# On “One of the Few” Objects

You Wu[†] Pankaj K. Agarwal[†] Chengkai Li[‡] Jun Yang[†] Cong Yu[§]
[†]Duke University, [‡]University of Texas at Arlington, [§]Google Research

## ABSTRACT

Objects with multiple numeric attributes can be compared within any “subspace” (subset of attributes). In applications such as computational journalism, users are interested in claims of the form: *Karl Malone is one of the only two players in NBA history with at least 25,000 points, 12,000 rebounds, and 5,000 assists in one’s career.* One challenge in identifying such “one-of-the-$k$” claims ($k = 2$ above) is ensuring their “interestingness.” A small $k$ is not a good indicator for interestingness, as one can often make such claims for many objects by increasing the dimensionality of the subspace considered. We propose a uniqueness-based interestingness measure for one-of-the-few claims that is intuitive for non-technical users, and we design algorithms for finding all interesting claims (across all subspaces) from a dataset. Sometimes, users are interested primarily in the objects appearing in these claims. Building on our notion of interesting claims, we propose a scheme for ranking objects and an algorithm for computing the top-ranked objects. Using real-world datasets, we evaluate the efficiency of our algorithms as well as the advantage of our object-ranking scheme over popular methods such as Kemeny optimal rank aggregation and weighted-sum ranking.

## 1 Introduction

Raw data in various domains are becoming more widely accessible, e.g., play-by-play game logs for sports, databases of campaign finance and voting records for politics, etc. An increasing number of news stories are driven by information extracted from data. Computational journalism [5, 6] is a nascent field about using computing to help improve effectiveness and reduce cost for journalism, which serves a vital role in our society. One important goal in computational journalism is *(semi-)automatic lead identification* from data, i.e., finding interesting information nuggets from raw data that lead to further investigation and/or news stories around them. In this paper, we take a first step toward this goal by considering a popular form of claims exemplified by the following:

- *There is no player in NBA history with more points, more rebounds, and more assists than Oscar Robertson in one’s career.*
- *Rick Perry is one of the only three candidates in the 2012 US federal election cycle to have received at least \$600k from “lawyers & lobbyists” (an interest group that is usually pro-Democrat) and \$400k from “energy & natural resources” (usually pro-Republican).*

These two claims share a common structure: both are about an object being one of the few that “stand out” when compared according to a set of numeric attributes. More precisely, given a set of objects $\mathcal{O}$, each with a set $\mathcal{A}$ of numeric attributes, a *one-of-the-few* claim has the following form:

***Object $o$ is dominated by fewer than $k$ objects in a non-empty subset $\mathcal{B} \subseteq \mathcal{A}$ of attributes.***

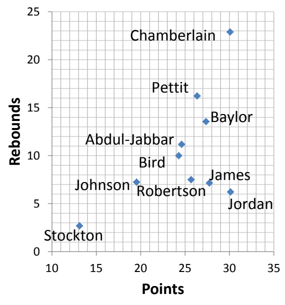
Here, we say $o'$ dominates $o$ in $\mathcal{B}$ if $o'$ is no worse than $o$ for all attributes in $\mathcal{B}$, and $o'$ is strictly better than $o$ for at least one attribute in $\mathcal{B}$. Journalists are interested in two tasks: 1) finding all “interesting” one-of-the-few claims from a given dataset; 2) ranking objects based on what one-of-the-few claims can be made about them. The first task allows journalists to identify claims that can be used in stories or serve as leads for further investigation. The second task provides an object-centric view that allows journalists to prioritize their investigation of particular objects (especially when there are many interesting one-of-the-few claims).

**Task 1: Finding Claims** One-of-the-few claims are closely related to the concept of *$k$-skyband* for multi-dimensional data [12]. Intuitively, the $k$-skyband of a set $\mathcal{O}$ of objects in subspace $\mathcal{B}$ is the subset of objects each dominated by fewer than $k$ other objects in $\mathcal{B}$. (The better-known notion of *skyline* is a special case of $k$-skyband where $k = 1$.) From the $k$-skyband in $\mathcal{B}$, a one-of-the-few claim can be generated for each object in the skyband straightforwardly. While various algorithms exist for computing a $k$-skyband given $k$ and $\mathcal{B}$, they do not address the key challenge of ensuring “interestingness” of the claims they find in a way that is easy for users to control and interpret. Although the parameter $k$ can be tuned, it is a poor indicator of interestingness, as illustrated by the following example.

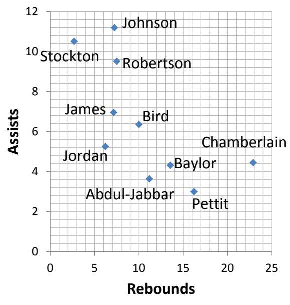
**Example 1.** *Consider the set of nearly* 4000 *players in NBA history, with stats such as career total* points, rebounds, *and* assists. *Being one of the top* 50 *leading scorers—i.e., in the* 50*-skyband for the single-attribute subspace* {points}*—is quite an impressive feat. However, if we expand the subspace to* {points, rebounds}, 142 *players now fit the bill—i.e., each is dominated by fewer than* 50 *others in* {points, rebounds}. *If we further include* assists *in the subspace,* 324 *(almost* 9% *of all) players will be in the* 50*-skyband.*

Example 1 clearly illustrates why we cannot use a universal $k$ to ensure interestingness of one-of-the-few claims for different subspaces: the size of the skyband tends to increase rapidly as dimensionality goes up. For example, the expected number of 1-skyband (skyline) objects is $O(\ln^{d-1}|\mathcal{O}|/(d-1)!)$ for a $d$-dimensional subspace [1], assuming no correlation between attributes. With a fixed $k$, too many claims can be in high-dimensional subspaces, making them less interesting. Clearly, $k$ needs to be adjusted when $|\mathcal{B}|$ changes. Furthermore, the appropriate setting for $k$ cannot be expressed simply as a function of dimensionality, because data characteristics—for example, correlation among attributes—also matter, as illustrated below.

**Example 2.** *We plot a subset of NBA players, with attributes* points, rebounds, *and* assists *per game, as points in subspaces* {points, rebounds} *(Figure 1a) and* {rebounds, assists} *(Figure 1b). It is*



(a) Positive: *points* vs. *rebounds*



(b) Negative: *rebounds* vs. *assists*

Figure 1: Correlation in NBA player stats.

*easy to see that the skybands in Figure 1a tend to be smaller than those in Figure 1b, because of the positive correlation between* points *and* rebounds *and the negative correlation between* rebounds *and* assists*. For example, the* 3*-skyband in* {points, rebounds} *contains* 5 *players (Jordan, Chamberlain, James, Baylor, and Pettit), which translate into* 5 *one-of-the-*3 *claims; on the other hand, the* 3*-skyband in* {rebounds, assists} *contains* 9 *players (all except Jordan). Hence, no single choice of $k$ is appropriate for these two subspaces of the same dimensionality.*

Example 2 above clearly illustrates that we cannot hope to define interestingness, which is data-dependent, by a function of $k$ and $|\mathcal{B}|$ alone. Asking the user to pick the right $k$ manually for each and every subspace is also infeasible. Our quest is to find an effective way of ensuring claim interestingness such that: 1) users are not required to tune lots of parameters; 2) the results are easy to understand and explain in layman's terms. Both properties are critical for computational journalism, where journalists may be non-technical and the results need to be explained to the general public in stories.

**Task 2: Ranking Objects** Even with an appropriate definition of "interestingness," many objects may be the subject of at least one interesting one-of-the-few claim in some subspace. The task of ranking objects allows users to prioritize their effort in investigating objects. From our experience analyzing real data and preliminary user studies, different data domains and user preferences call for some degree of customization in ranking, as illustrated below.

**Example 3.** *Both John Stockton and Larry Bird are inductees into the Naismith Memorial Basketball Hall of Fame, but they have very different playing styles. Stockton has the second highest assists per game in NBA history, but is not very impressive in points or rebounds. Bird ranks 17th in points, 60th rebounds, and 44th in assists. Stockton and Bird exemplify what we call "specialized" and "well-rounded" objects, respectively. How to rank specialized objects relative to well-rounded ones often depends on the context in which the ranking will be used, or may simply be a matter of personal opinion.*

A popular method for ranking objects according to multiple attributes is *Kemeny optimal rank aggregation* [7], or *Kemeny* for short. It produces a "consensus" ranking that minimizes the number of pairwise disagreements (in the relative ordering of two objects) with respect to the rankings under individual attributes. Kemeny tends to downgrade objects that rank extremely high in very few attributes but considerably low in other attributes. For Example 3 above, Bird, who is well-rounded, would be ranked as the 9th by Kemeny, while Stockton, who is specialized, would be as low as the 139th, which may not be acceptable to some.

While Kemeny leaves no option for customization, another popular method, *weight-sum ranking*, exposes too many knobs. With weighted-sum, a user specifies a preference vector, whose components represent weights assigned to individual attributes; objects are then ranked according to their projection onto this vector (i.e., weighted combination of their attribute values). For a $d$-dimensional dataset, the user needs to properly specify $d$ weights; even if we learn these weights automatically, training examples must be provided by the user. Such requirements may overwhelm journalists with little time or technical expertise. Our goal is to devise a ranking scheme with as few knobs as possible, which would allow customization without overwhelming users.

**Main Contributions** First, we propose a simple but effective definition for the interestingness of a claim based on its "uniqueness." For a one-of-the-few claim (with a particular $k$ in a particular subspace $\mathcal{B}$) to be interesting, we require that this claim (with the same $k$ and $\mathcal{B}$) cannot be made for more than $\tau$ objects. Unlike $k$, $\tau$ is a user-defined threshold that applies universally to all subspaces, significantly reducing the burden on the user to define interestingness. For each subspace, our definition automatically adapts $k$ in a data-dependent way, and naturally excludes those high-dimensional subspaces where no claims are unique enough. Furthermore, $\tau$ is also easy to understand for non-technical users.

Based on this definition, we introduce the problem of finding all interesting one-of-the-few claims across all non-empty subspaces, given the uniqueness threshold $\tau$. The fact that our $k$ is data-dependent and not fixed raises unique challenges not addressed by previous work on computing skylines and skybands. We devise efficient algorithms that avoid redundant computation. In particular, we are able to improve the worst-case complexity of finding all interesting claims in a subspace from $O(|\mathcal{O}|^2)$ to $O(\tau|\mathcal{O}|)$, which is attractive because $\tau$ in practice is small for claims to be unique.

Building on the definition of interestingness, we propose a novel scheme for scoring and ranking objects based on the aggregated interestingness of claims involving them. One key insight distinguishing our scheme from others such as Kemeny and weighted-sum is that we in effect aggregate ranks across all non-empty subspaces as opposed to just individual attributes. Our scheme supports tuning by a single parameter $\alpha$, which captures user preference between specialized and well-rounded objects, and overcomes the inflexibility of Kemeny without resorting to an overwhelming number of knobs like weighted-sum. Extending the algorithms for finding all interesting claims, we show how to compute top-ranked objects efficiently given $\alpha$. We experimentally demonstrate, on real datasets, that our scheme is able to produce rankings comparable to Kemeny and weighted-sum while offering more effective customization.

## 2 Finding One-of-the-Few Claims

**Preliminaries** Consider a set $\mathcal{O}$ of $n$ objects, each with $d$ numeric attributes $\mathcal{A} = \{A_1, A_2, \ldots, A_d\}$. A *subspace* is a subset of the attributes and the set of all subspaces of $\mathcal{A}$ form a lattice. We say that subspace $\mathcal{B}_1$ is an *ancestor* (*descendant*) of subspace $\mathcal{B}_2$ if $\mathcal{B}_1 \subseteq \mathcal{B}_2$ (resp. $\mathcal{B}_1 \supseteq \mathcal{B}_2$). We say $\mathcal{B}_1$ is a *parent* (*child*) of $\mathcal{B}_2$ if $\mathcal{B}_1 \subseteq \mathcal{B}_2$ (resp. $\mathcal{B}_1 \supseteq \mathcal{B}_2$) and their cardinalities differ by one.

We say $o_1$ *dominates* $o_2$ in subspace $\mathcal{B}$, denoted $o_1 \succ_{\mathcal{B}} o_2$, if i) $\forall A \in \mathcal{B}, o_1.A \geq o_2.A$, and ii) $\exists A \in \mathcal{B}, o_1.A > o_2.A$. Clearly, dominance is transitive: if $o_1 \succ_{\mathcal{B}} o_2$ and $o_2 \succ_{\mathcal{B}} o_3$, then $o_1 \succ_{\mathcal{B}} o_3$.

**Definition 1** (Dominating Subset, $k$-Skyband, Skyline, Tier)**.**

- *The* dominating subset *of $o \in \mathcal{O}$ in subspace $\mathcal{B} \subseteq \mathcal{A}$, denoted $\mathfrak{D}_{\mathcal{B}}(\mathcal{O}, o)$, is the subset of objects that dominate $o$ in $\mathcal{B}$; i.e., $\mathfrak{D}_{\mathcal{B}}(\mathcal{O}, o) = \{o' \in \mathcal{O} \mid o' \succ_{\mathcal{B}} o\}$. Let $\delta_{\mathcal{B}}(\mathcal{O}, o) = |\mathfrak{D}_{\mathcal{B}}(\mathcal{O}, o)|$ denote the size of the dominating subset.*

- *The* $k$-skyband *($k \geq 1$) of $\mathcal{O}$ in subspace $\mathcal{B}$, denoted $\mathfrak{S}^k_{\mathcal{B}}(\mathcal{O})$, is the subset of objects in $\mathcal{O}$ that are each dominated by fewer than $k$ other objects in $\mathcal{O}$; i.e. $\mathfrak{S}^k_{\mathcal{B}}(\mathcal{O}) = \{o \in \mathcal{O} \mid \delta_{\mathcal{B}}(\mathcal{O}, o) < k\}$.*
- *The* skyline *of $\mathcal{O}$ in subspace $\mathcal{B}$ is $\mathfrak{S}^1_{\mathcal{B}}(\mathcal{O})$, i.e., the 1-skyband.*
- *The* $i$-th tier *($i \geq 1$) of $\mathcal{O}$ in subspace $\mathcal{B}$ is the subset of objects in $\mathcal{O}$ that are each dominated by exactly $i - 1$ other objects in $\mathcal{O}$; i.e. $\{o \in \mathcal{O} \mid \delta_{\mathcal{B}}(\mathcal{O}, o) = i - 1\}$.*

Clearly, by definition, the $k$-skyband $\mathfrak{S}^k_{\mathcal{B}}(\mathcal{O})$ is the disjoint union of all $i$-th tiers with $i \leq k$, and the difference between the $k$-skyband and the $(k-1)$-skyband is the $k$-th tier.[1]

To illustrate, consider the set $\mathcal{O}$ of 10 NBA players and the subspace $\mathcal{B} = \{\textit{rebounds}, \textit{assists}\}$ shown in Figure 1b. $\mathfrak{D}_{\mathcal{B}}(\mathcal{O}, \text{Stockton}) = \{\text{Johnson}\}$, so $\delta_{\mathcal{B}}(\mathcal{O}, \text{Stockton}) = 1$. $\mathfrak{S}^1_{\mathcal{B}}(\mathcal{O}) = \{$Johnson, Robertson, Bird, Chamberlain$\}$ is the skyline (or 1-skyband), which is also the 1-st tier. $\mathfrak{S}^2_{\mathcal{B}}(\mathcal{O}) = \mathfrak{S}^1_{\mathcal{B}}(\mathcal{O}) \cup \{$Stockton, Baylor, Pettit$\}$ is the 2-skyband, where Stockton, Baylor, and Pettit are in the 2nd tier and each dominated by exactly one object in $\mathcal{O}$. $\mathfrak{S}^3_{\mathcal{B}}(\mathcal{O})$, the 3-skyband, additionally includes the 3rd tier $\{$James, Abdul-Jabbar$\}$, leaving only Jordan out, as mentioned in Example 2.

**Problem Statement** As motivated in Section 1, while each object in the $k$-skyband translates into a one-of-the-few claim, we measure the interestingness of this claim by the number of objects for which similar claims can be made, i.e., the size of the $k$-skyband. Instead of struggling with setting $k$, which depends on the subspace and object distribution, a user should be able to specify a single threshold $\tau$ that caps the number of similar claims. Therefore, we introduce the concept of *top-$\tau$ skyband* below. While the concept is closely related to $k$-skyband, a crucial difference is that a top-$\tau$ skyband is defined by its size, while a $k$-skyband, defined by its number of tiers, can be arbitrarily large.

**Definition 2** (Top-$\tau$ Skyband)**.** *Given $\tau \geq 1$, the* top-$\tau$ skyband *of a set of objects $\mathcal{O}$ in subspace $\mathcal{B}$ is the largest skyband whose size does not exceed $\tau$. In other words, it is the $\hat{k}$-skyband where $\hat{k} = \max\{k \mid \tau \geq |\mathfrak{S}^k_{\mathcal{B}}(\mathcal{O})|\}$.*

The fact that an object $o$ belongs to the top-$\tau$ skyband with the $k$-th tier as its last non-empty tier—or alternatively, the $k$-skyband with size no more than $\tau$—translates into the following statement:

*Object $o$ is dominated by fewer than $k$ objects in $\mathcal{B}$, and this claim cannot be made for more than $\tau$ objects.*

Intuitively, $\tau$ measures the uniqueness of the claim made by the first part of the statement above. For example, in Figure 1a, suppose we set $\tau = 3$. The top-3 skyband is the 1-skyband $\{$Chamberlain, Jordan$\}$. The 2-skyband would be too big, because it additionally contains Pettit, Baylor, and James and has size $5 > 3 = \tau$. Note that the 3rd tier is empty—no player is dominated by exactly two others in this example—so the 3-skyband (recall Example 2) is the same as the 2-skyband.

The problem of finding all interesting one-of-the-few claims can now be formulated as follows:

**Definition 3** (Finding Top-$\tau$ Skybands in All Subspaces)**.** *Given the set $\mathcal{O}$ of objects and a user-specified threshold $\tau$, find, for every non-empty subspace $\mathcal{B} \subseteq \mathcal{A}$, the top-$\tau$ skyband of $\mathcal{O}$ in $\mathcal{B}$.*

For each subspace, the membership of an object in the top-$\tau$ skyband corresponds a one-of-the-few claim that cannot be made for more than $\tau$ objects. This definition leads naturally to some desired features. In a single-attribute subspace, the top-$\tau$ skyband contains essentially the top $k$ objects ranked by this attribute (but with better handling of ties). As the subspace dimensionality goes up, $k$ decreases in an automatic, data-dependent manner until the $k$-skyband contains no more than $\tau$ objects. Thus, with this problem formulation, users do not need to pick $k$ manually for different subspaces. To illustrate, consider again Example 2. Suppose the user sets $\tau = 8$. For subspace $\{\textit{points}, \textit{rebounds}\}$ (Figure 1a), the top-8 skyband would be the 5-skyband. In contrast, for $\{\textit{rebounds}, \textit{assists}\}$ (Figure 1b), the top-8 skyband would be the 2-skyband. Finally, in very high-dimensional subspaces where so many objects are on the skyline that no claims are interesting, our problem formulation correctly leads to an empty top-$\tau$ skyband.

**Overview of Solutions** The rest of this section describes our algorithms for finding all interesting one-of-the-few claims. At a high level, we 1) traverse the lattice of subspaces in some manner, and 2) compute the top-$\tau$ skyband for each subspace we visit. Techniques for improving efficiency exist both across subspaces and within each subspace. For finding the top-$\tau$ skyband in a given subspace $\mathcal{B}$, we propose two algorithms in Sections 2.1 and 2.2. We then show in Section 2.3 how to explore the lattice of subspaces using these algorithms as subroutines.

Note that computing the top-$\tau$ skyband for a single-attribute subspace $\{A\}$ simply involves finding the top $\tau$ objects sorted by $A$; algorithms in Sections 2.1 and 2.2 are needed only when $|\mathcal{B}| > 1$.

Also note that it is possible to devise solutions by adapting existing techniques from literature. We present one such solution here, which we call *Baseline*. For lattice traversal, Baseline adopts the strategy of *bottom-up skyline (BUS)* of Pei et al. [13], who study the problem of computing the skyline in every subspace. BUS can be extended to compute $k$-skyband if $k$ is given. In each subspace, Baseline starts with $k = 1$, and computes the $k$-skyband and increments $k$, iteratively, until the $k$-skyband contains at least $\tau$ objects. Obviously, this iterative process of finding the right $k$ leads to a lot of redundant computation. Our new algorithms avoid such redundant computation, and we experimentally validate their advantages over Baseline in Section 4.

## 2.1 Progressive Top-$\tau$ Skyband Algorithm

As it turns out, all objects in the $(k+1)$-th tier must lie on the skyline of the set of remaining objects after those in the $k$-skyband are taken out. This simple but useful observation has probably been made in other contexts too; we state it as a lemma here for completeness.

**Lemma 1.** $\mathfrak{S}^{k+1}_{\mathcal{B}}(\mathcal{O}) \setminus \mathfrak{S}^k_{\mathcal{B}}(\mathcal{O}) \subseteq \mathfrak{S}^1_{\mathcal{B}}\big(\mathcal{O} \setminus \mathfrak{S}^k_{\mathcal{B}}(\mathcal{O})\big)$.

*Proof.* Let $\overline{\mathfrak{S}}$ denote $\mathcal{O} \setminus \mathfrak{S}^k_{\mathcal{B}}(\mathcal{O})$. Consider any $o$ in the $(k+1)$-th tier. Clearly $o \in \overline{\mathfrak{S}}$ because $\delta_{\mathcal{B}}(\mathcal{O}, o) = k$. If $o \notin \mathfrak{S}^1_{\mathcal{B}}(\overline{\mathfrak{S}})$, then there exist $o' \in \overline{\mathfrak{S}}$ such that $o' \succ_{\mathcal{B}} o$. By transitivity of dominance, every object that dominates $o'$ also dominates $o$, so $\delta_{\mathcal{B}}(\mathcal{O}, o) > \delta_{\mathcal{B}}(\mathcal{O}, o') \geq k$, a contradiction. □

Lemma 1 suggests the following strategy, which we call *Progressive* (Algorithm 1), for computing the top-$\tau$ skyband for a given subspace. Given $\tau$, Progressive computes the answer set $\Phi$ tier by tier, starting from the first. By Lemma 1, to obtain the next non-empty tier, we first compute (Line 3) the skyline $S$ for the set of remaining objects (those outside the current $\Phi$). Objects in the next non-empty tier ($\Delta\Phi$ on Line 5) are those in $S$ whose dominating subsets in $\mathcal{O}$ are the smallest in size. We add this tier to $\Phi$ as long as the resulting answer set has no more than $\tau$ objects. It is easy to see that at the end of each iteration, the invariant that $\Phi = \mathfrak{S}^k_{\mathcal{B}}(\mathcal{O})$ holds. Progressive terminates if the addition of the next non-empty tier causes the size of the answer set to exceed $\tau$.

[1]Note that tiers differ from the well-known concept of *maximal layers*, where the next maximal layer is defined as the skyline of the set of objects outside all previous layers.

**Algorithm 1:** Progressive$(\mathcal{O}, \mathcal{B}, \tau)$

```
  Data: object set $\mathcal{O}$, subspace $\mathcal{B}$, and size threshold $\tau$
  Result: the top-$\tau$ skyband of $\mathcal{O}$ in $\mathcal{B}$
1 $\Phi \leftarrow \varnothing$; $k \leftarrow 0$;
2 while true do
3     $S \leftarrow \mathfrak{S}^1_{\mathcal{B}}(\mathcal{O} \setminus \Phi)$;
4     $k' \leftarrow \min\{\delta_{\mathcal{B}}(\Phi, o) + 1 \mid o \in S\}$;
5     $\Delta\Phi \leftarrow \{o \in S \mid \delta_{\mathcal{B}}(\Phi, o) + 1 = k'\}$;
6     if $|\Phi \cup \Delta\Phi| > \tau$ then break;
7     $\Phi \leftarrow \Phi \cup \Delta\Phi$; $k \leftarrow k'$;
8 return $\Phi$;
```

Note that on Lines 4 and 5 we compute the size of the dominating subset for $o \in S$ in $\Phi$ instead of $\mathcal{O}$. This optimization is correct because, being on the skyline of $\mathcal{O} \setminus \Phi$, $o$ is not dominated by any object in $\mathcal{O} \setminus \Phi$, so $\delta_{\mathcal{B}}(\Phi, o) = \delta_{\mathcal{B}}(\mathcal{O}, o)$.

**Further Optimizations** For clarity of presentation, Algorithm 1 leaves out some details, which we describe below. Suppose that in the previous iteration, the skyline computed was $\check{S}$, and $\Delta\check{\Phi} \subseteq \check{S}$ was added to the answer set. For the current iteration, it is easy to see that $\check{S} \setminus \Delta\check{\Phi} \subseteq S$; i.e., all remaining objects in $\check{S}$ will appear again in the skyline $S$ to be computed. This observation leads to two optimizations. **1)** We can speed up successive skyline computations on Line 3 across iterations. Specifically, we adapt the *SUBSKY* algorithm of Tao et al. [15] (although in general any skyline algorithm can be used). The original SUBSKY uses an index whose size is linear in the $|\mathcal{O}|$ to help compute the skyline of $\mathcal{O}$ in any subspace. During execution, SUBSKY always maintains the skyline for the subset of objects that it has examined. In our adaption of SUBSKY, we remove from (a copy of) the SUBSKY index any object added to $\Phi$, and we "seed" each invocation of SUBSKY with $\check{S} \setminus \Delta\check{\Phi}$ instead of starting it with an empty skyline. **2)** We can reduce the number of dominance tests involved in determining $\delta_{\mathcal{B}}(\Phi, o)$ for $o \in S$ on Lines 4 and 5. For any $o \in \check{S} \setminus \Delta\check{\Phi} \subseteq S$, we have already computed $\delta_{\mathcal{B}}(\mathcal{O}, o)$ in the previous iteration, so there is no need to recompute it.

**Complexity** Following convention [15], we measure the performance of our algorithms by the number of dominance tests. Progressive performs two types of such tests: 1) those involved in computing the skyline of the remaining objects, and 2) those involved in computing the sizes of the dominating subsets in $\mathcal{O}$ for objects on that skyline. The number of type-1 tests depends on the skyline algorithm; Tao et al. [15] describes various techniques to reduce it for SUBSKY. However, in the worst case, $|\Phi|$ is almost $\tau$ in the final iteration, while $|S|$ can be roughly $|\mathcal{O}| - \tau$ (i.e., the next non-empty tier contains nearly all remaining objects). In this case, the total number of dominance tests would be $\Theta(|\mathcal{O}|^2)$.

## 2.2 One-Pass Top-$\tau$ Skyband Algorithm

Progressive has poor worst-case complexity: it computes the entire next tier in order to decide whether to add that tier to the answer, but that tier can be much larger than $\tau$. In this section, we show how to tame the complexity in terms of $\tau$ with another algorithm *OnePass*. This algorithm works by considering object one by one in a particular order while maintaining an answer set $\Phi$. Dominance tests are only performed between the object being considered and those in the current $\Phi$. The processing order is chosen in a "safe" way, as defined below, which allows OnePass to cap $|\Phi|$ at $\tau$ at all times, thereby bounding the number of dominance tests to $\tau|\mathcal{O}|$.

**Definition 4** (Safe Order)**.** *A order for a set $\mathcal{O}$ of objects is* safe *(for OnePass) if $o'$ precedes $o$ whenever $o' \succ_{\mathcal{B}} o$.*

Algorithm 2 describes OnePass. The details of implementing the safe order will be given later in this section. Here, we first explain the algorithm and establish its correctness, the crux of which is captured by the lemma below.

**Lemma 2.** *The following invariants are true at the end of each iteration of OnePass's main loop, where $\mathcal{O}^\star$ denotes the set of all objects processed so far.*

*(I-1) For all $o' \in \Phi$, $c[o'] = \delta_{\mathcal{B}}(\mathcal{O}, o')$.*

*(I-2) $\Phi = \mathfrak{S}^k_{\mathcal{B}}(\mathcal{O}^\star)$.*

*(I-3) $|\mathfrak{S}^{k+1}_{\mathcal{B}}(\mathcal{O})| > \tau$ (if $|\mathcal{O}| > \tau$).*

*Proof.* We prove the lemma by induction. All invariants obviously hold before the beginning of the first iteration. Suppose all invariants hold before the start of an iteration processing $o$. We show that they also hold when the current iteration ends.

For clarity, let $\check{\Phi}$ and $\check{k}$ denote the values of $\Phi$ and $k$, respectively, at the start of the current iteration; let $\check{\mathcal{O}}^\star = \mathcal{O}^\star \setminus \{o\}$ denote the set of objects processed before the current iteration.

(I1): At the end of the iteration, for any $o' \in \Phi$ that was also in $\check{\Phi}$, $c[o'] = \delta_{\mathcal{B}}(\mathcal{O}, o')$ stays true as OnePass does not update $c[o']$. It remains to be shown that if $o$ is added to $\Phi$, then $c[o] = \delta_{\mathcal{B}}(\mathcal{O}, o)$. Note that $c[o] = \delta_{\mathcal{B}}(\check{\Phi}, o) < \check{k}$. It suffice to show that all objects in $\mathcal{O}$ dominating $o$ are in $\check{\Phi}$. Suppose that is not the case. Let $o'$ denote the first object in safe order such that $o' \succ_{\mathcal{B}} o$ and $o' \notin \check{\Phi}$. Because of the safe processing order, we know $o' \in \check{\mathcal{O}}^\star$, and $\delta_{\mathcal{B}}(\check{\mathcal{O}}^\star \setminus \check{\Phi}, o') = 0$. Meanwhile, $\delta_{\mathcal{B}}(\check{\Phi}, o') \leq \delta_{\mathcal{B}}(\check{\Phi}, o) < \check{k}$. Therefore, $\delta_{\mathcal{B}}(\check{\mathcal{O}}^\star, o') = \delta_{\mathcal{B}}(\check{\mathcal{O}}^\star \setminus \check{\Phi}, o') + \delta_{\mathcal{B}}(\check{\Phi}, o') < \check{k}$, implying that $o' \in \mathfrak{S}^{\check{k}}_{\mathcal{B}}(\check{\mathcal{O}}^\star)$. However, by the inductive hypothesis, $\mathfrak{S}^{\check{k}}_{\mathcal{B}}(\check{\mathcal{O}}^\star) = \check{\Phi}$, contradicting the assumption that $o' \notin \check{\Phi}$.

(I2): First, for any object $o' \in \check{\mathcal{O}}^\star$, $o'$ cannot be dominated by $o$ because of the safe processing order. Therefore, $\delta_{\mathcal{B}}(\check{\mathcal{O}}^\star, o') = \delta_{\mathcal{B}}(\mathcal{O}^\star, o')$, which implies $o' \in \mathfrak{S}^j_{\mathcal{B}}(\check{\mathcal{O}}^\star) \Leftrightarrow o' \in \mathfrak{S}^j_{\mathcal{B}}(\mathcal{O}^\star)$ for any $j$. To complete the proof we need to consider the possible membership of $o$ in $\Phi$. There are two cases. Case 1: $o$ is not added to $\Phi$ because $\delta_{\mathcal{B}}(\check{\Phi}, o) \geq \check{k} = k$. In this case, $\delta_{\mathcal{B}}(\mathcal{O}^\star, o) \geq \delta_{\mathcal{B}}(\check{\Phi}, o) \geq k$, which means $o \notin \mathfrak{S}^k_{\mathcal{B}}(\mathcal{O}^\star)$. It is easy to verify that (I2) holds. Case 2: $o$ is added to $\Phi$. As shown in the proof for (I1), $c[o] = \delta_{\mathcal{B}}(\mathcal{O}, o) < \check{k}$, and all objects in $\mathcal{O}$ dominating $o$ are in $\check{\Phi} \subseteq \mathcal{O}^\star$. Therefore, $o \in \mathfrak{S}^{\check{k}}_{\mathcal{B}}(\mathcal{O}^\star)$. It is easy to verify that $\Phi = \mathfrak{S}^{\check{k}}_{\mathcal{B}}(\mathcal{O}^\star)$ right after Line 8. If Lines 9–11 are not executed, $k = \check{k}$ and (I2) obviously holds. Otherwise, consider Lines 9–11. For any $o' \in \Phi$ right before Line 11, $c[o'] = \delta_{\mathcal{B}}(\mathcal{O}, o')$ as shown by the proof for (I1). Because of the safe processing order, no object in $\mathcal{O} \setminus \mathcal{O}^\star$ dominates $o'$, so $c[o'] = \delta_{\mathcal{B}}(\mathcal{O}^\star, o')$. Therefore, Lines 9–11 in effect remove the last non-empty tier of $\Phi = \mathfrak{S}^{\check{k}}_{\mathcal{B}}(\mathcal{O}^\star)$ and update $k$ accordingly, so (I2) also holds.

(I3): If $k$ is not updated in the current iteration, (I3) obviously remains valid. Suppose $k$ is updated. As shown in the proof for (I2) above, $\check{\Phi} \cup \{o\} = \mathfrak{S}^{\check{k}}_{\mathcal{B}}(\mathcal{O}^\star)$, and Line 11 simply removes the last non-empty tier of this $\check{k}$-skyband, which is the $(k+1)$-th tier where $k + 1 \leq \check{k}$. Hence, $|\mathfrak{S}^{k+1}_{\mathcal{B}}(\mathcal{O}^\star)| = |\mathfrak{S}^{\check{k}}_{\mathcal{B}}(\mathcal{O}^\star)| = |\check{\Phi} \cup \{o\}| > \tau$ (by Line 9). Because of the safe processing order, no object in $\mathcal{O}^\star$ is dominated by any in $\mathcal{O} \setminus \mathcal{O}^\star$, so for all $o' \in \mathcal{O}^\star$, $\delta_{\mathcal{B}}(\mathcal{O}^\star, o') = \delta_{\mathcal{B}}(\mathcal{O}, o')$; therefore, $\mathfrak{S}^{k+1}_{\mathcal{B}}(\mathcal{O}^\star) \subseteq \mathfrak{S}^{k+1}_{\mathcal{B}}(\mathcal{O})$, so $|\mathfrak{S}^{k+1}_{\mathcal{B}}(\mathcal{O})| \geq |\mathfrak{S}^{k+1}_{\mathcal{B}}(\mathcal{O}^\star)| > \tau$. □

Consider the next object $o$ in safe order. OnePass first checks whether $o$ is dominated by at least $k$ objects in $\Phi$ (Lines 3–6). If yes, $o$ is ignored because it would be in the $(k+1)$-th or later tier (by (I-1)) and therefore cannot be in the top-$\tau$ skyband of $\mathcal{O}$

**Algorithm 2:** OnePass($\mathcal{O}, \mathcal{B}, \tau$)

```
   Data: object set $\mathcal{O}$, subspace $\mathcal{B}$, and size threshold $\tau$
   Result: the top-$\tau$ skyband of $\mathcal{O}$ in $\mathcal{B}$
1  $\Phi \leftarrow \varnothing$; $k \leftarrow \tau$;
2  foreach $o \in \mathcal{O}$ in safe order (Definition 4) do
3      $c[o] \leftarrow 0$; // count the number of objects in $\Phi$ dominating $o$
4      foreach $o' \in \Phi$ do // heuristically in ascending order of $c[o']$
5          if $o' \succ_{\mathcal{B}} o$ then $c[o] \leftarrow c[o] + 1$;
6          if $c[o] \geq k$ then break; // no need to count further
7      if $c[o] < k$ then
8          $\Phi \leftarrow \Phi \cup \{o\}$;
9          if $|\Phi| > \tau$ then // $\Phi$ is too big; remove the last tier
10             $k \leftarrow \max\{c[o'] \mid o' \in \Phi\}$;
11             $\Phi \leftarrow \Phi \setminus \{o' \in \Phi \mid c[o'] = k\}$;
12             if $k = 0$ then break;
13 return $\Phi$;
```

(by (I-3)). We stop counting as soon as $c[o]$ reaches $k$. Heuristically, we check $o$ against "better" objects in $\Phi$ (i.e., those with smaller dominating sets) first, in hope of reaching $k$ sooner with fewer dominance tests.

If $c[o] < k$, OnePass adds $o$ to $\Phi$ (Lines 8) and remember the count $c[o]$. If doing so makes $|\Phi|$ exceed $\tau$ we remove the last tier from $\Phi$ and update $k$ accordingly (Lines 9–11), to preserve (I-2) and (I-3). If $k$ drops to 0 (Line 12), that means even the skyline has size bigger than $\tau$ (by (I-3)), so we can terminate (and return $\varnothing$) without processing the remaining objects.

After all objects in $\mathcal{O}$ are processed, $\Phi$ is the top-$\tau$ skyband of $\mathcal{O}$ in $\mathcal{B}$, by (I-2) and (I-3).

**Producing a Safe Order** The simplest approach for OnePass to produce a safe processing order is to sort $\mathcal{O}$ by $\mathcal{B}$, which would take $O(|\mathcal{O}| \log |\mathcal{O}|)$ time. To avoid the cost of a full sort, we precompute, for each of the $d$ attributes in $\mathcal{A}$, a version of $\mathcal{O}$ sorted by that attribute. Given a subspace $\mathcal{B}$, OnePass picks the attribute $A \in \mathcal{B}$ with the largest domain (a heuristic also adopted in [2, 13]), and use the version of $\mathcal{O}$ sorted by $A$. During processing, if OnePass finds that a group of objects tie in $A$, OnePass further sorts this group by the full $\mathcal{B}$. As a further optimization, before sorting this group, OnePass first performs Lines 3–6 on each object $o$ in the group, and removes those with $c[o] \geq k$; the filtered group is then sorted and further processed. In the worst case, OnePass still needs to pay $O(|\mathcal{O}| \log |\mathcal{O}|)$ for sorting. In practice, however, we have found our heuristics very effective in eliminating sorting, because most ties tend to occur later in processing; by that time, most objects will be eliminated by Lines 3–6. Furthermore, the cost of filtering is low because $\tau$ and $k$ are typically small.

**Complexity** Because OnePass caps $|\Phi|$ at $\tau$ at all times, the number of dominance tests is bounded by $\tau|\mathcal{O}|$. As explained above, sorting adds $O(|\mathcal{O}| \log |\mathcal{O}|)$ in the worst case, though in practice the extra cost is rarely incurred. Furthermore, In high-dimensional subspaces, $k$ decreases rapidly as OnePass examines more objects. It is likely that an object will be discarded after a few dominance tests. OnePass also detects the case when the size of the skyline would exceed $\tau$, and is able to terminate without examining the whole $\mathcal{O}$. Thus, OnePass is expected to run much faster in practice than the bound suggests, particularly for high-dimensional subspaces.

## 2.3 Lattice Traversal

Finding all interesting one-of-the-few claims requires computing the top-$\tau$ skyband in every non-empty subspace. We now describe, on a high level, how to accomplish this task using either Progressive or OnePass as the building block.

For Progressive, computation in every subspace starts with finding the skyline. For this part, we directly apply the techniques of Pei et al. [13], who study the problem of computing the skyline in every subspace. Their techniques traverse the lattice of subspaces in either bottom-up or top-down order, and try to share computation across subspaces. For a given subspace $\mathcal{B}$, once we have computed the skyline using these techniques, we proceed with Progressive to compute the rest of the top-$\tau$ skyband as discussed in Section 2.1.

For OnePass, we traverse the lattice of subspaces top-down, i.e., going from low- to high-dimensional subspaces. We use the top-down technique from Pei et al. [13] to help compute the skyline in a subspace using those found in its parent subspaces. Moreover, we use the top-$\tau$ skybands in parent subspaces to fine-tune the safe processing order in the current subspace.

Furthermore, during a top-down lattice traversal, we use two tests to prune uninteresting high-dimensional subspaces from the search. 1) If the "distinct-count" of skyline objects in $\mathcal{B}$ (i.e., the number of distinct projections onto $\mathcal{B}$) is greater than $\tau$, all descendants of $\mathcal{B}$ must have empty top-$\tau$ skybands. 2) Given $\mathcal{B}$, if the union of skyline objects from $\mathcal{B}$'s parents already contains more than $\tau$ skyline objects in $\mathcal{B}$, $\mathcal{B}$ must have an empty top-$\tau$ skyband.

## 3 Ranking Objects

This section presents our solution for ranking the set $\mathcal{O}$ of objects with attributes $\mathcal{A}$, based on what one-of-the-few claims can be made about them across subspaces, and how interesting these claims are. Section 3.1 proposes a novel scoring scheme that captures varying user preferences with a single parameter, while Section 3.2 describes how the algorithms in Section 2 can be leveraged to compute top-ranked objects.

## 3.1 Scoring Objects

A common approach for ranking a set $\mathcal{O}$ of objects (e.g., political candidates) is to combine multiple ranked lists of them (e.g., by voters). Traditionally, the scores for objects in a single list are assigned according to a *positional scoring vector (or function)* [19], $v$, which maps a rank $i$ ($1 \leq i \leq |\mathcal{O}|$) to a numeric score $v(i)$, such that $v(i) \geq v(i+1)$. Some examples include *Borda*, where $v(i) = |\mathcal{O}| - i$, and *Plurality*, where $v(1) = 1$ and $v(i) = 0$ for $i > 1$. Then, the overall object ranking is done according to the aggregate score of each object, usually defined as the sum of its scores across all ranked lists.

A straightforward approach would be to have one ranked list of $\mathcal{O}$ by each attribute in $\mathcal{A}$, and sum up an object's scores across these lists. However, this approach has serious drawbacks, particularly in handling data with correlation. Suppose $o_1$ is ranked high for two correlated attributes (e.g., contributions from finance and real estate sectors, or minutes played and points scored), while $o_2$ is ranked equally high in two anti-correlated attributes (e.g., contributions from oil companies and environmental groups, or rebounds and assists). Since it is harder to rank high for two anti-correlated attributes, we should score $o_2$ higher than $o_1$ overall. However, the approach of summing scores over individual attributes will assign the same score to $o_1$ and $o_2$ (assuming all other factors are equal).

**Aggregate Positional Scoring with Ties (APST)** To resolve the issue above, we propose *Aggregate Positional Scoring with Ties*. The key novelty is to aggregate object scores not just across individual attributes, but instead over all non-empty subspaces. The dominance relationship in a multi-dimensional subspace $\mathcal{B}$ likely does not induce a totally ranked list; hence, we draw insight from Section 2 to score objects using the partial order in a way that reflects the uniqueness of one-of-the-few claims in $\mathcal{B}$. The result is a scoring scheme that naturally adapts to data distributions.

**Definition 5** (Aggregate Positional Scoring with Ties (APST))**.** *For each non-empty subspace $\mathcal{B} \subseteq \mathcal{A}$, sort $\mathcal{O}$ in ascending order of $\delta_{\mathcal{B}}(\mathcal{O}, o)$, the size of the dominating subset for each $o \in \mathcal{O}$; ties are broken arbitrarily. Denote this ranking by $\pi_{\mathcal{B}}$, where $\pi_{\mathcal{B}}(o) \in [1, |\mathcal{O}|]$ is the rank of $o$ in this ranking. Let $[\pi^{\vdash}_{\mathcal{B}}(o), \pi^{\dashv}_{\mathcal{B}}(o)]$ denote the range of ranks occupied by objects that tie with $o$ in $\delta_{\mathcal{B}}(\mathcal{O}, o)$; i.e., $\pi^{\vdash}_{\mathcal{B}}(o) = \min\{\pi_{\mathcal{B}}(o') \mid \delta_{\mathcal{B}}(\mathcal{O}, o') = \delta_{\mathcal{B}}(\mathcal{O}, o)\}$ and $\pi^{\dashv}_{\mathcal{B}}(o) = \max\{\pi_{\mathcal{B}}(o') \mid \delta_{\mathcal{B}}(\mathcal{O}, o') = \delta_{\mathcal{B}}(\mathcal{O}, o)\}$. Given a positional scoring function $v$ where $v(1) > v(2) > \cdots > v(|\mathcal{O}|)$, the* (APST) score of object $o$ in $\mathcal{B}$*, denoted $\gamma_{\mathcal{B}}(o)$, is given by:*

$$\gamma_{\mathcal{B}}(o) = \frac{1}{\pi^{\dashv}_{\mathcal{B}}(o) - \pi^{\vdash}_{\mathcal{B}}(o) + 1} \cdot \sum_{i \in [\pi^{\vdash}_{\mathcal{B}}(o), \pi^{\dashv}_{\mathcal{B}}(o)]} v(i).$$

*Overall, the* APST *score of $o$, denoted $\Gamma(o)$, is the total of $o$'s score over all non-empty subspaces: $\Gamma(o) = \sum_{\mathcal{B} \subseteq \mathcal{A}, \mathcal{B} \neq \varnothing} \gamma_{\mathcal{B}}(o)$.*

Intuitively, for each subspace $\mathcal{B}$, APST sorts objects by tiers, and each tier occupies a range of ranks. The total score assigned by $v$ to such a range of ranks is distributed equally among objects in the corresponding tier, as we consider them ties in $\mathcal{B}$. The larger the tier, the less "unique" each object, and the smaller the share each object will receive. A number of desirable properties follow directly from the definition.

- Suppose two tiers (in different subspaces) have sizes $\Delta < \Delta'$, but the ranges of ranks they occupy start from the same position. APST scores objects in the smaller tier (of size $\Delta$) higher than those in the other tier, in their respective subspaces.
- Suppose two tiers (in different subspaces) are of the same size, but occupy different ranges of ranks starting from $r$ and $r'$, respectively, where $r < r'$. APST scores objects in the tier occupying the earlier range (starting with $r$) higher than those in the other tier, in their respective subspaces.
- Consider two tiers in the same subspace, where the range of ranks occupied by the first tier precedes that occupied by the second. APST scores objects in the first tier higher than those in the second, in this subspace.

Furthermore, aggregating scores over combinations of attributes make APST naturally adaptive to correlations in data. Consider again the example earlier in this section, where $o_1$ is ranked high for two correlated attributes $\{A_1, B_1\}$ and $o_2$ is ranked equally high in two anti-correlated attributes $\{A_2, B_2\}$. Since it is rare to rank high for anti-correlated attributes, few objects will be in $o_2$'s tier or an earlier tier in $\{A_2, B_2\}$; therefore, APST will score $o_2$ high in $\{A_2, B_2\}$. On the other hand, for $o_1$, since $A_1$ and $B_2$ are correlated, there will likely be more objects dominating $o_1$ in $\{A_1, B_1\}$ than those dominating $o_2$ in $\{A_2, B_2\}$. Thus, all other factors being equal, APST will score $o_2$ higher than $o_1$ overall.

**Adjustable Discounted APST (APST-$\alpha$)** What remains to be discussed is the choice of the positional scoring function $v$. To make our scoring scheme easy to use, we have so far consciously avoided introducing any tuning parameters. However, as motivated in Example 3, we would like some degree of customization for ranking "specialized" objects relative to "well-rounded" ones. Recall that a specialized object is exceptional in very few attributes (such as Stockton in assists), while a well-rounded object is exceptional in none, but reasonably good in many, so as to be exceptional when all those attributes are combined (such as Bird). To capture user's wide ranging preferences for one or the other, we design our positional scoring function with a single tunable parameter to achieve that flexibility while retaining APST's desirable properties.

**Definition 6** (Adjustable Discounted APST (APST-$\alpha$))**.** *Let $\alpha$ be a real number in $(0, 1)$. The* adjustable discounted APST (APST-$\alpha$) score *of an object $o \in \mathcal{O}$ is $o$'s APST score with positional scoring function $v_\alpha(i) = \alpha^{i-1}$.*

Intuitively, the discounting factor, $\alpha$, controls how much more higher ranked objects weigh against lower ranked objects, thereby affecting how specialized and well-rounded objects score relatively overall. In a higher-dimensional subspace $\mathcal{B}$, tiers tend to be larger. Hence, an object that is dominated by few others in $\mathcal{B}$ (but by more objects in subsets of $\mathcal{B}$) tend to score lower in $\mathcal{B}$, compared with an object with the same number of dominating objects in a lower-dimensional subspace $\mathcal{B}'$. With a smaller $\alpha$, the score gap is wider, so overall, it is more difficult for well-rounded objects to surpass specialized ones, whose scores come mostly from contributions by low-dimensional subspaces. In Section 4, we will see how effective $\alpha$ is as a tuning knob on real data—compared with weighted-sum (discussed in Section 1), APST-$\alpha$ has only 1 knob instead of $d$, but is able to produce comparable rankings as highly tuned weighted-sum.

**Comparison with Kemeny Optimal Rank Aggregation** In addition to flexible approaches where users are allowed to specify their preferences, studies in social choice theory have also been investigating an alternative approach where some notion of "optimality" is defined so that the resulting ranks achieve that optimality. A popular representative of this approach is *Kemeny (optimal rank aggregation)*. As discussed in Section 1, given a set of rankings for $\mathcal{O}$, a Kemeny optimal rank aggregation is a ranking that minimizes the total number of pairwise disagreements between this ranking and the input rankings.

We give the formal definition below for completeness.

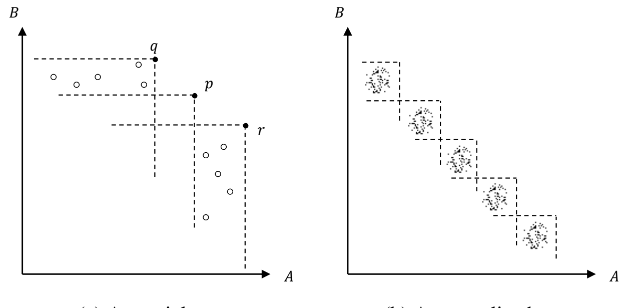
**Definition 7** (Kendall Tau Distance, Kemeny Optimal Aggregation)**.** *A ranking $\pi$ of $\mathcal{O}$ is a linear ordering of objects in $\mathcal{O}$, and $\pi(o) \in [1, |\mathcal{O}|]$ denotes the rank of $o$. The* Kendall tau distance *between two rankings $\pi$ and $\pi'$, denoted $K(\pi, \pi')$, is defined as the number of pairwise disagreements between $\pi$ and $\pi'$; i.e., $K(\pi, \pi') = |\{(o, o') \in \mathcal{O} \times \mathcal{O} \mid \pi(o) < \pi(o') \wedge \pi'(o) > \pi'(o')\}|$.*

*Given a set of $d$ rankings $\pi_1, \ldots, \pi_d$ of $\mathcal{O}$, a Kemeny optimal aggregation is a ranking that minimizes the total Kendall tau distance to each of the $d$ given rankings, i.e., $\arg\min_\pi \sum_{i=1}^{d} K(\pi, \pi_i)$.*

It seems natural to apply Kemeny to the $d$ rankings according to the individual attributes in $\mathcal{A}$. However, there are several issues. First, finding a Kemeny optimal aggregation is NP-hard in $|\mathcal{O}|$ [7]. We would prefer an approach that is computationally more tractable. Indeed, ranking objects using APST-$\alpha$ has complexity polynomial in $|\mathcal{O}|$.

Second, by definition, Kemeny is strongly biased against specialized objects, which rank high in very few attributes but low in many, because ranking them high overall will incur many disagreements. As discussed in Section 1 following Example 3, based on points, rebounds, and assists per game, Kemeny would rank John Stockton low, even though he is a Hall-of-Famer who has the second highest assists per game in NBA history. In contrast, with a small $\alpha$, APST-$\alpha$ would rank it high, because it likely lies on a small skyband for roughly half of the $2^d - 1$ non-empty subspaces—namely those containing the attribute in which the object specializes.

Third, Kemeny sometimes fails to suggest a rank for a worthy object $p$. More precisely, there may be many consensus rankings that are all optimal in the Kemeny sense, where $p$ is ranked differently, sometimes below objects that can be considered obviously inferior to $p$. We illustrate this issue with an example below, and show how APST-$\alpha$ avoids this issue.



(a) A special case (b) A generalized case

Figure 2: Kemeny fails to recognize a worthy object ($p$) but APST-$\alpha$ succeeds.

**Example 4.** *Figure 2(a) depicts a set $\mathcal{O}$ of $2m+1$ objects with two attributes $A$ and $B$. Three most worthy objects, $p$, $q$, and $r$, are on the two-dimensional skyline; $q$ and $r$ each dominate $m$ objects, while $p$ dominates none. Although $p$ is mediocre in either $A$ or $B$, no object dominates $p$ in both $A$ and $B$, and $p$'s balance between $A$ and $B$ cannot be offered by any other object.*

*For Kemeny, note that any object that dominates $p$ in $A$ (or $B$) is dominated by $p$ in $B$ (or $A$, respectively). Since $p$ represents the median in both $A$ and $B$, no matter where $p$ is placed in the consensus ranking, it induces the same number of disagreements ($2m$). Thus, $p$ can be ranked anywhere in a Kemeny optimal aggregation.*

*Under APST-$\alpha$, the scores of $p$, $q$, and $r$ are:*

$$\Gamma(p) = \gamma_{AB}(p) + \gamma_A(p) + \gamma_B(p) = (1+\alpha+\alpha^2)/3 + 2\alpha^m;$$
$$\Gamma(q) = \Gamma(r) = (1+\alpha+\alpha^2)/3 + 1 + \alpha^{m+1}.$$

*To bound the rank of $p$ according to APST-$\alpha$, let $k$ denote the number of objects in $\mathcal{O} \setminus \{p,q,r\}$ with score no less than $\Gamma(p)$. The sum of their scores is at least $k\Gamma(p)$ but obviously at most $\sum_{o\in\mathcal{O}\setminus\{p,q,r\}} \Gamma(o)$. Hence, $k \le \sum_{o\in\mathcal{O}\setminus\{p,q,r\}} \Gamma(o)/\Gamma(p)$. The rank of $p$ is no worse than:*

$$\begin{aligned} k+3 &\le \textstyle\sum_{o\in\mathcal{O}\setminus\{p,q,r\}} \Gamma(o)/\Gamma(p) + 3 \\ &= \left(\textstyle\sum_{o\in\mathcal{O}} \Gamma(o) - (\Gamma(p)+\Gamma(q)+\Gamma(r))\right)/\Gamma(p) + 3 \\ &= \tfrac{3(1-\alpha^{2m+1})/(1-\alpha)-(1+\alpha+\alpha^2+2+2\alpha^m+2\alpha^{m+1})}{(1+\alpha+\alpha^2)/3+2\alpha^m} + 3 \\ &\approx 3 \cdot \frac{1+2\alpha}{1-\alpha^3}. \end{aligned}$$

*The simplification in the last step above is obtained by ignoring $\alpha^{2m+1}$, $\alpha^m$, and $\alpha^{m+1}$, assuming that $m$ is sufficiently large.*

*Hence, with a small enough $\alpha$, APST-$\alpha$ can rank $p$ as high as the third—better than all other objects except $q$ and $r$.*

*This example can be generalized to a case consisting of multiple object clusters where Kemeny fails while APST-$\alpha$ succeeds (Figure 2(b)). Suppose any two objects from different clusters do not dominate each other. For Kemeny, objects from different clusters are indistinguishable, by the same argument for the simple case above. Thus, it is possible for a skyline object in some cluster to rank lower than all objects from other clusters. In contrast, APST-$\alpha$ favors skyline objects in each cluster, especially those close to either the $A$ or $B$ axis.*

Finally, Kemeny does not allow for customization—if it fails to recognize a worthy object there is little that a user can do. On the other hand, the discounting factor $\alpha$ in APST-$\alpha$ provides a single, effective knob that allows users to choose their preference between specialized and well-rounded objects. Indeed, we have seen from above how the choice of $\alpha$ helps overcome the issues faced by Kemeny. More detailed results on real data are shown in Section 4.

## 3.2 Finding Top $\kappa$ Objects

Computing the exact APST-$\alpha$ scores of all objects in subspace entails computing $\delta_{\mathcal{B}}(\mathcal{O}, o)$ for all $o \in \mathcal{O}$ and non-empty $\mathcal{B} \subseteq \mathcal{A}$, which takes $O(2^d|\mathcal{O}|^2)$ time using a naive algorithm. However, for the purpose of identifying objects most worthy of further investigation, we care only about the top-ranked objects. This observation leads to the following problem definition:

**Definition 8** (Finding Top $\kappa$ Objects). *Given a set $\mathcal{O}$ of objects, a discounting factor $\alpha \in (0,1)$, and $\kappa \ge 1$, find the top $\kappa$ objects in $\mathcal{O}$ ranked by APST-$\alpha$ scores.*

We show how to extend the algorithms in Section 2 to efficiently compute approximate answers to the above question. The key observation is that, in each subspace, we need to compute only enough number of tiers in order to not miss any top objects overall, because score contributions from memberships in subsequent tiers are so small that they have little influence over whether an object can be in the top $\kappa$ overall.

Given an error allowance $\epsilon$, we can compute a list of objects, where each object $o$ gets an approximate score $\tilde{\Gamma}(o) \in (\Gamma(o) - \epsilon, \Gamma(o)]$. We divide the error allowance $\epsilon$ evenly among the subspaces to be considered. Within a subspace $\mathcal{B}$ with share $\epsilon_{\mathcal{B}}$ of error allowance, we use Progressive or OnePass to "grow" the skyband up to some top-$\tau$ skyband such that any object outside it will receive a score below $\epsilon_{\mathcal{B}}$ in $\mathcal{B}$. We add the objects in this skyband to the output list (or update their scores if they are already in it). With some care, we can ensure that OnePass retains its advantage over Progressive; i.e., it avoids computing the entire "next" tier, which could include all remaining objects. The idea is that we only need to see enough number of objects in this tier before knowing that all its objects must score below $\epsilon_{\mathcal{B}}$, because APST scores get "diluted" by larger tiers. When we are done with $\mathcal{B}$, we can derive a tighter bound (hopefully much less than $\epsilon_{\mathcal{B}}$) for errors in $\mathcal{B}$, and use it to update the error allowance for remaining subspaces.

The procedure described is formally presented as follows. $N_{\mathcal{A}}$ denotes the number of subspaces that haven't been visited. $\hat{\epsilon}$ denotes the maximum possible error accrued so far.

1. Initialize $N_{\mathcal{A}} = 2^d - 1, \hat{\epsilon} = 0$.
2. For each subspace:
   - 2.1 $\tau \leftarrow \lceil \log_\alpha((\epsilon - \hat{\epsilon})/N_{\mathcal{A}}) \rceil$
   - 2.2 Compute the top-$\tau$ skyband and the next tier if the top-$\tau$ skyband contains less than $\tau$ objects.
   - 2.3 Suppose the next tier starts at $x+1$ and ends at $x+\Delta$, where $x < \tau < x+\Delta$. If $\alpha^x \cdot \frac{1-\alpha^\Delta}{(1-\alpha)\cdot\Delta} > (\epsilon - \hat{\epsilon})/N_{\mathcal{A}}$, output this tier together with the top-$\tau$ skyband and update $\hat{\epsilon} \leftarrow \hat{\epsilon} + \alpha^{x+\Delta}$. Otherwise, ignore the last tier for score aggregation, and update $\hat{\epsilon} \leftarrow \hat{\epsilon} + \alpha^x \cdot \frac{1-\alpha^\Delta}{(1-\alpha)\cdot\Delta}$
   - 2.4 $N_{\mathcal{A}} \leftarrow N_{\mathcal{A}} - 1$.

Now we show that step 2.2 above can be done by extending either Progressive or OnePass .

**Extending Progressive** We change Progressive so that it computes not the largest at most $\tau$, but the smallest skyband that contains at least $\tau$ objects. (removing line 6 and while condition be $|\Phi| \ge \tau$). This change does not affect the running time of Progressive , because progressive compute the next tier anyway.

**Extending OnePass** Suppose in each iteration, the last tier computed so far starts at $x+1$ and ends at $x+\Delta$. We change the condition on line 9 to $\alpha^x \cdot \frac{1-\alpha^{\Delta}}{(1-\alpha)\cdot\Delta} \leq (\epsilon - \hat{\epsilon})/N_{\mathcal{A}}$. The worse case running time of OnePass is then modified to $O(ny)$, where $y$ is the smallest integer such that $\frac{1-\alpha^y}{(1-\alpha)\cdot y} < \epsilon/(2^d - 1)$.

The user does not need to choose $\epsilon$ manually. A reasonable default is $\alpha^{\kappa-1}$, the value of the positional scoring function at rank $\kappa$. With this setting, we can guarantee that any object not in the output list cannot be among the top $\kappa$ overall. The output list may contain more than $\kappa$ objects, and $\epsilon$ can be used to determine which part of this ranking is guaranteed to be accurate. As future work, we are developing an "online" version of the algorithm where $\epsilon$ is incrementally tightened when given additional time or until the top $\kappa$ objects can be accurately separated from the rest.

## 4 Experiments

### 4.1 Datasets

All algorithms were implemented in C++. We conducted all experiments on a machine with Intel Core i7-2600 3.4GHz processor and 7.8GB memory. We used the following real and synthetic datasets.

**Real datasets**

- *NBA players' career total statistics (NBA1)* [2]: This dataset contains the total career performance of ∼4K players who have played in NBA ($n$≈4K), where the performance is captured by 15 numeric attributes ($d$=15) including *number of games played*, *points*, *rebounds*, etc.
- *NBA players' career average statistics (NBA2)*: This dataset captures the career average performance of each player, i.e., the attribute values for each player are derived by dividing the corresponding values in the above NBA1 dataset by the number of games played by the player. Hence it also has ∼4K players ($n$≈4K) and all the numeric attributes in NBA1, except *number of games played* ($d$=14).
- *NBA players' game-by-game statistics (NBA3)*: This dataset contains the performance of each player in each game, with in total ∼400K individual performance records ($n$≈400K) on 14 numeric attributes ($d$=14).
- *National Research Council survey data on 127 Computer Science programs (NRC)* [3]: This dataset contains the assessment of 127 CS programs in the US ($n$=127) on 20 numeric attributes ($d$=20). The data was used by NRC to rank these programs.

**Synthetic datasets**: We used three types of synthetic datasets– correlated (CORR), anti-correlated (ANTICORR), and independent (IND), under varying dimensionality $d$, dataset cardinality $n$, and skyband size $\tau$. CORR and ANTICORR were generated in the following way. We randomly sampled points by multi-variate Gaussian distribution. We then stretched all the points in the direction of $(1, 1, \ldots, 1)$ to produce CORR, and we shrunk them in the direction of $(1, 1, \ldots, 1)$ to yield ANTICORR. This way, all the attributes are pairwise correlated or anti-correlated.

### 4.2 Efficiency of Top-$\tau$ Skyband Algorithms

Given a dataset with $d$ numeric attributes, a particular $\tau$ value, and an algorithm, we used the algorithm to find the top-$\tau$ skyband for each and every subspace. The total elapsed time for the $2^d-1$ nonempty subspaces is used to measure the algorithm's efficiency. We compared the efficiency of Baseline, Progressive (Algorithm 1), and OnePass (Algorithm 2). Baseline was also introduced in Section 2. It iteratively computes $k$-skyband using a simple extension of the BUS algorithm in [13], from $k$=1 to the $k$ value such that the corresponding $k$-skyband contains at least $\tau$ objects.

[2]NBA datasets were from `http://www.databasebasketball.com/`.

[3]The NRC data is from `http://www.nationalacademies.org/nrc/`.

#### 4.2.1 *Efficiency Experiments on Real Datasets*

Figure 3 shows the execution elapsed time of Baseline, Progressive and OnePass, under varying $\tau$=10, 20, . . . , 100, on both the 4K-tuple NBA1 dataset and the 400K-tuple NBA3 dataset. On both the small and large datasets, our algorithms significantly outperformed the baseline approach.

On the small NBA1 dataset (Figure 3(a)), OnePass was less efficient for small $\tau$ and started to outperform Progressive as $\tau$ increased. This is because the dimensionalities of subspaces in which top-$\tau$ skyband is empty but cannot be pruned (by techniques in Section 2.3) increase as $\tau$ increases. Computing the skylines for these subspaces gets more expensive as SUBSKY becomes less efficient.

On the large NBA3 dataset (Figure 3(b)), we also observe the running time of Progressive grew faster than OnePass as $\tau$ increased, due to the same reason for Figure 3(a). Besides, attributes in NBA3 have smaller correlations than NBA1 attributes, which induces larger skylines for the same subspace and makes Progressive even less efficient. This is why OnePass already outperforms Progressive when $\tau$ is small.

#### 4.2.2 *Efficiency Experiments on Synthetic Datasets*

Our experiments on the synthetic datasets CORR/ANTICORR/IND verified the scalability of Progressive and OnePass.

**Varying $d$, fixed $n$=100,000 and $\tau$=100:** Figure 4 shows that both Progressive and OnePass were faster than Baseline by orders of magnitude (vertical axis in logarithmic scale). Progressive run faster than OnePass on (a) correlated data. Their performances were comparable on (b) anti-correlated and (c) independent data. We observe that the elapsed time on CORR is much longer than that on ANTICORR and IND for all three algorithms. This is because the top-$\tau$ skyband in CORR may be non-empty even in high dimensional subspaces, while for ANTICORR and IND, top-$\tau$ skyband for a small $\tau$ is empty in most subspaces. Such cases can be quickly detected and pruned as discussed in Section 2.3. When the dimensionality is high, due to the way ANTICORR was generated for ensuring pairwise anti-correlation among all attributes, ANTICORR becomes similar to IND. So we omit the results on ANTICORR in the following discussion.

**Varying $n$, fixed $d$=15 and $\tau$=100:** From Figure 5, we observe again that Progressive and OnePass outperformed Baseline by orders of magnitude, and Progressive slightly outperformed OnePass. Their elapsed time increased roughly linearly by $n$.

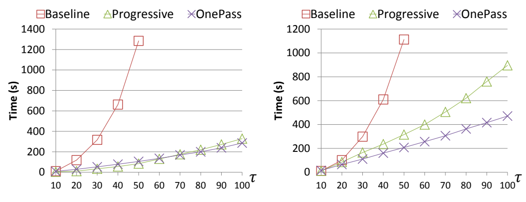
**Varying $\tau$, fixed $n$=100,000 and $d$=15:** Figure 6 shows that Progressive and OnePass significantly outperformed Baseline on both CORR and IND. Their elapsed time grew nearly exponentially by $\tau$ for all three algorithms (vertical axis in logarithmic scale). Progressive was slightly faster than OnePass on CORR. On IND dataset, OnePass was slower than Progressive under small $\tau$ but became faster than the latter as $\tau$ increased.

### 4.3 Quality of Ranking by APST-$\alpha$

We evaluated the quality of the rankings produced by APST-$\alpha$ and Kemeny on the NBA and NRC datasets, and weighted-sum ranking on NBA.

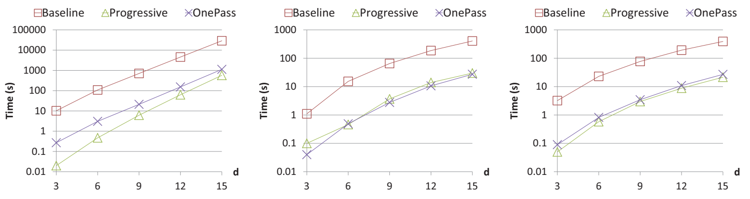
#### 4.3.1 *APST-$\alpha$ vs. Weighted-Sum on NBA Dataset*

In this experiment, we used APST-$\alpha$ and weighted-sum to rank NBA players by their career average performance (NBA2). We varied $\alpha$ to produce different APST-$\alpha$ rankings. For each ranking results, we measure its quality by the number of Hall of Fame inductees (HoFers for short) in the top-$k$ players according the the ranking, under various $k$ values.



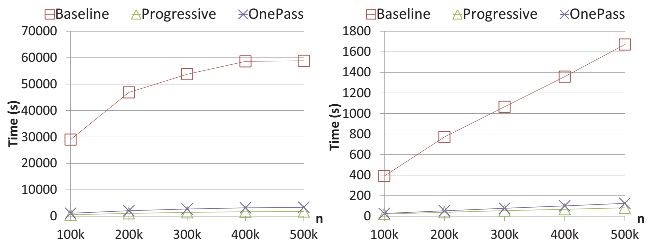
(a) NBA1 (b) NBA3

Figure 3: Algorithm elapsed time on NBA1, NBA3 (varying $\tau$)



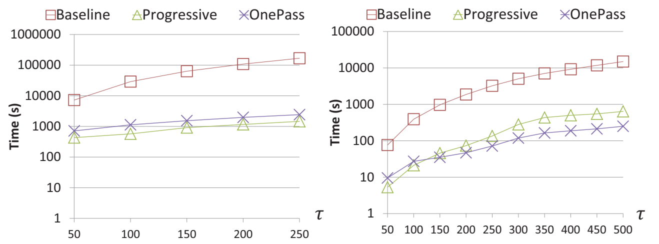
(a) CORR (b) ANTICORR (c) IND

Figure 4: Elapsed time of algorithms on synthetic data (varying $d$)



(a) CORR (b) IND

Figure 5: Elapsed time of algorithms on synthetic data (varying $n$)



(a) CORR (b) IND

Figure 6: Elapsed time of algorithms on synthetic data (varying $\tau$)

In NBA2, 7 out of the 14 attributes are used for ranking because statistics such as *steals, blocks*, and *minutes* were not recorded for games played before 1971 in the dataset. Thus, including these attributes would underrate the HoFers from the early days. Moreover, we did not use career total performance data (NBA1) in this experiment because that will also underrate earlier HoFers, as they did not play as many games as recent players.

For weighted-sum, the weights on the attributes were determined as follows. We first found a linear classifier to separate HoFers from non-HoFers that minimizes the number of inaccurately classified players. We then used the (normalized) vector perpendicular to the linear classifier as the weight vector in weighted-sum ranking.

The result of this experiment is shown in Figure 7. It compares eighted-sum, APST-.99, APST-.5, and APST-.01, by the numbers of HoFers in the top-$k$ of their rankings, respectively. We can see that APST-$\alpha$ contains visibly more HoFers in top-$k$ under most considered combinations of $\alpha$ and $k$. Note that the performance of all ranking methods were negatively affected by the existence of active players who are future HoFers but not eligible for induction until 5 years after retirement. For the sake of accuracy, these players were not considered in obtaining the weight vector but were included in ranking.

### 4.3.2 *APST-$\alpha$ vs. Kemeny on NBA Dataset*

We also used Kemeny for the same experiment described in Section 4.3.1 and the result is also shown in Figure 7. We can see that in most cases APST-$\alpha$ rankings contain visibly more HoFers in top-$k$ than Kemeny ranking. However, we should note that APST-$\alpha$ and weight-sum rankings were produced for all 4K players in NBA2, while we ranked only 200 NBA players for Kemeny. There are 91 HoFers in the 4K players in NBA2 and 59 of them are in the 200 players for Kemeny. The 200 players is the set union of 7 lists on the 7 attributes participated in ranking, where each list contains the top-40 players on the corresponding attribute.

We used only 200 players due to practical reasons as follows. To compute Kemeny optimal rank aggregation, we modeled it as an Integer Programming (IP) problem and used IBM CPLEX Optimizer (`http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`) to solve it. CPLEX failed to find solutions when there are more players due to precision in computation. Moreover, finding Kemeny optimal is known to be NP-hard under 4 or more attributes [7]. The corresponding IP problem is also memory-intensive as it requires O($n^3$) variables for $n$ objects (i.e., NBA players). Hence CPLEX caused memory overflow with close to 300 players.

For fairness in comparison, we also used APST-$\alpha$ to rank the 200 objects only, in which it showed comparable performance to Kemeny. Due to space limitations, curves for this result are omitted from Figure 7.

### 4.3.3 *APST-$\alpha$ vs. Kemeny on NRC Dataset*

The National Research Council ranked 127 CS programs by a linear combination of 20 numeric attributes in the NRC dataset. The accuracy of NRC's data and appropriateness of their ranking method still remains a controversial topic. Nevertheless, we ranked the same 127 programs by APST-$\alpha$ and Kemeny using the same 20 attributes. We assessed the quality of their rankings by the sizes of their intersections with NRC's reference rankings in top-$k$, under various $k$ values. We observed comparable performance from Kemeny and APST-$\alpha$. Due to space limitations, we omit the figure from presentation.

## 4.4 Effect of $\alpha$ in APST-$\alpha$

To better understand the parameter $\alpha$ in APST-$\alpha$ and investigate how it is related to promoting specialized or all-round objects, we injected an artificial player into the NBA1 and NBA2 datasets. Again, we used 7 (instead of all) attributes, as mentioned in Section 4.3.2, to avoid underrating earlier players. The artificial player has the highest value on attribute *points* but the lowest values in all other attributes. Figure 8 shows how this player's rank changed in APST-$\alpha$ ranking by varying $\alpha$. As $\alpha$ decreased from 1, the artificial player quickly got ranked high among real players. Its rank converged as $\alpha$ approached 0, but never became the first because it cannot surpass the players that have the highest values on some other attributes. Similar observation was made when the artificial player is specialized in other attributes. This result agrees with the discussion in Section 3 that a smaller $\alpha$ favors those objects that specialize in a few dimensions over well-rounded players. An example of specialized player is John Stockton, a HoFer, who is second only to Magic Johnson in assists per game, but is not ranked

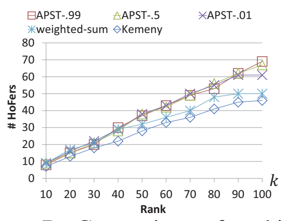


Figure 7: Comparison of rankings by number of HoFers in top-$k$

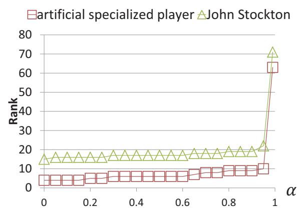


Figure 8: Rank of artificial player with different $\alpha$

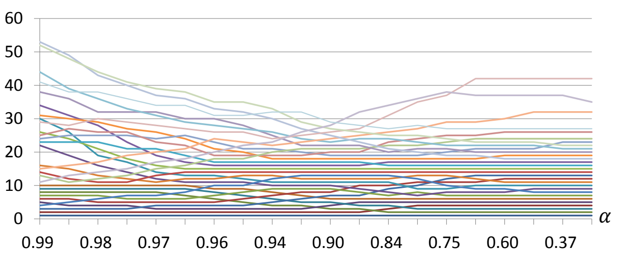


Figure 9: How the APST-$\alpha$ ranks of NBA players change by $\alpha$

high in any other considered attributes. His ranking positions under various $\alpha$ values are also plotted in Figure 8, and his curve shows similar trend to that for the artificial player.

In Figure 9, we focus on the top-30 players by APST-.9 on NBA2, under the three most frequently used attributes in assessing player performance– *points, rebounds* and *assists*. The figure shows how their ranking positions change by $\alpha$ (horizontal axis in logarithmic scale). At the bottom of the figure, the top-9 players always remain in top-13, due to their exceptional performance in multiple attributes. The ranks of the remaining players change drastically by $\alpha$. These players can be mainly categorized into two types: I) specialized players, who are exceptional on very few attributes; II) well-rounded players, who are not exceptional on any single attribute, but are reasonably good on multiple attributes and hence are exceptional with regard to those attributes combined. In Figure 9, the curves of Type-I players go down as $\alpha$ decreases, because small $\alpha$ favors them more against other players in at least half of the subspaces that contain the attribute on which they specialize. On the contrary, the curves of Type-II players go up as $\alpha$ decreases. They get rewarded in higher dimensional subspaces for not being dominated by many other players in such subspaces. However, they can be easily dominated by others on individual dimensions.

## 5 Related Work

Our work is closely related to skyline computation for multidimensional data, which can roughly be grouped into two classes. The first class [3, 4, 8] include Block Nested Loop (BNL), Divide and Conquer (D&C) and their variants, and do not require precomputed indices. The second class of algorithms [14, 9, 12] adopt $B^+$-tree and R-tree indices to prune away unnecessary computation. None of those considers all subspaces or $k$-skybands. Skyline computation for multiple (or all) subspaces are first studied by Pei et al [13] (BUS and TDS algorithms) and by Tao et al [15] (SUBSKY algorithm). The former are extensions of BNL and D&C algorithms that traverse the subspace lattice without using indexing structured, while the latter follows the index-based approaches. Extending skyline to $k$-skyband was first studied in [12].

While our notion of top-$\tau$ skyband is the first to provide a universal parameter suitable for selecting the right set of objects regardless of the subspaces being considered, there are previous works on various advanced selection criteria when the size of the skyline is too large. For example, BBS [12] and PBT [18] focus on selecting points that dominate the most other points, and Lin *et al* [10] focus on selecting a fixed sized set of points that collectively dominate the most other points. More recently, Lu *et al* [11] propose the notion of layered skylines, where each layer is defined as the skyline after objects in previous layers are removed. Neither of those works consider the need for compatibility across all subspaces as we do.

Linear ranking of objects for the purpose of answering top-$k$ query have been studied extensively, including those using skyline and skybands [16, 17]. Those studies focus on performance improvements and do not consider providing better control of the semantics of the ranking functions. Kemeny ranking [7], on the other hand, was proposed in social choice theory to achieve good semantics. It aims to minimize the disagreements between the aggregated ranking and the input rankings, a problem turns out to be NP-hard (if more than 3 rankings are to be aggregated). While useful, Kemeny does not provide a flexible knob, such as our $\alpha$ parameter, for adjusting the ranking based on the users' preference towards different types of prominent objects.

## 6 Conclusion

The tasks of finding one-of-the-few claims from data and ranking objects by such claims are important to the nascent field of computational journalism. We have introduced a uniqueness-based measure of interestingness, which leads to a simple and intuitive problem formulation for finding all interesting claims, using a single uniqueness threshold $\tau$ that automatically adapts to data characteristics and applies to all subspaces. We have proposed a novel scheme for ranking objects, overcoming the inflexibility of Kemeny without resorting to a large number of knobs like weighted-sum. We have devised efficient algorithms for both tasks, using techniques such as pruning and approximation to tame complexity. We have seen promising results on real data, and we believe that our attention to usability will appeal to journalists and citizens alike.

## References

[1] J. Bentley, H. Kung, M. Schkolnick, and C. Thompson. On the average number of maxima in a set of vectors and applications. *JACM*, 25(4):536–543, 1978.

[2] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cube. *ACM SIGMOD Record*, 28(2):359–370, 1999.

[3] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, 2001.

[4] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, 2003.

[5] S. Cohen, J. T. Hamilton, and F. Turner. Computational journalism. *Communications of the ACM*, 54(10):66–71, 2011.

[6] S. Cohen, C. Li, J. Yang, and C. Yu. Computational journalism: A call to arms to database researchers. In *CIDR*, 2011.

[7] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, 2001.

[8] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *VLDB*, 2005.

[9] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*, 2002.

[10] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, 2007.

[11] H. Lu, C. Jensen, and Z. Zhang. Skyline ordering: A flexible framework for efficient resolution of size constraints on skyline queries. Technical report, Aalborg University, 2010.

[12] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *TODS*, 30(1):41–82, 2005.

[13] J. Pei, Y. Yuan, X. Lin, W. Jin, M. Ester, Q. Liu, W. Wang, Y. Tao, J. Yu, and Q. Zhang. Towards multidimensional subspace skyline analysis. *TODS*, 31(4):1335–1381, 2006.

[14] K. Tan, P. Eng, B. Ooi, et al. Efficient progressive skyline computation. In *VLDB*, 2001.

[15] Y. Tao, X. Xiao, and J. Pei. Subsky: Efficient computation of skylines in subspaces. In *ICDE*, 2006.

[16] P. Tsaparas, T. Palpanas, Y. Kotidis, N. Koudas, and D. Srivastava. Ranked join indices. In *ICDE*, 2003.

[17] A. Vlachou, C. Doulkeridis, K. Nørvåg, and M. Vazirgiannis. Skyline-based peer-to-peer top-k query processing. In *ICDE*, 2008.

[18] M. Yiu and N. Mamoulis. Efficient processing of top-k dominating queries on multi-dimensional data. In *VLDB*, 2007.

[19] H. Young. Social choice scoring functions. *SIAM Journal on Applied Mathematics*, pages 824–838, 1975.