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.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*data mining*; H.2.4 [**Database Management**]: Systems—*query processing*; K.4.0 [**Computers and Society**]: General

Keywords

Computational journalism, fact checking, skyband, 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, we consider *one-of-the-few* claims of the following form:

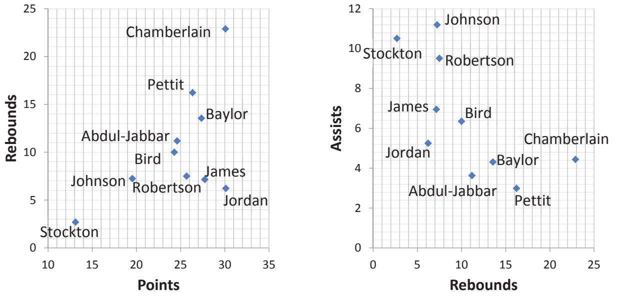
> ***Object o is dominated by fewer than k objects in a non-empty subset $\mathcal{B} \subseteq \mathcal{A}$ of attributes.***

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 Our goal is to find interesting one-of-the-few claims in all “subspaces” (i.e., subsets of attributes). One-of-the-few claims are closely related to the concept of k-*skyband* for multi-dimensional data [15]. 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* $\{\text{points}\}$*—is quite an impressive*

KDD’12, August 12–16, 2012, Beijing, China.



(a) Positive: *points* vs. *rebounds* (b) Negative: *rebounds* vs. *assists*

Figure 1: Correlation in NBA player stats.

feat. However, if we expand the subspace to $\{\text{points}, \text{rebounds}\}$, 142 *players now fit the bill—i.e., each is dominated by fewer than* 50 *others in* $\{\text{points}, \text{rebounds}\}$*. 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 $O(\ln^{|\mathcal{B}|-1}|\mathcal{O}|/(|\mathcal{B}|-1)!)$, if object ranks by individual attributes are uniformly and independently distributed [1], e.g., when objects are uniformly distributed in a box. With a fixed k, too many claims may be in high-dimensional subspaces, making them uninteresting. Clearly, k must be adjusted as $|\mathcal{B}|$ changes. Furthermore, the appropriate setting for k is not simply as a function of dimensionality, because data characteristics—e.g., 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* $\{\text{points}, \text{rebounds}\}$ *(Figure 1a) and* $\{\text{rebounds}, \text{assists}\}$ *(Figure 1b). It is 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* $\{\text{points}, \text{rebounds}\}$ *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* $\{\text{rebounds}, \text{assists}\}$ *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 are often non-technical and the results need to be explained to the general public in stories. Furthermore, given the interestingness measure, we need to address the challenge of finding interesting one-of-the-few claims in all subspaces efficiently.

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 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, or may simply be a matter of personal opinion.*

A popular method for ranking objects based on multiple input rankings 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). When aggregating object rankings under individual attributes, Kemeny tends to downgrade objects that rank extremely high in 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, *weighted-sum ranking*, exposes too many knobs. With weighted-sum, a user specifies a preference vector, whose components represent weights assigned to attributes; objects are 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 specify d weights; even if we learn these weights automatically, training examples must be provided. 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 allows 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 τ objects. Unlike k, τ 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, τ is 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 τ. 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 τ 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 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 α, 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 α. 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 (more precisely, spanned by) a subset of the attributes. The set of all subspaces of $\mathcal{A}$ forms a lattice under containment relation. 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 $\mathcal{B}_1 \subset \mathcal{B}_2$ (resp. $\mathcal{B}_1 \supset \mathcal{B}_2$) 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 $\mathcal{O}$ where each is dominated by fewer than k 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 $\mathcal{O}$ where each is dominated by exactly $i - 1$ 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 subspace $\mathcal{B} = \{\textit{rebounds}, \textit{assists}\}$ shown in Figure 1b. $\mathfrak{D}_{\mathcal{B}}(\mathcal{O}, \text{Stockton}) = \{\text{Johnson}\}$, so $\delta_{\mathcal{B}}(\mathcal{O}, \text{Stockton}) = 1$. $\mathfrak{S}^1_{\mathcal{B}}(\mathcal{O}) = \{\text{Johnson}, \text{Robertson}, \text{Bird}, \text{Chamberlain}\}$ is the skyline (or 1-skyband), which is also the 1-st tier. $\mathfrak{S}^2_{\mathcal{B}}(\mathcal{O}) = \mathfrak{S}^1_{\mathcal{B}}(\mathcal{O}) \cup \{\text{Stockton}, \text{Baylor}, \text{Pettit}\}$ 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 $\{\text{James}, \text{Abdul-Jabbar}\}$, 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 τ that caps the number of similar claims. Therefore, we introduce the concept of *top-τ skyband* below. While the concept is closely related to k-skyband, a crucial difference is that a top-τ skyband is defined by its size, while a k-skyband, defined by its number of tiers, can be arbitrarily large.

Definition 2 (Top-τ Skyband)**.** *Given $\tau \geq 1$, the* top-τ skyband *of a set of objects $\mathcal{O}$ in subspace $\mathcal{B}$, denoted $\Phi^{\tau}_{\mathcal{B}}(\mathcal{O})$ (or Φ^{τ} if context is clear), is the $\hat{k}$-skyband where $\hat{k} = \max\{k \mid \tau \geq |\mathfrak{S}^k_{\mathcal{B}}(\mathcal{O})|\}$.*

In other words, the top-τ skyband is the largest skyband whose size does not exceed τ.

The fact that an object o belongs to the top-τ skyband with the k-th tier as its last non-empty tier—or alternatively, the k-skyband with size no more than τ—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 τ objects.*

Intuitively, τ measures the uniqueness of the claim made by the first part of the above statement. For example, in Figure 1a, suppose we set $\tau = 3$. $\Phi^{\tau} = \{\text{Chamberlain}, \text{Jordan}\}$ is the 1-skyband. The 2-skyband would be too big, because it also contains Pettit, Baylor, and James, and has size $5 > \tau$. 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-τ Skybands in All Subspaces)**.** *Given the set $\mathcal{O}$ of objects and a user-specified threshold τ, find $\Phi^{\tau}_{\mathcal{B}}(\mathcal{O})$ for every non-empty subspace $\mathcal{B} \subseteq \mathcal{A}$.*

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

Overview of Solutions The rest of this section discusses how to find all interesting one-of-the-few claims. At a high level, we 1) traverse the lattice of subspaces in some manner, and 2) compute Φ^{τ} for each subspace we visit. Techniques for improving efficiency exist both across subspaces and within each subspace. For finding Φ^{τ} 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 Φ^{τ} for a single-attribute subspace $\{A\}$ simply involves finding the top τ 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. [17], 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 τ 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-τ Skyband Algorithm

As it turns out, all objects of $\mathcal{O}$ 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 (see [23] for proof):

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

[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 that are not in previous layers.

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

```
  Data: object set $\mathcal{O}$, subspace $\mathcal{B}$, and size threshold $\tau$
  Result: $\Phi^{\tau}_{\mathcal{B}}(\mathcal{O})$
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 $\Phi$;
```

Lemma 1 suggests the following strategy, which we call *Progressive* (Algorithm 1), for computing the top-τ skyband for a given subspace. Given τ, Progressive computes the answer Φ 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 Φ). 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 Φ as long as their set union has no more than τ 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 Φ to exceed τ.

Note that in Lines 4 and 5 we compute the size of the dominating subset for $o \in S$ in Φ 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 $\bar{S}$, and $\Delta\bar{\Phi} \subseteq \bar{S}$ was added to the answer set. For the current iteration, it is easy to see that $\bar{S} \setminus \Delta\bar{\Phi} \subseteq S$; i.e., all remaining objects in $\bar{S}$ 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. [19] (in general any skyline algorithm can be used). The original SUBSKY uses an index whose size is linear in $|\mathcal{O}|$ 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 Φ, and we "seed" each invocation of SUBSKY with $\bar{S} \setminus \Delta\bar{\Phi}$ 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 $o \in \bar{S} \setminus \Delta\bar{\Phi} \subseteq S$, 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 [19], 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. [19] describes various techniques to reduce it for SUBSKY. However, in the worst case, $|\Phi|$ is almost τ 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-τ 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 larger than τ. In this section, we show how to tame the complexity with another algorithm *OnePass*. This algorithm works by considering objects one by one in a particular order. Each object is either added to the answer set Φ or dropped. Once Φ has more than τ objects, the last tier is peeled off. The processing order is chosen in a "safe" way, as defined below, which allows OnePass to cap $|\Phi|$ at τ at all times, thereby bounding the number of dominance tests to $\tau|\mathcal{O}|$, in contrast to $O(|\mathcal{O}|^2)$ by Progressive.

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

```
   Data: object set $\mathcal{O}$, subspace $\mathcal{B}$, and size threshold $\tau$
   Result: $\Phi^{\tau}_{\mathcal{B}}(\mathcal{O})$
 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 $\Phi$;
```

Definition 4 (Safe Order [2]). *An 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 (see [23] for proof).

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 $|\mathcal{O}| > \tau$).

Consider the next object o in the safe order. OnePass first checks whether o is dominated by at least k objects in Φ (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 $\Phi^{\tau}_{\mathcal{B}}(\mathcal{O})$ (by (I-3)). We stop counting as soon as $c[o]$ reaches k. Heuristically, we check o against "better" objects in Φ (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 Φ (Lines 8) and remember the count $c[o]$. If doing so makes $|\Phi|$ exceed τ we remove the last tier from Φ 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 τ (by (I-3)), so we can terminate (and return $\varnothing$) without processing the remaining objects. After all objects in $\mathcal{O}$ are processed, Φ is the top-τ 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 $\mathcal{B}$ (lexicographically, with an arbitrary ordering among the attributes), which takes $O(|\mathcal{O}| \log |\mathcal{O}|)$ 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, 17]), and

[2] Note that a safe order is equivalent to a topological order.

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 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 τ and k are typically small.

Complexity Because OnePass caps $|\Phi|$ at τ 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 τ, and is able to terminate without examining the whole $\mathcal{O}$. Thus, OnePass is expected to run 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 Φ^τ 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; details can be found in [23].

For Progressive, computation in every subspace starts with finding the skyline. For this part, we directly apply the techniques of Pei et al. [17], 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 Φ^τ 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. [17] to help compute the skyline in a subspace using those found in its parent subspaces. Moreover, we use the Φ^τ 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 τ, Φ^τ must be empty for all descendant subspaces. 2) Given $\mathcal{B}$, if the union of skyline objects from parent subspaces already contains more than τ objects, $\Phi^\tau_\mathcal{B}(\mathcal{O})$ must be empty.

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)* [25], 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).

All-Subspace Positional Scoring with Ties (APST) To resolve the issue above, we propose *All-Subspace 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 (All-Subspace Positional Scoring with Ties (APST))**.** *For each non-empty subspace $\mathcal{B} \subseteq \mathcal{A}$, sort $\mathcal{O}$ in non-descending 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. For example, APST favors tiers that are smaller in size or occupy earlier ranges of ranks; see [23] for a more formal discussion.

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-α) 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 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-α))**.** *Let α be a real number in $(0, 1)$. The* adjustable discounted APST (APST-α) 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, α, 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 α, 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 α is as a tuning knob on real data—compared with weighted-sum (discussed in Section 1), APST-α has only 1 knob instead of d, but produces 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 investigated an alternative where some notion of "optimality" is defined for the resulting ranks. 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.

One natural way to use Kemeny is to apply it to the d rankings according to the individual attributes in $\mathcal{A}$. This approach, which we call *Kemeny-d*, has 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-α has complexity polynomial in $|\mathcal{O}|$.

Second, by definition, Kemeny-d is strongly biased against specialized objects, which rank high in 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-d 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 α, APST-α 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-d 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. For an illustrative example, as well as how APST-α avoids this issue, see [23].

Fourth, Kemeny-d 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 α in APST-α 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 α helps overcome the issues faced by Kemeny. More detailed results on real data are shown in Section 4.

Finally, instead of Kemeny-d, it is conceivable to use Kemeny to aggregate all $2^d - 1$ rankings according to all non-empty subsets of attributes in $\mathcal{A}$.[3] We are not aware of any previous work exploring this approach. While this approach could potentially alleviate the second and third issues above, it still suffers from the first (high computational complexity) and last (lack of any customization). This approach would require the same amount of effort as APST-α just to prepare the $2^d - 1$ input rankings in all subspaces; then, its rank aggregation is more expensive and not tunable.

3.2 Finding Top κ Objects

Computing the exact APST-α 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:

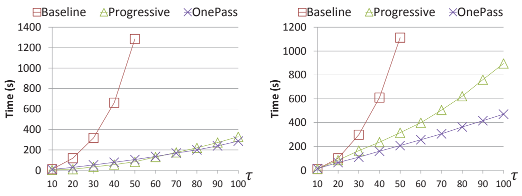
Definition 7 (Finding Top κ Objects)**.** *Given a set $\mathcal{O}$ of objects, a discounting factor $\alpha \in (0, 1)$, and $\kappa \geq 1$, find the top κ objects in $\mathcal{O}$ ranked by APST-α scores.*

We show how to extend the algorithms in Section 2 to efficiently compute approximate answers to the above question. We provide only high-level insights here; see [23] for a full discussion with more technical details. 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 κ overall.

Given an error tolerance $\epsilon > 0$, we can compute a list of objects, where each object o gets an approximate score $\tilde{\Gamma}(o) \in (\Gamma(o) - \epsilon, \Gamma(o)]$. Roughly speaking, we divide the error tolerance ϵ among the subspaces to be considered. Within a subspace $\mathcal{B}$ with share $\epsilon_{\mathcal{B}}$ of error tolerance, we use Progressive or OnePass to "grow" the skyband up to some top-τ 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 tolerance for remaining subspaces.

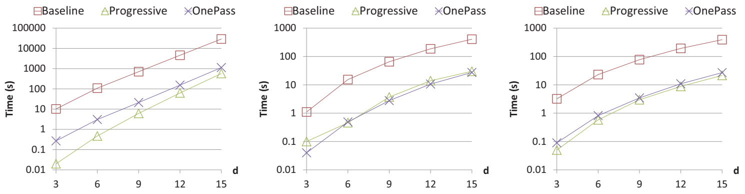
The user does not need to choose ϵ manually. A reasonable default is $\alpha^{\kappa-1}$, the value of the positional scoring function at rank κ. With this setting, we can guarantee that any object not in the output list cannot be among the top κ overall. The output list may contain more than κ objects, and ϵ 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 ϵ is incrementally tightened when given additional time or until the top κ objects can be accurately separated from the rest.

[3] More precisely, such rankings are partial orders induced by skyband tiers in their respective subspaces; Kemeny readily generalizes to partial orders.



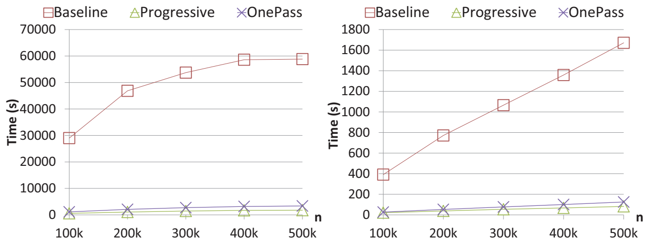
(a) NBA1 (b) NBA3

Figure 2: Running time on NBA1, NBA3 (varying τ).



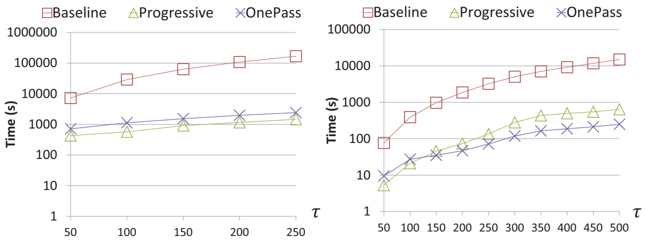
(a) CORR (b) ANTICORR (c) IND

Figure 3: Running time on synthetic data (varying d).



(a) CORR (b) IND

Figure 4: Running time on synthetic data (varying n).



(a) CORR (b) IND

Figure 5: Running time on synthetic data (varying τ).

4 Experiments

All algorithms were implemented in C++ and tested on a machine with Intel Core i7-2600 3.4GHz processor and 7.8GB memory.

We use three datasets on NBA players.[4] **NBA1** contains the *career total* statistics for $n \approx 4$K players. There are $d = 15$ performance attributes, including the total *number of games played*, *points*, *rebounds*, etc., over the players' whole careers. **NBA2** contains the *career average* statistics for the same set of $n \approx 4$K players. It has the same set of attributes as NBA1 except *number of games played* (hence $d = 14$), and the attribute values for a player are derived by dividing the corresponding values in NBA1 by the number of games played by the player. **NBA3** contains the game-by-game statistics for each player. There are a total of $n \approx 400$K records with $d = 14$ performance attributes.

We also use synthetic datasets to test algorithm performance—correlated (**CORR**), anti-correlated (**ANTICORR**), and independent (**IND**), with varying size and dimensionality. CORR and ANTICORR are generated by first sampling data points randomly from a multivariate Gaussian distribution; then, we stretch all points in the direction of $(1, 1, \ldots, 1)$ to produce CORR, and we shrink them in the direction of $(1, 1, \ldots, 1)$ to yield ANTICORR. This way, the attributes are pairwise correlated or anti-correlated.

Additional results are in [23], including those on the National Research Council survey of 127 computer science programs.

4.1 Efficiency of Top-τ Skyband Algorithms

Given a dataset, a particular τ value, and an algorithm, we use the algorithm to find the top-τ skyband for each and every subspace. The total elapsed time for the $2^d - 1$ nonempty subspaces measures the algorithm's efficiency. We compare Baseline (Section 2), Progressive (Algorithm 1), and OnePass (Algorithm 2).

4.1.1 *Efficiency on Real Datasets*

Figure 2 shows the execution time of Baseline, Progressive, and OnePass, under varying $\tau = 10, 20, \ldots, 100$, on both NBA1 ($n = 4$K) and NBA3 ($n = 400$K). On both the small and large datasets, our algorithms significantly outperform the baseline approach.

On the small NBA1 dataset (Figure 2(a)), OnePass is less efficient for small τ and starts to outperform Progressive as τ increases. The reason is that the dimensionalities of subspaces in which top-τ skyband is empty but cannot be pruned (by techniques in Section 2.3) increase with τ. Computing the skylines for these subspaces gets more expensive as SUBSKY becomes less efficient. Results on NBA2 are similar to NBA1 and hence omitted.

On the large NBA3 dataset (Figure 2(b)), we also see that the running time of Progressive grows faster than OnePass as τ increases, for the same reason above. Moreover, attributes in NBA3 have smaller correlations than in NBA1, which induce larger skylines, making Progressive even less efficient. Hence, OnePass starts to outperform Progressive when τ is small.

On both datasets, we observe the running time of OnePass to be roughly linear in τ, confirming the analysis in Section 2.2.

4.1.2 *Efficiency on Synthetic Datasets*

Varying d, Fixed $n = 100$K and $\tau = 100$: Figure 3 shows that both Progressive and OnePass are faster than Baseline by orders of magnitude (vertical axis has logarithmic scale). Progressive runs faster than OnePass on correlated data (a). Their performances are comparable on anti-correlated (b) and independent data (c). We observe that the running times on CORR are much longer than those on ANTICORR and IND for all algorithms. The reason is that top-τ skybands in CORR may be non-empty even in high dimensional subspaces, while for ANTICORR and IND, they tend to be empty and can be quickly detected and pruned as discussed in Section 2.3. When the dimensionality is high, due to the way ANTICORR is generated for ensuring pairwise anti-correlation among all attributes, ANTICORR becomes similar to IND; hence, we omit the results on ANTICORR in the following discussion.

Varying n, Fixed $d = 15$ and $\tau = 100$: From Figure 4, we observe again that Progressive and OnePass outperform Baseline by orders of magnitude, and Progressive slightly outperforms OnePass. Their running times increase roughly linearly by n.

Varying τ, Fixed $n = 100$K and $d = 15$: Figure 5 shows that Progressive and OnePass significantly outperform Baseline on both CORR and IND. For all three algorithms, their running times grow nearly exponentially in τ (vertical axis has logarithmic scale). Progressive is slightly faster than OnePass on CORR. On IND, OnePass is slower than Progressive under small τ but becomes faster as τ increases.

[4] `http://www.databasebasketball.com/`

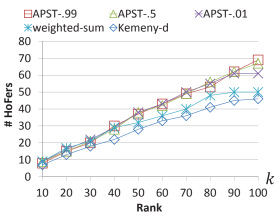


Figure 6: Comparison of rankings by number of HoFers in top-k.

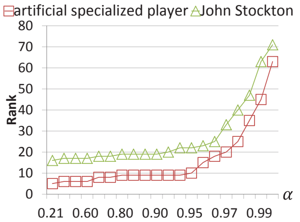


Figure 7: Ranks of an artificial player vs. Stockton as α varies.

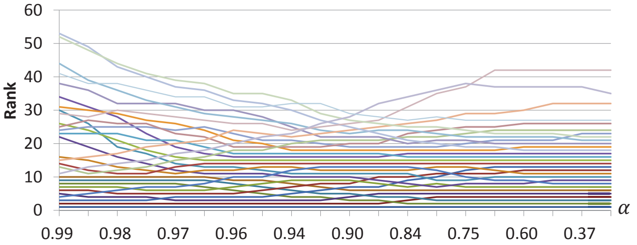


Figure 8: Effect of α on the APST-α ranking of NBA players.

Overall, these experiments further confirm the significant performance advantage of Progressive and OnePass over Baseline.

4.2 Quality of Ranking by APST-α

We evaluate the quality of the rankings by APST-α, Kemeny-d, and weighted-sum for the NBA2 dataset (career average statistics). For each ranking, we measure its quality by the number of Hall of Fame inductees (HoFers for short) found among the top-k players according the the ranking, under various k values.

We use 7 out of the 14 attributes in NBA2 for ranking because the dataset did not record statistics such as *steals*, *blocks*, and *minutes* for games played before 1971. Including these attributes would unfairly underrate the players from the early days. We do not use NBA1 (career total performance) because it would also underrate earlier players, as they did not play as many games as recent ones.

APST-α vs. Weighted-Sum We vary α in APST-α to produce different rankings. For weighted-sum, the weights on the attributes are determined as follows. We first find a linear classifier to separate HoFers from non-HoFers that minimizes the number of inaccurately classified players. We then use the unit vector perpendicular to the linear classifier as the weight vector in weighted-sum.

Figure 6 compares weighted-sum, APST-.99, APST-.5, and APST-.01, by the numbers of HoFers found among the top k players in their rankings, respectively (ignore the curve for Kemeny-d for now). We can see that APST-α contains visibly more HoFers in top-k under most combinations of α and k considered. Note that the performance of all ranking methods are negatively affected by active players who would be good candidate HoFers in the future, but will not be eligible until 5 years after retirement.[5] For example, at $k = 80$, the top k players identified by weighted-sum include 48 HoFers. Among the remaining 32, 23 are not yet eligible; 9 are eligible but not HoFers.

APST-α vs. Kemeny-d For Kemeny-d, solving for the optimal ranking proved computationally challenging on this real dataset, reinforcing our discussion in Section 3.1. We model the optimization as an integer program (IP) and solve it with CPLEX.[6] Recall that there are about 4K players. On our reasonably powerful machine, CPLEX caused memory overflow with close to 300 players, and it failed to find solutions with more than 200 players because of precision issues. Thus, we improvise by first identifying 200 players, who are the set union of the 7 top-40 lists for the 7 attributes used for ranking. Then, we run Kemeny-d on these 200 players only.

In Figure 6, we can see that in most cases APST-α rankings contain noticeably more HoFers in top-k than Kemeny-d. The scale limitation Kemeny-d is to blame: there are 91 HoFers in the 4K players in NBA2, but only 59 of them are among those 200 considered by Kemeny-d. In contrast, APST-α and weighted-sum have been computed over all 4K players. For comparison disregarding Kemeny's scale limitation, we have also used APST-α to rank the same subset of 200 players; in that case APST-α and Kemeny-d showed comparable performance (see [23] for results).

4.3 Effect of α in APST-α

To understand how the parameter α in APST-α helps in promoting specialized vs. well-rounded objects, we inject an artificial player into NBA2. Again, we use 7 attributes, as explained in Section 4.2. This artificial player has the highest *points* but the lowest values for all other attributes. Figure 7 shows how this player's rank changes in the APST-α ranking when α varies. As α decreases from 1, the artificial player quickly gets ranked high among real players. Its rank converges as α approaches 0, but never becomes the first because it cannot surpass real players who have the highest values on some other attributes. We have observed similar behaviors when the artificial player specializes in other attributes. This result agrees with the discussion in Section 3 that a smaller α favors specialized objects over well-rounded ones. An example of a specialized player is John Stockton, a HoFer, who is second only to Magic Johnson in assists per game, but is not ranked high in any other attributes considered. His ranks under various α values are also plotted in Figure 7, which show a similar trend to that for the artificial player.

In Figure 8, we focus on the top 30 players bound by APST-.9 in NBA2, under the three attributes: *points*, *rebounds*, and *assists*. The figure shows how their ranking positions change by α (horizontal axis has logarithmic scale). At the bottom of the figure, the top 9 players always remain in the top 13, because of their exceptional performance in multiple attributes. The ranks of the remaining players respond well to changes in α. We can categorize these players 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. In Figure 8, the curves of Type-I players go down as α decreases, because small α favors them more against other players in at least half of the subspaces, i.e., those containing the attribute on which they specialize. On the contrary, the curves of Type-II players go up as α decreases. They get rewarded in higher-dimensional subspaces for not being dominated by many players in such subspaces, while they can be easily dominated by others on individual dimensions.

For a concrete example, consider the lowest-ranked players at largest and smallest α, respectively. The lowest ranked curve (53rd) at the largest α (on the leftmost) corresponds to Nate Thurmond, a HoFer, ranked 277th in *points*, 6th in *rebounds*, and 579th in *assists*. As a representative Type-I player, when α decreases, his overall rank becomes higher, ending eventually at 18th. In contrast, the lowest ranked player (42nd) at the smallest α, Charles Barkley (another HoFer), ranked 33rd, 23rd, and 223rd respectively in *points*, *rebounds*, and *assists*. As a representative Type-II player, when α decreases, his rank goes all the way down from 11th. A tunable α thus allows a user to specify personal preference between Thurmond and Barkley effectively.

[5] For weighted-sum, we excluded these players when training the weight vector, but included them in ranking as with other methods.

[6] `http://www.ibm.com/software/integration/optimization/cplex-optimizer/`

5 Related Work

Our work is closely related to skyline computation for multidimensional data, which can be roughly grouped into two classes. The first class [3, 4, 8] includes Block Nested Loop (BNL), Divide and Conquer (D&C) and their variants, and do not require precomputed indices. The second class [18, 11, 15] adopts B^+-tree or R-tree indices to prune unnecessary computation. Skyline computation for multiple (or all) subspaces are first studied by Pei et al. [17] (BUS and TDS algorithms) and by Tao et al. [19] (SUBSKY algorithm). The former are extensions of BNL and D&C algorithms that traverse the subspace lattice without using indices, while the latter follows the index-based approaches. Extending skyline to k-skyband was first studied in [15].

While our notion of top-τ 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 proposing various advanced selection criteria when the size of the skyline is too large. For example, BBS [15] and PBT [24] focus on selecting points that dominate the most other points, and Lin et al. [13] focus on selecting a fixed sized set of points that collectively dominate the most other points. More recently, Lu et al. [14] propose the notion of layered skylines, where each layer is defined as the skyline after objects in previous layers are removed. Compared with these works, we believe our notion of top-τ skyband is more appealing in terms of usability, because it has a simple interpretation and a single parameter works across all subspaces.

Linear ranking of objects for the purpose of answering top-k queries has been studied extensively, including approaches using skyline and skybands [21, 22]. These 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 and offer good semantics. As discussed in Section 3.1, however, it does not provide a flexible knob, such as our α parameter, for adjusting the ranking based on the users' preference towards different types of interesting objects. Moreover, computing Kemeny is NP-hard.

Several lines of work on data mining, to various extents, share our focus on finding interesting claims that can be translated to simple, intuitive English statements. Local pattern discovery [26, 20] and subgroup discovery [12, 10] aim to find semantically describable subsets of data whose properties deviate strongly from the overall distribution. However, our subsets of interest, based on dominance, do not fit in their frameworks. Redescription mining [16] seeks to describe a given subset of data using a sequence of set operations. In contrast, we mine the subset in the first place, and our desired description of the subset involves dominance and counting instead. [9] finds "prominent streaks" (consecutive high/low values in a data sequence) and captures the significance of such streaks by skyline dominance test on streak length and value.

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 simple and intuitive problem formulation for finding all interesting claims, using a single uniqueness threshold τ 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 believe that our attention to usability will appeal to journalists and citizens alike.

Acknowledgments Y.W., P.K.A., and J.Y. are supported by NSF grants CNS-05-40347 and IIS-07-13498. P.K.A. is additionally supported by NSF grants CCF-06-35000, CCF-09-40671, CCF-10-12254, by ARO grants W911NF-07-1-0376 and W911NF-08-1-0452, and by an ARL award W9132V-11-C-0003. J.Y. is additionally supported by NSF grant IIS-09-16027. C.L. is supported by NSF Grants IIS-10-18865 and CCF-11-17369. Both J.Y. and C.L. are also supported by HP Labs Innovation Research Awards.

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] X. Jiang, C. Li, P. Luo, M. Wang, and Y. Yu. Prominent streak discovery in sequence data. In *KDD*, pages 1280–1288, 2011.

[10] B. Kavšek, N. Lavrač, and V. Jovanoski. Apriori-sd: Adapting association rule learning to subgroup discovery. *Advances in Intelligent Data Analysis V*, pages 230–241, 2003.

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

[12] N. Lavrač, F. Železnỳ, and P. Flach. Rsd: Relational subgroup discovery through first-order feature construction. *Inductive Logic Programming*, pages 149–165, 2003.

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

[14] 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.

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

[16] L. Parida and N. Ramakrishnan. Redescription mining: Structure theory and algorithms. *AAAI*, 20(2):837, 2005.

[17] 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.

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

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

[20] Y. Tian, G. Weiss, D. Hsu, and Q. Ma. A combinatorial fusion method for feature mining. In *KDD*, volume 7, 2007.

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

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

[23] Y. Wu, P. K. Agarwal, C. Li, J. Yang, and C. Yu. On "one of the few" objects. Technical report, Duke University, Feb. 2012. `http://www.cs.duke.edu/dbgroup/papers/2012-WuEtAl-oneoffew.pdf`.

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

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

[26] S. Zhang and M. Zaki. Mining multiple data sources: local pattern analysis. *KDD*, 12(2):121–125, 2006.