corresponding tuple is the representative. In each iteration, we replace the k tuples with a potentially new set of k tuples that improves the regret measure. This is done by choosing, for each convex region of functions, the tuple from the database $\mathbb{D}$ that is the best representative for that region (this is accomplished by making calls to the oracle). The function space is re-partitioned into k new convex regions, and the iterations continue until we converge to a local optima (i.e., the k tuples do not change).

Region	**Half-space 1**	**Half-space 2**	**Half-space 3**
Region for t_1	$t_1 - t_5$: $-20A_1 + 5A_2 + 4A_3 \geq 0$	$t_1 - t_7$: $-A_1 + 2A_2 - 3A_3 \geq 0$	$t_1 - t_9$: $-26A_1 + 12A_2 + 3A_3 \geq 0$
Region for t_5	$t_5 - t_1$: $20A_1 - 5A_2 - 4A_3 \geq 0$	$t_5 - t_7$:$19A_1 - 3A_2 - 7A_3 \geq 0$	$t_5 - t_9$: $-6A_1 + 7A_2 - A_3 \geq 0$
Region for t_7	$t_7 - t_1$: $A_1 - 2A_2 + 3A_3 \geq 0$	$t_7 - t_5$: $-19A_1 + 3A_2 + 7A_3 \geq 0$	$t_7 - t_9$: $-25A_1 + 10A_2 + 6A_3 \geq 0$
Region for t_9	$t_9 - t_1$ $26A_1 - 12A_2 - 3A_3 \geq 0$	$t_9 - t_5$: $6A_1 - 7A_2 + A_3 \geq 0$	$t_9 - t_7$: $25A_1 - 10A_2 - 6A_3 \geq 0$

Table 2.1: Table containing the inequalities that define each region of the representative $\{t_1, t_5, t_7, t_9\}$.

2.3.2 Details

The key in the design of *URM* is that a set of k representative tuples *partition* the function space into k *convex* regions, such that each of the tuples t_i is the "representative" for all functions its region R_i. This enables adopting the K-MEDOID technique for clustering the function space.

Theorem 2.3.1. *Consider the set $S : \{t_1, \cdots, t_k\}$ as the compact representative of database $\mathbb{D}$ and the region of functions that are represented by t_i be R_i. Then the followings hold:*

1. *The regions $R_1, \cdots, R_k$ partition the function space.*
2. *For each tuple t_i, R_i is convex.*

3. *Each region R_i is the intersection of $(k-1)$ half-spaces.*

Proof. Let us consider a ranking function f from the space of all ranking functions F. From equation 2.2, the regret ratio $r = rr(f, S)$ is the regret ratio of the set S for the function f. The tuple from the set S which achieves this regret ratio is given by,

$$t = \text{argmin}\, \forall_{i \leq k} \frac{f(t) - f(t_i)}{f(t)}$$

This tuple $t \in S$ is the representative for the function f. The region R_i consists of all the functions that have t_i as the representative. As the above property is applicable to the set of all ranking functions, regions $R_1, \cdots, R_k$ partition the function space into k regions. This proves property 1.

The region of functions for which the tuple t_i is ranked higher than tuple t_1 is a half-space denoted by H_{i1}^+, which is defined by the inequality,

$$\sum_{i=1}^{k} w_i \cdot (t_i[i] - t_1[i]) \geq 0 \tag{2.5}$$

The hyper-plane represented by the left-hand side of equation 2.5 is denoted as ordering exchange hyper-plane as it divides the space into two regions, one in which tuple t_i is better than t_1 and vice versa in the other. As equation 2.5 represents a half-space and half-spaces are convex, H_{i1}^+ is a convex region. Similarly, let H_{ij}^+ be the region of functions where t_i is better than t_j. As R_i is the region of functions where t_i is the representative, we need to consider the function space where t_i is better than all the other tuples, which is given by,

$$R_1 = \cap_{j=1}^{k} H_{ij}^+ \tag{2.6}$$

Region R_i is an intersection of $(k-1)$ half-spaces. This proves property 2.

We know that, the intersection of convex regions is convex. As half-spaces are convex, region R_i, which is an intersection of $(k-1)$ half-spaces is convex. This proves property 3. □

An interesting observation from Theorem 2.3.1 is that each region is described as a list of $(k-1)$ half-spaces (equations), which is *independent of the number of dimensions*.

As an example let us consider Figure2.1, which shows the regions for the tuples $\{t_1, t_5, t_7, t_9\}$ in Example 1. Table 2.1 shows the three half-spaces define each of the regions. For instance, the region of t_1 is described by the half-spaces between t_1 and t_5, t_7, and t_9. Each of these regions are drawn in Figure 2.1. One can verify that these regions are convex and partition the function space.

Theorem 2.3.1 shows that the regions of a set of tuples $S : \{t_1, .., t_k\}$ *partition* the function space into k non-overlapping convex clusters. Now we show how this enables adopting the K-MEDOID algorithm. Consider a database $\mathbb{D}$ and an initial set $S : \{t_1, .., t_k\}$ of k representative tuples in the first iteration of the algorithm. Let the region of functions R_i for tuple t_i be represented as the intersection of $(k-1)$ half-spaces. Based on Definition 2.3.2, t_i is preferred over all tuples $t_j \in S \backslash \{t_i\}$ for any function $f \in R_i$. *But it does not necessarily mean that t_i is preferred over all tuples $t_j \in \mathbb{D} \backslash \{t_i\}$.* So, for each i, there may be another tuple in the dataset that can introduce a smaller regret compared to t_i for R_i. This is a key observation in designing the subsequent iterations of the algorithm. To do so, we rely on the existence of the "regret optimisation oracle", that finds the best representative tuple in the database for R_i. This oracle is dependent on the variant of the regret measure we are seeking to optimise.

Regret optimisation oracle: The regret optimisation oracle is the variant-specific part of our overall approach. It is used to find a representative tuple from the database which best optimises a specific regret measure a convex region of ranking functions R_i (e.g., max regret, average regret, more general ℓ_p-norm regret, etc). Formally, given a database $\mathbb{D}$, a particular variant of regret measure of interest, and a convex region of functions R_i defined by the intersection of half-spaces, the regret optimisation oracle finds the tuple (and the

corresponding regret measure) from the database which minimises the regret measure over all functions in R_i.

For now, we assume that such a variant-specific oracle exists, and we proceed with describing how our unified algorithm *URM* can leverage such an oracle (details about the design of the oracle for different problem variants are deferred to § 2.4 and § 2.5). At every iteration of the algorithm, for every region of functions R_i, the regret optimisation oracle is used to find the new representative for the region R_i. The set of k tuples obtained from k calls to the regret optimization oracle form the new set of representatives for the next iteration of the algorithm. The iterations continue until the representative set of tuples ceases to change.

The pseudo code of the *URM* algorithm is given in Algorithm 1. Also, Algorithm 2 shows the pseudo code for finding the region R_i for a tuple t_i, in the form of the intersection of $(k-1)$ half-spaces.

Compared to the existing literature for regret ratio minimising problem, the *URM* algorithm has some unique and important features. *URM* provides a unified framework for the ℓ^p and ℓ^∞ classes of regret ratio measures. Our iterative algorithm has the any-time property where the user may stop at any time/iteration and still find a set of compact representatives. The any-time property can be very useful in case of real time query answering systems and other time-sensitive applications.

2.3.3 Proof of convergence for ℓ^p and ℓ^∞ norms

A critical factor for iterative algorithms is the guarantee of convergence. In this section, we prove that our algorithm converges to a local optima, for the ℓ^p and ℓ^∞ norm class of regret ratio measures.

Theorem 2.3.2. *In each successive iteration of the URM algorithm, the regret measure for the set $S : \{t_1, \cdots, t_k\}$ improves for the ℓ^p norm regret measure.*

Algorithm 1 *URM* Algorithm

Input: Database $\mathbb{D}$, Regret Oracle Orc, Initial set of k tuples $initial$

Output: Representative S

$S \leftarrow initial$
repeat
    $S' \leftarrow$ new List
    for tuple t in S **do**
        $R_t \leftarrow$ **RegionOf**(t, $S \setminus \{t\}$)
        $t' \leftarrow Orc(\mathbb{D}, R_t)$ `// the best tuple for` R_t `based on` Orc
        Add t' to S'
    end for
    $S \leftarrow S'$
until Convergence
return S

Algorithm 2 RegionOf

Input: A set of k tuples S and a tuple $t \in S$

Output: R_t in the form of intersection of $(k-1)$ half spaces

$R_t \leftarrow$ *new set*
for tuple t_j in $S \setminus \{t\}$ **do**
`//` H_j `is defined as as` $\sum_{i=1}^{d} H_j[i] \times w_i \geq 0$
    $H_j \leftarrow$ *new list* of size d
    for $i \leftarrow 1$ to d **do** $H_j[i] = t[i] - t_j[i]$
    Add H_j to R_t
end for
return R_t

Proof. Let us consider a set $S : \{t_1, \cdots, t_k\}$ as the compact representatives before the iteration and let set $S' : \{t'_1, \cdots, t'_k\}$ be the compact representatives after the iteration. The ℓ^p norm for the set S can be described by

$$RR = \sqrt[p]{\int_{f \in F} (rr(f, S))^p} = \sqrt[p]{\sum_{i=1}^{k} \int_{f \in R_i} (rr(f, t_i))^p}$$

During the iteration, for every region R_i the regret minimisation oracle finds a tuple from the database $\mathbb{D}$ that minimises the ℓ^p norm regret measure. Hence, we know that,

$$\underset{i=1}{\overset{k}{\forall}} \int_{f \in R_i} (rr(f, S))^p \geq \int_{f \in R'_i} (rr(f, t'_i))^p \tag{2.7}$$

As the equation for RR' can be written as,

$$RR' = \sqrt[p]{\sum_{i=1}^{k} \int_{f \in R_i} (rr(f, t'_i))^p} \tag{2.8}$$

Using equation 2.7 in equation 2.8, we get

$$\sqrt[p]{\sum_{i=1}^{k} \int_{f \in R_i} (rr(f, t_i))^p} \geq \sqrt[p]{\sum_{i=1}^{k} \int_{f \in R_i} (rr(f, t'_i))^p}$$

As the region of t_i differs from that of t'_i, we can conclude

$$rr(f, t'_i)) \geq (rr(f, S') \tag{2.9}$$

$$RR \geq RR'$$

□

A similar proof for ℓ^∞ norm can be proved.

Theorem 2.3.3. *In each successive iteration of the URM algorithm, the regret measure for the set $S : \{t_1, \cdots, t_k\}$ improves for the ℓ^∞ norm regret measure(maximum regret ratio).*

Proof. Let us consider a set $S : \{t_1, \cdots, t_k\}$ as the compact representatives before the iteration and let set $S' : \{t'_1, .., t'_k\}$ be the compact representatives after the iteration. The ℓ^∞ norm regret measure for the set S can be described by

$$RR = \max_{i=i}^{k} \sup_{f \in R_i} rr(f, t_i)$$

During the iteration, for every region R_i the regret minimisation oracle finds a tuple from the database $\mathbb{D}$ that minimises the ℓ^∞ norm regret measure. Hence, we know that

$$\forall_{i=1}^{k} \sup_{f \in R_i} rr(f, t_i) \geq \sup_{f \in R_i} rr(f, t'_i) \tag{2.10}$$

Using equations 2.10 and 2.9 we get,

$$\max_{i=i}^{k} \sup_{f \in R_i} rr(f, t_i) \geq \max_{i=i}^{k} \sup_{f \in R_i} rr(f, t'_i) \geq \sup_{f \in F} rr(f, S')$$

$$RR \geq RR'$$

□

2.3.4 Running Example

To provide a better understanding of the algorithm, in this section we provide a run of the algorithm over Example 1, while considering max as the target regret ratio measure. Let the set $S = \{t_1, t_5, t_7, t_9\}$ be the initial representative for the algorithm. The region of each tuple is highlighted in Table 2.1. Based on Definition 2.3.2, each tuple t_i is in charge of its own region R_i, highlighted in Figure 2.1, i.e., t_i is the best tuple in S for all the ranking functions lying inside R_i. In the next iteration the *URM* algorithm goes through each of these regions to find better representatives. To do this, it calls the oracle *MaxO*. We shall provide the details of this oracle in § 2.4. For each region R_i, the oracle

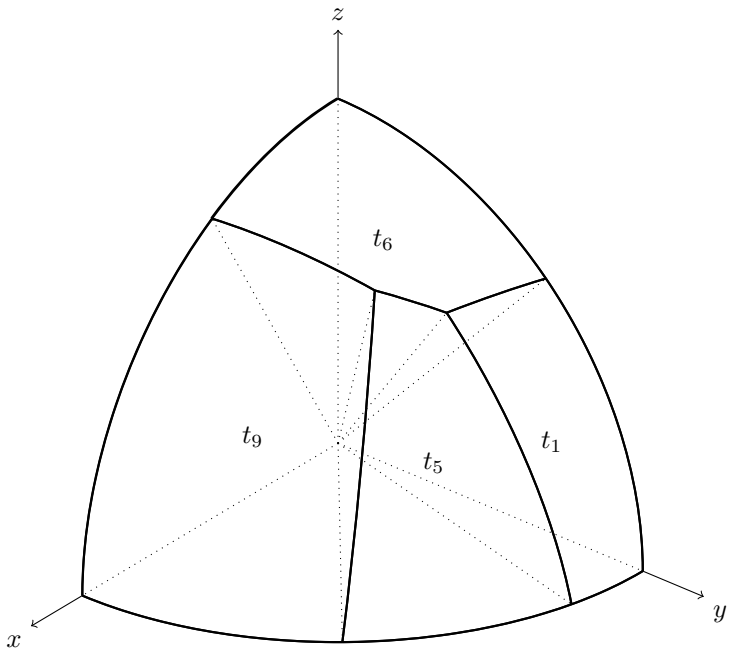


Figure 2.2: The tuples in the representative $\{t_1, t_5, t_6, t_9\}$ partition the entire function space into convex regions. We have obtained this updated representative after completion of one iteration of the *URM* algorithm with $\{t_1, t_5, t_7, t_9\}$ as the initial set. This change in the representative reduces the max regret ratio score to 0.0444.

finds the best tuple in the entire dataset that has the minimum value of max regret ratio the functions in R_i. After the first iteration, the representative changes to $\{t_1, t_5, t_6, t_9\}$. This new representative creates a different partitioning of the function space, shown in Fig. 2.2. We report the max regret ratio scores of this iteration in Table 2.2. As we can see, the new representative has a lower score of max regret ratio, which is due to the convergence property of the algorithm, discussed in § 2.3.3. The algorithm then calls the oracle *MaxO* to find the best tuples in the dataset for the new regions and continues this process until convergence.

Similar illustration for the initial representative $\{t_2, t_8, t_9, t_{10}\}$ is shown below in Table 2.3 and Figures 2.3, 2.4, 2.5, 2.6 and 2.7.

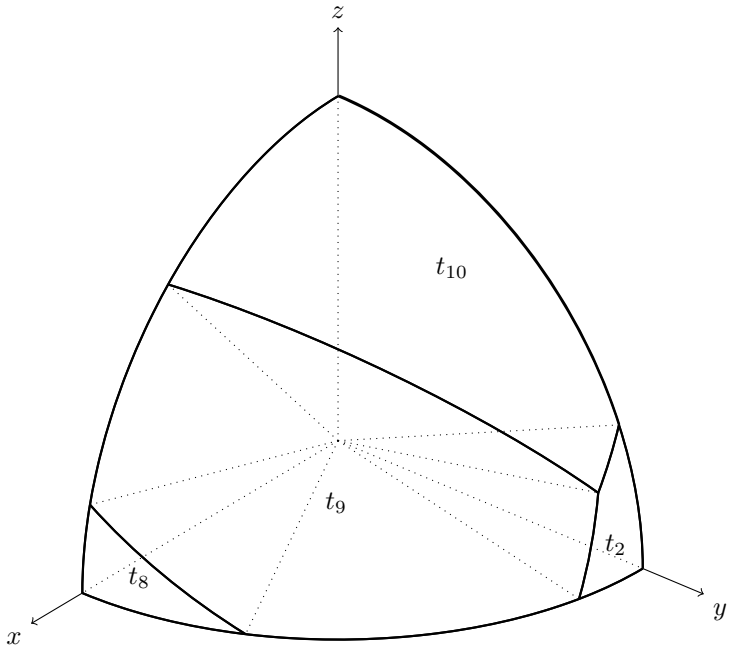


Figure 2.3: The tuples in the representative $\{t_2, t_8, t_9, t_{10}\}$ partition the entire function space into convex regions. The maximum regret for the representative is 0.0989.

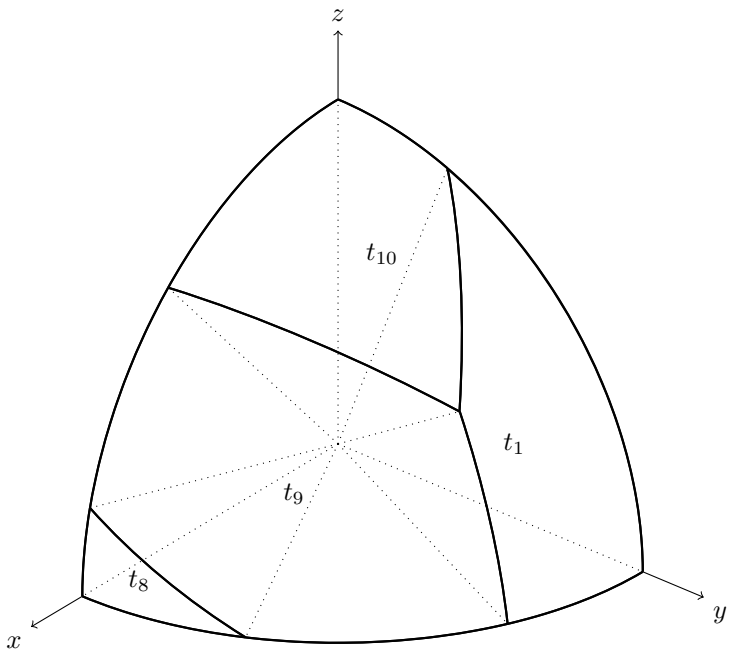


Figure 2.4: In the first iteration, the representative is updated to $\{t_1, t_8, t_9, t_{10}\}$ with a maximum regret of 0.0989.

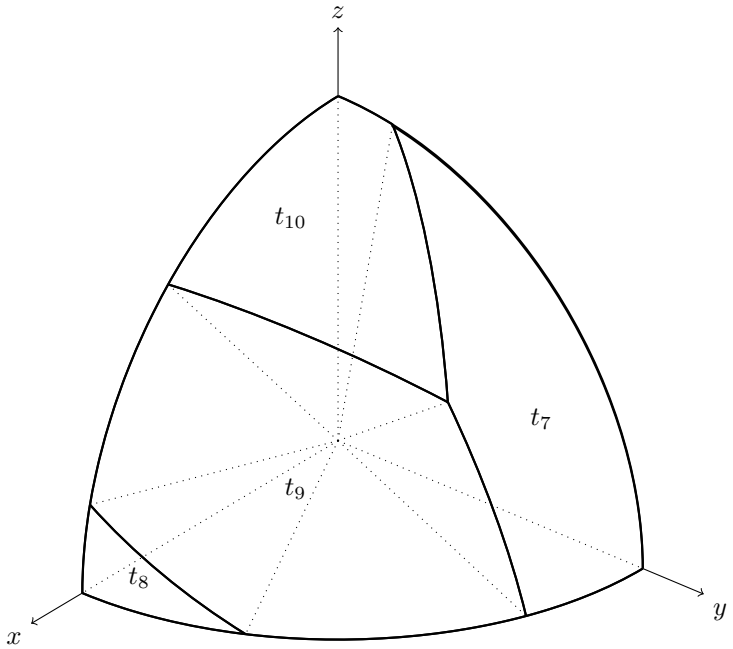


Figure 2.5: In the second iteration, the representative is updated to $\{t_7, t_8, t_9, t_{10}\}$ with a maximum regret of 0.0989.

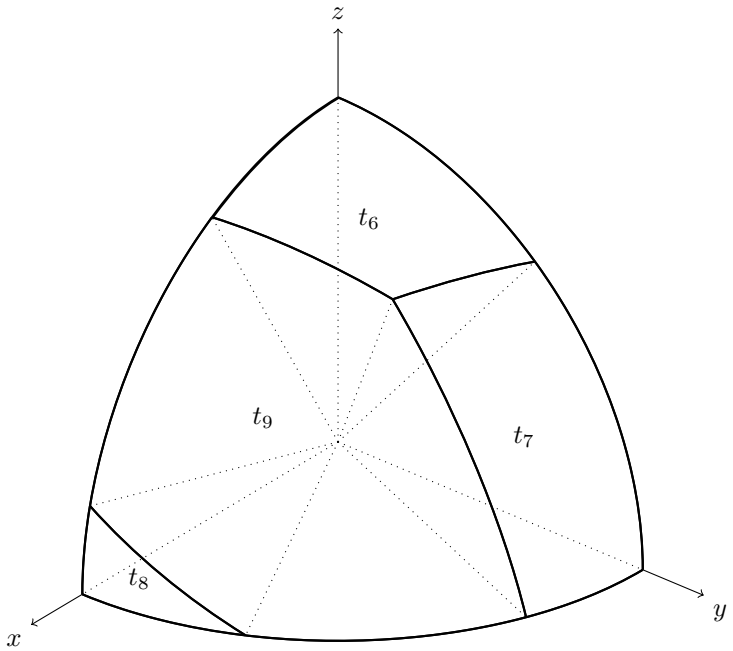


Figure 2.6: In the third iteration, the representative is updated to $\{t_7, t_8, t_9, t_6\}$ with a maximum regret of 0.0430.

Initial Representative		Representative after one Iteration	
Tuple	**RR**	**Tuple**	**RR**
t_1	0.0	t_1	0.0103
t_5	0.0223	t_5	0.0177
t_7	0.1208	t_6	0.0230
t_9	0.0938	t_9	0.0444
	0.1208		**0.0444**

Table 2.2: An iteration of the *URM* algorithm corresponding to the max regret ratio problem for example 1. Regret ratio for each of the tuples in representative is shown. The last row gives the max regret ratio score for the representatives.

Initial		Iteration 1		Iteration 2		Iteration 3		Iteration 4	
Tuple	**RR**	**Tuple**	**RR**	**Tuple**	**RR**	**Tuple**	**RR**	**Tuple**	**RR**
t_2	0.0969	t_1	0.0709	t_7	0.0840	t_7	0.0430	t_1	0.0378
t_8	0	t_8	0	t_8	0	t_8	0	t_8	0
t_9	0.0869	t_9	0.0378	t_9	0.0430	t_9	0.0430	t_9	0.0378
t_{10}	0.0989	t_{10}	0.0989	t_{10}	0.0989	t_6	0.0231	t_6	0.0231
	0.0989		**0.0989**		**0.0989**		**0.0430**		**0.0378**

Table 2.3: An iteration of the *URM* algorithm corresponding to the max regret ratio problem for running example Regret ratio for each of the tuples in representative is shown. The last row gives the max regret ratio score for the representatives.

2.4 Max Regret Ratio ORACLE

So far in this chapter, we discussed the regret-minimising problem in general, assuming the existence of an oracle for computing the regret. In this section we focus on the original (and dominant) measure of regret-ratio: max regret-ratio [12]. Given a database $\mathbb{D}$ and a region of ranking functions R, the oracle *MaxRROrc* finds the tuple that has the least maximum regret ratio score for all functions in R. The region of functions R is formulated as the intersection of half spaces, $R = \{H_1, .., H_{k-1}\}$. The objective is to design an efficient oracle for this case.

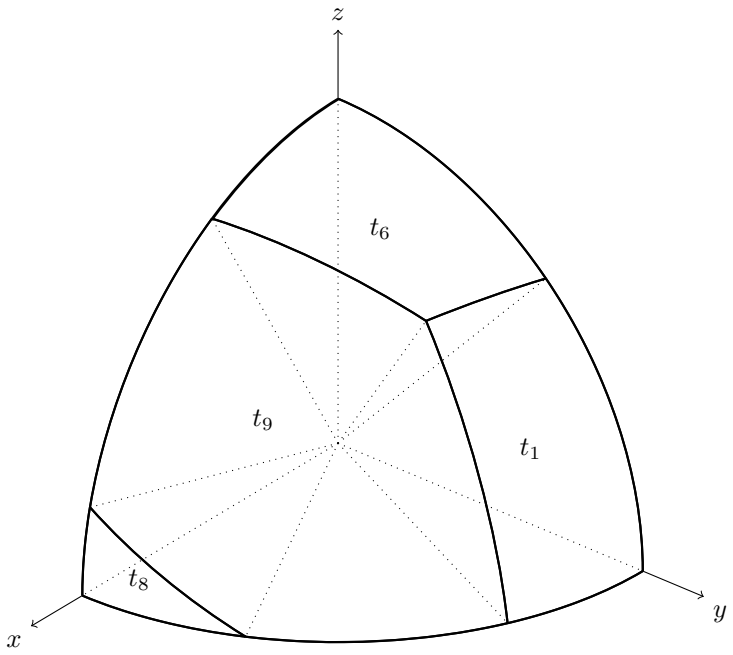


Figure 2.7: In the final iteration, the algorithm converges with the local optimum representative $\{t_1, t_8, t_9, t_6\}$ with a maximum regret of 0.0378.

We first model the problem into a weighted directed complete graph, and use it for calculating the max regret ratio. Then, we propose three optimisation techniques to make the oracle efficient and scalable.

2.4.1 Graph Transformation

In order to find the tuple with the least max regret ratio in a region R, we transform the problem into a graph exploration instance. Consider a weighted directed complete graph G, as illustrated in Figure 2.8, with n nodes and $n(n-1)$ edges such that:

- Every tuple $t \in \mathbb{D}$ translates to the node t in G.
- The weight $w_{t \to t'}$ of an edge $t \to t'$ (from node t to node t') is equal to the max regret ratio of replacing t' with t in the region R.

In order to compute the weight of an edge $t \to t'$ we use the (fixed size – independent of n) linear programming (LP) shown in Equation 2.11:

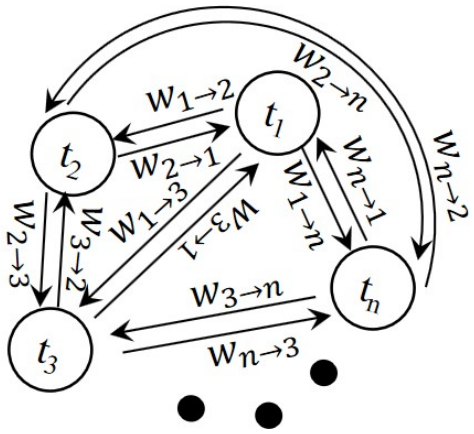


Figure 2.8: Graph transformation for max regret ratio oracle

$$\max \quad r \tag{2.11}$$

$$s.t. \qquad r \geq 0$$

$$\sum_{j=1}^{d} w_j \times t'[j] = 1$$

$$\sum_{j=1}^{d} w_j \times (t'[j] - t[j]) \geq r$$

$$\forall 0 < \ell < k :$$

$$\sum_{j=1}^{d} H_\ell[j] \times w_j \geq 0$$

Having the graph G constructed, the max regret ratio of assigning a tuple t as the representative (removing all other tuples $t' \in \mathbb{D}\backslash\{t\}$) of the region R, is the maximum weight of its outgoing edges. This is proved in Theorem 2.4.1.

Theorem 2.4.1. *Given the graph G of a database $\mathbb{D}$, a space of raking functions R the max regret ratio of a tuple $t \in \mathbb{D}$ in R is:*

$$RR(t, \mathbb{D}, R) = \max_{\forall t' \in \mathbb{D}\backslash\{t\}} w_{t\to t'}$$

Proof. Let $f' \in R$ be the ranking function for which the tuple t has the maximum regret ratio score. Let $r'_{f'}$ be the maximum regret ratio score and tuple $t_{f'}$ be tuple which has the maximum score from the database $\mathbb{D}$ for the function f'. By definition, the equation for maximum regret ratio $r'_{f'}$ is

$$r_{f'} = \frac{f'(t_{f'}) - f'(t)}{f'(t_{f'})}$$

An important observation that we will use in this proof is that the equation for $r_{f'}$ is dependent only on the tuple $t_{f'}$ from the database.

The computation of edge weight $w_{t \to t'}$ can be formulated as,

$$w_{t \to t'} = \sup_{f \in R} \frac{f(t') - f(t)}{f(t')}$$

As we compute the edge weights between tuple t and all the other tuples from the database D, the maximum value is computed when the comparison between tuples t and $t_{f'}$. Hence, $RR(t, \mathbb{D}, R)$ is equal to $r_{f'}$. □

The representative tuple for a region of functions R is the node that has the minimum max-weight over its outgoing edges. Therefore, after constructing the graph G, the oracle can make a pass over the graph and find the representative node to assign to the region.

Given that the LPs have a constant size, the construction of the graph G and finding the representative tuple for a region R based on it has the time complexity of $\mathcal{O}(n^2)$.

Even though the construction of graph G enables a polynomial algorithm for the max regret ratio oracle, it is still a quadratic algorithm which is not efficient and scalable in practice. Therefore, in the rest of this section we propose different approaches for making the oracle more efficient.

The idea is to find the representative tuple of the region without the complete construction of G. For instance, following the convergence property of the *URM* algorithm, we know that, at every iteration, the max regret ratio of the representative of a region R is not more than the one for the representative from the previous iteration. We can exploit this

property by using the regret ratio value of the representative for the region R_i as a threshold during the regret ratio value computation for the region R_i. That is, for any node in graph G, we ignore computing the weights of its outgoing edges, as soon as we find a edge that is not smaller than threshold for this region.

Following the idea of not computing the weights of all edges in G, next, we propose a threshold-based algorithm for the max regret ratio oracle.

2.4.2 Threshold-based Algorithm for Max Regret Ratio Oracle

Threshold-based algorithms are proven to be effective in practice and are the de-facto solution for many important problems such as top-k query processing [4, 22]. When the objective value is minimisation, the idea is to sort the tuples based on a lower bound on their objective values, ascending. Then starting from the top of the list, while maintaining a threshold, we continue processing the tuples in the sorted order until the lower bound value of the remaining tuples in the list is larger than the current threshold. The algorithm can stop then, as the objective value of the remaining tuples cannot be less that the best known threshold.

We apply a similar strategy here. But, first, we need to find the lower bounds on the max regret ratio of the tuples in region R. We apply two strategies for finding the lower bound: (i) function sampling and (ii) edge sampling, explained in § 2.4.2.1 and § 2.4.2.2, respectively. The objective is to construct a sorted vector $\mathcal{V}$ based on the lower bound values on the max regret ratio of the tuples.

2.4.2.1 Lower bound based on function sampling

The first strategy is to use function sampling for finding the lower bounds. We use the existing work [7] for sampling unbiased functions from the region R. Using a set of N IID function samples drawn from R, we construct a n by N table $\mathcal{T}$ that every row in

tid	f_1	$\cdots$	f_j	$\cdots$	f_N
t_1					
$\vdots$					
t_i			$rr_{f_j}(t_i)$		
$\vdots$					
t_n					

Figure 2.9: Illustration of table $\mathcal{T}$

it is a tuple and every column is one of the sampled functions. Every cell $\mathcal{T}[i, j]$ is the regret ratio of the tuple t_i on the sampled function f_j (Figure 2.9). In order to identify the cell values in $\mathcal{T}$, we first make a pass over the matrix and fill every cell $\mathcal{T}[i, j]$ with the value of $f_j(t_i)$. Also, for every column j, we keep track of its maximum value max_j. After finishing the first pass over the matrix, we do a second pass replacing each cell value $\mathcal{T}[i, j]$ by $(max_j - \mathcal{T}[i, j])/max_j$.

After the table $\mathcal{T}$ is constructed, the lower bound on the max regret ratio of each tuple t_i is the max value on row $\mathcal{T}[i]$. That is,

$$lower_{RR}(t_i, R) = \max_{j=1}^{N} \mathcal{T}[i, j] \tag{2.12}$$

Algorithm 3 uses the above idea and returns a sorted vector of tuples based on the lower bound estimation of their max regret ratio. Making two passes over $\mathcal{T}$ and then sorting the vector $\mathcal{V}$, Algorithm 3 is in $\mathcal{O}(n(N + \log n))$. Note that, considering a fixed sampling budget, the algorithm is linearithmic.

2.4.2.2 Lower bound based on edge sampling

Function sampling works well for fat regions i.e. regions which have a large volume where sampling from these regions is easy. But if the space of function space is a thin region then function sampling from it would end up being costly. We propose weighted sampling of edges of G for these cases.

Algorithm 3 SortedLB$_{FS}$

Input: Database $\mathbb{D}$, Set of Function Samples F

Output: Sorted vector of tuples based on the lower bound of their max regret ratio

for $j \leftarrow 1$ to $|F|$ **do**
  $max_j \leftarrow 0$
  for $i \leftarrow 1$ to n **do**
    $\mathcal{T}[i,j] \leftarrow F[j](t_i)$
    if $\mathcal{T}[i,j] > max_j$ **then** $max_j \leftarrow \mathcal{T}[i,j]$
  end for
  for $i \leftarrow 1$ to n **do**
    $\mathcal{T}[i,j] \leftarrow \frac{max_j - \mathcal{T}[i,j]}{max_j}$
  end for
end for
for $i \leftarrow 1$ to n **do**
  $\mathcal{V}[i] \leftarrow (i, \max_{j=1}^{N} \mathcal{T}[i,j])$
end for
Sort $\mathcal{V}$ on second column
return $\mathcal{V}$

Recall that the max regret ratio of a tuple t_i in a region R is the max of the weights of its outgoing edges in G (c.f. Theorem 2.4.1). A loose lower bound on the max regret ratio value can be obtained by uniformly sampling a few edges from the graph and computing their weights. To make it more effective, we use a weighted sampling of the edges to obtain reasonably tight lower bounds. In order to obtain the weights for the sampling process, we start with a set of ranking functions, chosen within the region R. To obtain the functions, a linear program is used to find the Chebyshev centre [23], which is the centre of largest

inscribed hyper sphere inside of R. Using a normal distribution, we sample a few functions and transform these functions to lie on the surface of the hyper sphere described by the Chebyshev centre. We use the summation of the scores of the tuples as a guidance for the weighted sampling. The idea is that the tuples with the higher scores are more likely of being representative of the region. Hence, we use the normalised vector of the score aggregates for the tuples, as the probability distribution for edge sampling.

Weighted sampling is performed using these weights to obtain a few edges of the graph. Computing the weights of these edges gives us a tighter lower bound value for max regret ratio. These lower bound values are then used in a similar manner to the threshold based algorithm § 2.4.2.1. Algorithm 4 shows the pseudo code for the weighted edge sampling algorithm.

Having the sorted list of tuples $\mathcal{V}$, we are now ready to design our threshold-based algorithm (Algorithm 5). Starting from the first tuple in the list, the algorithm computed the weights for the outgoing edges the current tuple in graph G (Equation 2.11). The max regret ratio of the current tuple is the max of its outgoing edges. While making a pass over $\mathcal{V}$ and computing the max regret ratio of the tuples, the algorithm keeps track of the best known solution (the least value of max regret ratio) as the threshold, and stops as soon as the lower bound values of the remaining tuples are higher than threshold.

2.5 Average Regret Ratio ORACLE

In this section, we shift our focus to a different measure of regret ratio, namely, average regret-ratio (ARR). An exact solution for the ARR computation oracle requires to partition the region R into the "maxima sub-regions" such that a specific tuple t_i is the maxima for each and every of the functions in each sub-region. Let $T_M \subseteq \mathbb{D}$ be the set of tuples that have the maximum score for at least one function in R. Also let $n' \leq n$ be

Algorithm 4 SortedLB$_{ES}$

Input: Database $\mathbb{D}$, Number of samples N, Number of edge samples N_{edge}, Convex region of functions R

Output: Sorted vector of tuples based on the lower bound of their max regret ratio

$center, radius \leftarrow$ Compute Chebyshev Center for region R

$F \leftarrow Sample(N)$ `// draw N samples`

for $j \leftarrow 1$ to $|F|$ **do**

  for $i \leftarrow 1$ to n **do**

    $\mathcal{P}[i] \leftarrow \mathcal{P}[i] + F[j](t_i)$

  end for

  $total \leftarrow total + \mathcal{P}[i]$

end for

for $i \leftarrow 1$ to n **do**

  $\mathcal{P}[i] \leftarrow \frac{\mathcal{P}[i]}{total}$ // `Normalize` $\mathcal{P}$

end for

for $i \leftarrow 1$ to n **do**

`//` `draw` N_{edge} `samples from distribution` $\mathcal{P}$

  $E \leftarrow Sample(\mathcal{P}, N_{edge})$

  for $j \leftarrow 1$ to N_{edge} **do**

    $w \leftarrow max(w,$ compute $w_{i \rightarrow E[j]}$ based on Eq. 2.11$)$

  end for

  $\mathcal{V}[i] \leftarrow (i, w)$

end for

Sort $\mathcal{V}$ on second column

return $\mathcal{V}$

Algorithm 5 MaxO

Input: Database $\mathbb{D}$, Convex region of functions R, Sampling budget N, representative t, threshold τ

Output: The representative tuple for R

$F \leftarrow$ Sample(R,N) `// draw N samples from R`
if F is not empty **then** $\mathcal{V} \leftarrow$ SortedLB$_{FS}$($\mathbb{D}$, F)
else $\mathcal{V} \leftarrow$ SortedLB$_{ES}$($\mathbb{D}$, R, N)
for $i \leftarrow 1$ to n **do**
    if $\mathcal{V}[i,2] \geq \tau$ **then break**
    $rr \leftarrow 0$
    for $j \leftarrow 1$ to n where $j \neq \mathcal{V}[i,1]$ **do**
        $w \leftarrow$ compute $w_{\mathcal{V}[i,1] \rightarrow j}$ based on Eq. 2.11
        if $w > rr$ **then** $rr \leftarrow w$
        if $rr \geq \tau$ **then continue**
    end for
    if $rr < \tau$ **then**
        $\tau \leftarrow rr$; $t \leftarrow \mathcal{V}[i,1]$
    end if
end for
return t

the size of T_M. Similar to Definition 2.3.2, the set of functions that each tuple $t \in T_M$ is the maxima is a convex region, defined as the intersection of $n' - 1 = O(n)$ half-spaces. For every function f in $R_{t'}$ (the maxima sub-region of a tuple $t' \in T_M$), the regret ratio of a tuple $t \in \mathbb{D}$ is $(f(t') - f(t))/f(t')$. This, in the end, provides an exact solution for computing the ARR of a tuple in a region R. However, requires to compute the volume

under the regret ratio curves across the sub-regions. Unfortunately, even though the total number of sub-regions is in $O(n)$ and each sub-region is defined as the intersection of linear number of half spaces, the computation of ARR within each of the sub-regions is not computationally feasible in higher dimensions. That is because, as proven by Dyer et al. [18], the exact computation of the volume of a convex shape described as the intersection of linear number of half-spaces is #P-hard.

Fortunately, although exact volume computation is usually costly, Monte-carlo methods [24] combined with tail inequalities [25] provide strong estimation methods for the problem. Following this, Zeighami et. al [2] have shown that the ARR value can be approximated using N samples of ranking functions with an error bound of ϵ and a confidence of $1 - \sigma$, where the relation between N, ϵ, and σ is shown in the following equation.

$$\epsilon = \sqrt{\frac{3 \ln \frac{1}{\sigma}}{N}} \tag{2.13}$$

Essentially, what [2] shows is that *discretizing* the continuous function space to a set of N uniform function samples, and using the samples for finding the maxima representatives guarantee an error ϵ, with the confidence interval of $1 - \sigma$, as specified in Equation 2.13. We follow this in the design of the ARR oracle. That is, to use the discrete set of N uniform function samples for finding the representative tuples.

Consider a database $\mathbb{D}$ and a set of representative tuples S and their regions. For each of the regions, we want to find the tuple $t \in \mathbb{D}$ for which the ARR score is the lowest. For any region R_i, we select the functions which lie inside R_i. We use this set of ranking functions to estimate the ARR score of each of the tuples in the database. Next, for each region, the tuple with the lowest ARR score replaces the previous representative of that region. The algorithm is given in 6.

Algorithm 6 AvgO

Input: Database $\mathbb{D}$, Convex region of functions R, Function samples F in R

Output: The representative tuple for R

for $j \leftarrow 1$ to $|F|$ **do** $max[j] \leftarrow 0$ `// max score for each function`
for $i \leftarrow 1$ to n **do**
    for $j \leftarrow 1$ to $|F|$ **do**
        $scores[i, j] \leftarrow F[j](t_i)$
        if $max[j] < scores[i, j]$ **then** $max[j] \leftarrow scores[i, j]$
    end for
end for
$min \leftarrow \infty$
for $i \leftarrow 1$ to n **do**
    $arr[i] \leftarrow \sum_{j \leftarrow 1}^{|S|} \frac{max[j] - scores[i,j]}{max[j]}$
    if $min > arr[i]$ **then**
        $min \leftarrow arr[i]; t \leftarrow i$
    end if
end for
return t

2.6 Experiments

2.6.1 Experimental setup

Datasets: For evaluating our algorithms, we have used the following datasets. We generated two synthetic datasets - *Surface* and *Scaled* along the lines of *Sphere* and *SkyPoints* in [14].

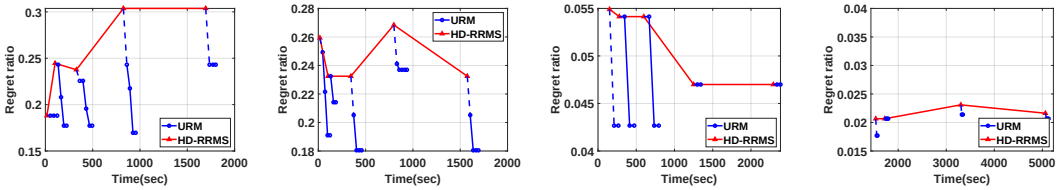
- (Real dataset) *Colors [26]:* This is one of the commonly used datasets for the evaluation of regret ratio and skyline problems [12, 14, 27]. In our experiments, we have used the

Color Histogram dataset. It contains 68,040 tuples, each being a color image. For every image, it contains 32 attributes, where each attribute is the density of a color in the entire image.

- (Synthetic) *Surface:* We generated the Surface dataset by uniformly sampling points on the surface of a unit hypersphere. Therefore, by construction, all the points in the Surface dataset belong to the skyline. The dataset contain 20,000 tuples, over 12 attribute, in range $[0, 1]$.
- (Synthetic) *Scaled:* For the Scaled dataset, we uniformly generated points inside a unit hypersphere. Since, in a high dimensional space, a large portion of the total volume of a hypersphere lies near the surface area, most of the points in the *Scaled* dataset is also present in the skyline. The scaled dataset contains a set of 20,000 tuples, each defined over 12 attributes in range $[0, 1]$.
- (Real dataset) *DOT [28]:* The flight on-time dataset is published by the US Department of Transportation(DOT). It records, for all flights conducted by the 14 US carriers in January 2015, attributes such as scheduled and actual departure time, taxiing time and other detailed delay metrics. The dataset consists of 457,013 tuples and 7 ordinal attributes.
- (Real dataset) *NBA [29]:* NBA dataset contains the points for the combination of player,team,season up to 2009. It contains 21,961 tuples and 17 ordinal attributes: *gp, minutes, pts, oreb, dreb, reb, asts, stl, blk, turnover, pf, fga, fgm, fta, ftm, tpa, tpm.*

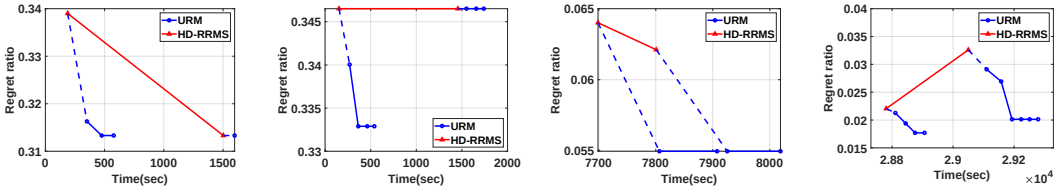
Hardware and Platform: All our experiments were performed on a Core-i7 machine running Ubuntu 16.04 with 64 GB of RAM. The algorithms were implemented in Python.

Evaluations: In order to asses the performance of our algorithm, we focus on two main criteria namely, efficiency and efficacy. Concretely, we evaluate based on the following metrics - (i) the quality of the results produced, i.e. the regret ratio measure of the result (ii) the amount of time taken by the algorithm.



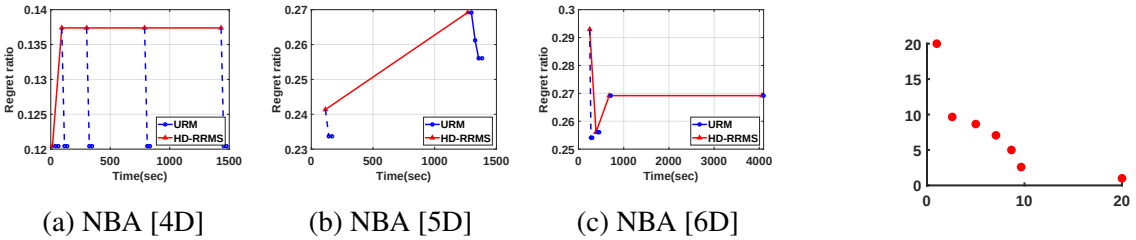
(a) Surface [4D] (b) Scaled [4D] (c) Colors [4D] (d) DOT [4D]

Figure 2.10: Maximum regret ratio when *URM* uses the compact representative produced by HD-RRMS as the initial set [4D, K=5].



(a) Surface [5D] (b) Scaled [5D] (c) Colours [5D] (d) DOT [5D]

Figure 2.11: Maximum regret ratio when *URM* uses the compact representative produced by HD-RRMS as the initial set [5D, K=5].



(a) NBA [4D] (b) NBA [5D] (c) NBA [6D]

Figure 2.12: Maximum regret ratio when *URM* uses the compact representative produced by HD-RRMS as the initial set for NBA dataset.

Figure 2.13: 2D Dataset Example

Algorithms Evaluated: For the max regret ratio representative problem, we have used HD-RRMS as the baseline [13], one of the recent and advanced algorithms for finding the max regret ratio representative that guarantees a tunable additive approximation. To do so, it discretizes the ranking function space and models the problem as a discrete matrix min-max problem. A combination of binary search technique and transforming the problem into fixed-size set covers are applied for solving the problem. To make the problem practical,

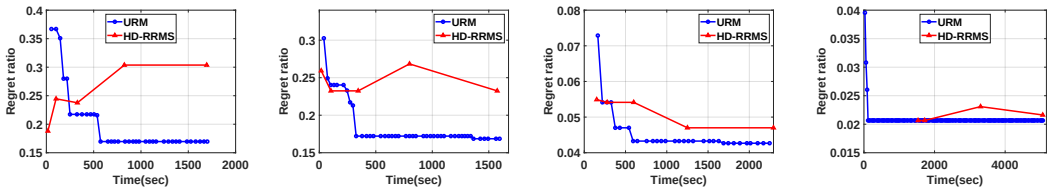
the greedy approximation algorithm is used for the set cover instances. As we shall later show in this section, the extra approximation induced by the greedy set cover in the HD-RRMS algorithm shows up in our results and can be seen in the HD-RRMS curve when the regret ratio does not reduce with time in some cases. For the average regret ratio case, we have implemented the GREEDY-SHRINK algorithm [2]. This algorithm starts with the entire database as its representative and iteratively removes the tuple for which the increase in the average regret ratio is the least. This process is continued until k tuples are left. In addition to GREEDY-SHRINK, we also compute the global minima, using N samples from the function space, drawn based on Eq. 2.13 with $\epsilon = 0.01$ and a confidence as 0.999. We filtered the dataset to only the skyline points. Using the N samples, a table containing the regret-ratio values for the filtered tuples is generated, similar to table 2.9. For every combination of k tuples as a representative, we compute the average regret-ratio value. It is important to note here that existing algorithms are all approximation algorithms and do not exhibit the *anytime* property. As a result, *URM* is not directly comparable to the existing work. In our experiments we focus on specific properties of our algorithm.

We also compare our algorithm with several skyline reducing algorithms (which are discussed in more detail in § 2.7). We have implemented the KRSPGREEDY algorithm from [30], EIQUE algorithm from [31], NAIVEGREEDY algorithm from [32], ϵ-*ADR* greedy algorithm from [33] and SKYCOVER algorithm from [34].

2.6.2 Summary of experimental results

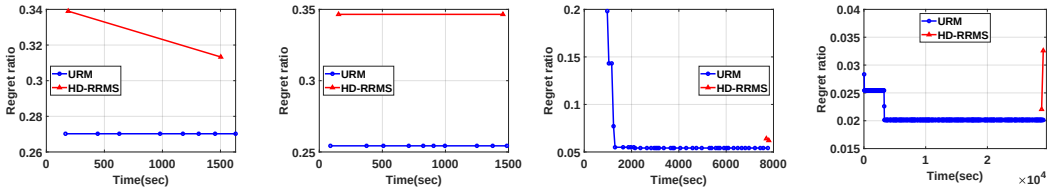
At a high level, the experiments verified the quality and efficiency of our proposal. We consider regret ratio score as the measure of quality. For the max regret ratio representative problem, *URM* improves the quality of the representatives provided by HD-RRMS when used as a starting point. When provided with a time budget *URM* qualitatively outperforms HD-RRMS algorithm. In case of the average regret ratio representative problem, *URM* qual-

itatively outperforms GREEDY-SHRINK when provided with the same time budget. Our experiments show that *URM* converges to the local minima very fast which increases the chance of discovering the global optima. In addition, the experiments demonstrate some of the useful properties of our approach, namely, (a) *anytime* property - even if the execution of the algorithm is terminated at any point of time, the algorithm will still have a representative (b) provision for getting better results - by restarting with a different set of initial points, *URM* can achieve better representatives.



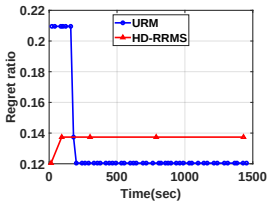
(a) Surface [4D] (b) Scaled [4D] (c) Colors [4D] (d) DOT [4D]

Figure 2.14: Maximum regret ratio when both *URM* and HD-RRMS uses a fixed time budget [4D, K=5].

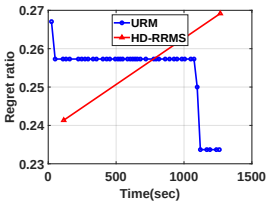


(a) Surface [5D] (b) Scaled [5D] (c) Colors [5D] (d) DOT [5D]

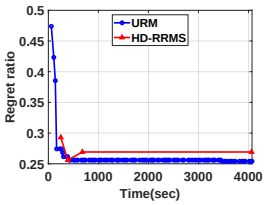
Figure 2.15: Maximum regret ratio when both *URM* and HD-RRMS uses a fixed time budget [5D, K=5].



(a) NBA [4D]



(b) NBA [5D]



(c) NBA [6D]

Figure 2.16: Maximum regret ratio when both *URM* and HD-RRMS uses a fixed time budget for NBA dataset.

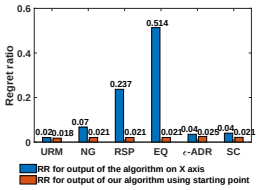
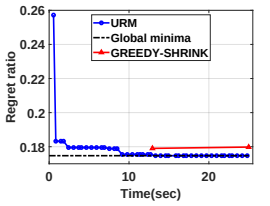
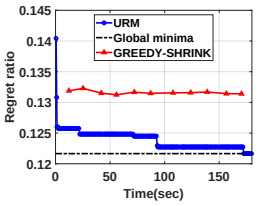


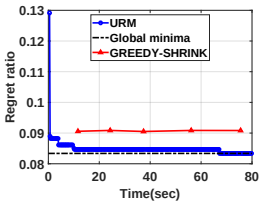
Figure 2.17: Comparing skyline reducing alg. with *URM* for DOT dataset with 4 dimensions



(a) Colors, Set size = 4 [9D]



(b) Colors, Set size = 5 [9D]



(c) Colors, Set size = 6 [9D]

Figure 2.18: Average regret ratio, we are comparing our results against the global best representatives [9D].

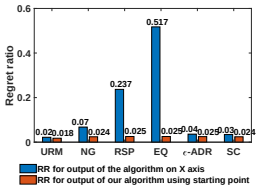


Figure 2.19: Comparing skyline reducing alg. with *URM* for DOT dataset with 5 dimensions

2.6.3 Results for Max Regret-Ratio

Results when initial representative is given: Similar to other clustering-inspired algorithms, the quality of the results produced by the *URM* algorithm depends on the initial set of tuples. Often feeding the output of an existing approximation algorithm as the input to these iterative algorithms yield good results. In this set of experiments, we have used the compact representative returned by HD-RRMS as the initial set of tuples to the *URM* algorithm. To be fair in the assessment, while creating the plots for *URM*, we have taken the time to generate the initial points into consideration. Concretely, we have added the running time of HD-RRMS to the running time of *URM* when creating the plots for the *URM* algorithm. We run the experiments on the *Surface*, *Colors*, *NBA*, *DOT*,and the *Scaled*

datasets. For *Surface*, *Colors*, *DOT* and *Scaled* datasets we run *URM* for 4 and 5 dimensions with 20k points each. We use the same configuration used in [13] for *NBA* and *DOT* datasets. While we use 20k tuples with 4, 5 and 6 dimensions for *NBA* dataset, we use 400k tuples with 4 and 5 dimensions for *DOT* dataset. Fig. 2.10 and 2.11 plot the obtained max regret ratio to the time taken by each algorithm for *Surface*, *Colors*, *Scaled* and the *DOT* datasets for dimensions 4 and 5, respectively. Fig. 2.12 compares the *URM* and HD-RRMS algorithms for the *NBA* datasets.

To get different results from HD-RRMS, we used different values of γ for discretization. Every red point shows the experimental result of one individual run of the HD-RRMS algorithm. While HD-RRMS is expected to provide better results as γ increased, in several settings, one can see an increase in the max regret ratio of the generated output. The reason is that HD-RRMS uses the greedy approach for solving the (theoretically fixed-size) set cover instances. This adds one more level of approximation to the algorithm which, in the end, results in the non-decreasing behavior of it in some cases.

In the figures, each red point is connected to a blue point by a dotted line, which represents feeding the HD-RRMS algorithm's output to the *URM* algorithm. The string of blue points connected by the solid blue line show the performance of the *URM* algorithm over the subsequent iterations. First, our experiments show that *URM* quickly reaches the local minima of regret ratio in this setting. Also, in most of the experiments, *URM* improves the results of the HD-RRMS algorithm. The ones where *URM* could not improve the result of HD-RRMS were the ones that HD-RRMS had, by chance, discovered a local optima. Apart from the performance of the algorithm, it is interesting to see the existence of multiple local optima and the impact of the starting point on the optima discovered. For instance in Fig. 2.10b, each setting discovered a different local optima. The fast convergence of *URM* enables multiple runs of the algorithm with multiple starting points, which increases the chance of discovering the global optima.

Results on a fixed time budget: While *URM* is an *anytime* algorithm, HD-RRMS is not. That is, HD-RRMS returns a representative only after it is finished. Therefore, in order to make a fair assessment, we run the HD-RRMS algorithm with different values of γ and allocate the exact same amount of time taken by the HD-RRMS algorithm as the time budget for our *URM* algorithm. We run our experiments with *Surface*, *Colors*, *NBA*, *DOT*,and the *Scaled* datasets. This set of experiments demonstrate two important properties of the *URM* algorithm. First, as we can see from Fig. 2.14, 2.15 and 2.16 the regret ratio scores reduce monotonically. This is due to the convergence property of the algorithm, proved in § 2.3.3. In addition, the *URM* algorithm produces reasonably good regret ratio scores even when the allocated time budget is small. This means that a user can still get a reasonably good set of representatives even if she terminates the algorithm before it finishes. Another important result is that *URM* actually allows the user to find a better representative. With a fixed time budget, we restart the algorithm with a random set of starting points and repeat this process until our time budget expires. Essentially, with subsequent repetition of the *URM* algorithm, we find a different set of compact representatives with a better score of max regret-ratio. Another important property that our algorithm exhibits is the any time property. As *URM* is an iterative hill climbing algorithm, the best representative among the ones it has already visited can be consumed by the user even before the end of the time budget. This can be seen in Fig. 2.14 and Fig. 2.15 where if we were to stop the algorithm at any point of time we would get a representative marked by the blue circles. First, looking at the figures, one can see the monotonically decreasing behaviour of *URM*, compared to HD-RRMS. Also, HD-RRMS needs to finish at least once or will not provide any output. For instance, in Fig. 2.15(c), HD-RRMS did not provide any result for a time budget less than 8000 seconds. In contrast, having the anytime property, *URM* has an output to offer at any point of time.

Proof of concept by an example: Using the NBA dataset, we highlight a concrete example. The dataset contains performance records of different Basketball players across different seasons. Consider the case where the decision criteria are Points, Rebounds, Assists and Steals. More than 60 tuples belonged to the skyline. Offering such a large set to the user is overwhelming. Instead, using the *URM* algorithm while setting the output size to 6 we find the set { Wilt Chamberlain - 1967, Don Buse -1975, Nate Archibald - 1972, Michael Jordan - 1987, Wilt Chamberlain - 1961, John Stockton - 1988} with maximum regret-ratio of 0.058. That is, the user can make selection between these 6 tuples, yet be sure that the score (quality) of its selection is not more than 5.8% percent worse than the optimal choice. For example, for the ranking function Assists+Steals, the optimal tuple is *John Stockton 1990* with the score of 1398. The tuple *John Stockton - 1988* in the representative set has the score of 1381 which shows how close to optimal it is.

2.6.4 Results for Avg Regret-Ratio

Comparison against the global minima: *URM* is an iterative algorithm where the regret ratio score monotonically decreases over time. That is, it produces better results as time passes. In contrast, the GREEDY-SHRINK algorithm generates results only at program termination. For a fair comparison between the two algorithms, we run GREEDY-SHRINK with different parameter values of ϵ and σ. We also demonstrate the ability of our algorithm to find the global minima given sufficient time. Therefore, in addition to the GREEDY-SHRINK algorithm we compare against the global minima, the computation of which is described in § 2.6.1. Concretely, we let the *URM* algorithm finish multiple iterations and recorded the regret ratio at the end of each iteration. Finally, we have compared the progression of these scores against the global minima along with the results from GREEDY-SHRINK. For this set of experiments, we have used 20,000 points from the *Colors* dataset, in a 9 dimensional space. As discussed in § 2.5, instead of computing the exact

volumes, we approximate the average regret ratio score using N samples with an ϵ of 0.01 and a confidence of 0.999 as described in 2.13. We present the results of our experiments in Fig. 2.18. The black dotted line indicates the global minima while the red triangles show the output from the GREEDY-SHRINK algorithm. The *URM* algorithm outperforms the GREEDY-SHRINK algorithm for different output sizes of k. In all cases *URM* finds a representative with small value of average regret ratio in a small amount of time. In fact, our experiments show that it is possible to reach the global minima of average regret ratio in a short time.

2.6.5 Results for skyline reducing algorithms

To compare our algorithm with skyline reducing algorithms, we have implemented the algorithms KRSPGREEDY [30], ϵ-*ADR* [33], EIQUE [31], NAIVEGREEDY [32], and SKYCOVER [34]. NAIVEGREEDY takes the skyline and a starting tuple as input to output a representative. To be fair in comparison, we passed every point from the skyline as a starting point and chose the set with least regret ratio. ϵ-*ADR* and SKYCOVER take an error parameter ϵ as the input and output the smallest set size satisfying ϵ-*ADR* optimisation criteria. To find a set of size *k* or smaller, we find a large ϵ which satisfies the set size criteria and apply a binary search to get the smallest ϵ satisfying the size requirement. The results for the DOT dataset with dimensions 4 and 5 are presented in Fig. 2.17 and Fig. 2.19. We denote KRSPGREEDY as *RSP*, NAIVEGREEDY as *NG*, EIQUE as *EQ*, ϵ-*ADR* greedy as ϵ-*ADR* and SKYCOVER as *SC* in the plots. The blue bar shows the regret ratio of the output of the skyline reducing algorithms, with the name of the algorithm on the *X* axis. The orange bars show the regret ratio of the output of our (*URM*) algorithm when initialised with the corresponding algorithm's output. As expected *URM* outperforms other algorithms.

2.7 Related Work

Over the past few decades, a major amount of research has focused on the generation of a representative of the dataset to assist users with multi-criteria decision making. Most of these published works can be grouped into skyline discovery, skyline reduction and *regret* based compact representative computations. In this section, in addition to discussing the related works in these three categories we also provide a brief summary of the clustering techniques that have inspired the *URM* algorithm.

Regret minimizing representatives: As an effective solution to the problem of finding compact representatives, Nanongkai et al. [12] introduced regret-ratio minimising representative. The notion of regret introduced in the paper deals with the amount of dissatisfaction the user would express when she is shown the top item from the representative set instead of the top item from the dataset when provided with the user preference. Many different variations of the original regret-minimising representative problem have been studied since this paper [16, 35]. One specific variant, finding a representative set that minimises the maximum regret ratio, has been extensively studied by several researchers. Agarwal et al. [14] proved that this problem is NP-complete for dimensions larger than 2. Asudeh et al. [13] introduce function space discretization and the transformation of the problem to set-cover instances. [13,14] propose approximation algorithms with similar tunable approximation guarantees for the problem. Another interesting variant of this problem is finding the average regret ratio problem. It was introduced by Zeigami et al. [2, 15]. However, it is important to note that while others have studied and proposed solutions for single variants of the regret measures, we have proposed an unifying algorithm that can work with a class of regret measures.

Skyline and convex hull: In the pursuit of efficiency, finding a small set of representatives of the entire dataset has been of key interest in recent years. In the absence of user preferences, skyline [10, 36, 37] and convex hull based algorithms obtain a part of the dataset

which behave like representatives. While convex hull is more compact than the skyline, the computation of a convex hull is significantly more costly in large dimensions. Bentley et al have studied approximation algorithms with tight approximation ratios for convex hulls [38]. However, as we move to higher dimensions, most points in the dataset appear on the skyline or convex hull. As a result, even though these skyline/convex hull based algorithms are effective in lower dimensions, the large size of skyline and convex hull render them ineffective as representatives.

Skyline reducing algorithms: Existing works have studied the problem of reducing the size of the skyline in various scenarios [30–34, 39–41]. To the best of our knowledge, none, except the regret-minimising literature, incorporate user customisable functions for skyline reduction nor can be directly translated to the regret ratio problem. [30] and [39] reduce the skyline size by choosing a subset of size k of it which maximises the number of dominated tuples by this subset. [32] proposes distance-based skyline, a subset of size k that minimises the sum of the distances between the points and their closest representative. [41] and [31] find a subset of size k with maximum diversity. [42] propose the concept of skyline ordering, which is skyline-based partitioning of a given data set such that an order exists among the partitions. This ordering is then exploited to reduce the set size to k. We note that all of the aforementioned have a different objective function than regret-ratio and cannot provide any guarantee on how good it is for an arbitrary function.

Similarly, [33] and [34] propose a new measure, ϵ-*ADR* query, which chooses a subset of the skyline such that the tuples of the skyline when scaled by (1+ ϵ), dominate the rest of the skyline tuples. We illustrate the difference between the optimisation measure of ϵ-*ADR* query and regret ratio with an example. Consider a dataset with n points (in 2D), out of which (n-2) are equi-angularly placed on the surface of a circle with radius *10* and the two other points are $\langle 20, 1\rangle$ and $\langle 1, 20\rangle$. Note that all the points belong to the skyline. Figure 2.13 shows this example for $n = 7$. The optimal set for size 2 is (the convex

hull) $\{\langle 20, 1\rangle, \langle 1, 20\rangle\}$ with max regret ratio 0. On the other hand, ϵ-*ADR* query chooses a subset of the skyline such that the tuples of the skyline when scaled by (1+ ϵ), dominate the rest of the skyline tuples. As both $\langle 20, 1\rangle$ and $\langle 1, 20\rangle$ have one of their attributes set to *1*, the scaling required for these tuples to dominate the other tuples is large making them less attractive to the ϵ-*ADR* query. For the example of Figure 2.13, ϵ-*ADR* [33] returns $\{\langle 8.66, 5.0\rangle, \langle 5.0, 8.66\rangle\}$. Note that, instead of $\langle 20, 1\rangle$ and $\langle 1, 20\rangle$, we could add points $\langle X, 1\rangle$ and $\langle 1, Y\rangle$ with X and Y such that there exists a tuple $\langle a, b\rangle$ from the database which has the property $\frac{X}{a} < b$ and $\frac{Y}{b} < a$. For example, we can increase X and Y to *49* as this dataset contains the point $\langle 7.07, 7.07\rangle$ for which $\frac{49}{7.07} = 6.93 < 7.07$.

Clustering: Clustering have long been studied to partition data into similar groups. One of the major variants in this field is the K-MEDOID algorithm [17]. Like many other clustering algorithms, K-MEDOID is also an iterative algorithm that partitions the points into clusters by minimizing the distance between the points from the cluster center. However, the difference between the K-MEDOID algorithm and other clustering techniques is that K-MEDOID always chooses the cluster centroids from the existing data points. In the literature, there have been several variations of K-MEDOID that have explored several avenues of choosing the best centroid. Few of the most prominent are PAM, CLARA [43] and CLARANS [44].

CHAPTER 3

Fairness-Aware Range Queries for Selecting Unbiased Data

We are being constantly judged by automated decision systems that have been widely criticised for being discriminatory and unfair. Since an algorithm is only as good as the data it works with, biases in the data can significantly amplify unfairness issues. In this paper, we take initial steps towards integrating fairness conditions into database query processing and data management systems. Specifically, we focus on selection bias in range queries. We formally define the problem of fairness-aware range queries as obtaining a fair query which is most similar to the user's query. We propose a sub-linear time algorithm for single-predicate range queries and efficient algorithms for multi-predicate range queries. Our empirical evaluation on real and synthetic datasets confirms the effectiveness and efficiency of our proposal.

3.1 Introduction

In the era of big data and advanced computation models, we are all constantly being judged by the analysis, algorithmic outcomes, and AI models generated using data about us. Such analysis are valuable as they assist decision makers take wise and just actions. For example, the abundance of large amounts of data has enabled building extensive big data systems to fight COVID-19, such as controlling the spread of the disease, or in finding effective factors, decisions, and policies [45]. Similar examples can be found in almost all corners of human life including resource allocation and city policies, policing, judiciary system, college admission, credit scoring, breast cancer prediction, job interviewing, hir-

ing, and promotion, to name a few. In particular, let us consider the following as a running example:

Example 2. *(Part 1) Consider a company that would like to make a policy decision, targeted at its "profitable" employees. Following our real experiment in § 3.5.2, suppose the company has around 150K employees. Using salary as an indicator of how profitable an employee is, the business management office of the company considers the query* `SELECT * FROM EMP WHERE salary≥$65K`*, which includes around 18% of employees. Surveying this group, the company wants to develop some mechanisms to motivate and retain these employees.*

Looking at these analyses through the lens of fairness, algorithmic decisions look promising as they seem to eliminate human biases. However, *"an algorithm is only as good as the data it works with"* [46]. In fact, the use of data in all aforementioned applications have been highly criticised for being discriminatory, racist, sexist, and unfair [47,48]. Probably the main reason is that real-life social data is almost always "biased" [46]. Using biased data for algorithmic decisions create fairness dilemmas such as impossibility and inherent trade-offs of fairness [49–51]. Besides historical biases and false stereotypes reflected in data, other sources such as *selection bias* can amplify unfairness issues [46]. To highlight a real example, let us continue with Example 2:

EXAMPLE 1. *(Part 2) As we shall later elaborate in § 3.5.2, it turns out the company has more female employees than male. Still, due to the known historical discrimination [52], the selected group of employees contain noticeably more males. As a result, targeting this group for the analysis, the company will end up favouring the preferences of the male employees, which is unfair to female employees and will, in a feedback loop, result in losing more of the "profitable" female candidates.*

Fortunately, recently different computer science communities, such as machine learning and, in particular, data management, have taken fairness issues seriously. In past three

years alone, there have been many publications in related topics such as fairness-aware data repair, cleaning, and integration [53–56], data bias detection/resolution [57–63], and data/model annotation [64, 65], systems [66–68], ranking [69–72], crowdsourcing [73], as well as different keynotes [74, 75] and tutorials [47, 76, 77] dedicated to this topic in premier database conferences, that underscore this community's role in properly addressing this problem.

Despite extensive efforts within the database community, there is still a need to integrate fairness requirements with database systems. Existing work is limited to query formulation for achieving data coverage (minimum count on demographic (sub)groups) [78–80]. To our knowledge, this paper is *the first to integrate fairness (parity on counts) with (selection) query answering*. In particular, as our first attempt, we consider range queries and pay attention to the facts: (i) the conditions in the range query may be selected intuitively by the human user. For instance, in Example 2 the user could have chosen \$65K as the query bound because it was (roughly) a good choice that would make sense for them; (ii) considering the ethical obligations and consequences, the user might be interested in accepting a "similar enough" query to their initial choice, if it returns a "fair" outcome.

In Example 2, we note that the company could, for instance, in a post-query processing step, remove some male employees from the selected group, or it could add some females to the selected pool, even though they do not belong to the query result. While such fixes are technically easy, those are illegal in many jurisdictions [81], because those amount to *disparate-treatment* discrimination: "when the decisions an individual user receives change with changes to her sensitive attribute information" [82], [46, 69]. For instance, one cannot simply increase or decrease the grade of a student, because of their race or gender. Instead, they should design a "fair rubric" that is not discriminatory. Therefore, instead of practising disparate treatment, we propose to adjusting the range to find a range (similar to finding a rubric for grading) with a fair output.

Following the above argument, our system allows the user to specify the fairness and similarity constraints (in a declarative manner) along with the selection conditions, and we return an output range that satisfies these conditions. To further clarify this, let us continue with Example 2 in the following.

EXAMPLE 1. *(Part 3) Being aware of the historical discrimination, ethical obligations, and the potential negative impacts on the company, besides knowing that the choice of salary lower-bound has been fuzzy, the business management office would like to find a query whose output is similar enough to the initial query and the number of male employees returned is at most 1000 (around 5%) more than the females. Using our system, they can issue a SQL query to find such a set. As explained in § 3.5.2, our system found the most similar fair range as* `SELECT * FROM EMP WHERE $60.5K≤salary≤$152K`. *Its outcome is 75% similar to the initial range query, and satisfies the fairness requirement. Observing the high Jaccard similarity between these two sets, the company now has the option to use this for their analysis, to make sure they are not discriminating against their female employees, hence not losing their valuable candidates.*

Our system provides *an alternative* to the initial query provided by the user. This is useful since often the choice of filtering ranges is ad-hoc, hence our system helps the user responsibly tune their range. If the discovered range is not satisfactory to the user, they can change the fairness and similarity requirements and *explore different choices* until they select the final result in a responsible manner [69, 83].

Summary of contributions Initiating fairness-aware query answering, in this paper we tackle non-trivial technical challenges and propose proper algorithms to address them. In particular, we make the following contributions in this paper:

- We initiate research on integrating fairness conditions into database query processing and data management systems. While the specific problem investigated in our paper focuses

on fairness in range queries, we hope this work will spur further research in this important and emerging field.

- We study the problem of fairness-aware range queries. That is, finding the most similar fair range to a user-provided range query for the database $\mathbb{D}$. We propose using SQL for declaring the fairness-aware queries.
- For single-predicate (SP) range queries, we propose the algorithm *SPQA* with *sub-linear* query time. The algorithm uses an innovative linear-size index *Jump pointers*.
- We model the problem for multi-predicate (MP) range queries as the traversal over a graph where nodes represent different queries and there is an edge between two nodes if their outputs differ by one tuple. In particular, we propose *Best First Search MP (BFSMP)* algorithm that, starting from the input range, efficiently explores neighbouring nodes to find the most similar fair range.
- Inspired by the A* algorithm, we propose *Informed BFSMP* that improves *BFSMP*, using an upper-bound on the *Jaccard similarity* for effective graph exploration.
- We conduct comprehensive experiments to evaluate our SP and MP algorithms. Our results demonstrate the efficiency and efficacy of *SPQA* for SP queries. Similarly, *IBFSMP* performs well for MP queries.

3.2 Preliminaries

3.2.1 Database Model

Database We consider a relation $\mathbb{D}$ with *n* objects, t_1 to t_n. Each object in $\mathbb{D}$, consists of numeric attributes A_1 to A_d. We refer to the value of attribute $A_j \in \mathbb{R}$ of object t_i as $t_i[j]$. While the numeric attributes of the database $\mathbb{D}$ can be used for range queries, objects may also consist of categorical attributes. These categorical attributes are used for filtering the objects based on the user's criteria.

id	A_0	A_1	colour	id	A_0	A_1	colour
t_0	3.1	1.5	red	t_7	13	5.4	red
t_1	0.7	2.3	red	t_8	11.3	2.6	blue
t_2	8	0.65	blue	t_9	2.3	8.4	blue
t_3	10.9	1.5	red	t_{10}	5.6	4.7	red
t_4	4.4	8.7	blue	t_{11}	12.7	2.8	red
t_5	1.2	4.1	red	t_{12}	7	0.3	blue
t_6	6.2	6.3	blue	t_{13}	9.1	9.4	red

Table 3.1: A toy example database $\mathbb{D}$ with two attributes A_0 and A_1 and the sensitive attribute *colour*.

A toy example of a database can be seen in Table 3.1 with 14 objects t_0 to t_{13}. The attributes of this database are in the form of A_i, namely A_0, A_1. The database also includes the non-ordinal attribute colour used in the fairness model. We use this database for illustrating various techniques across the paper.

Range Query Given a database $\mathbb{D}$, a range query is made up of conjunction of range constraints placed on some/all of the attributes of $\mathbb{D}$. A range constraint $\{start, end\}$ (aka a range predicate) on an attribute A_i filters the objects such that the attribute A_i lies within the filter range($start \leq A_i \leq end$). For instance, Query 1 is a query with one range predicate on the toy example of Table 3.1 that returns the objects $\{t_4, t_6, t_{10}, t_{12}\}$:

QUERY 1: `select id from` $\mathbb{D}$ `where` $4 \leq A_0 \leq 7$

In this paper, we consider the conjunction of multiple range predicates using "and" operation.

Similarity Measure The distance between two range queries is the dissimilarity of the two queries. Without loss of generality, the distance measured between two range queries can be normalised to lie in the range $[0, 1]$. The similarity between two range queries can be computed as one minus dissimilarity. Various similarity models can be used to measure the similarity between two range queries. Without loss of generality, we use *Jaccard similarity* and leave other models for future work. *Jaccard* similarity between two range queries can

be computed by the ratio of intersection between the output of the two range queries to the union of the output to the two range queries.

$$\text{SIM}(Q_1, Q_2) = \frac{out(\mathbb{D}, Q_1) \cap out(\mathbb{D}, Q_2)}{out(\mathbb{D}, Q_1) \cup out(\mathbb{D}, Q_2)}$$

where out is the output of the range query on $\mathbb{D}$.

QUERY 2: `select id from` $\mathbb{D}$ `where` $3 \leq A_0 \leq 6.2$

For instance, the similarity between the example queries Query 1 and Query 2 ($out(\mathbb{D}, \text{Query2}) = \{t_0, t_4, t_6, t_{10}\}$) is:

$$\text{SIM(Query 1, Query 2)} = \frac{|\{t_4, t_6, t_{10}\}|}{|\{t_0, t_4, t_6, t_{10}, t_{12}\}|} = 0.6$$

3.2.2 Fairness Model

Our definition of fairness is based on group fairness [84] and the notion of demographic parity, aka statistical parity and disparate impact [46, 47, 84–86].

Sensitive attribute Group fairness is defined as parity over different demographic groups such as `white` and `black`. The demographic groups are identified by a non-ordinal attribute, such as `race` or `gender`, known as *sensitive attributes*. In many of the existing applications, sensitive attributes are binary, separating a minority group (e.g. `female`) from the majorities (e.g. `male`). Therefore, in this paper, we follow the existing work such as [87] and consider the sensitive attribute to be binary in nature. We leave the non-binary sensitive attributes and more general cases for the future work.

As highlighted in our sample database in table 3.1, we use the attribute `colour` (with two values `red` and `blue`) to abstract the sensitive attribute (and the demographic groups).

Fairness constraint The fairness measure is defined as the parity over the demographic groups, identified by the colours `blue` and `red`. The parity condition is identified using a criteria that decides whether the output of a query is fair. Let C_r and C_b be the number of red and blue objects in the output.

In some application, the parity can be defined as having equal number of objects from the demographic groups in the output set. That is, $C_r = C_b$. In other words, the objects in the selected set should have equal chance of belonging to each demographic group. For instance, Query 1 in our toy example, returns three blue objects ($\{t_4, t_6, t_{12}\}$) and one red object ($\{t_{10}\}$) and does not satisfy the parity condition $C_r = C_b$, while Query 2 returns two blue ($\{t_4, t_6\}$) and two red object ($\{t_0, t_{10}\}$) – hence satisfies the parity condition.

Alternatively, some applications consider the underlying distributions and require that the set of selected objects to represent the underlying demographic from which they were chosen from. In other words, the objects from different demographic groups should have equal chances of being selected in the output set. That is,

$$C_r/n_r = C_b/n_b$$

where n_r and n_b are the total number of `red` and `blue` objects in $\mathbb{D}$. Similarly, different applications may require different notions of parity based on societal norms. To support all these cases under the same fairness model, we abstract the fairness constraint, using a weight parameter W as following:

$$W_r C_r = W_b C_b$$

Specifically, we refer to the case where $W_1 = W_2$ as *unweighted fairness* and other cases as *weighted fairness* model.

Achieving perfect demographic parity in form of equality is rarely practical in the real-world, hence we use a threshold ε to identify an acceptable disparity [85]. Using this threshold, the fairness constraint can be rewritten as

FAIR RANGE QUERY PROBLEM: *Given a database $\mathbb{D}$, a range query Q and a disparity value ε, find a fair range that is most similar to Q with a disparity value at most ε.*

Figure 3.1: Problem Formulation

DECLARATIVE FAIRNESS-AWARE QUERY:

```
SELECT ...  FROM DATABASE
WHERE
    RANGE-PREDICATES
SUBJECT TO
    |W_r C_r - W_b C_b| <= eps  and SIM >= tau
```

Figure 3.2: Declarative Query Model

$$|W_r C_r - W_b C_b| \leq \varepsilon \tag{3.1}$$

3.2.3 Problem definition

Having formally defined the database and fairness notions, we are now ready to provide our problem formulation. We consider the problem of finding the most similar fair range to a user provided range query for the database $\mathbb{D}$. Figure 3.1 provides the formal formulation of the fair range query problem. This problem formulation helps the data scientists to slightly change their query to find the data that is similar to their initial query output and is also fair.

Declarative query model Our problem formulation follows a *declarative fairness-aware query model* as specified in Figure 3.2. Using this declarative interface, we expect the user to easily formulate the fairness-aware queries. In particular, we realise that the user might not be interested to accept the queries that are far from their initial choices, hence might require to identify a constraint on the minimum similarity they would find relevant. This, along with the fairness constraint, is identified as part of the constraints, followed after the

"subject to" phrase. Note that there may be multiple queries *equally* near to the input range query that satisfy the constraints. The problem of finding all fair range queries nearest to the given query is an interesting direction for future work, discussed in section 3.7. For example, knowing that Query 1 does not satisfy the demographic parity for unweighted fairness, the user can reformulate the query in form of Query 3 to discover a similar query (with at least 80% similarity) that has at most a disparity of 1. **QUERY 3:** `select id from` $\mathbb{D}$ `where` $4 \leq A_0 \leq 7$

`subject to` $|C_b - C_r| \leq 1$ `and` SIM $\geq 80\%$

Formulating Query 3 as fair range query problem, the optimal solution, by changing the range predicate to $3.1 \leq A_0 \leq 7$, adds t_0 to the output set, satisfying both the fairness and similarity constraints specified by the user.

After providing the terms and discussing the problem formulation, we now turn attention to designing efficient algorithms for our problem. In particular, we realise that a large portion of the queries in practice have a single range predicate. Therefore, in § 3.3, we focus on this case, designing a tailored solution for it. Then in § 3.4, we will devise an algorithm for the queries with multiple range predicates. We show empirical results in § 3.5.

3.3 Single-predicate Range Queries

Next, we consider queries with a single range condition. We first provide definitions and theorems that form the basis for our algorithms, in § 3.3.1. Then we design *SPQA*, our algorithm for the unweighted case in § 3.3.2, discuss pre-processing details in § 3.3.3, and present weighted *SPQA* in § 3.3.4.

3.3.1 Jump pointers

Query answering systems usually conduct offline pre-processing (indexing) that facilitates online query answering. One extreme approach for finding fair range queries, that optimises for query time, is to precompute and store the answer to all possible range queries during pre-processing. Such an approach, using proper data structures, enables constant-time query answering. This, however, requires an extensive space of $\mathcal{O}(n^2)$ to store the answer to all possible single-predicate range queries, which might not be reasonable, specifically for databases with millions of objects. The other extreme is to optimise for the space and to delay the computation to online query answering time. This, however, might require enumerating $\mathcal{O}(n^2)$ possible ranges, which makes query answering inefficient – $\mathcal{O}(n^2)$. Our proposal is between these two extremes, by building a *linear*-size index (*Jump pointers*) that enables the *sub-linear* query answering time of $\mathcal{O}(\log n + disparity)$, where $disparity$ is the unfairness of the input query.

The idea behind the *Single Predicate Query Answering* (*SPQA*) algorithm is to quickly lookup fair ranges, each of which have a similarity of ε. Theorem 3.3.2 proves that for any unfair unweighted query, the nearest fair query has a disparity value of exactly ε. Among the fair ranges which have a disparity of ε, the ones which have a potential to be nearest by *Jaccard similarity* to the input range are explored by *SPQA* to find the most similar fair range.

Definition 3.3.1. *(Jump pointers): Consider a database $\mathbb{D}$ and the attribute A for the single-predicate range query model. A* right *(resp.* left*)* blue *jump pointer from location o_i points to the nearest/closest location b_r(resp. b_l) such that the number of blues in the range $[o_i + 1, b_r]$ (resp. $[b_l, o_i - 1]$) is equal to the number of reds plus one. Red jump pointers are also defined in the same manner.*

For the sample database of Table 3.1, Figure 3.4 depicts the right and left jump pointers for attribute A_0. The index is constructed on top of the sorted list of object ids

according to their values on A_0. Therefore, since $t_1[0] = 0.7$ is the minimum of A_0, t_1 is the first object in the list. For example, consider the object t_{12}, where $t_{12}[0] = 7$; the range $[8, 10.9]$ ($8 \leq A_0 \leq 10.9$) consists of the smallest range starting from 8 that has one additional *red* than the *blues* in the range. Hence, its right *red* jump pointer points to t_3 ($t_3[0] = 10.9$). Note that not all objects have jump pointers; for example there is no *right red* jump pointer from t_{10} as no location ahead of t_{10} has one additional *red* that than the number of *blues* in the same range. Even though there exists four jump pointers ({left *or* right} and {blue *or* red}) for every object in the list, two of those pointers are trivial. For example, the range $[8, 8]$ ($8 \leq A_0 \leq 8$) consists of t_2 and is the smallest range after t_{12} that consists of one additional *blue*. As this trivial information can be quickly determined in $\mathcal{O}(1)$ time, this pointer need not maintained and can be looked up at query time. Left jump pointer follows a similar pattern; for example the node t_8 maintains a left *blue* pointer to t_6 as the range $[6.2, 10.9]$ ($6.2 \leq A_0 \leq 10.9$) consist of one additional *blue* than *reds*.

We refer to travelling along the jump pointer from location o_i as following a jump pointer. Following a *blue jump pointer* k times from location o_i gives us the closest location o_j from o_i such that the range from o_i to o_j has k *blues* more than the range that would end at o_i. The algorithm to find jump pointers is described in § 3.3.3.

Lemma 3.3.1. *Following the red colored jump pointer k times from o_i lands at a location where the range ending at it is has k more reds than the same range ending at o_i.*

Proof. We provide the proof by induction:

Base case: By the definition of jump pointer, a *red* colored jump pointer points to the nearest location which has one additional red. This gives us the base case for $k = 1$.

Induction step: Assume that the lemma holds for $k - 1$ *red* jump pointers. Let the $k-1^{th}$ jump pointer point at location o_l and k^{th} jump pointer point at location o_m. Suppose

there exists a $o_{m'}$ which is closer to o_i than o_m while also satisfying the k additional reds criteria. If $o_{m'}$ lies to the left of o_l then we already have a contradiction as the $k-1^{th}$ jump pointer would lie to the left of $o_{m'}$. On the other hand if $o_{m'}$ lies to the right of o_l, then the red jump pointer should point to $o_{m'}$ as the range o_l to $o_{m'}$ consists of one additional red than blue. As both cases are not possible, $o_{m'}$ is the same as o_m.

□

Jump pointers will be used for two operations (expansion and shrink) in single-predicate range queries. An expansion operation expands a range to include more objects. Excluding objects by shrinking the range is done using shrink operations.

Definition 3.3.2. *(Cumulative sum): Consider an attribute A for the single-predicate range query model. The cumulative sum c_i at a location o_i is the difference between the number of reds and blues from the left most location along A to o_i.*

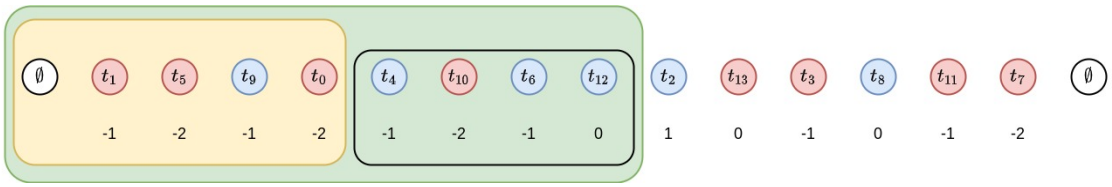


Figure 3.3: Disparity computation for the range $6.2 \leq A_0 \leq 10.9$ in the sample database of Table 7.

As an illustration from the sample dataset, consider computing the cumulative sum for the query range $6.2 \leq A_0 \leq 10.9$. We use figure 3.3 to illustrate the process. The cumulative sum at t_{12} refers to the weighted sum of all elements to the left (including) t_{12}, marked by the *green* colored rectangle. Similarly, the cumulative sum at t_0 refers to the weighted sum of all elements to the left (including) t_0, marked by the *yellow* colored rectangle. As we need the cumulative sum for the query range $6.2 \leq A_0 \leq 10.9$ which is marked by the black rectangle, the difference between the cumulative sum of the *green* and

yellow rectangles gives us the result. There is one blue and three reds until t_0 from the left most position t_1. Hence, the cumulative sum for t_0 is -2 $(1 - 3 = -2)$.

Given the two locations o_i and o_j ($[o_i, o_j]$), cumulative sums can be used to obtain the disparity between the start and end location in $\mathcal{O}(1)$ time.

$$disparity = c[i] - c[j-1] \tag{3.2}$$

Fox example, consider the example query 3.2.1, with range $4 \leq A_0 \leq 7$. The objects lying in the input range are $\{t_4, t_{10}, t_6, t_{12}\}$. With 3 *blues* and a *red*, the query has a disparity of 2 which can be computed using cumulative sum by $c[12] - c[0] = 2$. Note that any two locations with the same cumulative sum represent a range with *perfect parity*.

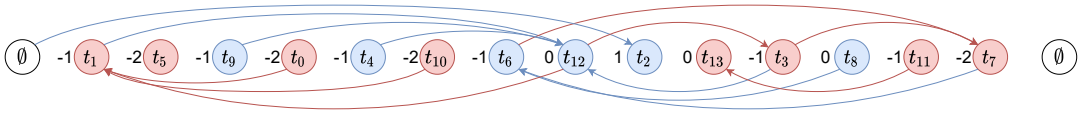


Figure 3.4: Right and left jump pointers for attribute A_0 of the sample database of Table 7.

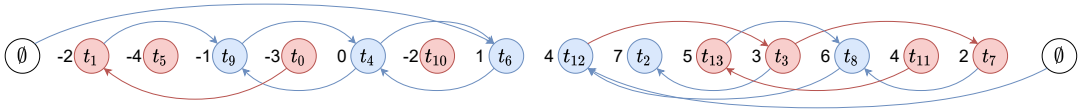
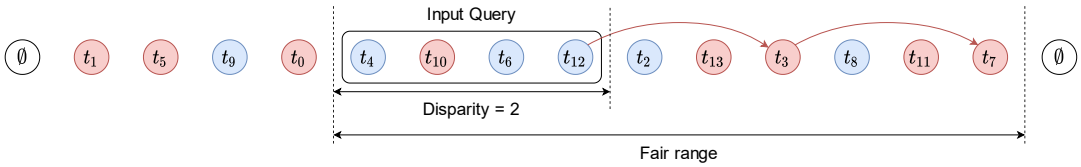


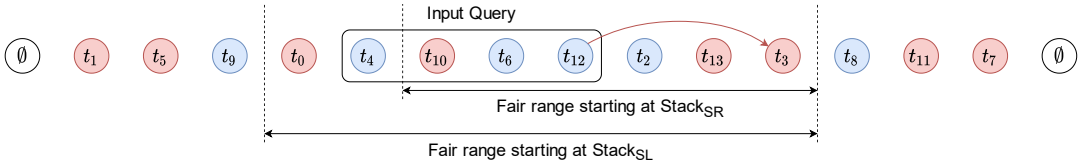
Figure 3.5: Jump pointers for a weighted fairness case

3.3.2 Query answering for unweighted fairness

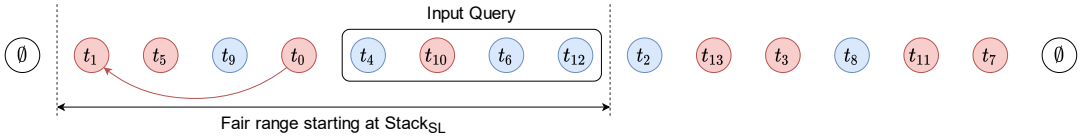
A range predicate is made up of two points, *start* and *end*. If one were to fix one of the two end points of the range query, to make the range query fair, the other end point can be moved to shrink the range or to expand it. Consider Figure 3.7, which shows the *red* jump pointer at t_{12}. Following the jump pointer, would get us to the closest point which has



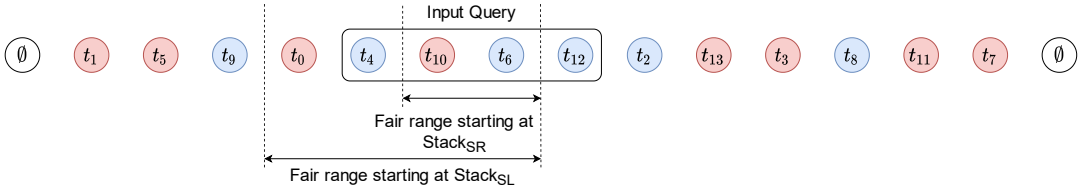
(a) Setting up the window by expanding the end point of the single predicate input query



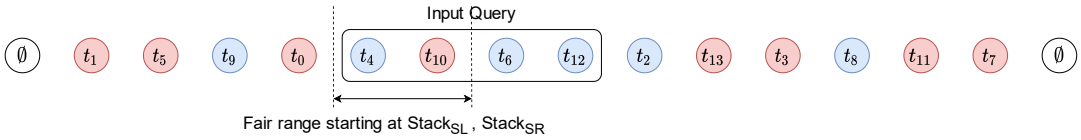
(b) Moving the window to the left from the initial position



(c) Window reaches the position where the *end* of the input lies



(d) The right end point of the window shrunk within the input query



(e) Exploration for all windows is completed

Figure 3.6: Step wise movement of the window over the course a run of the single predicate algorithm

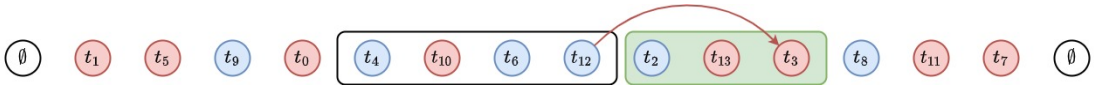


Figure 3.7: Intuition behind jump pointer for a single jump

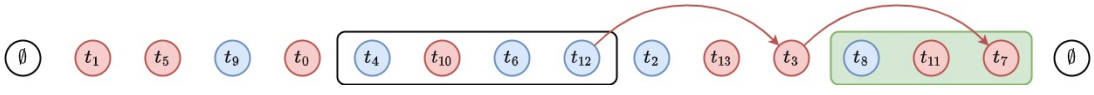


Figure 3.8: Intuition behind jump pointer for two jumps

one additional red. Similarly, Figure 3.8 shows the *red* jump pointer being followed twice at t_{12}, which points to closes point ahead of t_{12} which has 2 more *reds* than *blues*.

An expansion would require an addition of $|disparity - \varepsilon|$ deficient coloured objects to make it fair (proved later in theorem 3.3.2). For example, consider the input range query to be $[4.4, 7]$ ($4.4 \leq A_0 \leq 7$) with $\varepsilon = 0$. Figure 3.6a shows the query range $[4.4, 7]$ ($4.4 \leq A_0 \leq 7$) marked with an enclosing box. As there are two more *blues* in the range query, we require two additional *reds* to expand the query range to make it fair. Expanding the range by following two red jump pointers gives us a fair range. Figure 3.6a shows a fair range when the start of the range, t_4, is fixed and the other end point is allowed to expand to incorporate two additional reds, thus obtaining the range $[4.4, 13]$ ($4.4 \leq A_0 \leq 13$), marked by the dotted line. In general, one can follow the deficient pointer $|disparity - \varepsilon|$ times.

Our goal is to find the most similar range that resolves the disparity of $|disparity - \varepsilon|$. Such a disparity can be covered by moving either end points, that is $|disparity - \varepsilon|$ should equal to the sum of the changes on the right and left. For instance, figure 3.6b shows the end of the input query expanded by one red pointer and start expanded (resp. shrunk) by one red (resp. blue) pointer. Similarly, figures 3.6c, 3.6d and figure 3.6e shows the input query expanding/shrinking to cover the disparity of 2. We design a window-sweeping algorithm to find such a range.

Algorithm 7, *SPQA*, describes our approach, for finding the most similar fair range to the input query. Algorithm 9, JP describes the algorithm used for moving along a jump pointer.

Initially, the *start* end-point of the range is fixed and *SPQA* expands *end* end-point until a fair query is found, as shown in Figure 3.6a. The window is shown in the figure with the dotted lines indicating start and end of the fair range. The naming convention used for the boundaries is, S for start of input range and E for end of input range; L for left and R for right. Hence, SL (resp. SR) stands for start of input range expanded (resp. shrunk) to

the left (resp. right). When the window is swept to the left, the *start* end point can perform a shrink or an expansion as shown in Figure 3.6b. The remaining steps of the exploration by *SPQA* can be seen in Figures 3.6c, 3.6d and 3.6e. Among each of these, the output of the fair range which is most similar to the input query is provided as output to the user.

Theorem 3.3.2. *Given a database $\mathbb{D}$, the disparity threshold ε, and the input query $Q(A_j : [start, end])$, the optimal range has a disparity value of exactly ε.*

Proof. Let d be the disparity of the given range query. Let the disparity of optimal range be d_{opt}. Knowing that both the left and right end points of the input range could have moved, let the range for optimal range be $[l_{opt}, r_{opt}]$. Let us consider two new ranges, $[l_{opt}, start - 1]$, $[end + 1, r_{opt}]$ and let their corresponding disparities be dl_{opt} and dr_{opt}. The total disparity d_{opt} can be written as the sum of disparity of three different ranges, dl_{opt} for range $[l_{opt}, start - 1]$, dr_{opt} for range $[end + 1, r_{opt}]$ and d for range $[start, end]$.

$$d - d_{opt} = dl_{opt} - dr_{opt}$$

Suppose $d_{opt} \neq \varepsilon$. Let us now construct a range $[l'_{opt}, r'_{opt}]$ such that the disparity is exactly ε. To construct the range we will modify l_{opt} and r_{opt}. If l_{opt} lies on the left of $start$, then there was an expansion operation that has been performed. Instead of expanding it to cover a disparity of dl_{opt}, one can only expand it to a smaller extend such that the over all disparity is exactly ε. As the expansion was smaller the *Jaccard* similarity measure would be larger. Applying a similar approach in case l_{opt} was on the right of $start$ would also result in a larger *Jaccard* similarity measure as intersection between the two sets would be larger. A similar approach can be applied to the r_{opt} end point. The newly constructed range $[l'_{opt}, r'_{opt}]$ is more similar and has disparity of exactly ϵ. □

Lemma 3.3.3. *(Correctness) Given a database $\mathbb{D}$, the disparity threshold ε, and the input query $Q(A_j : [start, end])$, SPQA Algorithm 7 finds the optimal solution (the most similar fair range to q).*

Proof. To prove this theorem we use the property that the fair optimal range has a disparity of exactly ε, which is proved in theorem 3.3.2.

SPQA uses a windowed approach to explore all ranges around the input range which have a disparity value of ε. Let the disparity covered by moving the left the left end point $disparity([left, start - 1])$ be dl and the disparity covered by the right end point, dr be equal to $disparity([end + 1, right])$. The sum of dl, dr and d is ε.

$$dl + dr = \varepsilon - d$$

SPQA explores $4 \times disparity$ number of windows, where every pair of dl, dr satisfies the above equation. Thus the optimal result must lie in one of these pairs. □

Time complexity: The total amount of time taken by *SPQA* to answer the unweighted fairness queries with one range predicate is $\mathcal{O}(\log(n) + disparity(input))$, which can be seen in theorem below.

Theorem 3.3.4. *Given a database $\mathbb{D}$ with n tuples and an input range with disparity d the total time taken by SPQA algorithm is $\mathcal{O}(\log(n) + d)$.*

Proof. Searching the jump pointer data structure to reach the end points takes $\mathcal{O}(\log(n))$ time. Once the end points are found in the data structure, *SPQA* algorithm uses a window based approach to explore the fair ranges whose disparity is exactly equal to ϵ. There are a total of $\mathcal{O}(4d)$ such ranges, each of which takes $\mathcal{O}(1)$ time to compute *Jaccard* similarity. Thus, the total amount of time taken is $\mathcal{O}(\log(n) + d)$. □

Algorithm 7 *SPQA* Algorithm

Input : Database $\mathbb{D}$, Input query $Q(A_j : [start, end])$, acceptable disparity ε

Output : Most similar fair range query $fair$

$t_s \leftarrow binary\_search(\mathbb{D}, A_j, start)$

$t_e \leftarrow binary\_search(\mathbb{D}, A_j, end)$

$disparity \leftarrow c[j, t_s] - c[j, t_e]$

dColor $\leftarrow disparity > 0$ ▷ deficient: true-red, false-blue

if $disparity \leq \varepsilon$ **then** ▷ Input range is already fair

    return $[start, end]$

end if

$LEP \leftarrow t_s$ ▷ LEP stands for Left End Point

while $disparity > \varepsilon$ **do**

    Push $JP(\mathbb{D}\ , A_j, LEP, "left", deficient)$ to LEP, update disparity for $[t_{LEP}, t_e]$

end while

$fair \leftarrow \{\};\ sim \leftarrow 0$

$WindowSweep(t_{LEP}, t_e$, "shrink") ▷ Shift window by shrinking t_e

$WindowSweep(t_{LEP}, t_e$,"expand")

$WindowSweep(t_s, t_e$,"shrink") ▷ Shift win. by shrinking t_e

$WindowSweep(t_s, t_e$, "expand")

return $fair$

Algorithm 8 $WindowSweep$ algorithm

Input : Database $\mathbb{D}$, Attribute: A_j, start end-point: t_s, end end-point: t_e, operation, input query $Q(A_j : [start, end])$, reference to $fair$, sim

Output : Update $fair$ based on most fair range found

1: $disparity \leftarrow c[j, t_e] - c[j, t_s - 1]$

2: $dColor \leftarrow disparity > 0$

3: **if** $t_s < start$ **then** $t_s \leftarrow$ Pop(LEP)

4: **else** $t_s \leftarrow JP(\mathbb{D}\ , A_j, t_s, "right", !dColor)$ ▷ Shrink t_s

5: **while** $disparity > \varepsilon$ **do** ▷ Adjust t_e pointer

6: $\quad disparity \leftarrow c[j, t_e] - c[j, t_s - 1]$

7: $\quad dColor \leftarrow disparity > 0$

8: $\quad$ **if** $operation == "expand"$ **then**

9: $\quad\quad t_e \leftarrow JP(\ \mathbb{D}\ A_j, t_e, "right", dColor)$ ▷ Expand

10: $\quad$ **end if**

11: $\quad$ **else** $t_e \leftarrow JP(\mathbb{D}\ A_j, t_e, "left", !dColor)$ ▷ Shrink

12: **end while**

13: **if** $Jaccard([t_s, t_e], Q) > sim$ **then**

14: $\quad sim \leftarrow Jaccard([t_s, t_e], Q); fair \leftarrow [t_s, t_e]$

15: **end if**

16: $WindowSweep(t_s, t_e, operation)$

General positioning assumption: The algorithm and the definitions in the current section have been designed with the general positioning assumption. General positioning makes the assumption that no two points are co-located for the given attribute. In practice, with small modifications, our algorithms can handle the case when multiple points are present at a single location. Combining the co-located points into a single point with the aggregate weight would help us in creating a new dataset with no co-location. When the jump pointer

Algorithm 9 *JP* algorithm for *left* jump pointer

Input : Database $\mathbb{D}$, Attribute: A_j, Database object o_i, color c, direction dir

Output : *Database object o_j pointed by jump pointer*

1: **if** $dir ==$ "left" **then**
2: **if** $A_j[o_i - 1]$ is of color c **then**
3: return $o_i - 1$
4: **else**
5: return $LJP[o_i]$ ▷ LJP stands for Left Jump Pointer array
6: **end if**
7: **else**
8: **if** $A_j[o_i + 1]$ is of color c **then**
9: return $o_i + 1$
10: **else**
11: return $RJP[o_i]$ ▷ RJP stands for Left Jump Pointer array
12: **end if**
13: **end if**

encounters the new point with a variable weight, it needs to update the data structure with the variable weight value. Note though that the similarity function needs to take the number of points co-located into account while computing the similarity.

This algorithm holds unless there are too many of the same demographic data point at the same location so as to move a range from unfair because of lack of a group to unfair because of excess of that group. Note that this case is highly unlikely in practise. In such extreme cases, where the unfairness suddenly switches from one group being disadvantaged to the other, we choose two problems, one in which the aggregate point does not belong

thus limiting one side of the search space or the second in which we explore further by looking for the inverse jump pointers past that point.

3.3.3 Prepossessing

For every attribute in $\mathbb{D}$, a jump pointers index is created during the pre-processing. For this, the objects in every list are sorted based on the corresponding attribute (Figure 3.4) so that look ups can be performed quickly and then right and left pointers from each location of the database is calculated.

Finding jump pointer: *Jump* pointers play a key role in SP queries. A right *red jump* pointer points to the closest location to the right of the current location such that the number of *reds* in the range exceed the number of *blues* by 1. At any given location o_i, the range $[o_i+1, o_i+1]$ is a trivial range that satisfies this criteria. Hence, one of the 2 coloured *jump* pointers will point to o_{i+1}. The goal of the algorithm is to compute the other (non-trivial) right *jump* pointer.

In order to obtain the non-trivial right jump pointer, we need to find the location that differs by 1 (in the opposite sign than at location o_{i+1}). For example, in Figure 3.4, the cumulative sum at location t_{12} is 0 and as the colour at location t_2 is blue, the cumulative sum is 1. The non-trivial right jump pointer points the closest location with cumulative sum of -1.

The algorithm to find the jump pointers is given in Algorithm 10. The algorithm to find the jump pointers maintains a balanced binary search tree (BST) for the cumulative sums, which are used as keys for the *BST*. The indices which will be resolved when the specific cumulative sum is seen are stored as values within the *BST*. For example, when resolving the non-trivial red right jump pointer for t_{12}, closest location with a cumulative sum of -1 needs to be found. Hence, -1 is used as a key within the *BST* with the index t_{12} as a value.

The total time taken in the pre-processing step is $\mathcal{O}(n \log(n))$, as proved in theorem below.

Theorem 3.3.5. *Given a database $\mathbb{D}$ with n tuples and an attribute A_i pre-processing step takes $\mathcal{O}(n \log(n))$ time.*

Proof. The sorting step of pre-processing takes $\mathcal{O}(n \log(n))$ time for the given attribute A_i. Establishing the right and left *jump pointers* makes use of a balanced binary search tree (*BST*). A total of n indexes need to be inserted/deleted into the *BST*, which consumes $\mathcal{O}(n \log(n))$ time. Hence, the total time taken for building the jump pointer structure for given attribute A_i is in $\mathcal{O}(n \log(n))$. □

Note that, while the initial pre-processing takes $\mathcal{O}(n \log(n))$ time, query processing is sub-linear: $\mathcal{O}(\log(n) + d)$, where d is the disparity of input query.

Note on space complexity: During the pre-processing phase, *SPQA* algorithm creates a linear space data structure to aid in query processing. The query processing stores a total of disparity jump pointers to find the the most similar fair range which is small compared to the linear space data structure. Thus the total space complexity is $\mathcal{O}(n)$.

Algorithm 10 (Preprocessing) Building left jump pointers

Input : Database $\mathbb{D}$, attribute A_j

Output : *jump pointers*

Sort $\mathbb{D}$ along attribute A_j
$BST \leftarrow \{\}$
cumulative $\leftarrow 0$
for $i \leftarrow 0$ *to* n **do**
    cumulative $\leftarrow$ cumulative + colour(o_i)
    $c[j, i] \leftarrow cumulative$ ▷ Cumulative sum is contained in c
    if $cumulative$ present in BST **then**
        for $obj \in$ values of $BST[cumulative]$ **do**
            $LJP[j, obj] \leftarrow i$ ▷ Left jump pointer
        end for
    end if
    Insert i into $BST[cumulative - 2 * color(o_i)]$
end for

3.3.4 Generalisation to weighted fairness

So far, our attention has been on the unweighted fairness model. In this section, we move to our general model of fairness: weighted fairness. As explained in § 3.2, the fairness constraint for this model is in the form of $|W_r C_r - W_b C_b| \leq \varepsilon$, where W_r and W_b are the weights for the red and blue counts, respectively. That is, the difference between the weighted sum of the number of objects from the two demographic groups should not be bounded by the threshold ε. Note that any rational values for weights can be expressed

as integer weights by scaling these weights. For example, weights $W_r = 1.1$ and $W_b = 1.2$ are equivalent to $W_r = 11$ and $W_b = 12$.

The fair range query problem in the generalised case would refer to finding the most similar fair range to the input fair range such that the disparity is within a value of ε. Note that finding cases where the disparity is less than $max(W_r, W_b)/2$ would infer finding ranges with a level of precision less than a single unit of disparity (less than a single weighted coloured object). For the sake of simplicity and practicality, we omit such cases and assume that the value of $\varepsilon \geq max(W_r, W_b)/2$.

The algorithm that deals with the weighted case uses similar concepts like jump pointer and cumulative sum. In the general case, a *blue* (resp. *red*) right jump pointer from location o_i points to the closest location right of o_i, j_i such that the range $[o_i+1, j_i]$ contains more *blues* than *reds* (resp. *reds* than *blues*) by weight. A similar definition for left jump pointers can be defined. Jump pointers for the sample dataset presented in Table 3.1 is presented in Figure 3.5 using weights of 3 for *blue* and 2 for *red*.

For the weighted case, the cumulative sum at a location o_i indicates the weighted difference of *blues* and *reds*.

Theorem 3.3.6. *(Correctness) Given a database $\mathbb{D}$, the disparity threshold ε, weights W_r and W_b, and a query $Q(A : [start, left])$, algorithm 7 finds the the most similar fair range to Q.*

Proof. Let the input range be deficient in *blue* colour and would require *blue* objects to make it fair. We know from the fairness criteria that $\varepsilon \geq max(W_r, W_b)/2$. If an unfair range range had a disparity of $\delta > \varepsilon$, adding a single *blue* object can only fix a disparity of $max(W_r, W_b)/2$. Hence, adding a single blue cannot make any unfair range that is deficient in *blue*s into a unfair range that is excess in *blue*s. This property will be used for proving this theorem. While expanding the range towards the end of the range, the right jump

pointer of the deficient colour is used to reduce the disparity. The objects that are visited by this outward right jump pointer expansion until a fair range is reached are denoted as the *expansion jump path*. Note that the expansion jump path is restricted to within a disparity of $|disparity(input)| + max(W_r, W_b)/2$. The definition can be extended to *shrink jump path* in a similar manner. We prove that exploring the objects that lie along the *jump pointer paths* from the to end points is sufficient to get the optimum result.

Let us first describe an important property, that the objects that lie along the *jump paths* exhibit. Suppose the input range is unfair as it consists of more *blues* than *reds* i.e. in excess of ε. Let the object o_j be the right blue (deficient in general case) pointer from the end point o_i. The objects that lie in between o_i and o_j can cover at most one more *blue* worth of disparity in the jump. Hence, we can observe that *each of these points that lie along the expansion path cover monotonically more disparity that the previous object*.

Let $[l_{opt}, r_{opt}]$ be the optimal fair range and let the end points of the range not lie on expansion/shrink jump paths. Let us assume that r_{opt} was obtained because of an expansion operation. As the endpoint at r_{opt} is not on the expansion jump path, it would mean that the range ending at r_{opt} does not cover more disparity than the previous object (r_{prev}) on the expansion path. If the range $[l_{opt}, r_{prev}]$ is fair we have a contradiction. On the other hand, if the range is unfair, we can use the property that we explored before: *adding a single coloured point cannot change an unfair range in one colour to an unfair range in opposite colour*, leading to a contradiction. □

Time complexity: The total time taken by *SPQA* algorithm is same as the unweighted case, $\mathcal{O}(\log(n) + d)$, where d is the disparity in the input range. The details of the time complexity for the unweighted case which is mentioned in theorem 3.3.4 also applies to the weighted case.

Note on space complexity: During the pre-processing phase, *SPQA* algorithm creates a linear space data structure to aid in query processing. The query processing stores a total of disparity jump pointers to find the the most similar fair range which is small compared to the linear space data structure. Thus the total space complexity is $\mathcal{O}(n)$.

Pre-processing for the weighted fairness model In the weighted case, a *blue* pointer points to a location that has more *blue*s than *red*s. We use the same notation as that of the unweighted case and denote the cumulative sum at location o_i for the index of attribute A_j as $c[j, i]$. If the location next to o_i had a blue then the pointer would be trivial as it is pointing to the immediate next location. Let us consider the case where the immediate next location had a *red* instead. In such a case we need to find the nearest location whose cumulative sum is larger that the $c[j, i]$. In the opposite case where the neighbour was a blue one would like to find the closest location whose cumulative sum is smaller that the $c[j, i]$. Based on this approach, two data structures can be maintained. One for the locations whose pointers can be resolved if we found a cumulative sum with a smaller value than in the data structure. And the other data structure whose pointers can be resolved if we found a cumulative sum with a larger value. We use a *balanced binary search tree* for the two data structures. The pseudo-code to find the *jump pointers* for the weighted single predicate range query is in algorithm 11. The algorithm is an extension of the unweighted *jump pointers*. The algorithm uses a sort over the database as a the first step before adding and removing n items from a balanced BST in order to find the jump pointers, it takes $\mathcal{O}(n \log n)$ time.

General positioning assumption: The algorithm and the definitions in the current section have been designed with the general positioning assumption. General positioning makes the assumption that no two points are co-located for the given attribute. In practice, with small modifications, our algorithms can handle the case when multiple points are present at a single location. Combining the co-located points into a single point with the aggregate weight would help us in creating a new dataset with no co-location. When the jump pointer

encounters the new point with a variable weight, it needs to update the data structure with the variable weight value. Note though that the similarity function needs to take the number of points co-located into account while computing the similarity.

This algorithm holds until and unless we have too many of the same demographic data point at the same location so as to move a range from unfair because of lack of reds to unfair because of excess of reds. Note that this case is highly unlikely in practise. In such extreme cases, where the unfairness suddenly switches from one the advantaged group to the disadvantaged one, we choose two problems, one in which the aggregate point does not belong thus limiting one side of the search space or the second in which we try further exploration.

Algorithm 11 (Preprocessing) Left jump pointers for weighted case

Input : Database $\mathbb{D}$, attribute A_j

Output : *jump pointers*

Sort $\mathbb{D}$ along attribute A_j
$larger\_BST \leftarrow \{\}; smaller\_BST \leftarrow \{\}$
cumulative $\leftarrow 0$
for $i \leftarrow 0$ *to* n **do**
    if color(o_i) == *'blue'* **then** ▷ *Blue* always has a positive score
        $smaller\_BST[cumulative] \leftarrow i$
    else
        $larger\_BST[cumulative] \leftarrow i$
    end if
    $cumulative \leftarrow cumulative$ + $weight(color(o_i))$
    $iterator \leftarrow$Find $cumulative$ in $larger\_BST$
    while $iterator \neq \emptyset$ **do** ▷ Until end of $larger\_BST$
        $LJP[j, iterator] \leftarrow i$ ▷ Left jump pointer
        Increment $iterator$
    end while
    $iterator \leftarrow smaller\_BST.begin()$ ▷ Start of $smaller\_BST$
    while $cumulative > iterator$ **do**
        $LJP[j, iterator] \leftarrow i$
        Increment $iterator$
    end while
end for

3.4 Multi-predicate range queries

Next, we study the queries that contain multiple range predicates. Unfortunately, moving from single-predicate (SP) range queries to multi-predicate (MP) range queries complicates the problem significantly and the idea of jump pointers does not carry over. The reason is that in MP queries, there are different directions (along different angles) which a single jump can occur, while in SP there is only one direction (along x-axis) to make a jump. Moreover, while a SP query is identified by its two end-points, a MP query with d range predicates forms a hyper-cube with $2d$ sides. Hence, instead of the two end points of an SP range, one may need to move all sides of the hyper-cube to obtain the closest fair range, even when the disparity is slightly above the allowed fairness threshold, ε.

An observation that helps us with the MP cases is that the user may not be interested in fair ranges that are far away from the input query. Hence, the fair range query should be highly similar to the input range, otherwise it is not valuable for the user. We use this observation to design a *best-first search* (BFS) fair range query algorithm for the MP query.

3.4.1 Best First Search algorithm

At a high level, the BFS algorithm can be viewed as a "smart" traversal over a graph where every range is modeled as a node and there is an edge between two nodes if the outputs of their corresponding queries vary only in one tuple. That is, a node Q_2 is a *neighbour* of Q_1 if the output of query Q_1 differs from the output of query Q_2 by exactly 1 element. Mathematically, sets $out(\mathbb{D}, Q_1)$ and $out(\mathbb{D}, Q_1)$ have a symmetric difference of size 1.

The unfair input range provided by the user serves as a starting point in the graph traversal. This can be viewed as starting from the node with *Jaccard* similarity of 1 (*Jaccard distance* of 0), discovering its neighbours, deciding which node to visit next, and pruning

the blanket of nodes in the graph that their corresponding ranges have similarity less than the current best fair range discovered.

Starting from the node of the input query, the algorithm first needs to discover its neighbouring nodes in the graph. For this, we rely on the existence of an oracle $neighbors(Q)$ that discovers the neighbours of a query Q. It turns out, due to the frequency of calling this oracle, it can significantly impact the performance of the BFS algorithm. We shall provide a careful development of this oracle in § 3.4.2.

At any point of traversal, the algorithm selects the node that has the *maximum Jaccard similarity* with the *input query* for being visited next. The Jaccard similarity can be represented as the ratio of the intersection of two sets to their union, and the neighbouring range to a given range can differ only by a single element. Accordingly the neighbouring ranges are the ranges with the smallest Jaccard similarity.

Upon visiting a node, the algorithm checks if it satisfies the fairness requirements. If so, the algorithm stops and returns this range as the most similar fair range with query input. Otherwise, it calls the $neighbor$ oracle to discover the unseen neighbours of this node to be considered for traversal. The pseudo-code of the BFS algorithm is provided in Algorithm 12. It uses a max-heap for efficient traversal of the graph. Using the heap data structure, adding the new nodes to the list of discovered nodes and identifying the most similar node to the input range is done in logarithmic time to the size of heap.

Algorithm 12 Best-First Search algorithm for MP : BFSMP

Input : Database $\mathbb{D}$, attribute list A, input query Q

Output : *most similar fair range*

1: $Heap \leftarrow Q$
2: **while** $|Heap| \neq 0$ **do**
3: $top \leftarrow Heap.pop()$
4: **if** $fair(top)$ **then** return top
5: **for** $neighbor \in neighbors(top)$ **do**
6: $Heap.push(neighbor)$
7: **end for**
8: **end while**
9: return $\emptyset$

Lemma 3.4.1. *(Correctness) Algorithm 12 finds the most similar fair range to the input range.*

Proof. The *Jaccard* similarity of the set being explored is I/U, and the sets being added can have a reduced *Jaccard* similarity of $(I-1)/U$ or $I/(U+1)$. These are the smallest possible decreases in *Jaccard* similarity possible by removing or adding points.

Starting from the input range, let us now consider the neighbourhood path from the input range to the most similar fair range. As at every stage of the algorithm all the neighbourhood ranges which account to the smallest possible decreases in *Jaccard* similarity have been added to the heap, the fair output range that the algorithm produces is the most similar one. □

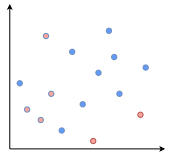


Figure 3.9: Sample set of points

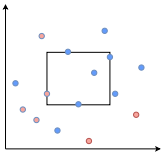


Figure 3.10: Sample input range

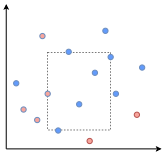


Figure 3.11: Expanding the rectangle downwards

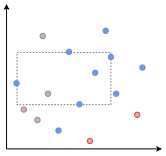


Figure 3.12: Expanding the rectangle towards left

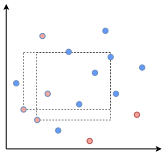


Figure 3.13: Neighbouring ranges in diagonal

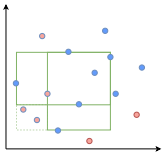


Figure 3.14: Skyline computation over a range query

3.4.2 Neighbouring range computation

Having explained the BFS algorithm, we now turn our attention to developing the $neighbors$ Oracle. Computing the neighbouring ranges is an important step of the BFS algorithm. The challenge here is to make sure *all* neighbours of a range have been discovered in an efficient manner.

To better explain the oracle, let us consider a sample dataset as shown in Figure 3.9. Consider a sample range be as shown in Figure 3.10. Suppose we want to expand the range outwards in order to add a new point. One simple approach of expansion that can be thought of is to move a side while maintaining either the height or the width constant. Figure 3.11 shows the expanded rectangle while moving the lower bound while maintaining the width constant. A similar approach can be performed on the left bound as seen in Figure 3.12. Note that these are not the only possible expansions. These expanded ranges will later be used to limit our search for finding the other neighbouring ranges along the diagonal direction. For 2D, there are 4 such expansions. As a generalisation, one can obtain $2d$ such expansions in d dimensions. Such points can be found out in $\mathcal{O}(\log^d n + k)$ using a range tree [88].

One can think of adding an additional point by moving a corner point along the diagonal direction. One such diagonal expansion can be seen in Figure 3.13. The bottom left corner can be expanded to add one more point along the diagonal. As one may observe,

such expansions are limited by the expansion of the sides which border the corner. The expanded boundary shown in Figures 3.11 shows the extent to which one may expand the bottom boundary downwards until they find a point. If a point laid in the diagonal beyond such a boundary, it would not account to a neighbourhood range as such an expansion would contain two points instead of one.

Figure 3.14 shows the expanded vertical and horizontal ranges that add a single point in solid green lines. The expanded boundaries form the limits of the region containing the diagonal expansion end points. *The problem of finding all possible diagonal expansion points can be formulated as finding a skyline* within the range shown by the dotted green lines: Given a corner point p_o for diagonal expansion[1] in a d-dimensional space, consider the bounding box specified by the side expansion (e.g. the dashed rectangle in the bottom-left of Figure 3.14). A point p_1 inside the bounding box dominates another point p_2 in the bounding box if $\forall 0 < i \leq d : |p_1[i] - p_o[i]| < |p_2[i] - p_o[i]|$. The skyline of the points in the bounding box is the set of points not dominated by any other point. Every skyline point is a valid diagonal expansion. As a result, in order to find all neighbours of a given range, it is enough to find (a) all neighbours by side expansion/shrinking and (b) all diagonal expansion points in the skylines. A MP query with d range predicates contains 2^d corners. Each corner can be expanded away from the centre of the MP query in order to find queries that differ by a single point, i.e. a neighbouring range. There can be many neighbouring ranges for each corner. A range skyline query can be constructed for each corner using the intersection of the boundaries of the side expansion as one of the end points of the range query and the corner point's coordinates itself as the other end point. One naive approach is to obtain the points that lie within the range using a R-tree and then apply a skyline algorithm on the points obtained. This is not efficient. We use studies that efficiently compute the skyline on range queries. In particular, we use the the *Range-*

[1] Shrinking a range is done similarly.

Skyline-Query algorithm by Janardan et. al. [89] for skyline discovery. This algorithm has a complexity of $\mathcal{O}((k+1)\log^d n)$, where k is size of skyline. Note that k should generally be a small number. In particular, as the number of dimensions increase, and as the size of the range grows, the expected number of points that occur within the corner ranges will decrease. That is because with each dimension the number of ranges it must occur within increases by one; and it will decrease with the size of the range, as more points that are potentially the nearest point will equate to a decrease in the size of the corner.

3.4.3 Informed best first search

The BFS algorithm discussed so far searches for the fair range by exploring the node with the maximum *Jaccard similarity* first. Branching out from a node to explore for a fair range requires discovering it neighbours, adding them to the heap, and repeating the same process for its neighbours in a recursive manner – which is time-consuming. On the other hand, given the amount of disparity at a node, it may be clear that its neighbour up to a certain number of hops cannot fill the disparity gap. That simply is because every neighbouring node has a difference of exactly one element with the current node and, hence, in the best case can drop the disparity *by one unit*. In other words, if the current disparity is equal to δ and the fairness threshold is $\varepsilon < \delta$, at least $\delta - \varepsilon$ hops are needed to fill the disparity gap.

Every hop in the path from the current node reduces the similarity from the initial query to a certain degree. As a result, combining the minimum number of hops to achieve fairness with the similarity decay per hop, we can compute an upper-bound threshold on the maximum similarity for a fair range (referred as U-threshold) that one can hope to achieve by branching out from the current node.

The above observation enables to design a more efficient algorithm, *Informed Best Frist Search algorithm for Multi-Predicate* (IBFSMP), with an early stop criteria, that de-

lays exploring the branches that their U-threshold is not the maximum. In other words, instead of selecting the most similar node to be explored next, IBFS selects the node with *maximum U-threshold* to be explored next. IBFSMP is inspired from the *A* algorithm* [90] which utilises the lower-bound on the remaining distance to the destination to perform an efficient search. However, IBFSMP differs from the A* in details and the way the bounds are calculated. We still need to compute the U-threshold of a node, which is done in Theorem 3.4.2.

Theorem 3.4.2. *The U-threshold of a node Q is:*

$$J_U(Q) = \begin{cases} \max\limits_{C'_r \leq \lceil \frac{\delta-\varepsilon}{W_r} \rceil} \frac{I - \lceil \frac{max(\delta-\epsilon-W_r \cdot C'_r, 0)}{W_b} \rceil}{U + C'_r} & W_r > W_b;\ \delta > \varepsilon \\ \max\limits_{C'_b \leq \lceil \frac{\delta-\varepsilon}{W_b} \rceil} \frac{I - C'_b}{U + \lceil \frac{max(\delta-\epsilon-W_b \cdot C'_b, 0)}{W_r} \rceil} & W_b > W_r;\ \delta > \varepsilon \\ \max\limits_{C'_r \leq \lceil \frac{\delta-\varepsilon}{W_r} \rceil} \frac{I - C'_r}{U + \lceil \frac{max(-\delta-\epsilon-W_r \cdot C'_r, 0)}{W_b} \rceil} & W_r > W_b; \delta < -\varepsilon \\ \max\limits_{C'_b \leq \lceil \frac{\delta-\varepsilon}{W_b} \rceil} \frac{I - \lceil \frac{max(-\delta-\epsilon-W_b \cdot C'_b, 0)}{W_r} \rceil}{U + C'_b} & W_b > W_r; \delta < -\varepsilon \end{cases} \tag{3.3}$$

where $\delta = W_b \cdot C_b - W_r \cdot C_r$.

Proof. Let the node Q have an intersection of I and union of U with the input range. Let the disparity of the unfair range Q be $\delta = W_b \cdot C_b - W_r \cdot C_r$. As the range Q is unfair, $|\delta| > \varepsilon$.

Let us consider the case that the range Q is unfair because of the presence of too many blues in Q compared to the reds.

$$\delta = W_b \cdot C_b - W_r \cdot C_r > \varepsilon$$

As we are trying to find the upper bound, we would like to maximise the *Jaccard similarity* such that such a range can *potentially* exist. In order to obtain a fair range, either blues can be removed, reds can be added or both can be done. Let B' be the blues that are removed and R' be the reds that are added to Q to make it a fair range.

$$
\begin{aligned}
-\varepsilon \leq W_b(C_b - C'_b) - W_r(C_r + C'_r) &\leq \varepsilon \\
-\varepsilon \leq \delta - W_b \cdot C'_b - W_r \cdot C'_r &\leq \varepsilon
\end{aligned}
$$

Moving around the terms, we get

$$
\delta - \varepsilon \leq W_b \cdot C'_b + W_r \cdot C'_r \leq \delta + \varepsilon \tag{3.4}
$$

In order to maximise the *Jaccard similarity*, various values of B' and R' need to be checked which satisfy the equation 3.4. Note that for a given value of B'(resp. R'), using the smallest R'(resp. B') that satisfies the equation 3.4 would provide a larger *Jaccard similarity*. Thus, given B' the smallest value of red satisfying the equation would be,

$$
C'_r = \lceil \frac{\delta - \varepsilon}{W_r} \rceil
$$

The *U-threshold* thus can be expressed as a maximisation in terms of R',

$$
\underset{0 \leq C'_r \leq \lceil \frac{\delta - \varepsilon}{W_r} \rceil}{max} \frac{I - \lceil max(\delta - \epsilon - W_r \cdot C'_r, 0)/W_b \rceil}{U + C'_r} \tag{3.5}
$$

Similarly, given R' the *U-threshold* thus can be expressed as a maximisation in terms of B' as,

$$
\underset{0 \leq C'_b \leq \lceil \frac{\delta - \varepsilon}{W_b} \rceil}{max} \frac{I - C'_b}{U + \lceil max(\delta - \epsilon - W_b \cdot C'_b, 0)/W_r \rceil} \tag{3.6}
$$

The amount of time taken to compute the *U-threshold* using the equation 3.5 is $\lceil \frac{\delta - \varepsilon}{W_r} \rceil$. The amount of time taken to compute the *U-threshold* using the equation 3.6 is $\lceil \frac{\delta - \varepsilon}{W_b} \rceil$. In case W_r is larger than W_b the complexity for exploring all the values for reds using equation 3.5

is better. Equation 3.6 can be used to explore all the values for blues when W_b is larger than W_r. A similar approach can be applied when the range is unfair because of excessive reds to obtain the final two cases in equation 3.3.

□

Replacing the selection criteria for traversing the graph with U-threshold, the only component of Algorithm 12 that needs to change is the max-heap and the rest remains unchanged, i.e., instead of structuring the heap according to similarity, IBFS builds the heap according to U-threshold (Equation 3.3).

Note that the IBFS algorithm is agnostic to the heuristic and similarity measure satisfying two important properties. (1) The similarity measure being used must be a set based similarity measure based on the points in the output range. (2) As can be seen from U-threshold, the heuristic must provide a upper-bound threshold on the maximum similarity for a fair range.

Note on space complexity: BFSMP algorithm explores neighbouring ranges to reach the fair range query that is nearest to the input query. Along the process a large number of ranges are explored and stored in memory in a *heap*. The space consumed by the algorithm depends on the number of neighbouring ranges explored. Thus the space complexity for BFSMP algorithm is $\mathcal{O}(\text{number of explored ranges})$.

3.4.4 Using MP algorithms for SP

Before concluding this section, we would like to note that MP algorithms also work for SP. However, *SPQA* has a provably better time complexity than *BFS*, in all instances. This is because *SPQA* takes advantage of pre-computed jump pointers which is only available when the possible changes in the bounds of the range are restricted to one degree of freedom. As explained in § 3.3.2, *SPQA* has a worst-case time complexity of $\mathcal{O}(\log(n) +$

$disparity$). One can easily establish a *best* case complexity for *BFS* algorithms that is at least as slow as this. First, in order to reach it's destination, *BFS* can adjust its range with each step by adding or removing a point. In the best case, it visits only points which monotonically decrease the disparity, and stops after *disparity* more steps. Additionally, *BFS* constructs a one dimensional range tree (which is equivalent to a balanced binary search tree) as pre-processing to find the closest point. This requires an initial setup time of $n\log(n)$. Therefore, the best case time-complexity of *BFS*s is $\Omega(n\log(n) + disparity)$. This demonstrates that the time-complexity of *SPQA* is comprehensively better, and accordingly, we favour it for SP queries.

3.5 Experiments

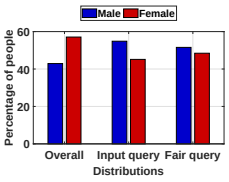


Figure 3.15: Demographic distributions in dataset, input query, and similar fair query.

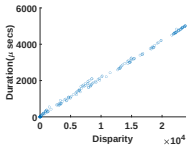


Figure 3.16: Time taken by *SPQA* v.s. disparity - *Texas Tribune* with gender as SA

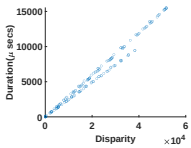


Figure 3.17: Amount of time taken by *SPQA* against disparity - *Texas Tribune* with race as SA

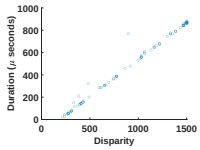


Figure 3.18: Amount of time taken by *SPQA* against disparity - COMPASS dataset

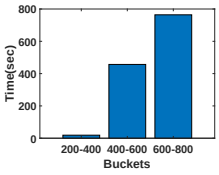


Figure 3.19: Amount of time taken by IBFSMP - Uniform dataset 3 range predicates

3.5.1 Experimental setup

Datasets: We used both real and synthetic datasets for our experiments. For the real world datasets we use *TexasTribune* and *UrbanGB* datasets. Along with the real world datasets, a synthetic dataset *Uniform* was generated for the experiments. Below we provide a brief description of these datasets.

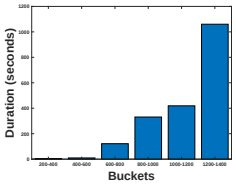


Figure 3.20: Average amount of time taken by IBFSMP algorithm for different bucket sizes - *UrbanGB* dataset

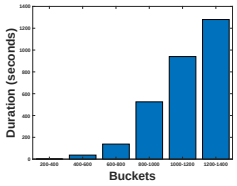


Figure 3.21: Average amount of time taken by IBFSMP algorithm for different bucket sizes - *Uniform* dataset

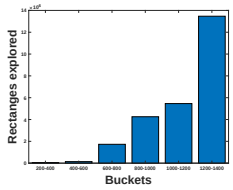


Figure 3.22: Rectangles explored by IBFSMP algorithm for different bucket sizes - *Urban GB* dataset

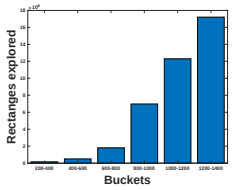


Figure 3.23: Rectangles explored by IBFSMP algorithm for different bucket sizes - *Uniform* dataset

Dataset name	Items	d	Sens. attribute	Weights
Texas Tribune [91]	149,481	21	gender,race	gen. (1:1) race (4:5)
COMPASS [92]	60,842	12	race	2:1
UrbanGB [93]	1,600,000	33	#vehicles in accident	2:1
(Synthetic)*Uniform*	10,000	4	Synthetic	1:1

- (Real dataset) *TexasTribune*[2]: Texas Tribune dataset consists of 149,481 records with the salary/compensation information for Texas state employees. The dataset has 21 attributes with gender and race being the main sensitive attributes and salary/compensation being numeric.
- (Real dataset) *COMPASS [92]*: COMPASS dataset (unprocessed) consists of 60,842 data points collected with 12 attributes with race as sensitive attribute and one real numbered attribute(*raw score*). The dataset has around 21K Caucasian (blue) and 39K non-Caucasian (red) warranting a ratio of 2:1.
- (Real dataset) *UrbanGB*[3]: UrbanGB dataset consists of 1.6 million records of accidents over a period 2000 and 2016. The dataset has 33 attributes, including latitude, longitude, accident severity, number of vehicles involved in the accident, date and time of accident. For the experiments, $10,000$ records from the *UrbanGB* dataset have been used. As

[2] https://salaries.texastribune.org/

[3] kaggle.com/daveianhickey/2000-16-traffic-flow-england-scotland-wales/data

UrbanGB dataset does not have a sensitive attribute inherent to it, we use the number of vehicles that were involved in the accident to create a sensitive attribute. There are 3,088 records where a single vehicle was involved in an accident and 6,912 records where more than one vehicle was involved. Hence, we use a weight of 2 for the 3,088 records and 1 for the 6,912 records.

- (Synthetic dataset) *Uniform*: The dataset consists of 10,000 points sampled uniformly from a cube which has a side of length 1,000 and a uniformly sampled binary sensitive attribute. The dataset has 4,967 blues and 5,033 reds.

The *Texas Tribune* and *COMPASS* datasets consist of one numerical attribute, a few other categorical attributes and sensitive attribute. Hence, the two datasets have been used with *SPQA*. As *Uniform* and *UrbanGB* datasets consist of multiple numeric attributes it is used for MPQA queries.

Our experiments were conducted on a Intel(R) Core(TM) i7-6850K CPU @ 3.60GHz with 64GB of main memory using Linux operating system (Ubuntu 18.04.5 LTS).

Algorithms implemented: Along with *SPQA*, weighted *SPQA* and IBFSMP algorithms, to evaluate multi-predicate range queries, we implemented a local search baseline algorithm.

- *Baseline*: The baseline approach for the multi-predicate range queries is based on limiting the search space to obtain a bounding box within which to search for fair queries. The maximum number of elements that can be added to the input range without violating the *Jaccard similarity* criteria is first computed. The boundaries of the expanded box are found by finding the smallest expansion along each of the directions/dimensions without violating the similarity criteria. This expansion limits the search space while still providing a valid box to search for the most similar fair range. This expanded range is then searched in a brute-force manner to obtain the closest fair range.
- *Coverage based algorithm* [79, 80]: We have used the code from [79, 80] to compare against our methods for multi-predicate range queries. The different techniques in the

papers [79, 80] modify the input range to produce a range which covers at least a given threshold number from each demographic group.

All the algorithms in the paper are implemented using C++. The implemented code can be found at the github location[4].

Experimental parameters: The value for ε plays an important role in simulating a real scenario. Satisfying perfect parity may not always be possible in practice and may require significant changes in the initial setting. In particular, in our problem setting, the ranges that are not similar enough to the user input may not be valuable. In our experiments, we allow a disparity of 5% between demographic groups. The corresponding value of ε is then computed as:

$$\varepsilon = \frac{0.05\ (|B\ W_b| + |R\ W_r|)|out(Q)|}{2(B+R)}$$

where B and R are the total number of blues and reds in the universe respectively. W_b (resp. W_r) refer to the weight of each blue (resp. red). The entity $(|B\ W_b| + |R\ W_r|)/(B+R)$ gives us the expected magnitude of the weight each point carries. The scaled weight for the given query Q with an allowed disparity of 5% thus turns out to be $0.05(|B\ W_b| + |R\ W_r|)|out(Q)|/(B+R)$.

Choosing an appropriate value of ε: Our system enables responsible data selection through exploration. While the choice of ε varies based on the application, an exploration based approach helps the application owner to choose an appropriate value of ε. That is, after the user specifies a range query and a value of ε, we return the most similar query, satisfying it. The user then has the choice to accept our recommendation, or to adjust the value of ε and/or the range and continue exploring until a satisfactory range is identified.

[4] https://github.com/surajshetiya/fairness-range-queries-icde-2022

3.5.2 Proof of Concept - TEXAS TRIBUNE

For a proof of concept, we use the Texas Tribune dataset. The dataset consists of 149,481 records of compensation for Texas state employees. We use `gender` as the sensitive attribute. As the representation of males and females in the dataset is similar (64,153 men and 85,328 female), the weights for male and female in the query are considered to be the same. The distribution of male and female employees in the overall dataset can be seen in Figure 3.15. Following Example 2 on this dataset, we assume the business office is interested in finding the employees who earn a salary of more than $65,000. There are a total of 14,803 men and 12,182 women earning more than $65,000, with a difference of 2,621 (around 10%). Considering the ethical considerations, the business management office would like to find a query whose output is similar to the initial query and within which there are at most 1000 (around 5%) males more than females.

Using *SPQA*, a fair query which is most similar to the input input query is determined. The output of the fair query ($60562 \leq salary \leq 152000$) consists of 32,064 employees with 16,532 male and 15,532 female. The *Jaccard* similarity of the fair query with the input query is around 75% (76.23%). Figure 3.15 shows the distribution of male and female employees in input query and the most similar fair query.

Extended PoC: A function $F(x)$ which for which datapoints with the same x values but from different demographic groups have different results will underperform on a given demographic group if that demographic group is unfairly represented in designing the function. In the TexasTribune dataset, there is a function that meets this requirement where $F(x)$ predicts the salary and the parameters x are their level (or lack of level) of employment and the part of the state for which of they are employed. We trained an auto-sklearn regressor on this task, using two datasets: the result of the original query in Example 2, and the result of the modified fair query. We then analyzed the dataset using R^2 scores over male and female data points. We observed that with the initial dataset there was a fractional

difference of 0.088, while with the unbiased data, that fractional difference was reduced to 0.022, where a lower fractional difference represents a regressor predicting the salary for men and women more consistently.

Systems Integration PoC: As an integration with a DBMS system, we create a thin web based interface based on *postgres*. For this PoC, we use the *TexasTribune* dataset. The *postgres* database is created with index on the numeric attribute - salary. For the PoC, we compare the $\mathcal{O}(n^2)$ naive algorithm with the jump pointer algorithm.

For the naive algorithm, the time is computed for processing the query. For the jump pointer algorithm, there is a pre-processing phase where we calculate the cumulative sum and jump pointers and populate a separate table with these details. We measure the time taken for naive query, pre-processing to create jump pointers and *SPQA* query processing. Average time taken by the pre-processing algorithm taken by *SPQA* 0.043 second. We run around 500 randomly sampled queries and measured the time taken by each of these queries. While average time taken by *SPQA* algorithm is 0.0054 seconds, average time taken by naive algorithm is 6.938 seconds. This shows the efficienly of *SPQA* when integrated with DBMS.

3.5.3 Performance of *SPQA* and weighted *SPQA*

The performance of *SPQA* depends on the disparity of the input query. For these series of experiments, we measure the amount of time taken by the *SPQA* algorithm when provided with an input query. The experiment were averaged over five runs of *SPQA* algorithm for more reliable time measurements. For these experiments, the*TexasTribune* dataset was used. As *TexasTribune* has 149,481 records with almost the same number of `male` as `female` we use `gender` as the sensitive attribute for the unweighted case. For the weighted case, we use `race` as the sensitive attribute while using `white` (majority) and `non-white` (minority) as the demographic groups. There are a total of $67,142$

`white` records and $82,339$ `non-white` records. As the ratio of `white` to `non-white` is very near to $4:5$ (0.815), we use weights of 4 and 5 for the weighted *SPQA*. For the SP queries, we use salary as the attribute for range predicates. A large part of the records of the database (95.6%) have a salary less than $100,000$. Hence, to set the range query boundary, we pick all points in multiples of $5,000$ between $5,000$ and $100,000$ as start and end points. For every query, time taken by *SPQA* algorithm is measured along with the input query's disparity. For both the weighted and unweighted case, the ε value was set to 500 for this set of experiments.

For the COMPASS dataset, the risk score varied between $-4.79 - 51.0$. Starting of the input range was generated between -4.79 and 51.0 with multiples of 3.0. Ending of the input range varied from the starting in multiples of 3.0. The *COMPASS* dataset has around 21K Caucasian (blue) and 39K non-Caucasian (red) warranting a ratio of 2:1.

Figures 3.16 and 3.17 show the scatter plot for amount of time for *SPQA* against the input query's disparity value for the *Texas Tribune* dataset and figure 3.18 shows the scatter plot for the weighted *SPQA* queries run on the *COMPASS* dataset. As a baseline, we ran IBFS for single range predicate (weighted and unweighted). On average, IBFS ran about 3 orders of magnitude slower than the jump pointer algorithm. The plots show a linear scaling of time with the input query's disparity. This empirically validates the running time of both the unweighted and weighted *SPQA* algorithms.

3.5.4 Performance evaluation of MP algorithms

For the multi-predicate range query evaluation, we use *UrbanGB* and *Uniform* datasets. For the experiments, $10,000$ records from the *UrbanGB* dataset have been used. There a total of 3,088 *blues* and 6,912 *reds*. Hence, we use a weight of 2 for the *blue* records and 1 for the *red* records. Latitude and longitude attributes from the database were used to form the range queries. The latitude values in the 10K records varied from -0.507015 to

0.297345 and the longitude values varied from 51.306584 to 51.660974. *Uniform* dataset consists of 10K records uniformly sampled from within a square of length $1,000$. The sensitive attribute is made of an almost equal number of blues and reds and hence we use a weight of 1 for both these colors. For all sets of experiments, a disparity of 5% between demographic groups is allowed which is indicated by the value of ε used. The baseline algorithm restricts the search to a bounding box and performs a thorough search of all the ranges in this rectangle. In this section we compare the run times of IBFSMP and baseline algorithms.

3.5.4.1 Effect of input query size on the run time

Query size is an important measure as it impacts the performance of our algorithms. It impacts the number of points that are being added or removed to find a fair range. While there are many other factors which may impact the performance of the query, we choose many queries in each bucket and repeat our experiments with each of these and aggregate our results to reduce the impact of other factors. For our experiments, query sizes vary from 200 to 1400 that are bucketized with intervals of 200. That is, if for example a query result contains 558 points, it fall in the query size bucket of 400-600. Each bucket has 20 range queries sampled for the experiment using rejection sampling. The input queries are chosen from different buckets using rejection sampling based on the points which satisfy the query. In each bucket, we execute 30 queries each and aggregate the results for comparison. The average run-time is measured for both the algorithms under different bucket sizes. For the queries in each bucket, the mean time taken during the run of the IBFSMP algorithm for *UrbanGB* and *Uniform* datasets is shown in Figure 3.20 and Figure 3.21, respectively. In case of the baseline algorithm which restricts the search space, experimental results for the bucket 200-400 show a mean of 697.4 seconds and 557.1 seconds for the *UrbanGB* and *Uniform* datasets respectively. The cases for the larger bucket sizes

did not complete even after 3 *hours* and thus are not tabulated. The aggregated values of mean show that IBFSMP outperforms the baseline algorithm by orders of magnitude. For each individual query, the IBFSMP outperforms the baseline algorithm. But, due to space constraints, the details of each query executed is not included.

IBFSMP shows similar trend when run in higher dimensions. The experiments with 3 range predicates show that the time taken grows with input range size as seen in figure 3.19. One difference we observed was that the larger part of the computation was spent in computing skylines than in lower dimensions. One can observe the increase in run times between the two and dimension charts even for small input sizes.

3.5.4.2 Effect of input query size on the number of ranges explored

For the next set of experiments, we evaluate the effect of input query size on the ranges explored. As the algorithm has a dependence on many factors, we choose input query arbitrarily to analyse the impact of input query size, *Jaccard similarity* from the input range on running time. The number of ranges explored by the IBFSMP algorithm is measured as a parameter along with the time taken. The input queries are chosen with different output sizes based on the number of points which satisfy the query. Query sizes vary from 200 to 1400 that are bucketized with intervals of 200. That is, if for example a query result contains 558 points, it fall in the query size bucket of 400-600. Each bucket has 20 range queries sampled for the experiment using rejection sampling. We use the same set of queries with different sizes, bucketized with intervals of 200, as in our previous experiment.

Intervals of 200 are used to chose buckets starting from 200 up to 1600. We arbitrarily choose multi-predicate queries with various input query sizes, and execute the IBFSMP algorithm. The queries are placed into buckets based on input range size.

For the queries in each bucket, mean and standard deviation of the run time and ranges explored during the run of the IBFSMP algorithm for *UrbanGB* and *Uniform* datasets is measured and tabulated in Table 3.2 and Table 3.3 respectively. As indicated by the mean values in Tables 3.2 and 3.3, the run-time increases with increase in query size. The number of ranges explored by *IBFSMP* for *UrbanGB* and *Uniform* datasets are shown in Figure 3.22 and Figure 3.23 respectively. As can be seen in the figures, the number of ranges explored grows significantly with increase in query size. As the amount of time taken is proportional to the number of ranges, the mean time taken grows with the number of ranges explored as can be seen in the both the figures.

We did not include the performance of *BFSMP* in the table as *IBFSMP* significantly outperformed it in all cases. For example, while *IBFSMP* on average required only 1.1 seconds for the 200-400 bucket in *Uniform* dataset, *BFSMP* on average took 11.1 seconds. That is because, on average, it explored 145K ranges (SD=394K) while this number was 15K for *IBFSMP*. Similarly, for *UrbanGB* dataset, *BFSMP* on average took 15.3 seconds while *IBFSMP* took 3.6 seconds for this experiment. The reason was that *BFSMP* on average explored 199K ranges , while this number was 51K for *IBFSMP*. In all cases, *IBFSMP* outperformed *BFSMP* for every query. In case of *BFSMP* algorithm, the amount of time taken for the 200-400 bucket has a mean of 11.1 seconds and standard deviation (SD) of 31.5 for the run-time for the *Uniform* dataset. a mean of 145314.9 and standard deviation of 394289.7 with the sampled queries from *Uniform* dataset. With the sampled queries from the *UrbanGB* dataset, *BFSMP* consumes 15.3 seconds of time on average with a SD of 22. The average number of ranges explored for this dataset stand at 199K with a SD of 275K. These results demonstrate the improvements that *IBFSMP* has over *BFSMP*.

Query size	μ time	σ time	μ #Ranges	σ #Ranges
200-400	3.6	10.3	51334.2	147597.0
400-600	9.5	18.9	138860.3	276703.1
600-800	121.4	288.7	1728136.6	4233843.5
800-1000	331.1	685.3	4254471.8	8698502.4
1000-1200	419.0	616.0	5460567.3	8111981.2
1200-1400	1060.2	1942.2	13468987.3	24934174.2

Table 3.2: Comparison of query run time (sec.) for various input range set sizes using IBFSMP for *UrbanGB* dataset

Query size	μ time	σ time	μ #Ranges	σ #Ranges
200-400	1.1	2.8	15495.1	39389.3
400-600	37.1	101.3	499156.7	1343447.2
600-800	138.1	292.3	1806451.6	3807519.0
800-1000	525.3	902.6	6974328.9	11961045.4
1000-1200	940.4	1919.6	12298647.0	25016749.7
1200-1400	1279.9	3123.1	17208695.7	41803978.6

Table 3.3: Comparison of query run time (sec.) for various input range set sizes using IBFSMP for *Uniform* dataset

3.5.5 Comparison with coverage based algorithms

Coverage based algorithms [79, 80] output a range query by modifying the given query such that at least a given number of items from each sensitive group are present. Note that coverage based *CRBase* makes use of a threshold value for each demographic group. On the other hand demographic parity measure is based on the notion of weighted difference between the demographic groups. As a range expands by addition of the minority group, items from the majority group are also added which may increase the disparity. To find ranges which satisfy demographic parity measure, we make use of numerous values of threshold to find different ranges that satisfy demographic parity fairness measure. Among these fair ranges, we record the ones which have the most similarity.

We have run these experiments with the *uniform* and *Urban GB* datasets. We measure the fair ranges from *CRBase* algorithm and record the one which has the most similarity.

CRBase algorithm was run with 4, 8, 16 and 32 bins. *CRBase* algorithm produces a fair range 33.9% of the time with the *Uniform* dataset. We measure the error by computing $1-CRSim/Optimal$, where $CRSim$ is the similarity of the $CRBase$ algorithm where as the $Optimal$ is the similarity of the optimal range. For the ranges where *CRBase* algorithm does not satisfy the fairness or similarity criteria we mark $CRSim$ as 0. An average error measure of 0 means that optimal range is always obtained, while an error of 1 means that the range produced never satisfies the criteria. The error produced by *CRBase* is 0.682 on average. For the *UrbanGB* dataset, we used a weighted fairness measure. $CRBase$ was able to produce a fair range for only 3 out of 120 sample ranges. The experiment shows that the two optimization problems and hence, the solutions are different in nature.

3.5.6 Summary of experimental results

At a high level, the experiments verify the efficiency and efficacy of our methods. Firstly, we empirically show the efficiency of the unweighted and weighted *SPQA* algorithm. Secondly, for a wide spectrum of range queries, we show that BFS algorithms outperform the baseline algorithm by orders of magnitude. Moreover, IBFSMP outperformed BFSMP since it explored far less number of ranges before it found the optimal solution. Finally, we also show the effect of input range size on IBFSMP, the larger the set size the more the time taken by IBFSMP to find the most similar fair range.

3.6 Related work

Query answering: Efficiency is critical requirement in query answering. A large amount of research has focused on different aspects of query answering over the past few decades. One of the popular methods that has been explored is the query answering using views [94–97], where the goal is to efficiently answer a query using a set of previously materialised views on the database. Srivastava et. al. [96] answer SQL queries with grouping and

aggregation in the presence of multi-set tables by detecting when the information existing in a view is sufficient to answer a query. Chaudhuri et. al. [97] solve the problem of optimising queries in the presence of materialised views. Approximately answering queries has also been studied extensively in many works [98–102]. While there have been many works in the area of query answering, none of these works can be modified to incorporate fairness into them.

Fairness: Reducing racial disparities has recently been a key research [47, 55, 69, 103–107]. Feldman et. al. [103] propose methods to make make data unbiased by modifying the fields/attributes. Hajian et. al. [104] propose a data transformation that can consider combination of attributes to perform data transformation. While [103, 104] perform data modification, we do not modify any data point to remove bias from data instead we provide the *nearest* fair data points to work with. While [105, 106] propose methods that learn to produce fair machine learning models from the given data they do not eliminate bias from the data itself.

Query reformulation: Salimi et. al. [108] created a system for detecting statistical dependencies which impact the result of the original query. In their work, they reformulate queries by modifying the attributes queried to account for these statistical anomalies. In other works [78–80], a system has been proposed which minimally relaxes a query to provide coverage for sensitive groups. The objective of [79, 80] is to modify the original query satisfying demographic coverage constraints (minimum number of items from a each group). Coverage constraint satisfaction involves only relaxing the constraints, which may not help in reducing disparity. Note that, trying to satisfy coverage can further *increase* the disparity between the groups. Similar to these works [78–80], our algorithms also modify the original query. However, unlike existing work, our objective is to find queries (i) similar to the initial query that (ii) satisfy a disparity (unfairness) threshold on counts from different demographic groups.

3.7 Discussion and future work

Fairness model There are many fairness models which one can consider when the data contains demographic sensitive attributes. In this paper, we have used the fairness model in which objects from different demographic groups have equal chances of being selected in the output set. There are other fairness models like the demographic parity based on ratio which we consider for future work. Such a fairness model has the form, $\delta \geq C_r/C_b \geq \delta^{-1}$.

Operators In our current work, we have considered a conjunctive operator to join different predicates. Query models like *SQL* support operators like *NOT* and *OR*. Note that the subset of operations (*OR* and *AND*) would allow the output queries to allow for union of ranges. We consider the addition of these different operators to the query model as an extension of the paper for future work.

All nearest fair ranges: The declarative query in 3.2 can have multiple range queries which are *equally* near while satisfying fairness constraints. An interesting area of research would be to enumerate all these nearest fair ranges.

Demographic group based extensions: Fairness problems based on binary demographic groups have been well studied [109–112] for various applications like clustering, PCA and other optimisation problems. We note that a significant portion of existing literature fairness and its definitions consider binary cases, as there usually is an advantaged/majority v.s. disadvantaged/minority group(e.g. COMPAS dataset(black vs non-black), adult and salary dataset(female vs male)). While binary case for fairness is an important case, extensions to these problems are valuable in many scenarios. We consider extending the fair range queries to non-binary demographic groups and demographic parity constraints on multiple sensitive attributes as future work.

3.8 Final remarks

In this paper, we initiated research on integrating fairness into data management systems. As our first attempt, we focused on selection bias in range queries, and proposed efficient algorithms. In particular, we proposed a sub-linear algorithm for single-predicate range queries and two algorithms based modeling the problem as graph traversal for multi-predicate range queries. Besides theoretical analysis, comprehensive experiments verified efficiency and effectiveness of our proposal.

We consider the extensive research required for the full integration of fairness, including a comprehensive database and query model with a broad coverage of bias, fairness notions, and a broad range of SQL operators as well as designing more efficient algorithms, for our future work.

CHAPTER 4

Shapley Values for Explanation in Two-sided Matching Applications

In this paper, we initiate research in explaining matchings. In particular, we consider the large-scale two-sided matching applications where preferences of the users are specified as (ranking) functions over a set of attributes and matching recommendations are derived as top-k. We consider multiple natural explanation questions, concerning the users of these systems. Observing the competitive nature of these environments, we propose multiple Shapley-based approaches for explanation. Besides exact algorithms, we propose a sampling-based approximation algorithm with provable guarantees to overcome the combinatorial complexity of the exact Shapley computation. Our extensive experiments on real-world and synthetic data sets validate the usefulness of our proposal and confirm the efficiency and accuracy of our algorithms.

4.1 Introduction

Beyond its traditional use [113–116], matching has been a core functionality of many of the modern two-sided online platforms [117–119], including dating applications such as `Tinder`, `OkCupid`, and `Bumble`[1], employment-oriented platforms such as `Linkedin`, `Indeed`, and `Zip- Recruiter`[2], and many more. The two-sided matching platforms provide matching recommendations between two types of stakeholders (users). To better explain the matchings, let us consider Example 3 as a running example across the paper.

[1] `tinder.com`; `okcupid.com`; `bumble.com`

[2] `linkedin.com`; `indeed.com`; `ziprecruiter.com`

Example 3. *(Part 1) Consider a two-sided employment-matching application with two types of users, job candidates and human resource (HR) users. The application provides matching recommendations to both job candidates and HR users. For example, an HR user who looks potential candidates for interview (either directly or indirectly) specifies a set of criteria and their preferences. Then the application returns a set of potential job candidates to the HR. It similarly finds matching job opportunities for the candidates.*

Matching in two-sided platforms can be modeled as a bipartite graph[3] where users on one side are matched to the users on the other side. For instance, Figure 4.1 models Example 3 as a bipartite graph, where job candidates and HR users are specified as red and blue nodes, respectively, while an edge $t_i \rightarrow t_j$ means that t_i has been recommended as a match for t_j. The application usually identifies the list of potential matches for each user (called *match list* in this paper) based on their *"individual preferences"*. While classic matching problems assume the each party *explicitly* specifies their preference as a ranking over the entire set on the other side, this assumption is not feasible for modern matching applications, simply due to their numerous number of users, the short attention span of users, and in some cases privacy considerations. As a result, the preferences are instead *implicitly* specified. That is, every user is associated with a set of *attributes* (aka features), and the preference of each user is defined as a function over the attributes of the other-side parties. The preference functions are either learned or specified by the users. The matching application uses the preference function to shortlist a limited list of candidates (the top-k) based on a user's preference function.

Lack of adequate *explanations* in these systems is a major issue where the users, impacted by the decision, may be interested to know more insights about the matching and

[3]As we shall explain in § 4.5, matching has many different formulations, properties, and applications. In this paper, our scope is limited only to bipartite many-many matching for two-sided online platforms.

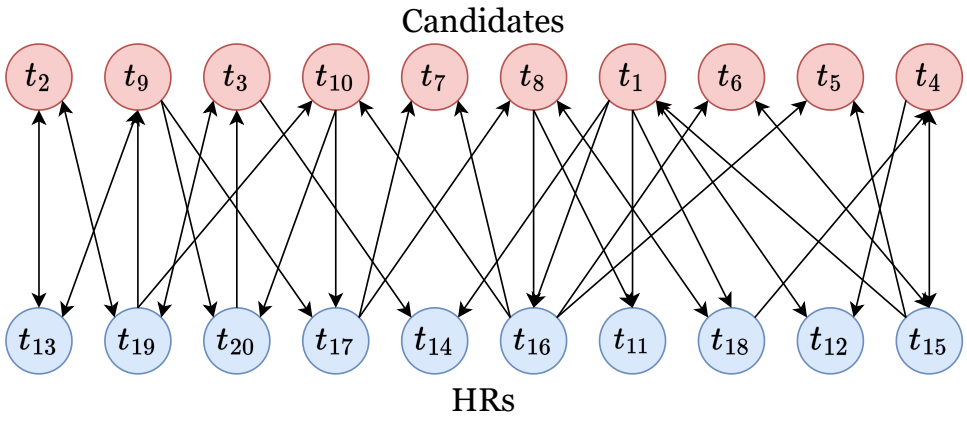


Figure 4.1: Illustration of the matching in Example 3.

	Python	R	PHP	JS
Candidate Attribute Values	2	2	1	1
HR Attribute Values	2	2	0	3
HR Ranking Weights	0.11	0.11	0.67	0.11
Shapley Values	0	-0.16	0.83	0.33

Table 4.1: The generated explanation for why Candidate t_3 is not in the Top-K of HR t_{20}

why they (or others) do/do-not appear in certain match lists. To further elaborate this, let us continue with Example 3.

EXAMPLE 3. *(Part 2) Looking back at Example 3-Part 1, suppose four attributes are considered for matching: Python, R, PHP, and Javascript. Each candidate has a skill level for each of these attributes, as does each HR, describing the nature of the job. Additionally, each Candidate and HR has an importance weight associated with each attribute, forming their preference as a linear function, while* $k = 2$. *Suppose candidate* t_3 *wants an explanations based on their matchings , which were either disappointing or pleasing. Currently the system provides no explanation for any of the four scenarios below, for which* t_3 *may be curious:*

	Python	R	PHP	JS
Candidate Attribute Values	2	2	1	1
HR Attribute Values	3	2	0	2
HR Ranking Weights	0.11	0.44	0.33	0.11
Shapley Values	0.25	0.91	-0.08	-0.08

Table 4.2: The generated explanation for why Candidate t_3 is in the Top-K of HR t_{19}

	Python	R	PHP	JS
Candidate Attribute Values	2	2	1	1
Candidate Ranking Weights	0.84	0.02	0.06	0.08
Shapley Values	0.61	0.28	0.0	0.11

Table 4.3: The generated explanation for why Candidate t_3's Top-K looks the way it does.

1. *t_3 was disappointed to not make it to the top-2 of HR t_{20}. An explanation would provide value to t_3.*
2. *t_3 was happy they made it into the top-2 of HR t_{19}. t_3 is eager to know as to what sets them apart from the rest.*
3. *Top-2 of t_3 consisted of $\{t_{19}, t_{20}\}$. An explanation for why these HRs are good recommendations would be useful.*
4. *Candidate t_3 made it into the top-2 of HRs t_{14}, t_{19}. t_3 wants to know why they made it into these specific top-2s.*

The lack of answers for these questions prevents understanding for Candidate t_3.

In this work, we create a framework to provide explanations to various queries which are commonly encountered by the users of two-sided matching applications. To the best of our knowledge, this is *the first paper in explaining matchings*. In particular, we observe that top-k (ranking) problem is inherently competitive. As a result, the outcome (score) of a preference function is not enough to realize if a user appears in a match list or not. What matters in these settings is the relative position (rank) of a user with respect to other users who "compete" for the top-k positions. Such competitive environments are naturally

	Python	R	PHP	JS
Candidate Attribute Values	2	2	1	1
Shapley Values	$0.48\bar{3}$	$0.23\bar{3}$	$0.33\bar{3}$	0.15

Table 4.4: The generated explanation for why Candidate t_3 appears in the Top-K it appears in.

explainable by *Shapley values* [120] – a game theory concept that identifies the contribution of each player (each attribute in our context) for deriving an outcome (e.g., a top-k match list). Shapley values have been proven repeatedly to solve explainability problems across different context [121–123].

Based on this observation, we consider Shapley values as the core of our system for our explanations. We consider a set of possible explanation queries, and provide Shapely-based approaches to answer them. Exact computation of Shapley values is a combinatorially hard problem, requiring algorithms that are exponential to the number of players. On the other hand, users might find accurate approximation of the values appropriate for explanation. Our system enables explanations as demonstrated in Example 3 (Part 3).

EXAMPLE 3. *(Part 3) Using Shapley-based methods, we generate an explanation for each of the previous queries. These explanations take the form of a value for each of the skills and other features on which the matching is generated. A high value indicates that the feature was largely responsible for each of the four cases. The individual can then be provided with a general explanation as to what about them resulted in the various outcomes.* [4]

1. *(From Table 4.1) t_3 is informed that they failed to be matched to HR t_{20} because they were not a good match with their PHP skills. However, based on their R skills alone, they would have been a good match for the job.*

[4]Due to space constraints, the example dataset is provided in the appendix.

2. *(From Table 4.2) t_3 can be told that the reason they were in the top-2 of HR t_{19} is because they were an excellent match on R skills. They can also be informed that their Python skills were less but still beneficial towards the matching as well.*
3. *(From Table 4.3) Top-2 of t_3 consisted of $\{t_{19}, t_{20}\}$. They are informed that this is largely because of the Python requirements of the HRs, and less so because of the R and JavaScript requirements.*
4. *(From Table 4.4) Candidate t_3 is provided the information that they made it into the Top-2 of HRs t_{14}, t_{19} because of their Python skills first, then their PHP skills, then their R skills, and finally their JavaScript skills, with each contributing slightly less than the previous.*

Summary of contributions. In summary, our contributions are a follows:

- In this paper, we initiate a study of a novel problem - that of providing explanations for matching and top-k recommendation systems. To the best of our knowledge, this paper is *the first to study explanation for matching.*
- We propose four explainability problems on the top-k matching model, which often arises from user's curiosity.
- Considering the competitive nature of our matching problem, we propose a Shapley-based approach to explain the queries and provide run time analysis for each of the problems. We show the need for alternate methods, as the run time is bound exponentially by the number of dimensions.
- We propose a sampling based approach to compute approximate Shapley values and prove guarantees on the trade-offs between number of samples and error rate. We adapt KernelSHAP as a practical heuristic to our problem.
- Extensive experimental analysis are provided for the various query settings and error guarantees, and our methods are evaluated in the real world via a user study.

4.2 Preliminaries

Data model: We consider a dataset $\mathbb{D}$ with a Boolean attribute for matching (*blue* and *red*), d numeric attributes $\mathbb{A} = \{A_1 \cdots A_d\}$. The dataset $\mathbb{D}$ consists of n entities t_1 to t_n, with a sizeable number of *blue*s and *red*s. We use the notation $t_i[j]$ to refer to the value of the attribute A_j for the entity t_i. Similarly, we use $t_i[m]$ to refer to the type of t_i, i.e. the value of the Boolean matching attribute on t_i. The values in the dataset $\mathbb{D}$ represent the scores of each entity for various attributes, which are used in the matching process.

Ranking functions: Each entity $t_i \in \mathbb{D}$ is associated with a ranking (aka preference or scoring) function that maps any given entity to a real valued score $f : \mathbb{R}^d \rightarrow \mathbb{R}^+$. The ranking function is used to express the preference of an entity during the matching process. As a hard criteria for matching, an entity with blue matching attribute only wants to match with an entity with red matching attribute and vice-versa. For instance, job recruiters (*blue*) and job seekers (*red*) are trying to match in a job matching scenario. The list of ranking functions F consists of n ranking functions corresponding to each of the entities. An entity t_i's ranking function is referred to as f_i throughout the paper. Some widely used types of ranking functions are linear, nearest-neighbor, and monotonic [124]. The techniques proposed in this paper are agnostic to the choice of ranking function. In this paper, the time taken to compute a score by the ranking function is referred to as C. An important requirement for the ranking functions we consider in this paper is the masking property. That is, given a ranking function, one can tune the function not to consider masked attributes when computing the scores. Masking is usually possible (including in linear, nearest-neighbor, and monotonic functions) by setting the values of the masked attributes as zero or null across all entities. That is, given an attribute A_j to mask, one can set $t_i[j] = 0, \forall t_i \in \mathbb{D}$. The set of attributes M which are set to 0s are known as masked attributes. The purpose of the masking function is to generate the outcome of a subset of non-masked attributes. In later sections, we explain the importance of the masking property for explanations.

Match list: As there are a large number of entities, any entity would like to see a small relevant set of entities as a potential match. Ranking functions are used to rank all the entities in the dataset belonging to the opposite matching attribute. In this work, top-k entries are shown for each entity using the rank of the entities as potential match. That is, given the ranking function f_i for an entity t_i, scores are assigned to all the entities $\{t_j \in \mathbb{D} \mid t_j[m] \neq t_i[m]\}$ using f_i. Those entities are then ranked and the top-k are chosen to be shown to t_i.

These top-k entries, known as a match list, are used to express the recommendations for matching. We denote a match list by l_i, such that each list consists of k entities. Given an entity t_i, let f_i^k represent the score for the k^{th} ranked entity using the scoring function f_i. Given an entity t_i,the match list can be mathematically expressed as,

$$l_i = \{t_j \mid f_i(t_j) \geq f_i^k \; and \; t_j[m] \neq t_i[m]\} \quad s.t. \qquad |l_i| = k$$

When a mask M is applied to obtain the top-k, the match list is represented as $l_i(M)$. Note that the value of k depends on the application and is not restricted to a fixed value for any individual. Without loss of generality, in this paper we assume a consistent k across all entities' ranking functions.

4.2.1 Problem definition

Our objective in this paper is to *increase responsibility in matching systems* by providing individuals with explanations about the matching and further information regarding why the matches occurred the way they did. The matching model for which explanations are being provided is called the "Top-k matching model", which is formally defined as:

Definition 4.2.1 (Top-k matching model)**.** *Given a data-set of entities $\mathbb{D}$, an integer k and ranking function for each of the entities, determine the match lists of each of the entities using the top-k from the ranked list.*

When a match list l_i is provided for a problem instance, it is not always immediately clear how the various attributes contributed to the outcome. In order to solve this problem, an *explanation* is provided which identifies the role of different attributes in producing the outcome.

Definition 4.2.2 (Explanation)**.** *Given an output of a function $y = f(A_1, A_2, ...A_d)$, determine the impact on producing the overall value y of each attribute A_i when A_i is included as a parameter.*

We now transition into explainability problems that arise from the curiosity of entities in the matching setting.

Point Queries: The first two types of explanations are when an entity queries about their presence or absence from another entity's match list. This type of explanation is simply called a "point query".

In Example 3, Candidate t_3 finds HR t_{20} in their match list, but t_3 and t_{20} were not a match due to t_3 not being present in t_{20}'s match list. In such a scenario, the curious and disappointed Candidate, t_3 would want an explanation for: *why t_3 is not present in t_{20}'s match list?* Such a problem/scenario where Candidate t_3 would like an explanation for why they were not present in t_{20}'s list, can be formally defined as follows:

PQ-NOTMATCH : *Given a dataset of entities $\mathbb{D}$ and entities t_i and t_j determine the contribution of attributes $A_1, A_2, \ldots, A_d$ in t_j* ***not appearing*** *in the match list of t_i.*

Conversely, again using Example 3, Candidate t_1 may also be interested in what factors made them present in the match list of HR t_{19}. In this scenario, the intrigued Candidate,

seeking to understand the scenario, might request for an explanation as to: *why t_1 is present in t_{19}'s match list?* This problem is formally defined as:

> **PQ-MATCH :** *Given a data-set of entities $\mathbb{D}$ and entities t_i and t_j determine the contribution of attributes $A_1, A_2, \ldots, A_d$ to t_j* ***appearing*** *in the match list of t_i.*

The next two queries deal with set based properties and hence, we term these as *Set queries*.

Set queries: A different type of query that Candidate t_3 from Example 3 might ask pertains to the match list. The candidate t_3 might also be curious, based on their ranking function f_3, *why the match list l_3 was generated some way*, either because they are happy or disappointed with the list that was provided to them. Formally, the query, *why does an entity's match list look a certain way*, can be formulated as,

> **SQ-SINGLE :** *Given a dataset of entities $\mathbb{D}$ and an entity t_i with a match list of l_i, determine the contribution of attributes $A_1, A_2, \ldots, A_d$ to t_i's match list looking like l_i.*

Finally, in addition to point queries and understanding their own ranking, Candidate t_3 from Example 3 may be concerned with the outcome of being ranked by others. Particularly, they may be interested in what attributes about them influence the set of match lists in which they appear.

Such an explanation can help Candidate t_3 understand the factors responsible for their current matches, for both cases where they are either pleased or displeased with the results. If Candidate t_3 is overall displeased with the HRs who match with them, knowing what attributes are responsible for this outcome can be informative. Equally, if they are consistently pleased with the HRs with whom they match, knowing the attributes that contribute to this outcome can provide insight into what went right. Formally, the problem

to determine factors that influence an entity appearing in the match list of other entities is defined as:

SQ-MULTIPLE : *Given a dataset $\mathbb{D}$ of entities and an entity t_i which is present in the match list of a set of T entities, determine the contribution of attributes $A_1, A_2, \ldots, A_d$ for t_i appearing in the match lists of T entities.*

4.3 Shapley value based solution

We start this section with discussing why Shapley values are suitable for explaining top-k and bipartite matching problems, followed by an overview of Shapley technique. Next, we show the transformation of our problem to Shapley, and discuss its scalability issue for higher dimensions. Then, we propose a sampling-based approximation technique with provable guarantees. We conclude the section by adapting KernelSHAP as a practical heuristic to solve our problem.

4.3.1 Why Shapley?

Scoring and ranking functions have been well-studied topics, while a major focus has been on explaining scoring functions. Even though there are similarities between scoring and ranking, their underlying requirements bring out drastic differences [47]. The score of an entity only depends on the (attributes) of the entity itself. On the other hand, the rank of an entity depends not only on its attribute values, but also on the attribute values of the other entities in the dataset. Consider an entity t in a dataset $\mathbb{D}$. $f(t)$ assigns a score to t. However, to find out the rank of t, one first needs to compute $f(t')$ for every entity in $\mathbb{D}$, and then find the position of t in the sorted list of entities based on f.

Because our problem is based on ranking, methods for scoring are no longer applicable for many queries. For example, in a (score-based) classification task, the attribute with the highest weight in the scoring function would be the most impactful. However, in a

ranking problem, the score values are important only in comparison with the score of other entities. As a result, it is not enough if the score of an entity is high; what matters is that it is higher than other entities in the dataset. In this situation, an attribute A_i with a *low weight* in the scoring function can become important if dataset entities have a *high variance* on it. This is because the ranking is determined by *competition* between the entities, which is not captured simply by the score on a single entity. Let us illustrate this with the following toy example.

Example 4. *(Part 1) Consider a sample dataset* $\mathbb{D}$ *with* 3 *attributes* $\mathbb{A} = \{A_1, A_2, A_3\}$ *and* 5 *entities, shown in Table 4.5. The entities belong to the same matching group. Let the scoring function for an entity belonging to the opposite group be the linear function* $f(t_i) = 5t_i[1] + 4t_i[2] + t_i[3]$. *For this example, let the value of* k *be equal to* 2. *The linear ranking function scores the entities* t_1, t_2, t_3, t_4 *and* t_5 *with scores* 62 , 25.4, 81.8, 54.5, *and* 81.7, *respectively. The entities ranked by their scores are* t_3, t_5, t_1, t_4 *and* t_2. *Hence, the match list looks like* $\{t_3, t_5\}$ *after the ranking function is applied. By looking at the entities, the entity whose ranking function is used for ranking might question* why is t_1 not in the top-k?

An explanation model based on scoring functions would emphasize on weights of the given query entity to obtain the contribution of the different attributes. The weights for $A_1(5)$ *is the highest, followed by* $A_2(4)$ *and* $A_3(1)$. *Looking at these weights, a scoring based explanation approach would conclude that either of* A_1 *or* A_2 *might be responsible for the query result. But upon masking attributes* $\{A_1\}$, $\{A_2\}$ *or* $\{A_1, A_2\}$ *and re-ranking one can observe that* t_3 *and* t_5 *are always scored higher than* t_1. *On the other hand upon masking attribute* $\{A_3\}$, *one can see that* t_1 *enters the match list. This illustrates the combinatorial feature importance in explaining ranking.*

	t_1	t_2	t_3	t_4	t_5	weights
A_1	10	1.2	9	9.5	10.1	5
A_2	1	1.1	1.2	1.5	1.8	4
A_3	8	15	32	1	24	1
$f(t_i)$	62	25.4	81.8	54.5	81.7	

Table 4.5: (Example 4) A sample dataset $\mathbb{D}$ with three attributes A_1, A_2 and A_3, and 5 entities.

Since ranking is based on the competition between entities, it is natural to map our problem using *coalitional game theory*. In a coalitional game, different players of a coalition in a competitive game compete for utility. Accordingly, we utilize coalitional game theory to capture the importance of different attributes.

Shapley value [120] is a concept in coalitional game theory which allows one to compute the importance of each of the players in the game to the outcome. Each game contains a set of n players and a utility function $v : 2^n \rightarrow \mathbb{R}$, which determines the worth of the subset of players. Note that the utility of an empty set of players is 0, $v(\emptyset) = 0$ as the value represents the worth of an empty set ($\emptyset$) of players. The average contribution by each of the players to the outcome of the coalitional game can be defined for each player i as the Shapley value Sh_i. Shapley value for the game is given by,

$$Sh_i = \frac{1}{n!} \sum_{X \in Sym(N)} (v(Pre_i^X \cup i) - v(Pre_i^X)) \tag{4.1}$$

where $Sym(N)$ represents the Symmetric group of set N, and Pre_i^X represents the players preceding i in permutation X. Equation 4.1 captures the marginal increase in contribution of a player i to its predecessors in permutation X.

An alternate form of Equation 4.1 can be used to compute the Shapley values. For a given subset set S of players, there exist $|S|!(n - |S| - 1)!$ permutations each of which

have the same utility value whose re-computation is avoided when using the alternate form given below,

$$Sh_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(n - |S| - 1)!}{(n!)} (v(S \cup \{i\}) - v(S)) \tag{4.2}$$

Even though the formula is helpful in reducing the complexity from $n!$ to 2^n, the computation is still exponential in nature. Accordingly, in practice, as the number of attributes grows this process tends to become infeasible.

4.3.2 Mapping Shapley value to matching

In Example 4, we argued that ideas such as using attribute-weights in the scoring function may not be effective for explaining top-k matchings. Alternatively, we propose to map attributes $\{A_1, A_2, \cdots, A_d\}$ as players of a coalitional game, compute the contribution of each attribute using the concept of Shapley values, and use these values to explain matchings. Indeed, an attribute's contribution is query dependent. That is, for different explainability queries, its contribution may vary from one query to the other. For instance, in Example 3, the contribution of the attribute (skill) *Java* for the query *"why is t_3 in the match list of t_{19}"* is different from *"why is t_3 not in the match list of t_{20}"*.

We propose a utility function $v : 2^d \rightarrow \mathbb{R}$ that maps a subset of attributes $S \subseteq \mathbb{A}$ to its utility. In our design we require to compute top-k matchings for a subset of attributes. Accordingly, the masking property mentioned in Section 4.2 can effectively be used to remove the impact of some of the attributes. Without the impact, it is equivalent to computing the utility on a subset of attributes. If there are no attributes in the subset, then a top-k cannot be computed. Accordingly, without the top-k, the utility value of all the four queries is 0 or in other words $v(\emptyset) = 0$. We provide design details of the utility function for different queries in Section 4.3.3.

4.3.3 Shapley value in matching

One of our contributions is to transform each of our explanation problems to Shapley value computation problem and define a utility function over the set of attributes. This section provides the details for the Shapley value based approach applied to the four types of explainability queries defined in Section 4.2.1.

PQ-NOTMATCH: Consider the scenario when a dataset of entities $\mathbb{D}$ and two entities t_i and t_j are provided for a query type of PQ-NOTMATCH. The attributes (players) responsible for t_j not appearing in t_i's match list would be valued high.

To reflect this we define a value function that returns a 1 (0 resp.) based on if an the entity t_j's absence (presence resp.) in t_i's match list l_i for a subset of attributes S. In this case, the set of attributes has contributed to a failed match and hence a value (reward) of 1. On the flip side, the utility function returns 0 if t_j is present in l_i for a given subset of attributes S. To explain further with Example 3 (1), the utility function would return 1 if t_{20} was not present in l_3^* and 0 otherwise. Assuming l_i^* is the match list computed based on strictly the attributes in set S, the utility function v can thus be defined as,

$$v(S) = \begin{cases} 0, & \text{if } S = \emptyset \textbf{ or } t_j \in l_i^* \\ 1, & \text{if } t_j \notin l_i^* \end{cases} \tag{4.3}$$

For Example 3 question (1), and the query PQ-NOTMATCH, the results are shown in Table 4.1. The exact algorithm generated the Shapley values 0, -0.16, 0.83, and 0.33 for the attributes Python, R, PHP, and JS respectively. As discussed earlier, this clearly shows that PHP is the largest contributor to why Candidate t_3 was not in HR t_{20}'s match list. Additionally, since Python has a negative value, the candidate's Python skill would cause the individual to appear in the match list.

PQ-MATCH: The next explainability query which can be calculated using Shapley value is the second query PQ-MATCH. A similar approach that we have seen so far can be extended to this scenario.

In the PQ-MATCH utility function, a value of 1 (reward) is returned if t_j is present in t_i's match list l_i for a given subset of attributes S. Alternatively, v returns 0 if t_j is absent from t_i's match list l_i for a subset of attributes S, as in this case S has not contributed to a successful match. Again, l_i^* is the match list computed based on strictly the attributes in set S. To illustrate this with Example 3 (2), the utility function v would return 1 if t_{19} was present in l_3^* and 0 otherwise. The utility function v for PQ-MATCH can thus be defined as,

$$v(S) = \begin{cases} 0, & \text{if } S = \emptyset \textbf{ or } t_j \notin l_i^* \\ 1, & \text{if } t_j \in l_i^* \end{cases} \tag{4.4}$$

Continuing with Example 3 question (2), the results for PQ-MATCH can be seen in Table 4.2. The exact algorithm generated the Shapley values 0.25, 0.91, -0.08, and -0.08 for the attributes Python, R, PHP, and JS respectively. By far the largest contributor to Candidate t_3 being in HR t_{19}'s match list is the candidate's R skills. Additionally, both PHP and JS can be seen as having a negative value, negatively impacting Candidate t_3 being in the match list.

SQ-SINGLE: Utility functions for queries based on match lists explain more complicated problems than the previous two queries. This is due to these queries being calculated for differences in the entire match list, instead of simply presence or absence from the match list.

Consider the scenario where a dataset of entities $\mathbb{D}$ and an entity t_i with a match list of l_i are provided for the query type of SQ-SINGLE. The utility function must capture the similarity of the computed top-k match list for the set of attributes S, l_i^*, with match lists for the full set of attributes $\mathbb{A}$, l_i. As the explanation relies on the top-k items, l_i, and its

similarity with l_i^*, we can use the Jaccard similarity between these two sets as the utility function. The Jaccard similarity is a value between 0 and 1. The value 0 indicates that the sets share no common elements and 1 indicates that the sets l_i and l_i^* are identical. To further explain with Example 3 (3), v is the Jaccard similarity of the match list of entity t_3 ($l_3\{t_{19}, t_{20}\}$) with l_3^*. The utility function v can be expressed as,

$$v(S) = \begin{cases} 0, & \text{if } S = \emptyset \\ \frac{|l_i \cap l_i^*|}{|l_i \cup l_i^*|}, & otherwise \end{cases} \tag{4.5}$$

Again, with Example 3 question (3), the results for SQ-SINGLE are shown in Table 4.3. The exact algorithm generated the Shapley values 0.61, 0.28, 0.0, and 0.11 for the attributes Python, R, PHP, and JS respectively. The largest contributor to Candidate t_3's *match list being the way it was*, is the Python requirement of the HRs, however to a lesser degree the R requirements contributed, and even lesser, the JS requirements contributed.

SQ-MULTIPLE: A similar approach of using Jaccard similarity can be used to find the contribution of each attribute for the SQ-MULTIPLE problem. Consider the scenario when a dataset of entities $\mathbb{D}$ and an entity t_i is provided. The entity t_i is present in the match list of $T \subseteq O$ entities. For a subset $S \subseteq \mathbb{A}$ of attributes, let T^* be the set of entities in whose match list t_i is present for the attributes S. The utility function represents the similarity between the set T^* and T.

A similarity of 0 indicates there are no shared elements between the two match lists, where a similarity of 1 indicates that the sets are identical. A smaller value indicates a smaller intersection or a higher union, and a larger value indicates a larger union or a smaller intersection. From the running Example 3 (4), the set of entities whose match list consist of t_3 is $T = \{t_{14}, t_{19}\}$. Hence, the function v for this case would compute the Jaccard similarity between $\{t_{14}, t_{19}\}$ and T^* for a subset of attributes $S \subseteq \mathbb{A}$. The value function v can be expressed as,

$$v(S) = \begin{cases} 0, & \text{if } S = \emptyset \\ 1, & \text{if } S \neq \emptyset \ \ \& \ \ |T \cup T^*| = 0 \\ \frac{|T \cap T^*|}{|T \cup T^*|}, & otherwise \end{cases} \tag{4.6}$$

Finally, for Example 3 question (4), the results for SQ-MULTIPLE are shown in Table 4.4. The exact algorithm generated the Shapley values $0.48\bar{3}$, $0.23\bar{3}$, $0.33\bar{3}$, and 0.15 for the attributes Python, R, PHP, and JS respectively. All of the candidates skills contributed somewhat, but Python, PHP, R, and JS contributed in descending order of importance.

Having mapped Shapley values to the top-k matchings, we can now explain matchings using the attribute contributions, as shown in Example 4 (Part 2).

EXAMPLE 4. *(Part 2) Looking at Table 4.5, the Shapley values of* $\{A_1, A_2, A_3\}$ *for the query* "why is t_1 not in the top-k?" *are computes as* $\{-0.1\bar{6},\ 0.3\bar{3},\ 0.8\bar{3}\}$*, using Equation 4.5. The high Shapley value of attribute* A_3 *indicates that* A_3 *can explain the query the most. The impact of attribute* A_3 *is partially seen empirically when we remove the attribute. Removal of* A_3*'s impact on the linear ranking function is reflected in the new scores the entities get,* t_1*,* t_2*,* t_3*,* t_4 *and* t_5 *get a score of* 54 *,* 10.4*,* 49.8*,* 53.5*,* 57.7 *respectively. These scores show that the entity* t_1 *is present in the match list when* $k = 2$ *when* A_3 *is removed, thus confirming its relative contribution.*

Time complexity analysis: Theorem 4.3.1 shows the time taken to compute the exact Shapley value is *exponential* to the number of attributes for all the four queries, making Exact Shapley value computation impractical when d is not small.

Theorem 4.3.1. *Given a data-set of entities* $\mathbb{D}$ *with ranking functions* F *with the other parameters for the* PQ-NOTMATCH*,* PQ-MATCH*,* SQ-SINGLE *and* SQ-MULTIPLE *problems, computing the exact Shapley value takes exponential time to number of the players (attributes).*

Proof. Computation of exact Shapley values using Equation 4.2 relies on computing the utility function efficiently over all subsets of $\mathbb{A}$. We analyse the running time of each of the value functions of the 4 problems and prove that the exponential nature arises solely from Shapley value computation from Equation 4.2. As noted in Section 4.2, we denote the amount of time taken by the ranking function as C.

PQ-NOTMATCH: Consider a dataset $\mathbb{D}$ and entities (t_i and t_j). The value function used for PQ-NOTMATCH is given in Equation 4.3. Given a subset of attributes $S \subseteq \mathbb{A}$, the time taken to recompute the ranking function for an entity is C. As there are n entities, the function v takes nC time to obtain the scores for all entities. Obtaining the top-k (match list l_i^*) can be efficiently performed using the selection algorithm which takes a total of $\mathcal{O}(n)$ time. Checking if the entity t_j is present in the match list takes k time. Hence, total time taken by function v is $\mathcal{O}(nC)$ for a given subset S.

PQ-MATCH: Computation of the value function is similar to the computation of PQ-NOTMATCHand hence consumes $\mathcal{O}(nC)$ time.

SQ-SINGLE: Consider a dataset $\mathbb{D}$ and entity t_i with match list l_i when using all attributes $\mathbb{A}$. The Shapley value function for SQ-SINGLE is given in Equation 4.5. The set based value function relies on *Jaccard similarity* between sets l_i and l_i^* to obtain a value. Hence, the first step is similar to PQ-NOTMATCH and PQ-MATCH problems, i.e. computation of l_i^*.

The Jaccard similarity computation can be efficiently performed by sorting the match lists l_i and l_i^*, followed by performing simultaneous linear scans on l_i and l_i^* to obtain both the intersection and union. This step consumes a total of $\mathcal{O}(k \log(k))$ time. Hence, the total time consumed is $\mathcal{O}(nC + k \log(k))$

SQ-MULTIPLE: Consider a dataset $\mathbb{D}$ and entity t_i which is present in the match list of entities T when using all attributes $\mathbb{A}$. The Shapley value function for SQ-SINGLE is given

in Equation 4.6. Given a subset of attributes S, T^* can be obtained by first computing match list l_j^* for all n entities and checking which ones contain t_i.

The computation of l_j^* and checking if t_i is present in l_j^* ($t_i \in l_j^*$) for a single entity consumes $\mathcal{O}(nC)$ as seen above. As there are n entities, obtaining T^* consumes a total of $\mathcal{O}(n^2C)$ time. Additionally, to compute the *Jaccard* similarity, (i) need to sort both T and T^*, (ii) perform simultaneous scans on T and T^* to obtain both intersection and union, and (iii) obtain the ratio. Steps (i) and (ii) consume $\mathcal{O}(n \log(n))$ time and (iii) consumes $\mathcal{O}(n \log(n))$ time. Hence, the overall time consumed is $\mathcal{O}(n^2C)$.

The Shapley value computation for each of the four cases relies on generating all subsets of the set of attributes $\mathbb{A}$. As there are d attributes, generating the sets consumes a total of $d\ 2^d$ time. Hence, the Shapley computation for all the four queries is exponential in d, the number of attributes $\mathbb{A}$. □

Note on limitations of Shapley based method: Kumar et al. [125] have shown certain limitations of Shapley-based methods while explaining machine learning models. During the masking process, these methods are shown to sample out-of-distribution data points which can affect the Shapley explanation model and create undesirable output. Matching-based applications are not subject to similar problems like machine learning models, as the whole process of top-k and bipartite matching is based on competition. These processes are less subject to changes in scores and instead rely on ranking between items. Nevertheless, we would like to note our dependence on the scoring functions to handle the NULL values generated during the masking process. Additionally, we would like to note that explanations generated from Shapley-based methods are not actionable and must be used as a means to *whitebox* the black box function.

4.3.4 Approximate sampling based approach

The prohibitive nature of the exact Shapley value algorithm has spurred research into approximate methods. Sampling based methods have been proposed to obtain approximate Shapley values. In this paper, we use the sampling based on permutations of attributes. Based on [126], the mean of the marginal contributions for each attribute A_i over the entire symmetric group $Sym(A)$ is equivalent to the corresponding Shapley value Sh_i. To approximate the value Sh_i, instead of calculating all members of the symmetric group, q members of the $Sym(A)$ are sampled. The estimated Shapley value for attribute A_i resulting from the samples is referred as $\hat{Sh}_i$ throughout the paper. The expected value of a random sample from the uniform distribution is equivalent to the mean of that distribution such that $Ex[\hat{Sh}_i] = Sh_i$.

A randomized approximation algorithm based on the random sampling of permutations for computing the explanation queries is defined as follows. Initially, random sampling is performed to select q permutations from $Sym(A)$. With $\hat{Sh}_i$ set to 0, $\hat{Sh}_i$ is increased by $\frac{1}{q} \times (v(Pre_i^X \cup A_i) - v(Pre_i^X))$ for each sampled permutation X. The value $\hat{Sh}_i$, is the approximate Shapley value for attribute A_i.

Sample size, *q*: The randomised approximate algorithm can be demonstrated to show that by varying q and specifying an approximation bound α, an error rate of ϵ can be given that $Pr(|\hat{Sh}_i - Sh_i| < \alpha) > 1 - \epsilon$. We prove in Theorem 4.3.2 that a for a given value of approximation bound α and error rate ϵ, there exists a fixed samples size q which satisfies $Pr(|\hat{Sh}_i - Sh_i| < \alpha) > 1 - \epsilon$ for our queries.

Theorem 4.3.2. *Given an approximation bound α and error rate ϵ, sampling $q \geq \frac{2\log(2/\epsilon)}{\alpha^2}$ random permutation members of the symmetric group, ensures that the inequality $Pr(|\hat{Sh}_i - Sh_i| < \alpha) > 1 - \epsilon$ is satisfied for all four problems.*

Proof. We prove this theorem using Hoeffding's inequality [127]. First, we prove for the four queries that each random variable Z_i varies within the range of $Z_i \in [-1, 1]$. The random variable Z_i in each of the four problems refers to the Shapley value function for the sampled permutation. We obtain the inequality by applying Hoeffding's inequality.

For the queries PQ-NOTMATCH and PQ-MATCH, the random variable is the difference between the Shapley value function when attribute A_i is added to permutation X i.e. $Z_i = (v(Pre_i^X \cup A_i) - v(Pre_i^X))$. Hence, the random variable Z_i can take values -1, 0 or a 1 based on how the Shapley value function changed.

For the set similarity based problems SQ-SINGLE and SQ-MULTIPLE Z_i is the difference between the Jaccard similarity for the random permutation $v(Pre_i^X \cup A_i)$ and $v(Pre_i^X)$. Hence, Z_i for these two problems is a continuous variable between -1 and 1.

As we have seen, in all four problems the random variable Z_i varies between -1 and 1. Applying Hoeffding's inequality with $b = 1$ and $a = -1$ we get,

$$\mathbf{P}\left(|\hat{Sh}_i - Sh_i| \geq \alpha\right) \leq 2e^{\frac{-q\alpha^2}{2}}$$

Restructuring the above equation and representing q in terms of α and ϵ we get $q = \frac{2\log(2/\epsilon)}{\alpha^2}$. Hence, proved. □

4.3.5 KernelSHAP

Another technique to approximate Shapley values is SHAP [123]. SHAP proposes a linear model to explain black box machine learning model-prediction for a given input data point. KernelSHAP is a generalised SHAP approximation technique proposed by Lundeberg et. al. [123]. The technique is built on top of LIME [128] to approximate Shapley values. KernelSHAP provides the parameters to be set in the optimisation function of LIME such that the linear LIME model finds the Shapley values. To the best of our

knowledge, KernelSHAP does not inherently provide any theoretical guarantees on sample size unlike the sampling approach, but empirically it performs better.

Dataset name	Attributes	Size
Synthetic	9	1000
Graduate admissions	6	500
Job candidates	22	390

Table 4.6: Details of the datasets

4.4 Experiments

We conduct extensive experiments on real-world and synthetic data to validate our proposal and to evaluate the performance of our algorithms. In the following, after discussing the experiment set up, we provide proof of concept experiments that focus on validating our results. Finally, the empirical evaluation of the different techniques is presented.

4.4.1 Experiments setup

Datasets: Often, dataset owners tend to not upload datasets involving real world entities due to anonymization concerns. As a result, there are few, and hard to find, datasets for matching. Hence, we have two real world datasets and 12 configurations of synthetic datasets for the experiments.

- Job candidates dataset (*real world dataset*)[5]: 390 candidates were parsed from 22 columns, including 18 numerical columns, 3 categorical columns, and 1 set based column. Values for numerical columns were normalized to be a value between 0 and 1. To generate HR entities, uniform random values were selected for each column from the set of possible values for that column in the candidate dataset.

[5] https://www.kaggle.com/datasets/saikrishna20/candidates-list

- Graduate admissions dataset [129](*real world dataset*): The dataset consists of application details of 500 students to the graduate program. There are a total of 6 numerical features and a categorical feature. Each data point in the dataset also a dependant variable, *chance of admit*, which is a score between 0 and 1.
- Synthetic dataset: There are many parameters that can be varied when generating the datasets. We have used three main factors, probability distribution, correlation between attribute, and number of dimensions. As our experimental study deals with assesing our method across different settings, we generate multiple datasets for each setting and aggregate results for each setting. For the probability distribution, we consider the Zipfian distributions for our experiments. Linear and non-linear ranking functions; correlated, anti-correlated, and independent attributes were used for the various experiments to have a wide variety of data for each type of query. For each of these 12 settings, 10 datasets were generated bringing the total to 120 datasets. Each dataset consists of 1000 items each with 9 dimensions.

Hardware and Platform: All our experiments were performed on a work station with a Core i9 Intel X-series 3.5 GHz machine running Linux Ubuntu with 128 GB of DDR4 RAM. The algorithms were implemented in Python 3.

Ranking function: As the ranking / preference functions were not available for the real world dataset, we generated n linear ranking functions. The ranking functions are based on proximity to the candidate's skills/ HR's requirements. In the user study, the ranking function used for contrasting LIME and Shapley values explanations is learned to predict the *chance of admit* based on the other features. For synthetic dataset, we have used both linear and non-linear ranking functions. For non-linear ranking functions, weighted squares of the attributes have been used to score and rank entities.

Algorithms Evaluated: We have implemented the brute force Shapley value algorithm and the approximate Shapley value algorithms. We use the KernelSHAP library from github by Lundberg [6].

As baselines to compare against, we use the weight based algorithm described in Section 4.3.1 and an attribute based baseline. The attribute based baseline ranks each of the attributes individually for a given query. More specifically, for a query we mask the set of attributes such that only attribute is unmasked. We measure the effect of the individual attributes and then rank these attributes. For the PQ-NOTMATCH (PQ-MATCH resp.) query, we use the highest (least resp.) rank achieved by the entity when only using a single attribute as a measure to compare all attributes and rank them. As SQ-SINGLE and SQ-MULTIPLE queries are set based queries, we use the Jaccard similarity measure instead to rank the attributes. The comparison for these baselines is provided in Section 4.4.2.2.

4.4.2 Proof of Concept

As our first set of experiments, we provide results to validate our proposal for explaining match lists using Shapley values. In particular, we first present a case study, discussing the explanations for specific cases in detail. Then, we provide an experiment to demonstrate the effectiveness of Shapley values for explanation.

4.4.2.1 Case Study

We begin our proof of concept experiments by a case study to illustrate the usefulness of our explanations. Aligned with our running example (Example 3), we select a user from our experiments on job-candidates dataset, and discuss the generated explanation in detail. Approximate Shapley values produced using the sampling algorithm for PQ-NOTMATCH are shown in Table 4.7.

[6] https://github.com/slundberg/shap

	Candiate Values	HR Function	Shapley Values
Python	1.0	0.002	0.0
R	0.0	0.005	0.09
Deep Learning	0.333	0.005	0.025
PHP	0.667	0.007	0.05
MySQL	0.667	0.007	0.075
HTML	0.667	0.007	0.035
CSS	0.0	0.005	0.085
JavaScript	0.667	0.005	0.065
AJAX	0.0	0.005	0.06
Bootstrap	0.0	0.006	0.07
MongoDB	0.0	0.005	0.045
Node.js	0.0	0.003	0.045
Reactjs	0.0	0.005	0.09
Performance_PG	0.791	0.06	-0.02
Performance_UG	0.7	0.06	0.015
Performance_12	1.0	0.06	-0.05
Performance_10	1.0	0.120	-0.05
Other Skills	['Algorithms', 'Data Structures', ...]	0.0769	0.065
Degree	Master of Science	0.0769	0.21
Stream	Computer Science	0.0769	0.05
Grad Year	2018	0.307	0.04
Current City	Bangalore	0.0769	0.005

Table 4.7: Candidate values, HR rankings, and PQ-NOTMATCH Shapley values.

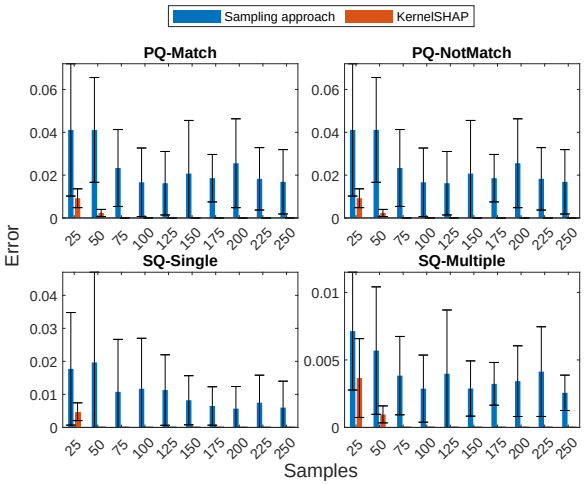


Figure 4.2: Error variations in sampling-based approach and KernelSHAP when varying number of samples for Candidates dataset

Among their programming skills, the largest contributors to this result were `R`, `React.js` and `CSS` with Shapley values of 0.09, 0.09, and 0.085. In each of these cases, the candidate performed poorly on these skills while HR ranked these skills fairly highly relative to other programming skills. The lowest Shapley among programming skills value was `Python`, where the individual had the maximum score, but the weight was also the lowest relative to all skills. Overall, the highest Shapley value was for the Candidate's `Master of Science` degree, indicating that this degree (as opposed to another one) was the main reason the candidate was not placed on the match list. Finally, the overall lowest weights were their performances in grade 12, 10, and post grad, with scores of -0.05, -0.05 and -0.02. The negative Shapley values indicate that that these skills worked against the candidate not being in the match list, and can be seen as skills that if evaluated solely on, the individual would appear in the match list. An explanation may appear as follows:

```
"You were not matched with this HR largely due to your degree
in Master of Science; the company was more interested in other
degrees. Among your programming skills, the company would like to
see more skill in R,  CSS, and React.js specifically. The
company was overall pleased with your Grade 10 Performance,
Grade 12 Performance, and Post-grad Performance, but considering
all factors, they did not want to match  with you at this time."
```

4.4.2.2 Effectiveness of Shapley values for explainability

For our first experiment, we empirically measure the effectiveness of the approximate algorithm in capturing the Shapley values. Given an explainability query, the brute force algorithm produces the exact Shapley values. Since our goal is to capture the exact Shapley values as accurately as possible, we compare these values with the results of various methods. To do this, we considered the *top ranked* attribute for each algorithm.

Distribution	Correlation	Function	APX - Q1	APX - Q2	APX - Q3	APX - Q4	WT - Q1	WT - Q2	WT - Q3	SCR - Q1	SCR - Q2	SCR - Q3	SCR - Q4
Uniform	Correlated	Linear	1.0	1.0	1.0	1.0	0.9	0.7	1.0	0.2	0.6	0.1	0.1
Uniform	Correlated	Non-Linear	1.0	1.0	1.0	1.0	1.0	0.5	0.9	0.5	.5	0.3	0.1
Uniform	Anti-Correlated	Linear	1.0	1.0	1.0	1.0	.8	0.7	1.0	0.5	0.7	0.1	0.2
Uniform	Anti-Correlated	Non-Linear	1.0	1.0	1.0	1.0	1.0	0.4	1.0	0.1	0.4	0.3	0.1
Uniform	Independent	Linear	1.0	1.0	1.0	1.0	1.0	0.6	1.0	0.4	0.7	0.3	0.4
Uniform	Independent	Non-Linear	1.0	1.0	1.0	1.0	1.0	0.5	1.0	0.2	0.5	0.5	0.2
Zipfian	Correlated	Linear	1.0	1.0	1.0	1.0	0.7	0.8	0.9	0.5	0.8	0.1	0.4
Zipfian	Correlated	Non-Linear	1.0	1.0	1.0	1.0	1.0	0.5	1.0	0.2	0.6	0.2	0.2
Zipfian	Anti-Correlated	Linear	1.0	1.0	1.0	1.0	0.6	0.9	0.8	0.3	0.5	0.8	0.3
Zipfian	Anti-Correlated	Non-Linear	1.0	1.0	1.0	1.0	0.9	0.5	0.7	0.8	0.2	0.0	0.8
Zipfian	Independent	Linear	1.0	1.0	1.0	0.9	0.8	0.8	0.9	0.2	0.8	0.1	0.1
Zipfian	Independent	Non-Linear	1.0	1.0	1.0	1.0	1.0	0.6	1.0	0.3	0.6	0.2	0.3

Table 4.8: The success measure of four methods in computing the same top value as Brute Force; APX=Approximate, WT=Weight, SCR=Attribute Score; and the four queries, Q1-Q4. For Q4, WT could not be used.

In this experiment, we measure the effectiveness of three methods in explaining the query. These methods are the Shapley values by sampling, the attribute with the highest weight (Section 4.3.1), and the attribute evaluated on the query function independently.

The datasets for this experiment consists of twelve settings on synthetic data. The settings are each combination of distribution, feature correlation types and scoring function type, i.e. $\{$*uniform, Zipfian distribution*$\}$ $\times$ $\{$ *independent, correlated or anti-correlated features* $\}$ $\times$ $\{$ *linear, non-linear scoring functions*$\}$. For each of these settings, ten trials were run with $k = 5$ and the results were recorded. For each experiment, highest rated attribute was removed and the output compared for its impact on the query.

The results are tabulated in Table 4.8. The results show that approximate method produced the same output as brute-force in 119 out of the 120 trials. The other methods performed consistently at best equal but generally worse across all queries. Measuring by weight performed almost always better than attribute based baseline. However, measuring by weight still often failed. It had a particularly low accuracy for PQ-NOTMATCH. Additionally, since weight based approach is not possible for SQ-MULTIPLE, it was considered to be an insufficient method for computing the highest Shapley value.

4.4.2.3 User study

In this experiment, we conduct a user study to validate our methods, using the graduate admissions dataset and the example dataset from Table 4.5. We included a quality control step to assess the participant's familiarity with scoring functions. Any candidate who scores at least 3 (out of 5) is considered for the study. *22 of the 26 participants qualified*. For the graduate admissions dataset, we create a black box ensemble model using *auto-sklearn* package that can predict the *chance of admit* given a student's admission data. We use this model to score students, and based on these scores obtain the top-k students who qualified during the admission process.

As a part of the first question, the participants were shown a scenario of a student who failed to qualify to the university. The participants were provided with the mean scores of the admitted students and the mean scores of all the students who applied. We showed the participants two explanations, one from LIME [128] and other from our method. The textual description for the explanations were created based on Molnar et al. [130]. The generated explanations were: *"The value of [FEATURE NAME] contributed [SHAP SCORE] to [INDIVIDUAL NAME] not being in the top-k compared to the average prediction for the dataset."* for Shapley values, and *"An increase of [FEATURE NAME] by one unit increases [INDIVIDUAL NAME] towards not being in the top-k by [LIME SCORE] units when all other feature values remain fixed."* for LIME. We asked the participants to choose among the two explanations. Around 63% of the participants found the explanation provided by our method more convincing.

Next, the participants were shown the scenario from Example 4. We provided the scoring function used ($5 \times A_1 + 4 \times A_2 + 1 \times A_3$) and asked the participants to explain, *why* t_5 *was not selected during the process*. As this is a subjective question, we compared the explanation provided in each answer against that of (i) LIME, (ii) function weights used

in the scoring, and (iii) our explanation. We classified the provided explanation in each answer into one of these three methods, based on its closeness to the explanation generated by these methods. While 32% and 13% preferred LIME and function weights, respectively, *more than 54% of the participants* preferred our explanation.

4.4.3 Performance Evaluation

Having discussed the experiments to validate our problem setting, we now present our performance evaluation results, where we study the approximation error and runtime of sampling based and KernelSHAP methods in comparison to exact Shapley values.

Impact of sampling size on error: The approximate sampling-based approach and KernelSHAP provide us with a trade-off between error and time consumed. As the number of samples taken increases, the error in the Shapley value decreases, but time increases. We want to measure the impact of sampling size on error and time consumed.

In this experiment, the number of samples that we use for the approximate Shapley value algorithm and KernelSHAP are varied and the runtime and error are measured. Match list size, k, is set to 5 for these experiments. Samples for the sampling based algorithm are chosen uniformly, with the sample size varying from 25 to 250 in increments of 25. For this experiment, we consider both, the synthetic datasets and the real world dataset. In each experiment setting, we also run the brute force algorithm to obtain exact Shapley values to measure against. For each of the configurations, we aggregate (average) the error across the d attributes and compute the standard deviation of the errors.

The measured average error in Shapley values and standard deviation of the average errors are plotted in Figure 4.2 for the Candidates dataset. As can be seen in the figure, KernelSHAP outperformed the sampling-based approach with a similar number of samples. The time consumed by KernelSHAP and sampling based approach are comparable and far

better than the brute force. Similar results can also be seen for the synthetic dataset with linear (Figure 4.4) and non-linear functions (Figure 4.3).

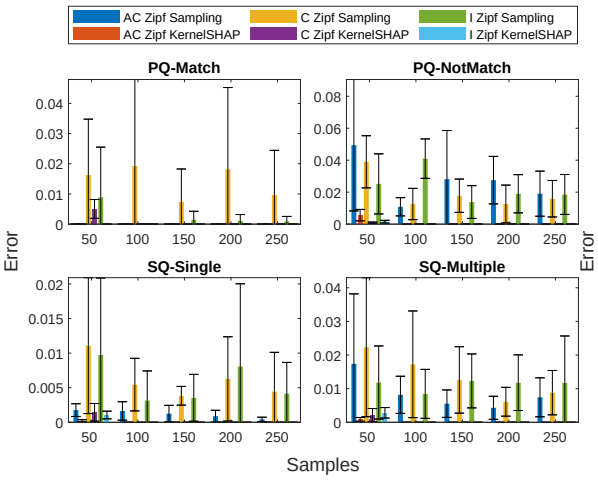


Figure 4.3: Error variations in sampling-based approach and KernelSHAP when varying number of samples for synthetic dataset with non-linear ranking functions

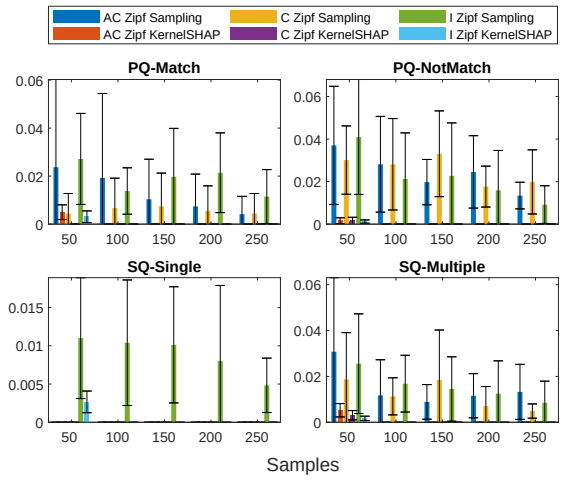


Figure 4.4: Error variations in sampling-based approach and KernelSHAP when varying number of samples for synthetic dataset with linear ranking functions

4.5 Related work

Matching and applications: Matching plays a critical role in the allocation of resources based on supply and demand like matching a region to medical needs [113], ecosystem services are matched based on changing land use [114], public health needs are matched by health insurance plans [115]. Matching systems rely on different mechanisms to capture preferences of both the supply and demand parties. Traditionally, these preferences are explicitly specified [131–134].

Explicitly specified matching relies on capturing preference lists from both the parties and then create a stable matching under specific conditions. Based on game theory, Gale and Shapley [135] designed a mathematical framework to attain stable matching and applied it to stable marriage and college admission. Such matching where one

party is matched only to one other party is known as stable marriage [136–138]. Similarly, many-to-one stable matching exists with applications such as: hospitals provide employment to many doctors [139] or student–project allocations [140] which was formalized by Roth and Sotomayor [141]. Many-to-many stable matching relies on matching many supply parties to many demand parties, which has applications in D2D-enabled cellular networks [132, 142], collaborator recommendations [143]. While there has been extensive work on explicit matching, to the best of our knowledge there has not been any explainability work in these fields.

Often, specifying the complete preference list is prohibitive in big data applications [144]. In such cases, the preference can be obtained via different means. In case of a search engine, users interact with the search engine to express their query until they reach their desired result (web-page). Traditionally TF-IDF [145, 146] and Latent Semantic Analysis [147, 148] based approaches were proposed to solve this problem. Recently, vector representation based approaches [149, 150] convert words into vector form and then use this embedded space representation to process the query. Another type of application where implicit preference is seen is user-item recommendation systems like Amazon[7], Netflix[8]. In user-item recommendation systems, items are recommended with prior interactions of the user with their systems. Numerous techniques like collaborative filtering [151], matrix factorisation [152], auto encoder based representations [153, 154] have been proposed to solve the problem. While some of these works have explanations built within the system [155, 156], to the best of our knowledge we are the first to work on explanations in bipartite matching scenarios.

Explainability in ranking: Ranking functions are popular for solving multi-criteria optimization. While ranking functions have been studied for many years, study of explain-

[7]https://www.amazon.com/

[8]https://www.netflix.com/

ability in ranking has been a recent trend. Verma et al. [157] explain queries based on a sampling approach in the neighbourhood of the query. Gale and Marian explore the topic of explaining ranking in multiple papers. First, in their 2019 paper [158] they assign scores to various attributes based on whether the items in the top-k for all methods are in the top-k for a particular attribute. They also demonstrate that the explanation for a ranking can be used to adjust a ranking function such that attributes are contributing to the amount required. The main difference between their work and ours is that they use this as an explanation for ranking. Top-k is similar but independent from ranking, and additionally our matching are bipartite. The difference is further shown by the fact that these algorithms cannot be easily modified to work for our queries. Next in 2020 [159], Gale and Marian expand upon their initial observations by also considering the weight of different parameters, and considering multiple metrics for the type of ranking produced by the function, namely disparity and diversity. Diversity and disparity in ranking has also been studied by works like [69]. Additionally, some work has been done on responsible ranking function design [69, 160], where the objective is to minimally change the weights in a ranking function to make the generated rankings (top-k) fair and stable.

Shapley values for explanations: Shapley values were introduced by Gale Shapley [120] to determine the contribution of each player to the success of the overall coalition. Shapley values have also been applied to the problem of explanations with a great degree of success. Here, the contribution of the various features to the overall prediction are calculated as Shapley values, and the Shapley values are used to explain the task [130]. Several notable contributions have been made to this. Štrumbelj et al. [122] devise a Monte Carlo sampling technique for explaining models, in order to avoid the exponential nature of exact Shapley value computation. Lundberg et al. [123] mapped the notion of Shapley values to the problem of model interpretability, introducing SHAP and specifically KernelSHAP which allows for regression-based, model agnostic computation of SHAP values. Lundberg et al.

expand upon this notion in 2018 [161] with the TreeSHAP method, which is capable of computing SHAP values for tree based models in polynomial time. While these methods have been studied extensively in machine learning, the problem of explainability in bipartite matching is novel and has not been studied.

Kumar et al. [125] have shown certain limitations of Shapley-based methods while explaining machine learning models. The limitations highlighted during the process include (i) out-of-distribution points generated during the masking process, (ii) explanations generated using Shapley are not actionable. We emphasise that bipartite matching is not subject to similar problems due to competition. Nevertheless, we have added a note about improper handling of NULL values during masking process. We would also like to point out that an important aspect of our paper is to open the blackbox model and thereby create trust towards the system.

4.6 Discussion

The concept of match list is modelled on real world matching websites and applications. Some more practical models may include more complex scenarios which can extend to more data types like non-binary matching values, matching preferences that extend to multiple matching values, probabilistic modelling of preference functions. While we present four different explainability queries that may be encountered in real life, there may be many more of these queries which may be of interest. We consider both these extensions important, and an avenue for future work.

While the model we propose is based on the real world, one might also want to consider other theoretical models of matching, like a complete ranked list by each individual instead of ranking functions. These lists could then be used in a stable marriage algorithm to produce a matching. Such alternate models which may be of theoretical interest can

be considered as alternate models and present an opportunity for a thorough theoretical analysis.

4.7 Conclusion

As the first attempt in explaining matchings, in this paper we considered the large-scale two-sided matching applications in which the user preferences are specified as ranking functions and matchings are derived as top-k. We considered four natural explanation questions that would benefit the users of these systems, and observing the competitive nature of rankings, proposed Shapley-based approaches for explanation. To overcome the combinatorial complexity of exact Shapley computation, we proposed an approximation algorithm with provable guarantees on run-time and accuracy. Furthermore, we conducted comprehensive experiments on real-world and synthetic data sets demonstrate the usefulness of our explanations and to evaluate the performance of our algorithms.

REFERENCES

[1] S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides, "Computing k-regret minimizing sets," *VLDB*, vol. 7, no. 5, 2014.

[2] S. Zeighami and R. C.-W. Wong, "Finding average regret ratio minimizing set in database," pp. 1722–1725, 2019.

[3] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A survey of top-k query processing techniques in relational database systems," *CSUR*, vol. 40, no. 4, p. 11, 2008.

[4] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *Journal of Computer and System Sciences*, vol. 66, no. 4, pp. 614–656, 2003.

[5] G. Das, D. Gunopulos, N. Koudas, and D. Tsirogiannis, "Answering top-k queries using views," in *VLDB*, 2006.

[6] A. Asudeh, N. Zhang, and G. Das, "Query reranking as a service," *PVLDB*, vol. 9, no. 11, 2016.

[7] A. Asudeh, H. Jagadish, G. Miklau, and J. Stoyanovich, "On obtaining stable rankings," *PVDLB*, vol. 12, no. 3, pp. 237–250, 2018.

[8] A. Asudeh, H. Jagadish, J. Stoyanovich, and G. Das, "Designing fair ranking schemes," in *SIGMOD*. ACM, 2019.

[9] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *ICDE*, 2001.

[10] A. Asudeh, S. Thirumuruganathan, N. Zhang, and G. Das, "Discovering the skyline of web databases," *PVLDB*, vol. 9, no. 7, pp. 600–611, 2016.

[11] A. Asudeh, G. Zhang, N. Hassan, C. Li, and G. V. Zaruba, "Crowdsourcing pareto-optimal object finding by pairwise comparisons," in *CIKM*, 2015.

[12] D. Nanongkai, A. D. Sarma, A. Lall, R. J. Lipton, and J. Xu, "Regret-minimizing representative databases," *VLDB*, 2010.

[13] A. Asudeh, A. Nazi, N. Zhang, and G. Das, "Efficient computation of regret-ratio minimizing set: A compact maxima representative," in *SIGMOD*. ACM, 2017.

[14] P. K. Agarwal, N. Kumar, S. Sintos, and S. Suri, "Efficient algorithms for k-regret minimizing sets," *LIPIcs*, 2017.

[15] S. Zeighami and R. C.-W. Wong, "Minimizing average regret ratio in database," in *SIGMOD*. ACM, 2016.

[16] M. Xie, R. C.-W. Wong, J. Li, C. Long, and A. Lall, "Efficient k-regret query algorithm with restriction-free bound for any dimensionality," in *SIGMOD*. ACM, 2018.

[17] L. Kaufman and P. J. Rousseeuw, "Clustering by means of medoids, statistical data analysis based on the l1 norm and related methods," *Y. Dodge, North-Holland*, 1987.

[18] M. Dyer and A. Frieze, "On the complexity of computing the volume of a polyhedron," *SIAM*, vol. 17, no. 5, pp. 967–974, 1988. [Online]. Available: https://doi.org/10.1137/0217060

[19] L. Kubáček, "On a linearization of regression models," *Applications of Mathematics*, vol. 40, no. 1, pp. 61–78, 1995.

[20] P. K. Agarwal and J. Pan, "Near-linear algorithms for geometric hitting sets and set covers," in *SOCG*. ACM, 2014.

[21] S. Thirumuruganathan, "A detailed introduction to k-nearest neighbor (knn) algorithm," *Retrieved March*, vol. 20, p. 2012, 2010.

[22] R. Fagin, R. Kumar, and D. Sivakumar, "Comparing top k lists," *Journal on Discrete Mathematics*, 2003.

[23] S. Boyd and L. Vandenberghe, *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[24] J. Hammersley, *Monte carlo methods*. Springer Science & Business Media, 2013.

[25] R. Motwani and P. Raghavan, *Randomized algorithms*. Cambridge university press, 1995.

[26] "Color dataset," https://archive.ics.uci.edu/ml/datasets/corel+image+features.

[27] I. Bartolini, P. Ciaccia, and M. Patella, "Efficient sort-based skyline evaluation," *ACM Trans. Database Syst.*, vol. 33, no. 4, pp. 31:1–31:49, Dec. 2008. [Online]. Available: http://doi.acm.org/10.1145/1412331.1412343

[28] "US Department of Transportation's dataset," http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time.

[29] "NBA dataset," www.databasebasketball.com/.

[30] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang, "Selecting stars: The k most representative skyline operator," in *ICDE*. IEEE, 2007, pp. 86–95.

[31] Z. Huang, Y. Xiang, and Z. Lin, "l-skydiv query: Effectively improve the usefulness of skylines," *Science China Information Sciences*, vol. 53, no. 9, pp. 1785–1799, 2010.

[32] Y. Tao, L. Ding, X. Lin, and J. Pei, "Distance-based representative skyline," in *ICDE*. IEEE, 2009, pp. 892–903.

[33] V. Koltun and C. H. Papadimitriou, "Approximately dominating representatives," in *ICDT*. Springer, 2005, pp. 204–214.

[34] S. Aggarwal, S. Mitra, and A. Bhattacharya, "Skycover: Finding range-constrained approximate skylines with bounded quality guarantees." in *COMAD*, 2016, pp. 1–12.

[35] D. Nanongkai, A. Lall, A. Das Sarma, and K. Makino, "Interactive regret minimization," in *SIGMOD*, ser. SIGMOD '12. New York, NY, USA: ACM, 2012, pp. 109–120. [Online]. Available: http://doi.acm.org/10.1145/2213836.2213850

[36] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *ICDE*. IEEE, 2001, pp. 421–430.

[37] M. F. Rahman, A. Asudeh, N. Koudas, and G. Das, "Efficient computation of subspace skyline over categorical domains," in *CIKM*. ACM, 2017, pp. 407–416.

[38] J. L. Bentley, F. P. Preparata, and M. G. Faust, "Approximation algorithms for convex hulls," *Commun. ACM*, vol. 25, no. 1, pp. 64–68, Jan. 1982. [Online]. Available: http://doi.acm.org/10.1145/358315.358392

[39] Y. Gao, Q. Liu, L. Chen, G. Chen, and Q. Li, "Efficient algorithms for finding the most desirable skyline objects," *Knowledge-Based Systems*, vol. 89, pp. 250–264, 2015.

[40] W. Jin, J. Han, and M. Ester, "Mining thick skylines over large databases," in *PKDD*. Springer, 2004, pp. 255–266.

[41] G. Valkanas, A. N. Papadopoulos, and D. Gunopulos, "Skydiver: a framework for skyline diversification," in *EDBT*. ACM, 2013, pp. 406–417.

[42] H. Lu, C. S. Jensen, and Z. Zhang, "Flexible and efficient resolution of skyline query size constraints," *TKDE*, vol. 23, no. 7, pp. 991–1005, 2010.

[43] L. Kaufman and P. Rousseeuw, "Finding groups in data: An introduction to cluster analysis," 1990.

[44] R. T. Ng and J. Han, "Clarans: a method for clustering objects for spatial data mining," *TKDE*, vol. 14, no. 5, pp. 1003–1016, Sep. 2002.

[45] S. Sen, "How data, analytics, and technology are helping us fight covid-19. minnpost," 2020.

[46] S. Barocas and A. D. Selbst, "Big data's disparate impact," *Calif. L. Rev.*, vol. 104, p. 671, 2016.

[47] A. Asudeh and H. Jagadish, "Fairly evaluating and scoring items in a data set," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 3445–3448, 2020.

[48] C. O'neil, *Weapons of math destruction: How big data increases inequality and threatens democracy*. Broadway Books, 2016.

[49] A. Chouldechova, "Fair prediction with disparate impact: A study of bias in recidivism prediction instruments," *Big data*, vol. 5, no. 2, pp. 153–163, 2017.

[50] J. Kleinberg, S. Mullainathan, and M. Raghavan, "Inherent trade-offs in the fair determination of risk scores," *arXiv preprint arXiv:1609.05807*, 2016.

[51] S. A. Friedler, C. Scheidegger, and S. Venkatasubramanian, "On the (im) possibility of fairness," *arXiv preprint arXiv:1609.07236*, 2016.

[52] F. D. Blau and L. M. Kahn, "The gender pay gap," *The Economists' Voice*, vol. 4, no. 4, 2007.

[53] B. Salimi, L. Rodriguez, B. Howe, and D. Suciu, "Interventional fairness: Causal database repair for algorithmic fairness," in *SIGMOD*, 2019, pp. 793–810.

[54] B. Salimi, B. Howe, and D. Suciu, "Database repair meets algorithmic fairness," *ACM SIGMOD Record*, vol. 49, no. 1, pp. 34–41, 2020.

[55] F. Nargesian, A. Asudeh, and H. V. Jagadish, "Tailoring data source distributions for fairness-aware data integration," *PVLDB*, vol. 14, no. 11, pp. 2519–2532, 2021.

[56] A. Yan and B. Howe, "Equitensors: Learning fair integrations of heterogeneous urban data," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2338–2347.

[57] Z. Jin, M. Xu, C. Sun, A. Asudeh, and H. Jagadish, "Mithracoverage: A system for investigating population bias for intersectional fairness," in *SIGMOD*, 2020.

[58] A. Asudeh, Z. Jin, and H. Jagadish, "Assessing and remedying coverage for a given dataset," in *ICDE*. IEEE, 2019, pp. 554–565.

[59] Y. Lin, Y. Guan, A. Asudeh, and H. Jagadish, "Identifying insufficient data coverage in databases with multiple relations," *PVLDB*, vol. 13, no. 12, pp. 2229–2242, 2020.

[60] A. Asudeh, N. Shahbazi, Z. Jin, and H. Jagadish, "Identifying insufficient data coverage for ordinal continuous-valued attributes," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 129–141.

[61] K. H. Tae and S. E. Whang, "Slice tuner: A selective data acquisition framework for accurate and fair machine learning models," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 1771–1783.

[62] E. Pastor, L. de Alfaro, and E. Baralis, "Looking for trouble: Analyzing classifier behavior via pattern divergence," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 1400–1412.

[63] Y. Moskovitch and H. Jagadish, "Countata: dataset labeling using pattern counts," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 2829–2832, 2020.

[64] C. Sun, A. Asudeh, H. Jagadish, B. Howe, and J. Stoyanovich, "Mithralabel: Flexible dataset nutritional labels for responsible data science," in *CIKM*, 2019, pp. 2893–2896.

[65] K. Yang, J. Stoyanovich, A. Asudeh, B. Howe, H. Jagadish, and G. Miklau, "A nutritional label for rankings," in *SIGMOD*, 2018, pp. 1773–1776.

[66] H. Zhang, X. Chu, A. Asudeh, and S. B. Navathe, "Omnifair: A declarative system for model-agnostic group fairness in machine learning," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 2076–2088.

[67] H. Zhang, N. Shahbazi, X. Chu, and A. Asudeh, "Fairrover: explorative model building for fair and responsible machine learning," in *Proceedings of the Fifth Workshop on Data Management for End-To-End Machine Learning*, 2021, pp. 1–10.

[68] A. Balayn, C. Lofi, and G.-J. Houben, "Managing bias and unfairness in data for decision support: a survey of machine learning and data engineering approaches to identify and mitigate bias and unfairness within data management and analytics systems," *The VLDB Journal*, pp. 1–30, 2021.

[69] A. Asudeh, H. Jagadish, J. Stoyanovich, and G. Das, "Designing fair ranking schemes," in *SIGMOD*, 2019, pp. 1259–1276.

[70] C. Kuhlman and E. Rundensteiner, "Rank aggregation algorithms for fair consensus," *PVLDB*, vol. 13, no. 12, pp. 2706–2719, 2020.

[71] A. Asudeh, H. Jagadish, G. Miklau, and J. Stoyanovich, "On obtaining stable rankings," *PVLDB*, vol. 12, no. 3, 2019.

[72] Y. Guan, A. Asudeh, P. Mayuram, H. Jagadish, J. Stoyanovich, G. Miklau, and G. Das, "Mithraranking: A system for responsible ranking design," in *SIGMOD*, 2019, pp. 1913–1916.

[73] Y. Zhao, K. Zheng, J. Guo, B. Yang, T. B. Pedersen, and C. S. Jensen, "Fairness-aware task assignment in spatial crowdsourcing: Game-theoretic approaches," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 265–276.

[74] L. Getoor, "Responsible data science," in *SIGMOD*, 2019.

[75] J. Stoyanovich, B. Howe, and H. Jagadish, "Responsible data management," *PVLDB*, vol. 13, no. 12, pp. 3474–3488, 2020.

[76] N. B. Shah and Z. Lipton, "Sigmod 2020 tutorial on fairness and bias in peer review and other sociotechnical intelligent systems," in *SIGMOD*, 2020, pp. 2637–2640.

[77] S. Venkatasubramanian, "Algorithmic fairness: Measures, methods and representations," in *PODS*, 2019, pp. 481–481.

[78] C. Accinelli, B. Catania, G. Guerrini, and S. Minisi, "covrew: a python toolkit for pre-processing pipeline rewriting ensuring coverage constraint satisfaction demonstration paper," 2021.

[79] ——, "The impact of rewriting on coverage constraint satisfaction." in *EDBT/ICDT Workshops*, 2021.

[80] C. Accinelli, S. Minisi, and B. Catania, "Coverage-based rewriting for data preparation." in *EDBT/ICDT Workshops*, 2020.

[81] M. J. Zimmer, "Slicing & dicing of individual disparate treatment law," *La. L. Rev.*, vol. 61, p. 577, 2000.

[82] M. B. Zafar, I. Valera, M. Gomez Rodriguez, and K. P. Gummadi, "Fairness beyond disparate treatment & disparate impact: Learning classification without disparate mistreatment," in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 1171–1180.

[83] A. Asudeh, "Enabling responsible data science in practice," *ACM SIGMOD Blog*, January 2021.

[84] A. Narayanan, "Translation tutorial: 21 fairness definitions and their politics," in *FAT**, 2018.

[85] S. Barocas, M. Hardt, and A. Narayanan, "Fairness and machine learning: Limitations and opportunities," fairmlbook.org, 2019.

[86] I. Žliobaitė, "Measuring discrimination in algorithmic decision making," *DATA MIN KNOWL DISC*, vol. 31, no. 4, pp. 1060–1089, 2017.

[87] F. Kamiran and T. Calders, "Data preprocessing techniques for classification without discrimination," *Knowledge and Information Systems*, vol. 33, no. 1, pp. 1–33, 2012.

[88] J. L. Bentley, "Decomposable searching problems." CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, Tech. Rep., 1978.

[89] S. Rahul and R. Janardan, "Algorithms for range-skyline queries," in *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, 2012, pp. 526–529.

[90] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[91] "Texas tribune dataset," https://salaries.texastribune.org/, visited: 2021.

[92] J. Monahan and J. L. Skeem, "Risk assessment in criminal sentencing," *Annual review of clinical psychology*, vol. 12, pp. 489–513, 2016.

[93] "Urbangb dataset," kaggle.com/daveianhickey/2000-16-traffic-flow-england-scotland-wales/data, visited: 2021.

[94] A. Y. Halevy, "Answering queries using views: A survey," *The VLDB Journal*, vol. 10, no. 4, pp. 270–294, 2001.

[95] G. Das, D. Gunopulos, N. Koudas, and D. Tsirogiannis, "Answering top-k queries using views," in *Proceedings of the 32nd international conference on Very large data bases*, 2006, pp. 451–462.

[96] D. Srivastava, S. Dar, H. V. Jagadish, and A. Y. Levy, "Answering queries with aggregation using views," in *VLDB*, vol. 96, no. September, 1996, pp. 318–329.

[97] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim, "Optimizing queries with materialized views," in *Proceedings of the Eleventh International Conference on Data Engineering*. IEEE, 1995, pp. 190–200.

[98] S. Chaudhuri, G. Das, and V. Narasayya, "Optimized stratified sampling for approximate query processing," *ACM Transactions on Database Systems (TODS)*, vol. 32, no. 2, pp. 9–es, 2007.

[99] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy, "The aqua approximate query answering system," in *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, 1999, pp. 574–576.

[100] S. Thirumuruganathan, S. Hasan, N. Koudas, and G. Das, "Approximate query processing for data exploration using deep generative models," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 2020, pp. 1309–1320.

[101] Q. Ma and P. Triantafillou, "Dbest: Revisiting approximate query processing engines with machine learning models," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 1553–1570.

[102] S. Shetiya, S. Thirumuruganathan, N. Koudas, and G. Das, "Astrid: Accurate selectivity estimation for string predicates using deep learning."

[103] M. Feldman, S. A. Friedler, J. Moeller, C. Scheidegger, and S. Venkatasubramanian, "Certifying and removing disparate impact," in *proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 2015, pp. 259–268.

[104] S. Hajian, J. Domingo-Ferrer, and A. Martinez-Balleste, "Discrimination prevention in data mining for intrusion and crime detection," in *2011 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*. IEEE, 2011, pp. 47–54.

[105] M. Hardt, E. Price, and N. Srebro, "Equality of opportunity in supervised learning," in *Advances in neural information processing systems*, 2016, pp. 3315–3323.

[106] S. Jabbari, M. Joseph, M. Kearns, J. Morgenstern, and A. Roth, "Fair learning in markovian environments," *arXiv preprint arXiv:1611.03071*, 2016.

[107] M. Joseph, M. Kearns, J. H. Morgenstern, and A. Roth, "Fairness in learning: Classic and contextual bandits," in *Advances in Neural Information Processing Systems*, 2016, pp. 325–333.

[108] B. Salimi, C. Cole, P. Li, J. Gehrke, and D. Suciu, "Hypdb: a demonstration of detecting, explaining and resolving bias in olap queries," *Proceedings of the VLDB Endowment*, vol. 11, no. 12, pp. 2062–2065, 2018.

[109] J. Dressel and H. Farid, "The accuracy, fairness, and limits of predicting recidivism," *Science advances*, vol. 4, no. 1, p. eaao5580, 2018.

[110] V. Zelaya, P. Missier, and D. Prangle, "Parametrised data sampling for fairness optimisation," *KDD XAI*, 2019.

[111] A. Backurs, P. Indyk, K. Onak, B. Schieber, A. Vakilian, and T. Wagner, "Scalable fair clustering," in *International Conference on Machine Learning*. PMLR, 2019, pp. 405–413.

[112] M. Olfat and A. Aswani, "Convex formulations for fair principal component analysis," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 663–670.

[113] H. Shao, C. Jin, J. Xu, Y. Zhong, and B. Xu, "Supply-demand matching of medical services at a city level under the background of hierarchical diagnosis and treatment-based on didi chuxing data in haikou, china," *BMC Health Services Research*, vol. 22, no. 1, pp. 1–12, 2022.

[114] S. H. Brunner, R. Huber, and A. Grêt-Regamey, "A backcasting approach for matching regional ecosystem services supply and demand," *Environmental Modelling & Software*, vol. 75, pp. 439–458, 2016.

[115] R. D. Lieberthal, *Matching Supply and Demand*. Cham: Springer International Publishing, 2016, pp. 145–171. [Online]. Available: https://doi.org/10.1007/978-3-319-43796-5_6

[116] K. Visscher, P. Stegmaier, A. Damm, R. Hamaker-Taylor, A. Harjanne, and R. Giordano, "Matching supply and demand: A typology of climate services," *Climate Services*, vol. 17, p. 100136, 2020.

[117] A. Goswami, F. Hedayati, and P. Mohapatra, "Recommendation systems for markets with two sided preferences," in *2014 13th International Conference on Machine Learning and Applications*. IEEE, 2014, pp. 282–287.

[118] G. K. Patro, A. Biswas, N. Ganguly, K. P. Gummadi, and A. Chakraborty, "Fairrec: Two-sided fairness for personalized recommendations in two-sided platforms," in *Proceedings of the web conference 2020*, 2020, pp. 1194–1204.

[119] L. Muzellec, S. Ronteau, and M. Lambkin, "Two-sided internet platforms: A business model lifecycle perspective," *Industrial Marketing Management*, vol. 45, pp. 139–150, 2015.

[120] L. S. Shapley, *A value for n-person games*. Cambridge University Press, 1988, p. 31–40.

[121] D. Janzing, L. Minorics, and P. Blöbaum, "Feature relevance quantification in explainable ai: A causal problem," in *International Conference on artificial intelligence and statistics*. PMLR, 2020, pp. 2907–2916.

[122] E. Štrumbelj and I. Kononenko, "Explaining prediction models and individual predictions with feature contributions," *Knowledge and information systems*, vol. 41, no. 3, pp. 647–665, 2014.

[123] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in neural information processing systems*, vol. 30, 2017.

[124] A. Asudeh, N. Zhang, and G. Das, "Query reranking as a service," *Proceedings of the VLDB Endowment*, vol. 9, no. 11, pp. 888–899, 2016.

[125] I. E. Kumar, S. Venkatasubramanian, C. Scheidegger, and S. Friedler, "Problems with shapley-value-based explanations as feature importance measures," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5491–5500.

[126] S. S. Fatima, M. Wooldridge, and N. R. Jennings, "A linear approximation method for the shapley value," *Artificial Intelligence*, vol. 172, no. 14, pp. 1673–1699, 2008.

[127] W. Hoeffding, "Probability inequalities for sums of bounded random variables," in *The collected works of Wassily Hoeffding*. Springer, 1994, pp. 409–426.

[128] M. T. Ribeiro, S. Singh, and C. Guestrin, "" why should i trust you?" explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.

[129] M. S. Acharya, A. Armaan, and A. S. Antony, "A comparison of regression models for prediction of graduate admissions," in *2019 international conference on computational intelligence in data science (ICCIDS)*. IEEE, 2019, pp. 1–5.

[130] C. Molnar, *Interpretable Machine Learning*. Lulu.com, 2020. [Online]. Available: https://books.google.com/books?id=jBm3DwAAQBAJ

[131] Y.-K. Che, J. Kim, and F. Kojima, "Stable matching in large economies," *Econometrica*, vol. 87, no. 1, pp. 65–110, 2019.

[132] T. Hößler, P. Schulz, E. A. Jorswieck, M. Simsek, and G. P. Fettweis, "Stable matching for wireless urllc in multi-cellular, multi-user systems," *IEEE Transactions on Communications*, vol. 68, no. 8, pp. 5228–5241, 2020.

[133] A. Abdulkadiroglu and T. Sönmez, "Matching markets: Theory and practice," *Advances in Economics and Econometrics*, vol. 1, pp. 3–47, 2013.

[134] R. Hakimov and D. Kübler, "Experiments on matching markets: A survey," WZB Discussion Paper, Tech. Rep., 2019.

[135] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, no. 1, pp. 9–15, 1962.

[136] A. E. Roth and M. Sotomayor, "Two-sided matching," *Handbook of game theory with economic applications*, vol. 1, pp. 485–541, 1992.

[137] G. S. Becker, "A theory of marriage: Part i," *Journal of Political economy*, vol. 81, no. 4, pp. 813–846, 1973.

[138] T. C. Bergstrom and M. Bagnoli, "Courtship as a waiting game," *Journal of political economy*, vol. 101, no. 1, pp. 185–202, 1993.

[139] N. Shimada, N. Yamazaki, and Y. Takano, "Multi-objective optimization models for many-to-one matching problems," *Journal of Information Processing*, vol. 28, pp. 406–412, 2020.

[140] D. J. Abraham, R. W. Irving, and D. F. Manlove, "Two algorithms for the student-project allocation problem," *Journal of discrete algorithms*, vol. 5, no. 1, pp. 73–90, 2007.

[141] A. E. Roth and M. Sotomayor, "The college admissions problem revisited," *Econometrica: Journal of the Econometric Society*, pp. 559–570, 1989.

[142] S. Qian, B. Wang, S. Li, Y. Sun, Y. Yu, and J. Wang, "Many-to-many matching for social-aware minimized redundancy caching in d2d-enabled cellular networks," *Computer Networks*, vol. 175, p. 107249, 2020.

[143] X. Kong, L. Wen, J. Ren, M. Hou, M. Zhang, K. Liu, and F. Xia, "Many-to-many collaborator recommendation based on matching markets theory," in *2019 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. IEEE, 2019, pp. 109–114.

[144] J. Ren, F. Xia, X. Chen, J. Liu, M. Hou, A. Shehzad, N. Sultanova, and X. Kong, "Matching algorithms: fundamentals, applications and challenges," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 5, no. 3, pp. 332–350, 2021.

[145] J. Ramos *et al.*, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, vol. 242, no. 1. Citeseer, 2003, pp. 29–48.

[146] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information processing & management*, vol. 24, no. 5, pp. 513–523, 1988.

[147] S. T. Dumais *et al.*, "Latent semantic analysis," *Annu. Rev. Inf. Sci. Technol.*, vol. 38, no. 1, pp. 188–230, 2004.

[148] T. Hofmann, "Probabilistic latent semantic indexing," in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, 1999, pp. 50–57.

[149] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck, "Learning deep structured semantic models for web search using clickthrough data," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2013, pp. 2333–2338.

[150] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil, "A latent semantic model with convolutional-pooling structure for information retrieval," in *Proceedings of the 23rd ACM international conference on conference on information and knowledge management*, 2014, pp. 101–110.

[151] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Advances in artificial intelligence*, vol. 2009, 2009.

[152] T. Tran, K. Lee, Y. Liao, and D. Lee, "Regularizing matrix factorization with user and item embeddings for recommendation," in *Proceedings of the 27th ACM international conference on information and knowledge management*, 2018, pp. 687–696.

[153] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie, "Autorec: Autoencoders meet collaborative filtering," in *Proceedings of the 24th international conference on World Wide Web*, 2015, pp. 111–112.

[154] Y. Wu, C. DuBois, A. X. Zheng, and M. Ester, "Collaborative denoising auto-encoders for top-n recommender systems," in *Proceedings of the ninth ACM international conference on web search and data mining*, 2016, pp. 153–162.

[155] D. Alvarez Melis and T. Jaakkola, "Towards robust interpretability with self-explaining neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[156] S. Teso, "Toward faithful explanatory active learning with self-explainable neural nets," in *Proceedings of the Workshop on Interactive Adaptive Learning (IAL 2019)*. CEUR Workshop Proceedings, 2019, pp. 4–16.

[157] M. Verma and D. Ganguly, "Lirme: locally interpretable ranking model explanation," in *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2019, pp. 1281–1284.

[158] A. Gale and A. Marian, "Metrics for explainable ranking functions," in *Proceedings of the 2nd International Workshop on ExplainAble Recommendation and Search (EARS 2019)*, 2019.

[159] ——, "Explaining monotonic ranking functions," *Proceedings of the VLDB Endowment*, vol. 14, no. 4, pp. 640–652, 2020.

[160] A. Asudeh, H. Jagadish, G. Miklau, and J. Stoyanovich, "On obtaining stable rankings," *Proceedings of the VLDB Endowment*, vol. 12, no. 3, 2018.

[161] S. M. Lundberg, G. G. Erion, and S.-I. Lee, "Consistent individualized feature attribution for tree ensembles," *arXiv preprint arXiv:1802.03888*, 2018.

BIOGRAPHICAL STATEMENT

Suraj Shetiya received his Bachelor of Engineering degree in Computer Science from Visvesvaraya Technological University, India, in 2010. Following this, he worked on cutting edge storage technologies like Snap-Mirror and SnapVault at NetApp, Bangalore. He started his Master's in Computer Science program at University of Texas at Arlington right after and completed it in 2017. Post his Master's degree, he started pursuing doctoral research under the supervision of Dr. Gautam Das. During the course of his doctorate, he has served as Graduate Teaching Assistant in the Computer Science Department in various courses from 2018 to 2023. He is the recipient of the STEM fellowship from 2017 to 2023. For his outstanding work as a PhD student, he has received **John S. Schuchman Outstanding Doctoral Student** and **Outstanding Doctoral Dissertation** from the Computer Science department. During his Ph.D., he has been co-author of 6+ peer-reviewed top-tier conference papers (SIGMOD, VLDB, ICDE) which have 35+ citations. His current research interests lie in responsible data management, compact representatives of databases and metric space problems in databases.