UNIVERSITY OF CALIFORNIA

Santa Barbara

Problems in Computational Geometry: Team Formation, Lottery Paths, and K-Dominance

A Thesis submitted in partial satisfaction of the
requirements for the degree Master of Science
in Computer Science

by

Thomas Gregory Schibler

Committee in charge:
Professor Subhash Suri, Chair
Professor Ambuj Singh
Professor Omer Egecioglu

September 2019

The thesis of Thomas Gregory Schibler is approved.

Ambuj Singh

Omer Egecioglu

Subhash Suri, Committee Chair

August 2019

ACKNOWLEDGEMENTS

I would especially like to thank Professor Subhash Suri for his excellent mentorship and collaboration on this thesis. I would also like to thank Professor Ambuj Singh for his collaboration on the team formation problem, and John Hershberger[1] for collaboration on the lottery problem.

[1]Mentor Graphics, Wilsonville, OR 97070

ABSTRACT

Problems in Computational Geometry: Team Formation, Lottery Paths, and K-Dominance

by

Thomas Gregory Schibler

This thesis is a compilation of three contributions to problems in computational geometry.

First, we consider a team formation problem in multi-dimensional space where the goal is to group a set of n agents into α teams, each of size β, to maximize their total performance. The performance of each team is measured by a score, which is the sum of h highest skill values in each dimension. We wish to maximize the sum of team scores. We prove that the problem is NP-hard if the dimension is $d = \Omega(\log n)$, even for scoring parameter $h = 1$ and team size $\beta = 4$. We then describe an efficient algorithm for solving the problem in two dimensions, as well an algorithm for computing a single optimal team in any constant dimension.

Second, we consider the following stochastic search problem, which we call the *lottery path* problem. The nodes of an edge-weighted graph contain (identical) prizes, with an independent but known probability distribution α, with at least one node guaranteed to yield prize. Whether or not a node has a prize is revealed only when the agent arrives at that node. The lottery path asks for a path of minimum expected length to reach a prize, starting from the initial node s. This problem is NP-hard, and our main result is a polynomial time algorithm with a constant factor approximation.

Lastly, we study the problem of k-dominance in a set of d-dimensional vectors, prove bounds on the number of maxima (skyline vectors), under both worst-case and average-case models, perform experimental evaluation using synthetic and real-world data, and explore an application of k-dominant skyline for extracting a small set of top-ranked vectors in high dimensions where the full skylines can be unmanageably large.

Contents

1 The Team Formation Problem **1**

1.1 Hardness of Team Formation 3

1.2 An Efficient Algorithm for 2 Dimensions 6

1.2.1 Forming 3-person teams in 2-dimensions 7

1.2.2 A Polynomial Time Algorithm 9

1.2.3 Optimal League Decomposition 10

1.2.4 Computing an Optimal League 11

1.2.5 Extension to Top h Scoring Rule 12

1.3 Team Formation in Higher Dimensions 13

2 Lottery Paths **14**

2.1 Preliminaries and Path Factorization 16

2.2 Approximation Algorithms 18

2.2.1 Lottery Paths When the $\boldsymbol{\alpha_i}$ Are Not Too Small 20

2.2.2 General Approximation of Lottery Paths 24

2.3 Lottery Paths vs. Multi-target Search 30

2.4 Polynomial Instances of Lottery Paths 31

2.4.1 **1**-Dimensional Lottery Path 31

2.4.2 Lottery Paths in DAGs 31

3 K-Dominance **32**

3.1 K-Dominance and the Worst-Case Bound 36

3.2 Average Case Analysis 39

3.3 Computing KDS in Subquadratic Time 43

3.4 Experimental Evaluation 45

3.4.1 KDS Size for Synthetic Data 46

3.4.2 KDS Size for Real-World Data 46
3.5 Application of KDS in Ranking of High-Dimensional Vectors 48
3.5.1 Top Movies and NBA Players 49
3.5.2 Jaccard Similarity 50
3.5.3 KDS vs. Aggregation-based Ranking 50

4 Concluding Remarks **53**

1 The Team Formation Problem

The problem of grouping a set of agents into teams with the objective of optimizing their collective performance is ubiquitous in a variety of organization settings, including team sports, project management, law, military, management consulting, academic ad hoc committees, to name a few. Mathematical models of team selection and performance, therefore, are an important area of research in social and management sciences. In these models, the skill set of each individual is typically modeled as an attribute vector. Research shows that while individual skills are clearly an important factor, the team's ability to search over vast and often ill-defined decision space crucially depends on its overall synergy and diversity [32, 47, 55, 59]. As a result, it is widely recognized that the performance of a team along a specific skill dimension should not depend on the average of the group members' values (so called weak synergy [31]) but rather on the skills of the best individuals on each dimension [10, 38, 56].

The selection of a single best team has been considered broadly in the literature [16, 20, 40, 41, 37, 39, 23, 52]. The more general problem of assembling multiple teams, however, is less well-understood, and has been studied mainly in the context of very specific performance objectives. For instance, Fitzpatrick and Askin [24] develop heuristics for assembling multiple 'multi-functional' teams using an integer programming formulation. The *coalition formation* problem in multi-agent (and multi-robot) systems also partitions agents into teams but the primary goal there is strategic utility maximization of completing a given set of tasks [53]. When the utility function is a simple sum of scalars, this becomes an easy-to-solve assignment problem in bipartite graphs [54], but under arbitrary set-valued functions the coalition formation is both NP-hard and inapproximable [53].

Against this backdrop, we investigate a simple and natural model for assembling multiple teams with multi-dimensional skills that allows us to explore the computational

complexity of multi-team formation as a function of the problem's intrinsic parameters: number of agents n, number of teams α, team size β, and dimension d of the skill vector. We place no constraints on the team structure except its prescribed size—any subset of agents can form a team—and use a simple additive function over independent attributes to measure team performance, thereby isolating the combinatorial aspects of the problem.

Specifically, we have an agent pool of n candidates, each modeled as a d-dimensional point $\mathbf{p} = (p_1, p_2, \ldots, p_d)$, where each dimension represents an independent real-valued skill. We want to form α teams, each of size β, for some integer values α, β, with $\alpha\beta \leq n$, so as to *maximize* the total score of all the teams. Each agent belongs to at most one team. In formulating the team score, we combine the two important aspects of a team performance: *strength* and *robustness* [20]. We measure the team strength by its coordinate-wise maxima but in order to add some degree of robustness we take the *top* h values for each coordinate, for a user-specified parameter $h \geq 1$. Thus, a team's score is defined as the *sum of* h *highest values of all dimensions*. We use the notation $\text{score}_h(T)$ to denote the score of team T using the top h scoring rule, which can be formally defined as

$$\text{score}_h(T) \quad = \quad \sum_{j=1}^{d} \max_{S \subset T, |S|=h} \sum_{i \in S} p_j^i,$$

where p_j^i is the jth coordinate of the ith point $\mathbf{p}_i$. Our problem then is the following: given a set of n agents, form α teams $T_1, T_2, \ldots, T_\alpha$, each of size β to maximize $\sum_i \text{score}_h(T_i)$. Figure 1 shows an example in two dimensions, where (A, C, D) is an optimal team of size 3 for the instance on the left using scoring parameter $h = 2$.

We show that the multi-dimensional multi-team formation problem is NP-hard even when each team has constant size, the scoring rule uses the top one value, namely, $h = 1$, and the dimension is logarithmic in n. Specifically, we show that the well-known 3-Dimensional Matching can be reduced to the team formation problem in dimension $d = \Omega(\log n)$ and scoring parameter $h = 1$. (If we consider very large dimensions, namely,

$d = \Omega(n)$, the problem becomes trivially hard because simply acquiring all necessary skills is a set covering problem. In most realistic settings, however, the dimension is much smaller than n, which is the focus of our work.) Our main result is a polynomial time algorithm for solving the 2-dimensional team formation problem optimally, for all scoring rules $h \geq 1$, using the following two-step algorithm. We first form a single team of size $\alpha \times \beta$, which we call a *league*, using a modified scoring rule. We prove that the total score of the league equals the score of the optimal team formation, and that an optimal league can be decomposed into an optimal solution of the team formation in polynomial time.

Our dynamic programming based algorithm can compute an optimal league in any fixed dimension. However, we show that the *league decomposition* lemma fails in higher dimensions, and so the optimal league's score no longer equals the score of the optimal team formation problem. Thus, forming multiple teams in more than two, but a constant, dimension remains an open problem.

1.1 Hardness of Team Formation

We begin with a brief reintroduction of the multi-team formation problem. Given an agent pool of n candidates, each candidate modeled as a d-dimensional point $\mathbf{p} = (p_1, p_2, \ldots, p_d)$, we want to form α teams, each of size β, for some integer values α, β, with $\alpha\beta \leq n$, so as to *maximize the sum* of team scores. Each agent belongs to at most team, and the score of a team T is defined as

$$\text{score}_h(T) \quad = \quad \sum_{j=1}^{d} \max_{S \subset T, |S|=h} \sum_{i \in S} p_j^i,$$

where p_j^i is the jth coordinate of the ith point $\mathbf{p}_i$, and $h \geq 1$ is the scoring parameter.

Theorem 1.1.1. *The multi-team formation problem is NP-hard.*

Proof. We show that the team formation problem is NP-hard in dimension $d = \Omega(\log n)$, even for scoring rule $h = 1$, by a reduction from the well known 3-dimensional matching problem (3DM). An instance of 3DM consists of three input sets X, Y, Z each of size n and a set of triples $W \subset X \times Y \times Z$. The problem is to decide if there is a subset $T \subseteq W$, with $|T| = n$, containing each element of $X \cup Y \cup Z$ once.

Given an instance of 3DM, we create an instance of the team formation problem as follows. The number of dimensions in our problem will be $6\ell + 4$, for a parameter $\ell = O(\log n)$. For each element of $X \cup Y \cup Z$, we associate a unique bit string of length 2ℓ, containing ℓ zeros and ℓ ones. We call this string the *tag* of that element. The bitwise complement of a tag will be denoted tag$'$. In particular, we will use the following types of bit strings:

1. $\text{tag}(x)$ = a unique bit string of length 2ℓ
   containing ℓ 0's and ℓ $1's$
2. $\text{tag}'(x)$ = bitwise complement of $\text{tag}(x)$
3. 0_ℓ = a string of ℓ $0's$

Using the fact that $\binom{2\ell}{\ell} \geq 2^\ell$, the choice of $\ell = 2\log n$ suffices for the creation of $3n$ distinct tags, one for each element of $X \cup Y \cup Z$. With the help of these tags we now create a point for each element of $X \cup Y \cup Z$ and each triple $t = (u, v, w)$ of W, in dimension $6\ell + 4$, as follows:

$x:$	$\text{tag}(x)$	$0_{2\ell}$	$0_{2\ell}$	1	0	0	0
$y:$	$0_{2\ell}$	$\text{tag}(y)$	$0_{2\ell}$	0	1	0	0
$z:$	$0_{2\ell}$	$0_{2\ell}$	$\text{tag}(z)$	0	0	1	0
$t:$	$\text{tag}'(u)$	$\text{tag}'(v)$	$\text{tag}'(w)$	0	0	0	1

Specifically, the first 2ℓ dimensions of $x \in X$ are its tag bits, followed by 4ℓ bits of 0's, and its last four bits are 1 0 0 0. The patterns for $y \in Y$ and $z \in Z$ are similar, as shown above. Next, the point corresponding to a triple $t = (u, v, w)$ has 6ℓ bits corresponding to the tag' strings of u, v, w, followed by the pattern 0 0 0 1. Altogether we have $3n + |W|$ points in dimension $O(\log n)$, which is polynomial in the input size.

We now prove the following: *the input 3DM instance is a yes instance if and only if our constructed instance admits formation of $\alpha = n$ teams, each of size $\beta = 4$, with total score at least $n(6\ell + 4)$.* To prove the forward direction, suppose the 3DM instance has a solution given by the set of triples T. For each $t = \{x, y, z\} \in T$, we create a team of size 4 using the points corresponding to x, y, z and t. Since $|T| = n$, and no element appears in more than one triple, we can form n disjoint teams. We now show that these teams achieve the target total score.

Each point's coordinate is either 0 or 1 along each of the $6\ell + 4$ dimensions, and so to reach the target score, each team must collect a 1 in each dimension, using its four points. Suppose the four points correspond to x, y, z and the triple (x, y, z). Then, by construction, in each of the first 6ℓ dimensions, we have a 1 in either tag() or tag$'$(), satisfying the requirement. Finally, the same holds for the last four dimensions, which is easy to check by inspection. Thus, assuming that the 3DM instance has a satisfying solution, we can construct n teams, each of size 4 with total score $n(6\ell + 4)$.

In the reverse direction, we show that any set of n teams with this score correspond to a perfect 3 dimensional matching. First, we observe that the optimal score requires that every team contributes exactly one 1 in each dimension. Considering the last 4 dimensions alone, this is only possible if the team contains exactly one point corresponding to a triple and each of the 3 elements in X, Y, and Z. Given this team structure, each of the first 3 sets of 2ℓ dimensions must collect a 1 from either the tag or tag$'$ of some element or triple respectively. To satisfy all 2ℓ dimensions, the tag and the tag$'$ must correspond to the same element, otherwise they will not be bitwise complements of each other.

Consequently, we must have elements x, y, z and triple $t = \{u, v, w\}$ with $x = u$, $y = v$, and $z = w$. If this property holds for all teams, then all selected triples must exactly cover each of the element sets, proving the existence of a 3DM solution. This completes the proof. □

Remark 1. The hardness proof is easily extended to any team size $\beta \geq 4$ by introducing an appropriate number of agents with null skills, namely, points with all 0 coordinates. The argument requires increasing the number of dimensions by $\Omega(\beta)$ to avoid the use of multiple triples in a single team. If the dimension is $\Omega(n)$, then computing a *single* team is also intractable. The proof can also be extended to scoring rule with $h > 1$ by introducing an appropriate number of points whose all coordinates are 1s.

Remark 2. It seems tempting to "compress" the ℓ-bits long strings into single numbers of polynomial size, and thereby reduce the dimension to a constant. However, the construction depends critically on being able to "add" each number to its complement to reach a target value, and breaks down if those bits are not used independently in the score.

1.2 An Efficient Algorithm for 2 Dimensions

Given the NP-hardness of the general problem, we now consider optimal team formation in small dimensions. In one dimension, the problem can be easily solved in $O(n \log n)$ time, as follows. We sort the agents in the increasing order of the skill level, say, the x axis. We then repeatedly select the top h unassigned agents, and assign them to the next team, until each of the α teams has h agents. Clearly, this assignment has the maximum sum of team scores. If needed, we can make each team's size to be exactly $\beta \geq h$, by arbitrarily selecting any of the unassigned agents since their scores do not contribute to the team scores.

In fact, a similar greedy strategy also solves the team formation problem for any dimension $d \geq 1$ if the team size is $\beta \geq hd$: repeat the earlier one-dimensional algorithm independently for each dimension. (It is possible for an agent to contribute a top score in more than one dimension, in which case a team may reach its maximum possible score with fewer than hd agents.) For team size $\beta < hd$, however, the problem becomes non-trivial even in dimension $d = 2$ and $h = 2$. This is the focus of the following discussion, where we consider the team formation problem in two dimensions.

1.2.1 Forming 3-person teams in 2-dimensions

In the interest of simpler exposition and proofs, we describe our algorithm using the scoring rule $h = 2$, and then discuss the minor adaptations needed for generalization to higher values of h. Therefore, in the following we drop the subscript h from the scoring notation; it is always assumed to be $h = 2$. Specifically, we focus on the case of team size $\beta = 3$, which helps illustrate some of the main difficulties of the problem. The case of $\beta = 1$ or $\beta = 2$ is easily solved greedily in two dimensions, and thus omitted from our discussion.

Somewhat surprisingly, the problem of forming teams of size 3 turns out to be non-trivial even if we want to form a single team, namely, $\alpha = 1$ with the scoring parameter $h = 2$. It serves as a test case both for disproving greedy schemes, and for our polynomial time algorithm. Using x and y as coordinates in two dimensions, suppose a 3-person team has agents with coordinates $(x_1, y_1), (x_2, y_2), (x_3, y_3)$. (Recall that we are using scoring rule of top two values, namely, $h = 2$.) Since the team score is composed of top two x and top two y values, and there are only 3 agents, at least one of them contributes both x and y to the team score. Let us call such an agent a 2-contributor. Each of the other agents contributes its x or y values (possibly these two agents are the same).

This property of the optimal 3-person team suggests a natural greedy algorithm: sort the agents by their x, y, and $x + y$ values. First take the agent with the maximum

$x + y$, remove it from all three lists, and then choose the agents with the maximum x and maximum y. Unfortunately, this simple algorithm is flawed. In fact, one can show that any algorithm that selects the team using only the *rank order* by x, y and $x + y$ coordinates fails. In particular, we construct two instances, each containing 4 agents, whose sorted orders by x, y, and $x + y$ are identical, yet their top scoring teams are different. The construction is shown in Figure 1.

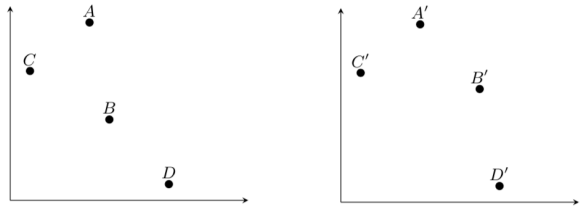

Figure 1: On the left, we show an instance of team formation with four two-dimensional agents: $A = (4, 11), B = (5, 5), C = (1, 8), D = (8, 1)$. On the right, we show a closely related instance with $A' = (4, 11), B' = (7, 7), C' = (1, 8), D' = (8, 1)$. The two instances have exactly the same sorted orders along $x, y, x + y$, but they lead to different optimal 3-person team solutions. The optimal team for the left instance is (A, C, D) with score 31 while the team for the right instance is (A', B', D') with score 33.

On the left, we have an instance with four agents $A = (4, 11), B = (5, 5), C = (1, 8), D = (8, 1)$. The optimal 3-person team for this instance is (A, C, D) with score of $31 = x(A) + x(D) + y(A) + y(C)$, with A contributing both x and y, C contributing y and D contributing x. On the right, we have another instance also with four agents, where only the coordinates of B' are different: $A' = (4, 11), B' = (7, 7), C' = (1, 8), D' = (8, 1)$, whose optimal team is (A', B', D') with score of $33 = x(B') + x(D') + y(A') + y(B')$. Yet, the two instances have the same ranking order by x, y, and $x + y$. *The crucial point of this example is that although A is an obvious choice for inclusion in the team, whether it contributes both x and y or just y depends on which other agents are in the team, namely, B or B'.*

Of course, since there are only $O(n^3)$ choices for a 3-person team, one can exhaustively

find an optimal one. But what about forming α teams? Even for the simple sum-of-team-scores objective function, the greedy strategy of iteratively computing the best 3-person team among the remaining agents fails, as shown by the following example of six agents that we want to group into two teams of size 3.

$$A = (20, 20); B = (10, 20); C = (20, 10); D = E = F = (0, 0)$$

The single optimal team is (A, B, C), with an score of 80, which leaves the remaining team of (D, E, F) with score 0. Instead, an optimal choice of two teams would be (A, B, D) and (C, E, F), which together have a score of 100.

1.2.2 A Polynomial Time Algorithm

In the following, we develop a polynomial time algorithm for solving the multi-team formation problem optimally in two dimensions. Our algorithm is based on the following idea:

1. First, identify the *union* of all agents that are in the optimal set of teams, and then
2. Partition this union into individual teams while preserving the total score.

A 3-person team in dimension $d = 2$ involves a total of $2d = 4$ individual skill scores, namely, top two scores in each of the two dimensions. Across α teams, therefore, we have a total of 4α scores. Instead of forming these teams, let us consider a slightly different problem. Find a group of 3α agents whose score is computed as follows: for each dimension, we take the top 2α skill values, and the group score is the sum of these 4α values.

For ease of reference, let us call such a group of 3α agents with this new scoring rule a *league*. Given a league L, let score(L) be the total score of L. Suppose $\mathcal{T}$ is the optimal set of 3-person teams, with total score score$(\mathcal{T})$. The question we ask is: what is the gap between score$(\mathcal{T})$ and score(L)? Clearly, score$(\mathcal{T}) \leq$ score(L), because the union of $\mathcal{T}$

is a valid league: a group of 3α agents, whose dimension-wise scores add up to score(L). But how much larger can the league score be compared to the team score? Our main result is that the two are always equal in two dimensions and, more importantly, (1) an optimal league can be partitioned in polynomial time into α teams of size $\beta = 3$ with the same total score, and (2) we can compute an optimal league in polynomial time. Together the two lead to a polynomial time algorithm.

1.2.3 Optimal League Decomposition

Let us first establish the league decomposition lemma.

Lemma 1.2.1 (League Decomposition)**.** *Given an instance of multi-team formation in two dimensions, let $\mathcal{T}$ be an optimal solution of α teams of size 3 each, and let L be an optimal league of size 3α. Then,* $\text{score}(\mathcal{T}) = \text{score}(L)$. *We can also partition L into an optimal multi-team solution in time $O(n)$.*

Proof. The score of a league sums the top 2α values in each dimension. We label each point a 2-contributor, x-contributor, y-contributor, or none, depending on how many coordinate values it contributes to the league score. We then observe the following:

1. there are at least α 2-contributors.

2. there are an equal number of x and y-contributors, and this number is at most α.

The first claim follows from the pigeon hole principle: 4α values are summed in scoring the league, but there are only 3α points, and so at least α points must contribute both of their coordinates. This leaves at most 2α values unaccounted for, which must be evenly split between x and y values. Thus, at most α values can come from an x-contributor, and α from a y-contributor.

We use these two facts to design a simple greedy algorithm for partitioning the league. The algorithm is shown above in Fig. 2. We first pair any 2-contributor of the league

```
1: procedure PARTITION 2D LEAGUE(p_1, ..., p_{3α})
2:     Initialize contributor lists X, Y, and XY.
3:     Initialize empty list of teams T.
4:     for i ∈ [0, len(X)] do
5:         Add team {XY[i], X[i], Y[i]} to T.
       j ← len(X).
6:     while j < len(XY) do
7:         Select any unused point p.
8:         Add team {XY[j], XY[j + 1], p} to T.
9:         j ← j + 2.
10:    Return T.
```

Figure 2: Partitioning a league into optimal teams.

with one x-contributor and one y-contributor. Because there are at least as many 2-contributors as x or y-contributors, we can continue this until there are no more 1-contributors left. By (2) above, we exhaust the x and y contributors at the same time. If any 2-contributors remain, we pair them arbitrarily together, along with an arbitrary extra point if we wish to maintain the team size.

To see that the resulting teams have the same total score as L, we note that exactly two x and two y values contributing to the league score are assigned to each team. Finally, the greedy algorithm only uses unsorted lists, and therefore runs in $O(n)$ time. This completes the proof. □

1.2.4 Computing an Optimal League

We now describe an algorithm for computing the optimal league L, using dynamic programming. Given a set of n d-dimensional points $\mathbf{p}_1, \cdots, \mathbf{p}_n$, we construct a 4-dimensional table A of size $n \times 3\alpha \times 2\alpha \times 2\alpha$ whose $A[i, j, k, l]$ entry stores $\text{score}_{k,l}(L_{i,j})$, where

$$\begin{aligned} L_{i,j} &= \text{an optimal league using at most } j \text{ points in } \{\mathbf{p}_1, \cdots, \mathbf{p}_i\} \\ \text{score}_{k,l}(L) &= \text{sum of top } k \text{ x-values and top } l \text{ y-values of } L \end{aligned}$$

The table is initialized as $L_{0,j} = L_{i,0} = 0$, for all i, j. Suppose we have computed all $L_{i-1,j-1}$ and want to compute $L_{i,j}$. Consider the new point $\mathbf{p}_i$. It is either not included in the league, or if it is included it serves in one of the three possible roles: x-contributor, y-contributor, or 2-contributor. We can, therefore, compute the table entry $L_{i,j}$ using the following dynamic program:

```
1: procedure 2D League(p_1, ..., p_n, α)
2:     Initialize n × 3α × 2α × 2α table A
3:     Let A[0, j, k, l] = 0, ∀j, k, l
4:     Let A[i, 0, k, l] = 0, ∀i, k, l
5:     for i ∈ [1, n] do
6:         for j ∈ [1, 3α] and k, l ∈ [0, 2α] do
7:             s_x ← A[i − 1, j − 1, k − 1, l] + p_i[x]
8:             s_y ← A[i − 1, j − 1, k, l − 1] + p_i[y]
9:             s_{x,y} ← A[i − 1, j − 1, k − 1, l − 1] + p_i[x] + p_i[y]
10:            s_0 ← A[i − 1, j, k, l]
11:            A[i, j, k, l] ← max(s_x, s_y, s_{x,y}, s_0)
12:    Return A[n, 3α, 2α, 2α]
```

Specifically, if $\mathbf{p}_i$ is an x-contributor, then $\text{score}_{k,l}(L_{i,j})$ is the x-coordinate of $\mathbf{p}_i$ plus the $\text{score}_{k-1,l}(L_{i-1,j-1})$; that is, the remaining points may only contribute $k-1$ x-values. We have similar cases for $\mathbf{p}_i$ being a y-contributor or a 2-contributor. The final optimal league score is found in the table entry $A[n, 3\alpha, 2\alpha, 2\alpha]$.

The table A has size $O(n\alpha^3)$, each entry can be computed in constant time, and so the algorithm runs in $O(n\alpha^3)$ time and space.

1.2.5 Extension to Top h Scoring Rule

The league decomposition lemma and the algorithm for computing the optimal league easily extend to scoring rule of top h, for all $h \geq 2$, as follows. Without loss of generality, we may assume that the team size satisfies $h \leq \beta < 2h$. Thus, the league size satisfies $\alpha\beta < 2\alpha h$. By the pigeonhole principle, the number of 2-contributors in the league is at least $\alpha(2h - \beta)$, and therefore we can assign to each team at least $(2h - \beta)$ of these

2-contributors, and fill the rest by 1-contributors arbitrarily. Similarly, the dynamic program algorithm is easily extended by changing the table size to $h\alpha$ instead of 2α. We summarize this result in the following theorem.

Theorem 1.2.1. *The multi-team formation problem in two dimensions can be solved optimally in worst-case time and space $O(n\alpha^3\beta h^2)$, where α is the number of teams, β the team size, h the scoring parameter, and n is the number of agents.*

1.3 Team Formation in Higher Dimensions

The dynamic programming algorithm of Section 1.2.3 can be extended to form an optimal *single* team of size $\beta < hd$ in polynomial time, for any fixed dimension d. Specifically, we compute a $(d+2)$-dimensional table A, whose first two dimensions are the same as before, namely, the first i points and the team size j. Each of the remaining d indices corresponds to the number of top scores in each dimension. In particular, $\text{score}_{k_1,\cdots k_d}$, where each $k_i \in [0, h]$, is the team score where top k_i values in dimension i have been accounted for. There are 2^d such combinations, so each table entry can be computed in $O(2^d)$ time. As mentioned earlier, when the team size $\beta \geq hd$, the problem can be easily solved in $O(dn)$ time using a greedy algorithm. We therefore have the following result.

Theorem 1.3.1. *We can compute an optimal single team of size β in d dimensions in time $O((2h)^d\beta n)$ time.*

The real difficulty in higher dimensions lies in forming *multiple teams*. In two dimensions, we used the League Decomposition Lemma as a key tool. Unfortunately, as we show below, in higher dimensions, this lemma no longer holds.

Theorem 1.3.2. *Let L be an optimal league, and $\mathcal{T}$ a set of optimal teams in dimensions $d \geq 4$. Then, there are instances for which* $\text{score}(\mathcal{T}) < \text{score}(L)$.

Proof. Consider the following set of 9 agents in four dimensions. $A = A' = (1, 1, 1, 0), B = B' = (1, 1, 0, 1), C = C' = (1, 0, 1, 1), D = D' = (0, 1, 1, 1)$, and $F = (0, 0, 0, 0)$. Suppose our goal is to form $\alpha = 3$ teams, each of size $\beta = 3$. Then, trivially, our league consists of all p points, where the scoring rule sums the top $2\alpha = 6$ values in each dimension. By construction, we have six 1s in each dimension, and so score$(L) = 24$.

However, any partition of these nine agents into 3 teams must assign the all-zero point F to one of the teams, which can therefore have a score of at most 6. On the other hand, no team has score more than 8, since the sum of top two entries in each of the four dimensions is two. Thus, the optimal team formation has score 22, proving that score$(\mathcal{T}) <$ score(L). This completes the proof. □

One can also show that an optimal partition of an optimal league L may not give an optimal team formation solution $\mathcal{T}$. For instance, imagine introducing one more agent $G = (1, 1, 1, 1)$ to the set of points in the previous example. Replacing F by G does not improve score(L), so L remains an optimal league. On the other hand, replacing F by G does improve score$(\mathcal{T})$. Finding an efficient algorithm for optimal or approximately optimal team formation in a constant dimension larger than 2 remains an interesting open problem.

2 Lottery Paths

Planning under uncertainty is a widely-studied topic in theoretical computer science, combinatorial optimization and operations research. One natural example of such planning is to explore a graph under a stochastic model of target location. Specifically, let $G = (V, E)$ be an edge-weighted undirected graph with n nodes. An agent must explore the nodes of G in search of a prize, where each node i contains a prize with an independent probability α_i. Whether or not a node contains a prize is only revealed to the agent when it reaches that node. Multiple nodes of the graph may contain prizes, meaning that

the sum $\sum_i \alpha_i$ can take any value between 0 and n, and all prizes are identical. Starting at its initial position s, the agent's goal is to reach a prize, if one exists, with minimum expected cost (path length).

Without loss of generality, we assume that at least one of the nodes contains a prize, and thus the agent's path length is always finite. Specifically, we can introduce an *anchor* node with $\alpha_i = 1$ at sufficiently large distance from all other nodes of G, without changing the outcome. The solution of the lottery path problem is a *policy*, which prescribes a path in G guaranteed to yield a prize with minimum expected length. It is easy to see the terminal node of such a policy must always be an anchor node, although the agent will likely stop earlier upon reaching a node containing a prize.

The lottery path problem is a game between an agent who wants to reach a target quickly and an adversary who wants to delay the agent but is constrained by the stochastic distribution of prize probabilities revealed to the agent at the start. Some of the work most closely related to lottery paths pertains to graph exploration problems with incomplete information. For instance, in one of the earliest papers, Papadimitriou and Yannakakis [49] studied the problem of "shortest paths without a map," in which the environment is dynamically specified. Their Canadian Traveler Problem (CTP) is a game between the agent and a *malicious* adversary: the agent is searching for a short s–t path, the adversary removes edges from the graph, and the agent learns about the state of an edge only upon reaching an adjacent node [3, 46, 49]. The Canadian Traveler Problem is PSPACE-complete. A *stochastic* version of search under uncertainty is the *graph search* problem (GSP) [35]: given an edge-weighted graph $G = (V, E)$, a start node s, and probability α_i for each node i such that $\sum_i \alpha_i = 1$, the goal is to find a tour through all the nodes that minimizes the *total weighted latency* $\sum_i \alpha_i C_i$, where C_i is the length of the subpath from s to i. While writing this, we became aware of a just-published work by van Ee and Sitters [58], which generalizes the graph search problem to *multi-target search* (MTS) in which multiple targets may exist independently among the nodes of the

graph, and the goal is to minimize the total weighted latency. This problem is closely related to the lottery search, but their approximation bound uses different assumptions, and is not directly comparable to ours. (See Sec. 4 for some discussion.) Specifically, they *condition* the search on finding a target, and ignore the search cost of any path that does not lead to a target. However, when applied to the same class of problems, our approximation bound improves the result of [58], from $14.4 + \epsilon$ to $11.7 + \epsilon$, for any $\epsilon > 0$.

A number of other problems, such as prize-collecting TSP or orienteering [2, 4, 5, 9, 15], also share the minimum-cost prize-winning aspect of lottery paths, but do not seem directly related. However, we make use of an approximation scheme for the orienteering problem.

Our main result is a polynomial time algorithm for approximating the lottery path within a constant factor. We do this in two steps. First, we present an $O(1/\alpha)$ factor approximation assuming that all prize probabilities satisfy $\alpha_i \geq \alpha$, for some $0 < \alpha \leq 1$. Then, to accommodate arbitrarily small values of α_i, we develop a subroutine that finds a path guaranteed to reduce the failure probability by a constant factor. We then iteratively use this subroutine to compose our final lottery path, which is guaranteed to be within a factor $11.7 + \epsilon$ of the optimal, for any $\epsilon > 0$.

Finally, we show that the lottery path can be computed optimally in polynomial time in some simpler environments, including one-dimensional space (a line) or a directed acyclic graph (DAG).

2.1 Preliminaries and Path Factorization

In the lottery problem, only the node identifiers and their pairwise distances matter, so we find it convenient to assume that the input is just an abstract *metric space* on n points $p_1, p_2, \ldots, p_n$ with pairwise distances $d(p_i, p_j)$, for all i, j. We can ensure this simply by taking the transitive closure of graph G using shortest path distances. Throughout, we

use points and nodes interchangeably. In addition to the metric space, the agent also knows the prize probabilities α_i for all the nodes. (Possibly only a subset of nodes may have $\alpha_i > 0$, but for the sake of generality we associate prize probabilities with all nodes.) Starting at the start point p_0, the agent explores the metric space in some order until a prize is found. In order to ensure finite expectation of the path length, we assume that at least one of the nodes has $\alpha_i = 1$. For convenience, we call any node with probability $\alpha_i = 1$ an *anchor*. Whether or not a node contains the prize is revealed to the agent only when it reaches the node. (Clearly, once a node is found not to contain the prize, any future visits to the same node also result in failure.) Any path not including an anchor fails to win the prize with non-zero probability, and any path extending beyond the first anchor cannot be shortest. Therefore, we have the following observation.

Lemma 2.1.1. *An optimal lottery path always terminates at an anchor.*

We begin with a useful factorization of the expected cost formula for the lottery path. Suppose the agent follows a path P from the starting point p_0 to some anchor point p_m. By renumbering the points, we may assume that the points along the path are $P = (p_0, p_1, \ldots, p_m)$. Let $\mathbb{E}(\lambda(P))$ be the expected path length of P. We can write this cost as

$$\mathbb{E}(\lambda(P)) = \sum_{i=1}^{m} \alpha_i F_i D_i,$$

where $F_i = \prod_{j=0}^{i-1}(1 - \alpha_j)$ is the joint probability that the agent reaches p_i, having failed at the first $i - 1$ points, and $D_i = \sum_{j=0}^{i-1} d(p_j, p_{j+1})$ is the total distance to p_i along P. The expected path length $\mathbb{E}(\lambda(P))$ is simply the sum over m independent events that the search terminates at point p_i, for $i = 1, 2, \ldots, m$. This path formula, however, is rather unwieldy because the contribution of any edge (p_i, p_{i+1}) is difficult to isolate. The following lemma simplifies the formula.

Lemma 2.1.2. *The expected length of a path $P = (p_0, p_1, \ldots, p_m)$, where p_m is an anchor*

point, can be written as

$$\mathbb{E}(\lambda(P)) \;=\; \sum_{i=1}^{m} d(p_{i-1}, p_i) F_i.$$

Proof. Let $\mathbb{E}(\lambda(P_k))$ be the *prefix cost* of the path terminating no later than p_k. (Equivalently, this is the path length assuming $\alpha_k = 1$.) Since p_m is an anchor, we have $\alpha_m = 1$ and therefore can write

$$\mathbb{E}(\lambda(P_m)) \;=\; \sum_{i=1}^{m-1} \alpha_i F_i D_i + 1 F_m D_m \;=\; \sum_{i=1}^{m-2} \alpha_i F_i D_i + \alpha_{m-1} F_{m-1} D_{m-1} + F_m D_m.$$

Using the equalities $F_m = (1 - \alpha_{m-1}) F_{m-1}$ and $D_m = D_{m-1} + d(p_{m-1}, p_m)$, we can write the last term as $(1-\alpha_{m-1}) F_{m-1}(D_{m-1} + d(p_{m-1}, p_m))$, which after simple algebraic manipulation gives

$$\begin{aligned} \mathbb{E}(\lambda(P_m)) &= \sum_{i=1}^{m-2} \alpha_i F_i D_i \;+\; \alpha_{m-1} F_{m-1} D_{m-1} + (1 - \alpha_{m-1}) F_{m-1}(D_{m-1} + d(p_{m-1}, p_m)) \\ &= \mathbb{E}(\lambda(P_{m-1})) + d(p_{m-1}, p_m) F_m, \end{aligned}$$

By expanding this recurrence, we get $\mathbb{E}(\lambda(P_m)) = \sum_{i=1}^{m} d(p_{i-1}, p_i) F_i$, which completes the proof. □

2.2 Approximation Algorithms

We first show that the lottery problem is *NP*-hard, by reduction from the Hamiltonian path problem.

Lemma 2.2.1. *The lottery problem is NP-hard.*

Proof. Given an undirected graph $G = (V, E)$, deciding if there is a simple path visiting every node of G exactly once is the well-known *NP*-complete Hamiltonian Path Problem [26]. We reduce it to the lottery path problem by creating a point p_i for each node of G, with the following distance metric. If (i, j) is an edge in the graph G, then we define

$d(p_i, p_j) = 1$, and otherwise $d(p_i, p_j) = 2$. All these points have prize probability $\alpha_i = \frac{1}{2}$. Finally, we introduce two additional points p_0 and p_{n+1}, with distances defined as follows: $d(p_0, p_i) = 1$ and $d(p_i, p_{n+1}) = L$, for $i = 1, 2, \ldots, n$, where L is a parameter to be fixed later. Any choice of $L \geq 4$ will suffice, but we keep L as a variable to make the calculations in our proof more transparent. Lastly, we set the distance $d(p_0, p_{n+1}) = L + 1$. The prize probability for p_0 is $\alpha_0 = 0$, while p_{n+1} is an anchor (the only one) with $\alpha_{n+1} = 1$.

Let us consider the optimal lottery path from p_0 to p_{n+1}. The direct path has cost $d(p_0, p_{n+1}) = L + 1$ but path p_0, p_i, p_{n+1} through any intermediate point p_i has cost at most $2 + \frac{1}{2}L$ (actually $1 + \frac{1}{2}L$) since $\alpha_i = 1/2$. This length is strictly smaller than $L + 1$, for any $L \geq 4$. Indeed, as long as there is any *unvisited* point p_j, we can improve the expected path length by inserting p_j just before p_{n+1}. Thus, the optimal lottery path must include all the input points $p_1, p_2, \ldots, p_n$. If G is Hamiltonian, then we use that path to construct the lottery path P from p_0 to p_{n+1} in which the first n edges have length 1 and the last has length L. Since the success probability at each intermediate point is $1/2$, the expected cost of this lottery path P is

$$\mathbb{E}(\lambda(P)) = \sum_{i=1}^{n} \left(\frac{1}{2}\right)^{i-1} + \left(\frac{1}{2}\right)^{n} L.$$

On the other hand, if G is not Hamiltonian, then every path from p_0 to p_{n+1} visiting all n nodes includes at least one edge of length 2, and therefore the expected path length is strictly larger than $\mathbb{E}(\lambda(P))$. Thus, a polynomial time algorithm for computing the optimal lottery path length can also decide whether G is Hamiltonian or not. □

Given the hardness of finding an exact solution to the lottery problem, we focus on designing a polynomial time approximation algorithm. Specifically, if the expected length of the optimal lottery path is λ^*, then our algorithm computes a path that finds the prize within expected length at most $(11.7 + \epsilon)\lambda^*$, for any $\epsilon > 0$. For ease of presentation, we describe the algorithm in two parts: we first design an algorithm (Section 2.2.1) that

works only if all prize probabilities are lower bounded by a constant; that is, none of the probabilities α_i is smaller than some pre-specified value α. In this case, the algorithm and its proof are quite simple; they form the basis of the main algorithm (Section 2.2.2), which does not require any assumption on the α_i values.

2.2.1 Lottery Paths When the α_i Are Not Too Small

Let us assume that the prize probabilities α_i at all nodes satisfy $\alpha_i \geq \alpha$, for some $0 < \alpha \leq 1$ and all $i > 0$. In this case, we show how to compute a lottery path whose expected length is $O(\frac{1}{\alpha})$ times the optimal. The algorithm, which we call the SUNFLOWER algorithm, works as follows:

1. Order the points in descending order using a ranking function (discussed below). Without loss of generality, suppose $p_1, p_2, \ldots, p_n$ is the rank order.

2. Construct the lottery path as a *sunflower pattern* p_0, p_1, p_0, p_2, p_0, ..., p_0, p_i, p_0, ..., p_m, where p_m is the first anchor point encountered.

That is, the algorithm performs an explore-and-retract, visiting each point p_i in turn and immediately returning to the origin p_0, until it reaches an anchor. We call this path a *sunflower* because it is a series of two-hop loops, all joined at the origin, as shown in Figure 3. In spite of its simplicity, we prove that this algorithm delivers a constant factor approximation with an appropriate ranking function.

Intuitively, the sunflower algorithm's performance rests on resolving the following dilemma: at any time, the agent has several unvisited points, offering a tradeoff between prize probabilities and costs. However, once the agent chooses to visit a point p, points closer to p may appear more attractive since they cost so little to visit. But this has the potential to lure the agent into blind alleys of the search space, ultimately incurring very high costs. The sunflower path avoids this pitfall by immediately returning to the origin after visiting p, and as our analysis shows, as long as prize probabilities are not

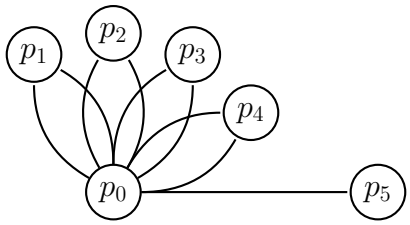

Figure 3: A sunflower path on 6 points. Each "petal" is formed by traveling to a non-anchor point and immediately back to the root.

too small, the retraction costs form a geometric series, incurring only a constant factor increase in the total path length.

Our proof of the algorithm rests on Lemma 2.2.2, which bounds the cost of converting an optimal lottery path into a sunflower path, and Lemma 2.2.3, which prescribes how to order the points for sunflower visiting.

Lemma 2.2.2. *Consider an instance of the lottery path problem in which all probabilities satisfy $\alpha_i \geq \alpha$, for some constant $\alpha > 0$. Given an optimal lottery path $P = \{p_0, p_1, \ldots, p_m\}$ for this instance, there exists a sunflower path P' whose expected cost is within a constant factor of P, namely, $\mathbb{E}(\lambda(P')) \leq (\frac{2}{\alpha} - 1)\mathbb{E}(\lambda(P))$.*

Proof. By Lemma 2.1.2, the expected cost of P can be written as

$$\mathbb{E}(\lambda(P)) = \sum_{i=1}^{m} d(p_{i-1}, p_i) F_i \tag{1}$$

We now consider the sunflower path $P' = \{p_0, p_1, p_0, p_2, \ldots, p_0, p_m\}$ visiting points in the same order as P. We compare the costs of these two paths by considering the contribution of each edge (p_{i-1}, p_i). In $\mathbb{E}(\lambda(P))$, this edge contributes $d(p_{i-1}, p_i)F_i$, namely, the edge length weighted by the failure probability of all preceding nodes.

In path P', the agent returns to p_0 after each p_i, and thus to advance from p_i to p_{i+1}, it must travel additional distance $d(p_i, p_0) + d(p_0, p_{i+1})$. We note two important properties: first, by the triangle inequality, $d(p_0, p_i) \leq \sum_{j=1}^{i} d(p_{j-1}, p_j)$, and second, although path backtracking before reaching p_{i+1} causes the distance $d(p_{i-1}, p_i)$ to be traversed two more

times, in the expected cost formula this distance increase is *pre-multiplied* by the failure probability of the prefix path. Since the extra contribution of $2d(p_{i-1}, p_i)$ occurs only for points p_j, $j = i+1, i+2, \ldots, m$, the total contribution of (p_{i-1}, p_i) to $\mathbb{E}(\lambda(P'))$ is:

$$\left(F_i + 2 \sum_{j=i+1}^{m} F_j \right) d(p_{i-1}, p_i).$$

Since $F_j = F_{j-1}(1 - \alpha_{j-1})$, we can factor out F_i from each term in the sum, leaving a product with $j - i$ terms. Because all probabilities satisfy $\alpha_i \geq \alpha$, this product is $\leq (1-\alpha)^{j-i}$. The contribution of $d(p_{i-1}, p_i)$ to $\mathbb{E}(\lambda(P'))$ is therefore upper bounded by $F_i \left(1 + 2\sum_{j=i+1}^{m} (1-\alpha)^{j-i}\right) d(p_{i-1}, p_i)$. We can bound the geometric series as follows: $\sum_{j=i+1}^{m}(1-\alpha)^{j-i} < \sum_{j=1}^{\infty}(1-\alpha)^j = \frac{1}{\alpha} - 1$. Thus, the contribution of (p_{i-1}, p_i) to $\mathbb{E}(\lambda(P'))$ is at most $(\frac{2}{\alpha} - 1)d(p_{i-1}, p_i)F_i$. Comparing to Eq. (1), therefore, it follows that

$$\mathbb{E}(\lambda(P')) \leq \left(\frac{2}{\alpha} - 1\right)\mathbb{E}(\lambda(P)),$$

which completes the proof. □

The next lemma establishes the structure of the optimal sunflower path.

Lemma 2.2.3. *The optimal sunflower path visits points in increasing order of* $(2/\alpha_i - 1)d(p_0, p_i)$.

Proof. In a sunflower path, if p_i is the ith point visited, for $i < m$, then its contribution to the expected path length is $F_i(1 + (1 - \alpha_i))d(p_0, p_i)$ because distance $d(p_0, p_i)$ is traveled with probability F_i, followed by the backtracking distance $d(p_0, p_i)$, which is traveled if the trial at p_i fails. Only the term for p_m does not include backtracking cost. Thus, the expected cost of the sunflower path visiting points in the order $p_1, p_2, \ldots, p_m$ is

$$\mathbb{E}(\lambda(P')) = \sum_{i=1}^{m-1} F_i(1 + (1 - \alpha_i))d(p_0, p_i) + F_m d(p_0, p_m)$$

In particular, the contribution of p_i is $F_i(2-\alpha_i)d(p_0,p_i)$, for $i < m$. Suppose for the sake of contradiction that the optimal sunflower path does not visit the points in the sorted (increasing) order as claimed. Then we must have two consecutive indices i and $j = i+1$ that are out of order, meaning

$$\left(\frac{2}{\alpha_i} - 1\right) d(p_0,p_i) > \left(\frac{2}{\alpha_j} - 1\right) d(p_0,p_j). \tag{2}$$

Let us consider the change in the expected cost if we swap the order of p_i and p_j. Since they are adjacent along the path, reordering them does not affect the terms preceding or following them. In particular, the failure probability term F_i for the prefix and suffix remain the same, and so it suffices to consider only the subpath (p_0,p_i,p_0,p_j,p_0). Before the swap, this subpath has cost $F_i(1+(1-\alpha_i))d(p_0,p_i) + F_j(1+(1-\alpha_j))d(p_0,p_j)$, which can be rewritten as

$$F_i(2-\alpha_i)d(p_0,p_i) + F_i(1-\alpha_i)(2-\alpha_j)d(p_0,p_j),$$

because $F_j = F_{i+1} = F_i(1-\alpha_i)$. The failure probability of the prefix F_i remains unchanged when we swap the order of p_i, p_j, and so the new cost of the subpath is

$$F_i(2-\alpha_j)d(p_0,p_j) + F_i(1-\alpha_j)(2-\alpha_i)d(p_0,p_i).$$

Optimality of the original path implies that

$$(2-\alpha_i)d(p_0,p_i) + (1-\alpha_i)(2-\alpha_j)d(p_0,p_j)$$

$$\leq (2-\alpha_j)d(p_0,p_j) + (1-\alpha_j)(2-\alpha_i)d(p_0,p_i),$$

which after simple algebraic manipulation implies

$$\left(\frac{2}{\alpha_i} - 1\right) d(p_0, p_i) \leq \left(\frac{2}{\alpha_j} - 1\right) d(p_0, p_j),$$

contradicting the inequality of Eq. (2). Thus, the optimal path must visit points in the claimed order. □

The pseudo-code for the final sunflower algorithm is shown in Algorithm 1; the main result of this section is summarized in Theorem 2.2.1.

Algorithm 1 SUNFLOWER $(p_0, \ldots, p_n, \alpha_0, \ldots, \alpha_n)$

1: Sort non-anchor points p_i in ascending order of $(\frac{2}{\alpha_i} - 1)d(p_0, p_i)$.
2: Compute the prefix cost and failure probability F_i for each p_i.
3: Let p_m be the anchor with minimum distance to p_0.
4: Compute the cost of sunflower path P_i visiting $\{p_0, \ldots, p_i\}$ in this order and terminating at p_m, for each i.
5: **return** the P_i with the minimum $\mathbb{E}(\lambda(P_i))$.

Theorem 2.2.1. *Given an instance of the lottery problem on an n-node graph, where all prize probabilities satisfy $\alpha_i \geq \alpha$, for $0 < \alpha \leq 1$, the* SUNFLOWER *algorithm computes a lottery path whose expected cost is within a factor $(\frac{2}{\alpha} - 1)$ of the optimal.*

The time complexity of the SUNFLOWER algorithm is $O(n \log n)$, dominated by sorting, plus $O(n)$ time needed to evaluate the prefix path costs and their failure probabilities F_i. However, this assumes that all distances $d(p_0, p_i)$ are given as part of the input. If not, we need to run one instance of Dijkstra's algorithm to compute distances from p_0 to all nodes, which can be done in $O(|E| + n \log n)$ time using Fibonacci heaps [19, 25], where $|E|$ is the number of edges in G.

2.2.2 General Approximation of Lottery Paths

Theorem 2.2.1 quickly loses its appeal if prize probabilities α_i become too small. For instance, if some of the points have $\alpha_i = 1/n$, the theorem only guarantees an $O(n)$

approximation. In this section, we describe our main result for approximating the optimal lottery path to within a factor of roughly 11.7 with no restriction on probability values. We begin by showing that all nodes with probability smaller than $1/n$ can be discarded without significantly affecting the lottery path cost.

Lemma 2.2.4. *Disregarding all points with probability $\alpha_i \leq 1/n$ increases the cost of the optimal lottery path by a factor less than the Euler number e.*

Proof. Suppose an instance A of the lottery path has an optimal path $P = \{p_0, p_1, \ldots, p_m\}$ visiting an arbitrary number of points with probability less than $1/n$. We consider a modified instance A' in which all probabilities smaller than $1/n$ are set to 0, namely, $\alpha'_i = 0$ if $\alpha_i \leq 1/n$, and $\alpha'_i = \alpha_i$ otherwise. Let P' be the path in A' visiting the points in the same order as P. Although the two paths visit the same sequence of points, $\mathbb{E}(\lambda(P'))$ is larger than $\mathbb{E}(\lambda(P))$ because some of the $(1 - \alpha_i)$ terms may increase to 1 due to zeroing of small probabilities. In particular, $\mathbb{E}(\lambda(P')) = \sum_{i=1}^{m} F'_i d(p_{i-1}, p_i)$, where $F'_i = F_i/B_i$, where $B_i = \prod_{0<j<i, \alpha_j \leq 1/n}(1 - \alpha_j)$. We now observe that $B_i \geq \prod_{0<j<n}(1 - 1/n) = (1 - 1/n)^{n-1} > 1/e$, and so $\mathbb{E}(\lambda(P')) < e\mathbb{E}(\lambda(P))$. □

We still have to deal with probability values between a constant and $O(1/n)$. The main conceptual problem is that when point probabilities are small, the sunflower path can lead to arbitrarily bad approximation because returning to the origin after each point is very costly. Our solution is a *quota* version of the sunflower algorithm, in which each "petal" subpath explores points until it has accumulated sufficiently large success probability before backtracking to the origin. More specifically, the subpath must reduce the failure probability by at least a factor of ϕ, for a fixed parameter ϕ, before backtracking. We show that an appropriate choice of ϕ, along with an algorithm to compute such an exploration path, leads to a constant factor approximation of the lottery path. Our algorithm, called QUOTA-SUNFLOWER, works as follows:

1. Compute a minimum length path such that the joint probability of failure at all those points is less than $1/\phi$.

2. Return to the root and repeat on the remaining points until termination.

Step 1 uses an orienteering path algorithm of [9] as a subroutine, and is described in detail later. For now, let us assume that we can find a prefix path of the minimum length that achieves our target probability, and discuss how to analyze the approximation guarantee of this algorithm. We do this by deriving a lower bound on the expected cost of the optimal path $\mathbb{E}(\lambda(P))$ and an upper bound on the cost of our quota-sunflower path $\mathbb{E}(\lambda(P'))$.

Suppose the optimal lottery path is $P = \{p_0, p_1, \ldots, p_m\}$, and let us evaluate the failure probability along P. In particular, the point p_i is reached with probability F_i, which is the joint probability of failure at all nodes preceding p_i along P. The initial failure probability is $F_0 = 1$. We will define a sequence of *marker points* $z_1, z_2, \ldots, z_k$ along P at which failure probabilities successively reduce by a factor of ϕ. More precisely, let z_1 be the first p_i along P at which the failure probability $F_i(1-\alpha_i)$ drops below $1/\phi$. In general, let z_j be the first point along P at which $F_j(1-\alpha_j) \leq (1/\phi)^j$. Using $d(z_i, z_j)$ to denote the distance along P from z_i to z_j, we can now derive a lower bound on the cost $\mathbb{E}(\lambda(P))$ as follows.

The agent must travel distance $d(z_0, z_1)$ with probability *at least* $1/\phi$ because its joint failure probability is $F_i > 1/\phi$ for all points p_i before z_1 along P. Similarly, it travels distance $d(z_1, z_2)$ with probability at least $1/\phi^2$, and so on. Summing these terms over all the markers $z_1, z_2, \ldots, z_k$, we have the following lower bound:

$$\mathbb{E}(\lambda(P)) \;\geq\; \sum_{i=1}^{k} \frac{1}{\phi^i} d(z_{i-1}, z_i).$$

We now derive an upper bound for a quota-sunflower path, in which the agent back-

tracks to the origin after each marker. Since in the first loop the agent reduces its failure probability to $1/\phi$ by following a minimum length path, it travels at most distance $d(p_0, z_1)$, because the prefix subpath of P from p_0 to z_1 is one such feasible alternative. With probability at most $1/\phi$, the agent then backtracks to p_0, which is at distance at most $d(p_0, z_1)$. In the next round, the agent finds a minimum length path that reduces the failure probability by another factor of $1/\phi$, and the length of this path is no worse than the prefix of P between p_0 and z_2. In general, we don't know the length of the minimum path that reduces the failure probability to $1/\phi^i$, but $\sum_{j=1}^{i} d(z_{j-1}, z_j)$, achieved by the prefix path of P, is an easy upper bound. The agent travels this distance with probability at most $1/\phi^{i-1}$, and then backtracks to the origin with probability at most $1/\phi^i$. Summing these distances gives an upper bound on the quota-sunflower path:

$$\begin{aligned}
\mathbb{E}(\lambda(P')) &\leq \sum_{i=1}^{k} \left(\frac{1}{\phi^{i-1}} + \frac{1}{\phi^i} \right) \sum_{j=1}^{i} d(z_{j-1}, z_j) = \sum_{i=1}^{k} d(z_{i-1}, z_i) \sum_{j=i}^{k} \left(\frac{1}{\phi^{j-1}} + \frac{1}{\phi^j} \right) \\
&\leq \sum_{i=1}^{k} d(z_{i-1}, z_i) \left(\frac{1}{\phi^{i-1}} + \sum_{j=i}^{k} \frac{2}{\phi^j} \right)
\end{aligned}$$

We can now bound the ratio $\mathbb{E}(\lambda(P'))/\mathbb{E}(\lambda(P))$ by dividing the ith term of the former by the corresponding term of the latter:

$$\frac{1/\phi^{i-1} + \sum_{j=i}^{k} 2/\phi^j}{1/\phi^i} \leq \phi + \sum_{j=0}^{k-i} \frac{2}{\phi^j} \leq \phi + \frac{2\phi}{\phi - 1}$$

The final expression is independent of i, and thus

$$\frac{\mathbb{E}(\lambda(P'))}{\mathbb{E}(\lambda(P))} \leq \phi + \frac{2\phi}{\phi - 1},$$

which is a constant that can be optimized by an appropriate choice of ϕ.

We now describe how to implement Step 1, which computes an approximately shortest path starting at p_0 and reduces the failure probability by a factor $\geq \phi$. We use an

approximation algorithm from [9] for the orienteering problem, which is a variant of prize-collecting Traveling Salesman. Specifically, given an edge-weighted graph G with prizes at nodes, a start node s, and a prize target T, the orienteering problem is to compute the shortest path from s that collects prizes totaling at least T. This problem is *NP*-hard, and Blum et al. present a polynomial-time algorithm for computing an orienteering path whose length is within a constant factor of the optimal. (The constant factor in that paper is $2 + \delta$, for any $\delta > 0$.)

In order to use the orienteering path for our problem, we need to adapt to two differences: first, the orienteering problem *sums* the prize values at nodes visited, while we wish to minimize the path's failure probability; second, the algorithm of [9] assumes that all node prizes are integers in the range $[0, n^2]$ while our probabilities are reals. We use scaling and rounding to address both these problems, as explained in the following lemma.

Lemma 2.2.5. *Given an instance of the lottery path problem, let $\lfloor n^2 \ln(1/(1-\alpha_i)) \rfloor$ be the prize assigned to a node whose success probability is α_i. An orienteering path that collects total prize $T = \lceil n^2 \ln \phi \rceil$ reduces the failure probability by a factor of at least ϕ.*

Proof. Since p_i has prize $w_i = \lfloor n^2 \ln(1/(1-\alpha_i)) \rfloor$, we have $\ln(1-\alpha_i) \leq -(w_i/n^2)$. If the path $P = p_0, \ldots, p_k$ collects the target prize, then $\sum_{i=1}^{k} w_i \geq T \geq n^2 \ln \phi$. This path reduces the failure probability by

$$\prod_{i=1}^{k}(1-\alpha_i) \;=\; \exp\left(\sum_{i=1}^{k}\ln(1-\alpha_i)\right) \;\leq\; e^{-\sum_{i=1}^{k}\frac{w_i}{n^2}} \;\leq\; e^{-\ln\phi} \;=\; \frac{1}{\phi}.$$

□

Remark. In fact, the optimal path shrinks the failure probability quite close to the prescribed amount. More specifically, since prizes are integer-valued, the total prize of the first $k-1$ nodes must fall short of the target: $\sum_{i=1}^{k-1} w_i < n^2 \ln \phi$. Next, since each node's

prize w_i is defined by the floor function, we have $\sum_{i=1}^{k-1}(w_i+1) \geq -n^2 \sum_{i=1}^{k-1} \ln(1-\alpha_i)$. From these, we can derive that

$$\ln \phi > -\frac{k}{n^2} - \sum_{i=1}^{k-1} \ln(1-\alpha_i) \geq -\frac{1}{n} - \sum_{i=1}^{k-1} \ln(1-\alpha_i).$$

By negating and exponentiating, we obtain $\prod_{i=1}^{k-1}(1-\alpha_i) > \frac{1}{\phi} e^{-1/n} \geq \frac{1}{\phi}(1-1/n)$. That is, a path that cannot be shortened overshoots its failure probability target by a relative error of at most $1/n$.

We now use the algorithm of Blum et al. to compute an orienteering path with approximation factor $(2+\delta)(\phi + \frac{2\phi}{\phi-1})$. The choice of $\phi = 1+\sqrt{2}$ minimizes the ratio to $(11.7+\delta')$ for any small constant $\delta' > 0$.[2] The perturbation of ϕ by $O(1/n)$ introduced by the floor and ceiling functions can be absorbed into the constant δ'. The pseudo-code for Quota-Sunflower is shown as Algorithm 2.

Algorithm 2 Quota-Sunflower$(p_0, \ldots, p_n, \alpha_0, \ldots, \alpha_n)$

1: $S \leftarrow \{p_0, \ldots, p_n\}$
2: Assign each p_i prize value $\lfloor n^2 \ln(1/(1-\alpha_i)) \rfloor$.
3: $j \leftarrow 0$
4: **while** S is not empty **do**
5: Apply Blum et al.'s orienteering path algorithm from s to all non-anchor choices of t with quota $\lceil n^2 \ln(1+\sqrt{2}) \rceil$.
6: Store the minimum distance solution as P_j.
7: Remove all p_i appearing in P_j from S.
8: Increment j.
9: Let p_m be the anchor that minimizes $d(p_0, p_m)$.
10: For each $j \in [0,k]$, compute the expected length of path $P_0, P_1, \ldots, P_j, p_m$.
11: **return** the minimum cost path in step 10.

[2] As defined, some prizes may be larger than n^2. We can treat these as a special case—a path that reaches a node with such a prize stops there, since $\phi < e$.

We now have the main result of this section.

Theorem 2.2.2. *Given an instance of the lottery problem on an n-node graph, the* Quota-Sunflower *algorithm computes in polynomial time a lottery path whose expected cost is within a constant factor of the optimal, where the constant is $(6+4\sqrt{2})+\epsilon$, for any $\epsilon > 0$, or about* 11.7.

2.3 Lottery Paths vs. Multi-target Search

While preparing this work, we learned of the recently published (and independent) work by van Ee and Sitters [58] on multi-target search. The following brief discussion is intended to highlight similarities and differences between our results.

In both the lottery path (LP) and multi-target search (MTS) problems, we are given an undirected graph with edge weights and stochastic prizes at the nodes, and the goal is to compute a path, starting from a fixed node, whose expected length before reaching a prize is minimum. However, there is a small but important difference between the two problems: LP assumes that at least one node (an *anchor*) has $\alpha_i = 1$, so successful termination is guaranteed, whereas MTS allows all nodes to have $\alpha_i < 1$, so there is a finite probability $\prod_i (1 - \alpha_i)$ that no prize is found. When this failure occurs, the cost of exhaustively searching the entire graph is *excluded* from the MTS bound.

As a result, the two problems are equivalent in terms of *exact* answers, but not for approximation. MTS can be reduced to LP by adding an anchor connected to all other nodes but very far away. The optimal lottery path visits all other nodes before reaching the anchor. Since the distances to the anchor are all equal, and the failure probability before the anchor does not depend on the order of the nodes before it, the prefix of the LP is identical to the optimal MTS path. In the other direction, LP reduces to MTS directly by forcing the probabilities of some nodes (anchors).

Since the approximation bound of van Ee and Sitters [58] ignores the cost of paths

that do not reach a prize, it is not strictly comparable to ours. However, for inputs for which the agent finds the prize with high probability ($\prod_i(1-\alpha_i)$ is small), the results are comparable, and our approximation factor of $11.7+\epsilon$ slightly improves the one in [58], which is $14.4+\epsilon$.

2.4 Polynomial Instances of Lottery Paths

Although the lottery path problem is *NP*-hard in general graphs, it can be solved optimally in polynomial time for some special classes of graphs. In this section, we discuss two such instances that may be of interest: lottery on a line (the 1-dimensional problem) and directed acyclic graphs.

2.4.1 1-Dimensional Lottery Path

The lottery path problem is non-trivial even if the search space is a one-dimensional linear chain: the agent's optimal path can switch directions (left $\leftrightarrow$ right) an arbitrary number of times. So, while simple greedy strategies do not work, we can show that a dynamic programming based algorithm can compute the optimal lottery path in $O(n^2)$ time.

2.4.2 Lottery Paths in DAGs

If the underlying graph is a directed acyclic graph (DAG), then the optimal lottery path can be found in linear time. We first compute a *topological ordering* of the nodes, which can be done in $O(|V|+|E|)$ time [19]. We then process the nodes in the *reverse* topological order, starting from the anchor(s) and, at each node, compute the optimal suffix cost, along with the best outgoing edge. When a node p_i is processed, all of its successors have been processed already. Therefore we can compute the optimal suffix path from p_i by taking the minimum over all of its outgoing neighbors, and weight this cost by $(1-\alpha_i)$.

The algorithm terminates when the root (source) is processed. The time to process each node is proportional to its outdegree, and therefore the entire algorithm takes $O(|E|)$ time and $O(|V|)$ space.

3 K-Dominance

In multi-criteria optimization and decision-making applications, there is often no single best answer, and a popular approach is to use pareto optimality. The set of pareto optimal points, which are the coordinate-wise *undominated* solutions, is called the *skyline*. Unfortunately as the dimension of the data grows, the size of the skyline tends to explode and most, if not all, of the input vectors can appear on the skyline [8, 13, 14]. (Although in theory all input points can appear on the skyline even in two dimensions, this pathological behavior is rarely observed in low-dimensional real-world data.) A database query for a car or a smart phone, for instance, can easily produce an overwhelming number of incomparable choices. In the National Basketball Association's (NBA) database of 21961 players in 17 dimensions (scoring attributes), more than 1400 players appear on the skyline. Figure 8 shows several real-world datasets with large skylines. The problem is even more pronounced in crowdsourced data such as movies or consumer product ratings—each input vector is the rating profile of a product by the users—where virtually every product can be highly ranked by some user, potentially elevating it to the skyline. While the classical result of Bentley et al [8] shows that the expected size of the skyline of n random vectors, whose components are chosen independently, is $O((\log n)^{d-1})$ in d-dimensions, the exponential dependence on d renders the skyline useless even in theory except in very low dimensions.

The *k-dominant skyline* $\mathcal{KDS}$ was introduced as a way to tame this curse of dimensionality, where by relaxing d-dominance to k-dominance, for $k < d$, many more points can be eliminated from the skyline, resulting in a smaller and more manageable set of

maxima. Formally, given a finite set of points $\mathcal{V}$ in R^d, a point u is said to *k-dominate* another point v if $u_i \geq v_i$ holds for k of the dimensions, $i = 1, 2, \ldots, d$, with strict inequality in at least one dimension. We use the notation $u \succ_k v$ to indicate k-domination of v by u, and write $u \not\succ_k v$ when u does not k-dominate v. The *k-dominant skyline* of $\mathcal{V}$, denoted $\mathcal{KDS}(\mathcal{V}, k)$, is the set of points that are not k-dominated:

$$\mathcal{KDS}(\mathcal{V}, k) \;=\; \{v \in \mathcal{V} \mid u \;\not\succ_k\; v,\; \forall\, u \in \mathcal{V} \setminus \{v\}\}.$$

We make both theoretical and applied contributions to the study of k-dominant skylines.

We analyze the cardinality of the k-dominant skyline under both the worst-case, where the points are chosen adversarily, and the average case, where the point coordinates are chosen independently from a distribution. Using synthetic and real-world data sets, we show that the $\mathcal{KDS}$ size in practice follows the exponential trend predicted by our average case bound. We then use movie ratings and NBA basketball datasets to show that $\mathcal{KDS}$ is an effective tool for high-dimensional ranking queries. The full-dimensional skylines of these data sets are unmanageably large, but $\mathcal{KDS}$ consistently finds top vectors using purely pareto property, without the need for ad hoc preference (utility) functions. (We discuss these issues further in Section 3.5.) Specifically, we make the following contributions:

1. [*Worst Case Bound:*] The worst-case cardinality of $\mathcal{KDS}$, when input points are chosen adversarily, is n for all $k > (d+1)/2$, and at most 1 for all $k \leq (d+1)/2$.

2. [*Average Case Bound:*] Let $\mathcal{V}$ be a set of n vectors in d dimensions where the components of each vector are distributed independently of each other, and for each component the magnitudes form a random permutation of $\{1, 2, \ldots, n\}$. We

show that the expected number of vectors appearing on the k-dominant skyline is

$$\begin{cases} O\left((\log n)^{2k-d-1}\right) & \text{for } k > \frac{d+1}{2} \\ O(1) & \text{otherwise} \end{cases}$$

Our result smoothly interpolates the cardinality of $\mathcal{KDS}$ for all values of k, and subsumes the classical result of Bentley et al. [8] as a special case for $k = d$.

3. [*Efficient Algorithm:*] The k-dominance relation does not obey transitivity, which is needed for the efficient (sub-quadratic) computation of skylines. We show that $\mathcal{KDS}$ can be computed from the traditional skylines of all k-dimensional projections of the input vectors. Our algorithm runs in worst-case time $O\left(d^{d-k} n \log^{k-1} n\right)$, which is subquadratic even for non-constant dimensions as long as $d \leq c \log n / \log \log n$, for some constant c.

4. [*Experiments:*] Our average case analysis assumes attribute distinctness and statistical independence of input vectors. While there is no reason to believe that real-world datasets meet these conditions, our experiments on a variety of data show that the $\mathcal{KDS}$ size follows the exponential growth predicted by our analysis.

5. [*Applications:*] We show that $\mathcal{KDS}$ is a useful tool for *robust ranking* in high dimensions. For instance, which players should be called the "10 most dominant NBA players" when more than 1400 are pareto optimal (undominated)? Our experiments show that the vectors that are the *first ones* to populate the k-dominant skyline, and therefore have the most longevity because k-dominance is a monotone property, are indeed the natural candidates for top ranking.

The skyline or maxima problem has a long history in computational geometry, mathematical optimization, and databases [7, 8, 22, 12, 11, 17, 28, 34, 36, 42, 51, 57, 27, 43, 48],

and the quest for efficient algorithms that scale to high dimensions and large input continues to this day. It has also been observed that as the dimension grows, the skylines can quickly lose their data-reduction utility because most of the input vectors may appear on the skyline [13, 18].

A number of approaches have been considered for identifying a smaller size representation of the full skyline in high dimensions. One simple method aggregates all the dimensions into a single *utility function*, and uses the ranking defined by this function. This approach while computationally attractive is problematic because it simply transfers all the burden to the "designer" of the utility function. Indeed, one of the important benefits of skyline-based approaches is to allow the users to explore various tradeoffs across different components such as price, location, brand etc. [13, 14]. Another approach is to compute a small set of input vectors that approximates the skyline over *all* preference functions. Unfortunately, these approaches are computationally intractable even for linear preference functions, as in the case of k-regret set [18, 45, 50, 1] or *approximate*-skyline [33].

Our focus is to both prove an upper bound on the size of the $\mathcal{KDS}$ and to explore its applications in ranking of high dimensional data. There exists a substantial and rich literature on estimating the cardinality of (conventional) skylines [8, 12, 27, 28, 42, 57], under a variety of data models, including in-memory, distributed, and data streams. However, very little is known about the cardinality of k-dominant skylines. The k-dominant skyline was introduced in [13] but the primary goal of that paper is to design efficient heuristics for computing the $\mathcal{KDS}$ scalably in practice. In a related work, Chan et al. [14] compute vectors that appear in many k-dominant skylines, but again the paper is concerned with the design and evaluation of an efficient heuristic. In [30], Hwang et al. consider certain threshold phenomena in k-dominant skylines under a continuous probability model, and derive limit bounds as $n, d \to \infty$. By contrast, we establish parametric upper bounds on the cardinality of $\mathcal{KDS}$ under both random and worst-case

inputs, similar to those for the conventional skylines obtained by Bentley et al. [8] or Buchta [12].

3.1 K-Dominance and the Worst-Case Bound

Let $\mathcal{V}$ be a set of n vectors in d-dimensional space. We will use the letters u and v to denote generic vectors of $\mathcal{V}$, and v_i as the ith coordinate of $v \in \mathcal{V}$. That is, $v = (v_1, v_2, \ldots, v_d)$ is the coordinate representation of vector v. We will use the terms vectors and points interchangeably since vectors are commonly viewed as points in d-dimensional space. Given two vectors $u, v \in \mathcal{V}$, we say that u *dominates* v, denoted $u \succ v$, if $u_i \geq v_i$ for $i = 1, 2, \ldots, d$, with strict inequality in at least one dimension, We say that a vector u *k-dominates* v if $u_i \geq v_i$ for at least k of the d possible dimensions, with strict inequality in at least one. We write this as $u \succ_k v$, generalizing the original definition of domination, which is equivalent to d-domination $\succ_d$. The *k-dominant skyline* of $\mathcal{V}$ is the set of vectors that are not k-dominated by any other vector. That is,

$$\mathcal{KDS}(\mathcal{V}, k) \;=\; \{v \in \mathcal{V} \mid u \;\not\succ_k\; v,\; \forall\, u \in \mathcal{V} \setminus \{v\}\}.$$

The k dimensions involved in k-dominance $u \succ_k v$ need not be the same as those involved in $u' \succ_k v'$, and so the k-dominant skyline does not equal the skyline of $\mathcal{V}$ after projection onto some fixed k-dimensional subspace. It also means that the k-dominance relation is *not transitive*: if $u \succ_k v$ and $v \succ_k w$, then we do not necessarily have $u \succ_k w$.

We begin by introducing the notion of *k-dominant graphs* that is useful for reasoning about the $\mathcal{KDS}$ and for establishing bounds on its cardinality. Given a set of input vectors $\mathcal{V}$ in d dimensions, we define a *directed graph* G_k, called the *k-dominant graph*, as follows: the vertices of G_k correspond to the vectors of $\mathcal{V}$, and its edges correspond to k-dominance relationships between pairs of vectors. That is, the graph G_k has a *directed* edge (u, v) whenever $u \succ_k v$, for $u, v \in \mathcal{V}$. The *in-degree* of a node v in G_k is the number

Vectors	Dim 1	2	3	4	5	6	7
1	6	2	6	4	3	6	4
2	5	1	4	2	6	5	2
3	1	3	5	5	5	1	5
4	4	5	3	1	4	4	1
5	3	4	2	6	2	3	6
6	2	6	1	3	1	2	3

Figure 4: An example for illustration of $\mathcal{KDS}$.

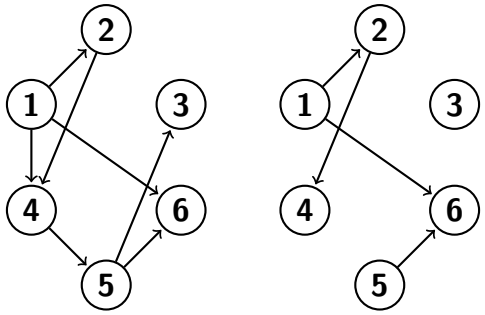

Figure 5: The graphs G_5 (left) and G_6 (right) for the vectors of Fig. 4. The $\mathcal{KDS}$ on the left contains a single vertex 1, and the one on the right has three vectors $\{1, 3, 5\}$.

of edges directed into v, namely, in-degree$(v) = |\{(u, v) \in G_k\}|$.

Figure 4 shows an example with $n = 6$ vectors in $d = 7$ dimensional space, and its corresponding 5-dominant and 6-dominant graphs are shown in Fig. 5. The non-transitivity of k-dominance can be seen in this example: 1 dominates 4, and 4 dominates 5, but 1 does not dominate 5.

The following two facts, which are used for proving the worst-case size bound of $\mathcal{KDS}$, easily follow from the k-dominant graph.

Lemma 3.1.1. *A vector v belongs to $\mathcal{KDS}(\mathcal{V}, k)$ if and only if v has in-degree zero in G_k.*

Proof. If v has in-degree zero, then no other vertex $u \in V$ can k-dominate it—otherwise (u, v) will be an edge directed into v. Conversely, if v belongs to $\mathcal{KDS}(\mathcal{V}, k)$, then it

has no incoming edge (u, v) because such an edge contradicts the claim that v is on the k-dominant skyline. □

It is also easy to see that the k-dominant skyline is a monotone function of k: all vectors on the k'-dominant skyline are also on the k-dominant skyline for $k' < k$. This was also observed in [13], however, we include it for completeness, along with an easier and more transparent proof using the k-dominant graph.

Lemma 3.1.2. *$\mathcal{KDS}(\mathcal{V}, k') \subseteq \mathcal{KDS}(\mathcal{V}, k)$, for $k' < k$.*

Proof. Clearly, $v \succ_k u$ implies $v \succ_{k'} u$ for all $k' < k$. Thus, if (u, v) is an edge in G_k, then it is also an edge in $G_{k'}$, which means that the set of edges in $G_{k'}$ is a superset of those in G_k. Since adding new edges to a graph cannot *increase* the number of nodes with in-degree zero, the zero in-degree nodes in $G_{k'}$ form a subset of the zero in-degree nodes in G_k, establishing $\mathcal{KDS}(\mathcal{V}, k') \subseteq \mathcal{KDS}(\mathcal{V}, k)$. □

With these preliminaries, we can now prove the following bound for the size of the $\mathcal{KDS}$ in the worst-case.[3]

Theorem 3.1.1. *The worst-case cardinality of $\mathcal{KDS}$ obeys the following bound:*

1. *For any set of n vectors in d-space and any k with $k \leq (d+1)/2$, $|\mathcal{KDS}(\mathcal{V}, k)| \leq 1$.*
2. *For any $n \geq 1$, and k, d such that $k > (d+1)/2$, there* exists *a set $\mathcal{V}$ of n vectors in d-space for which $|\mathcal{KDS}(\mathcal{V}, k)| = n$.*

Proof. We prove the first part by contradiction. Suppose $2k \leq d + 1$ and $\mathcal{KDS}(\mathcal{V}, k)$ contains at least two distinct vectors u, v. By Lemma 3.1.1, both u and v have in-degrees zero in the graph G_k, and thus neither (u, v) nor (v, u) is an edge in G_k. Consider the component-wise comparison between u and v. Since each can dominate the other on

[3]The first part of our theorem resembles a claim in [13] for $k < d/2$. Our result is more general with an easier proof.

$\leq k-1$ coordinates, and $2(k-1) < d$, we have at least one dimension unaccounted for, a contradiction, and the claim is proved.

To prove the second part, we describe an explicit construction with $|\mathcal{KDS}(\mathcal{V}, k)| = n$ and $d = 2(k-1)$. This suffices for the proof because this is the smallest dimension satisfying $2k > (d+1)$, and the bound for all other values of k follows from the monotonicity of k-dominant skylines asserted by Lemma 3.1.2. We represent the vectors as an $n \times d$ array, whose first $(k-1)$ column vectors are $(1, 2, \cdots, n)^T$, and whose last $(k-1)$ column vectors are $(n, n-1, \cdots, 1)^T$. Thus, the first vector has 1 as its first $(k-1)$ coordinates and n as the remaining coordinates, while the last vector has n as its first $(k-1)$ coordinates and 1 in the remaining coordinates. Figure 6 illustrates the construction for $d = 6$ and $k = 4$, which meets the condition $d = 2(k-1)$.

1	1	1	n	n	n
2	2	2	$n-1$	$n-1$	$n-1$
3	3	3	$n-2$	$n-2$	$n-2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	n	n	1	1	1

Figure 6: Proof of Theorem 3.1.1: $d = 6,\ k = 4$. No vector 4-dominates another and therefore all vectors belong to $\mathcal{KDS}$ for $k \geq 4$.

It is now easy to see that no vector u in this set k-dominates another vector v, because there are only $2k-2$ dimensions, and each dominates the other in precisely half of the dimensions, which is only $(k-1)$. Thus, we have $|\mathcal{KDS}(\mathcal{V}, k)| = |\mathcal{V}| = n$, and the proof is complete. □

3.2 Average Case Analysis

We now analyze the size of the k-dominant skylines when the input vectors are drawn randomly from a distribution. Our analysis uses the standard "attribute distinctness and statistical independence" model [8, 12, 27], which only assumes that the vector

components are distributed independently of each other, and for each component the magnitudes form a random permutation of $\{1, 2, \ldots, n\}$, namely, a total rank ordering. While not all data sets necessarily satisfy these assumptions, the model is sufficiently general, elegant and mathematically tractable for deriving non-trivial bounds.

We interpret the input set of n vectors as an $n \times d$ array, whose rows are the vectors and whose columns are permutations of $\{1, 2, \ldots, n\}$. The set of all possible permutations produces exactly $(n!)^d$ distinct vectors. We analyze the complexity of the k-dominant skyline for an input array $\mathcal{V}$ chosen uniformly at random from this set. What is the expected size of the k-dominant skyline for such an input $\mathcal{V}$?

We analyze the average size of $\mathcal{KDS}$ by setting up a recurrence. In order to aid that derivation, let $A(n, d, k)$ denote the average size of the k-dominant skyline for a set of n vectors in d dimensions, where the vectors are chosen under the random model described above. We assume, without loss of generality, that the first column of the input $n \times d$ array is sorted in the ascending order $(1, 2, \ldots, n)$—that is, the first coordinate of the ith vector is i, which if necessary can be realized by simply relabeling the vectors. See Figure 7 for illustration.

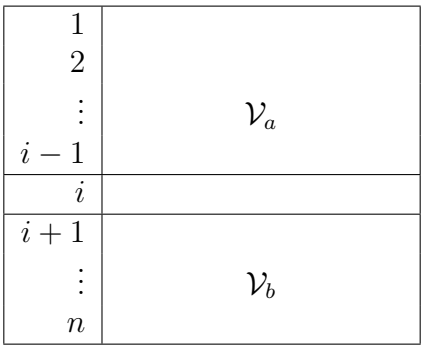

Figure 7: Average case analysis of $\mathcal{KDS}$. The vector v is the ith vector. The first $(i-1)$ vectors form the subset $\mathcal{V}_a$, and the last $(n-i)$ vectors form the subset $\mathcal{V}_b$.

Let us focus on a single but arbitrary vector v, and derive the probability that it belongs to the k-dominant skyline. Suppose the vector v is represented as the ith row,

which means its first coordinate is $v_1 = i$. We partition the remaining set of input vectors into two groups:

$$\mathcal{V}_a = \{u \in \mathcal{V} \mid u_1 < v_1\} \qquad \text{and} \qquad \mathcal{V}_b = \{u \in \mathcal{V} \mid u_1 > v_1\} \tag{3}$$

That is, $\mathcal{V}_a$ is the set of vectors that v dominates on the first coordinate, and $\mathcal{V}_b$ is the set of vectors that dominate v on the first coordinate. The key observation is that for v to be a $\mathcal{KDS}$ vector both of the following two events must occur:

1. None of the vectors in $\mathcal{V}_a$ k-dominates v in dimensions $\{2, 3, \ldots, d\}$, and
2. None of the vectors in $\mathcal{V}_b$ $(k-1)$-dominates v in dimensions $\{2, 3, \ldots, d\}$.

This follows because v can fail to be on the $\mathcal{KDS}$ only if either some vector in $\mathcal{V}_a$ or some vector in $\mathcal{V}_b$ k-dominates it. If some vector in $\mathcal{V}_a$ were to k-dominate v, it has to do so using all k dimensions from the set $\{2, 3, \ldots, d\}$ since v already dominates each vector of $\mathcal{V}_a$ on dimension 1. Condition (1) computes the probability that no $u \in \mathcal{V}_a$ k-dominates v. On the other hand, since each vector $u \in V_b$ already dominates v on the first coordinate, it needs to only find $k-1$ other dimensions among $\{2, 3, \ldots, d\}$ to achieve k-domination of v. Condition (2) computes the probability that no $u \in \mathcal{V}_b$ k-dominates v. The two events are independent because the remaining $d-1$ dimensions are independent of the first, and so the probability that v belongs to the $\mathcal{KDS}$ is the *product* of these two probabilities. The following two lemmas derive these probabilities.

Lemma 3.2.1. *The probability of event (1) is* $\frac{A(i,d-1,k)}{i}$.

Proof. Consider the $i \times (d-1)$ array consisting of the first i rows and the last $(d-1)$ columns of $\mathcal{V}$. This is a random set of i vectors in $(d-1)$-dimensional space, which for convenience we call the *reduced space*. By induction, the expected $\mathcal{KDS}$ size for this set is $A(i, d-1, k)$. The probability that v is one of these skyline vectors is $A(i, d-1, k)/i$,

by symmetry. Since v already dominates all the vectors of $\mathcal{V}_a$ in the first coordinate, it is on the $\mathcal{KDS}$ of $\mathcal{V}_a \cup \{v\}$ with the same probability. □

Lemma 3.2.2. *The probability of event (2) is* $\frac{A(n-i+1,d-1,k-1)}{n-i+1}$.

Proof. The proof is similar to (1). □

By combining the preceding two lemmas, we get the probability that v lies on the $\mathcal{KDS}$:

$$\Pr[v \in \mathcal{KDS}] \;=\; \frac{A(i,d-1,k)}{i} \times \frac{A(n-i+1,d-1,k-1)}{n-i+1}.$$

By summing over all n points, we get

$$A(n,d,k) \;=\; \sum_{i=1}^{n} \left(\frac{A(i,d-1,k)}{i} \times \frac{A(n-i+1,d-1,k-1)}{n-i+1} \right) \tag{4}$$

Since $\frac{A(i,d-1,k)}{i}$ is a probability in this recurrence, we can replace it with 1 and derive the following upper bound with a change of index:

$$A(n,d,k) \;\leq\; \sum_{i=1}^{n} \left(\frac{A(n-i+1,d-1,k-1)}{n-i+1} \right) \;\leq\; \sum_{i=1}^{n} \frac{A(i,d-1,k-1)}{i} \tag{5}$$

The function $A(n,d,k)$ is monotone non-decreasing with n because

$$A(n,d,k) \;=\; A(n-1,d,k) \;+\; \frac{A(n,d-1,k)}{n} \times \frac{A(1,d-1,k-1)}{1} \tag{6}$$

where the second term is non-negative. Therefore, we have the following upper bound:

$$A(n,d,k) \;\leq\; \sum_{i=1}^{n} \frac{A(i,d-1,k-1)}{i} \;\leq\; A(n,d-1,k-1) \times \sum_{i=1}^{n} \frac{1}{i} \tag{7}$$

Using the approximation for the harmonic number $H_n = \sum_{i=1}^{n} \frac{1}{i} \approx \ln n$ [19], we get the following after j iterations:

$$A(n, d, k) \;\leq\; A(n, d-j, k-j) \times (H_n)^j$$

The stopping condition for the recurrence is reached when $2(k-j)$ becomes less than or equal to $(d-j)+1$, at which point the skyline size drops to at most 1 by Theorem 3.1.1. We can show that the maximum number of iterations j is $j = 2k-(d+1)$, which leads to the following theorem.

Theorem 3.2.1. *Let $\mathcal{V}$ be a set of n random points in d-dimensional space under the attribute distinctness and statistical independence model. Then, the expected cardinality of their k-dominant skyline is*

$$\begin{cases} O\left((\log n)^{2k-d-1}\right) & \text{for } k > \frac{d+1}{2} \\ O(1) & \text{otherwise} \end{cases}$$

Theorem 3.2.1 smoothly interpolates between the two extreme cardinality bounds for $\mathcal{KDS}$ previously known, namely, $O(1)$ for $k \leq (d+1)/2$, and $O(\log^{d-1} n)$ for $k = d$. The classical result of Bentley et al. [8] emerges as a special case of this theorem for $k = d$. In database systems, a constructive way to utilize this result could be this: by reducing the value of k by one, we can expect the $\mathcal{KDS}$ to shrink by a significant fraction, namely, a factor of $O(\log^2 n)$. A query engine can therefore tune the parameter k to predictably control the (expected) number of skyline vectors that appear on $\mathcal{KDS}$.

3.3 Computing KDS in Subquadratic Time

Unlike full-dimensional dominance, the k-dominance relation is *not transitive*: that is, $a \succ_k b$ and $b \succ_k c$ does not guarantee $a \succ_k c$. The failure of transitivity means that the algorithms for traditional skylines [7, 28] cannot be used for computing $\mathcal{KDS}$, and to the best of our knowledge, no subquadratic time algorithm was known for k-dominant

skylines even when k is a constant. In this section, we describe such an algorithm.

Let $I_k \subset \{1, 2, \ldots, d\}$ be an index set of size k, namely, $|I_k| = k$. There are $\binom{d}{k}$ size k distinct index sets, and each such set defines a subset of k dimensions. The projection of the input set of points $\mathcal{V}$ along any particular set I_k is called a *k-projection* of $\mathcal{V}$. A k-projection is a set of k-dimensional points, and we refer to its skyline as the skyline of the k-projection. Then, the following simple observation is the key to our algorithm.

Lemma 3.3.1. *A point $v \in \mathcal{V}$ belongs to $\mathcal{KDS}(\mathcal{V}, k)$ if and only if v belongs to the skylines of all distinct k-projections of $\mathcal{V}$.*

We can, therefore, compute $\mathcal{KDS}(\mathcal{V}, k)$ by computing the skylines of all distinct k-projections of $\mathcal{V}$ and taking their common intersection.

Theorem 3.3.1. *Let $\mathcal{V}$ be a set of n points in d dimensions, where $d = O(\log n / \log\log n)$. The k-dominant skyline $\mathcal{KDS}(\mathcal{V}, k)$ is precisely the common intersection of the skylines of all possible k-projections of $\mathcal{V}$, and it can be computed in worst-case time $O(n \log^{d-1} n)$.*

Proof. The number of distinct k-projections of $\mathcal{V}$ is $\binom{d}{k} = \binom{d}{d-k}$. We can compute the skyline of each of these projections in time $O(n \log^{k-1} n)$ using the divide-and-conquer algorithm of [7]. The size of each of these skylines is at most n, and therefore we can compute their common intersection in $O(nd)$ time. The total running time of the algorithm is therefore $O\left(\binom{d}{d-k}\, n \log^{k-1} n\right)$. Since $\binom{d}{d-k} \leq (\frac{ed}{d-k})^{d-k} = O(d^{d-k})$, we can upper bound the running time as $O\left(d^{d-k} n \log^{k-1} n\right)$, which is sub-quadratic as long as $d \leq c \log n / \log\log n$, for an appropriate constant c. □

Finding a worst-case subquadratic algorithm in dimensions higher than $\Theta(\log n)$ remains a challenging open problem, even for conventional skylines.

3.4 Experimental Evaluation

Our theoretical bounds on the expected size of the $\mathcal{KDS}$ are predicated on certain properties of random data (attribute distinctness and statistical independence), which may or may not be satisfied by real-world datasets. In our experiments, we chose a number of diverse data sets to evaluate how their $\mathcal{KDS}$ size behaves in practice. We also generated synthetic data sets, following our theoretical distribution, to establish a baseline. (Since time complexity was not an important concern, we implemented a simple algorithm, which constructs an $O(dn)$ space data structure in $O(dn^2)$ time from which the $\mathcal{KDS}$ for any value of $k \leq d$ can be extracted in $O(n)$ time explicitly.)

Data Sets

Data	Size n	Dim d	Full Skyline
NBA Career	4051	17	80
NBA Season	21961	17	1466
Movie Lens 1	1682	943	1330
Movie Lens 2	3952	6040	3475
Wine Quality	4898	12	3094
Human Activity	7352	561	7352
Internet Ads	3279	1558	1976
Particle Signal	130065	50	129596

Figure 8: Data sets used in experiments and the size of their d-dimensional skyline.

The synthetic data sets are produced by generating random permutations with $n \approx 10^5$ and $d = 20$ or 30. For the real-world data, we use a number of popular high-dimensional data sets, as shown in Figure 8. The NBA basketball data is available at [6], the Movie Lens data at [44, 29], and all the remaining datasets are available from the UCI repository [21]. They vary in size from a few thousand points to more than hundred thousand points in dimensions ranging from 10 to several thousand. Altogether these data sets allow us to evaluate the behavior of $\mathcal{KDS}$ under highly diverse condi-

tions. In addition, for many of these sets, nearly all the input vectors show up on the full-dimensional skyline, providing a useful test for the utility of the $\mathcal{KDS}$. In all the experimental plots, the x-axis is the value of k and y-axis the size of $\mathcal{KDS}$.

3.4.1 KDS Size for Synthetic Data

We generated a number of random data sets in moderate dimensions $d \leq 30$, which proved sufficient to show the exponential growth rate of the k-dominant skylines.

Figure 9 shows the results for $n = 100K$, with $d = 20$ and $d = 30$; the plots for other values of n and d were found to be similar. (When the plots flatten out near the end, it means that the $\mathcal{KDS}$ size has reached the total number of input vectors.)

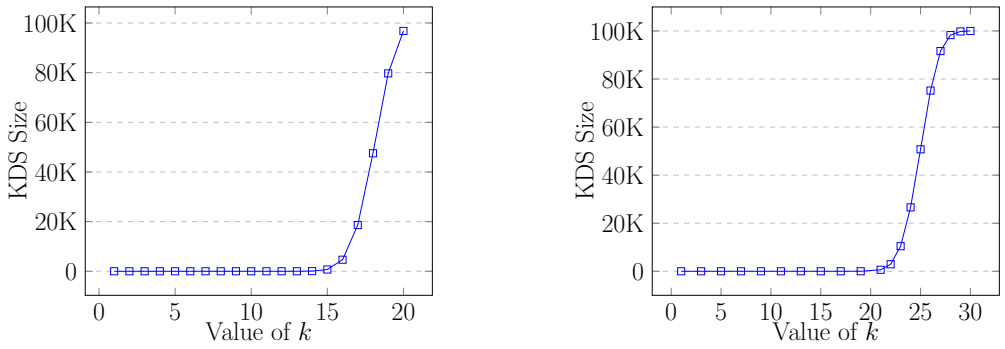

Figure 9: $\mathcal{KDS}$ size scaling for $n = 10^5$ random vectors in 20 (left) and 30 (right) dimensions.

3.4.2 KDS Size for Real-World Data

The NBA data sets [6] include scores in 17 different categories for 4051 basketball players. A higher score indicates better performance in each skill (dimension). The NBA-Career set has data aggregated over each player's entire career, while the NBA-Season set has a separate vector for each season in which a player was active (a total of 21961 vectors). Figure 10 shows the $\mathcal{KDS}$ size as a function of increasing k, for both NBA data sets, confirming a clear exponential rate of growth.

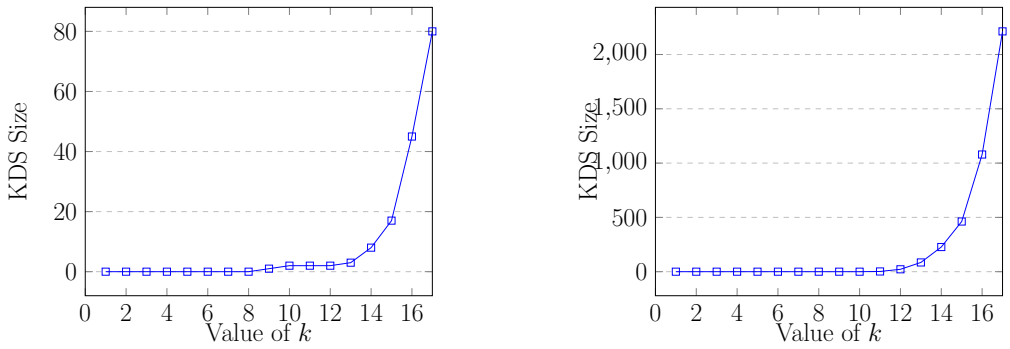

Figure 10: Results for the NBA player data sets, NBA-Career (left) and NBA-Season (right).

Figure 11 shows the results for the Movie Lens data [44, 29], in which each entry is a movie rating. We used first $100,000$ and first $1,000,000$ ratings to generate two different data sets. The former (Movie Lens 1) has $n = 1682$ distinct movies with $d = 943$ distinct reviewers, and the latter (Movie Lens 2) has $n = 3952$ movies with $d = 6040$ reviewers. (Each movie is rated on the scale of 1 (worst) to 5 (best), and we use the default value of 3 (average) for all blank entries.)

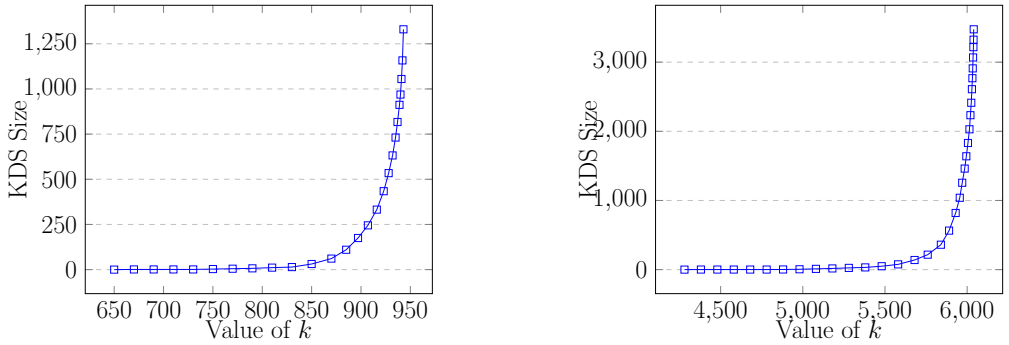

Figure 11: Results for Movie Lens 1 (left) and Movie Lens 2 (right) data sets.

Finally, the following figure shows the results for the remaining four data sets: wine quality, human activity, Internet ads, and particle signal.

In summary, across a diverse collection of data sets, the k-dominant skylines follow the exponential curve shown by our average case analysis, suggesting that the model of random data with attribute distinctness and statistical independence is potentially useful

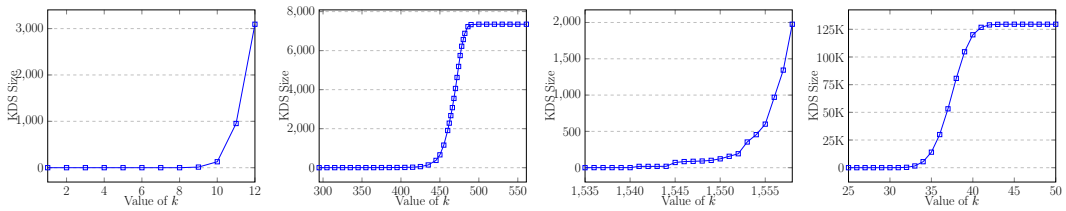

Figure 12: Results for wine quality, human activity, Internet Ads, and particle signal data sets.

in practice.

3.5 Application of KDS in Ranking of High-Dimensional Vectors

In multi-criteria decision problems, there is typically no single best answer and often too many incomparable answers are possible. For instance, in the NBA basketball data, almost 1500 players are *undominated*, and thus arguably a contender for *the* best player. One approach, exemplified by the top-k operator in databases, is to define a *utility function* that aggregates the user preferences across multiple dimensions, for instance, as a linear combination. Formulating an appropriate utility function, however, is quite challenging because it requires users to accurately quantify their preferences, and also requires different dimensions to be similar in scale. (In Section 3.5.3, we also compare simple aggregation-type rankings with the $\mathcal{KDS}$-based ranking.) The alternative approach of *skylines* uses the easier-to-apply principle of pareto optimality—no rational user prefers a solution that is dominated by another on all dimensions—but is stymied by the explosion of skyline size in higher dimensions.

The $\mathcal{KDS}$ suggests a natural approach for *pruning* the skyline using the control knob of parameter k, based on the following insight: many vectors may be undominated in high dimensions, but some are undominated only because they score highly in one or few dimensions, while others are undominated because they score highly in most of the

dimensions. Intuitively, the second kind are the more significant ones. The $\mathcal{KDS}$ based approach automatically selects the vectors that remain undominated on most dimensions, and thus appear on the *early* skyline when k is small. Using the asymptotic expected growth rate of $\mathcal{KDS}$ (cf. Theorem 3.2.1), a user can *control* the skyline to a desired size with parameter k.

In order to test this hypothesis, we carried out the following experiment. Suppose we consider a *random subset of dimensions* for the data, compute the $\mathcal{KDS}$ of the reduced set, and repeat this trial multiple times. How similar are the $\mathcal{KDS}$ in these trials? Intuitively, if $\mathcal{KDS}$ vectors lacked any intrinsic significance, then these skylines should not have much in common. On the other hand, if $\mathcal{KDS}$ vectors were fundamentally dominant, then they would persist in many subsets of dimensions, and show a high Jaccard index of similarity. We chose the Movie Lens and NBA data sets for these experiments because their "top vectors" are familiar names, and easy to interpret.

3.5.1 Top Movies and NBA Players

For the movies, we use the Movie Lens 2 data set, and drop about 10% of the dimensions at random, obtaining a set with 5426 dimensions out of the original 6040. We chose $k = 4800$, for which we are assured to get a small $\mathcal{KDS}$ size, based on Figure 11. We repeated this experiment 5 times, producing 5 different k-dominant skyline sets. The resulting skylines had size between 27 and 29. We then counted how many times each $\mathcal{KDS}$ vector (movie) appeared among these 5 skylines. Figure 13 shows the result: remarkably, the top 27 movies are common to all five $\mathcal{KDS}$ sets. Arguably, all of these movies have claims to be considered "top fan favorites," demonstrating the consistency and utility of $\mathcal{KDS}$ as an automatic data exploration tool.

For the basketball data, we use the NBA-Season data where we randomly dropped 2 of the 17 dimensions, and chose $k = 12$. We performed 5 trials of this experiment, and counted how many times each $\mathcal{KDS}$ vector appears among these 5 skylines. Figure 14

shows the results. Once again, most of the names found by the algorithm are all-time greats.

3.5.2 Jaccard Similarity

Among the five random trials of the Movie Lens data, the smallest $\mathcal{KDS}$ had 27 vectors, and the largest one had 29 vectors. To quantify the pairwise similarity among these 5 sets, we use the Jaccard similarity measure, which is defined as $\frac{\|A \cap B\|}{\|A \cup B\|}$, for two sets A and B. The Jaccard index of 1 indicates a perfect match. As the following table shows, the similarity index is between 0.93 and 1 in all cases. Among the five random trials of the NBA-Season data, the smallest $\mathcal{KDS}$ for $k = 12$ had 49 vectors, and the largest one had 68 vectors. The table below shows the Jaccard index of similarity among these 5 sets.

3.5.3 KDS vs. Aggregation-based Ranking

As a point of comparison, we now discuss some pitfalls suffered by alternative ranking methods based on simple utility functions. We use the movie database because the results are easier to interpret. We recall that in this dataset each vector represents a movie, with each dimension being one user's ratings on a 1–5 scale. The movie data, however, is incomplete since not all users rate all movies, and so we use the following two (natural) scoring functions to compare different vectors.

1. the weighted average of each movie's ratings, and

2. the raw sum of each movie's ratings.

The first method tends to highly rank those movies that have high scores but *very few ratings*. In fact, none of the movies ranked in the top 10 would be considered popular—each had an average score of 5 but rated by at most five users. Our second method corrects for this bias, and steers the ranking towards more popular movies by summing

Movie	Occur
Toy Story	5
The Usual Suspects	5
Braveheart	5
Star Wars: Ep. IV	5
Pulp Fiction	5
Shawshank Redemption	5
Forrest Gump	5
Schindler's List	5
Terminator 2	5
Silence of the Lambs	5
Fargo	5
The Godfather	5
Casablanca	5
One Flew Over Cuckoo's Nest	5
Star Wars: Episode V	5
The Princess Bride	5
Raiders of the Lost Ark	5
Groundhog Day	5
Back to the Future	5
L.A. Confidential	5
Saving Private Ryan	5
Shakespeare in Love	5
The Matrix	5
The Sixth Sense	5
American Beauty	5
Being John Malkovich	5
Gladiator	5

Figure 13: Occurrence frequency of top movies in 5 random trials of $\mathcal{KDS}$

NBA Player	Occur
Wilt Chamberlain 1961	5
Artis Gilmore 1974	5
Bob Mcadoo 1974	5
George Mcginnis 1974	5
K. Abdul-jabbar 1975	5
Julius Erving 1975	5
Artis Gilmore 1975	5
Moses Malone 1978	5
Michael Jordan 1984	5
Michael Jordan 1986	5
Charles Barkley 1987	5
Michael Jordan 1987	5
Michael Jordan 1988	5
Karl Malone 1988	5
Hakeem Olajuwon 1988	5
Karl Malone 1989	5
David Robinson 1990	5
Hakeem Olajuwon 1992	5
Shaquille O'neal 1993	5
David Robinson 1993	5
Dwight Howard 2007	5
Allen Iverson 2007	5
LeBron James 2007	5
Al Jefferson 2007	5
Amare Stoudemire 2007	5
Dwight Howard 2008	5
Dwyane Wade 2008	5
Kevin Durant 2009	5
Dwight Howard 2009	5
David Lee 2009	5
Charles Barkley 1987	4
Karl Malone 1988	4

Figure 14: Occurrence frequency of top NBA players in 5 random trials

	1	2	3	4	5
1	1	0.931	0.966	0.966	0.931
2	0.931	1	0.964	0.964	1
3	0.966	0.964	1	1	0.964
4	0.966	0.964	1	1	0.964
5	0.931	1	0.964	0.964	1

Figure 15: Jaccard Similarity for top movies

	1	2	3	4	5
1	1	0.603	0.716	0.638	0.776
2	0.603	1	0.459	0.493	0.568
3	0.716	0.459	1	0.718	0.658
4	0.638	0.493	0.718	1	0.585
5	0.776	0.568	0.658	0.585	1

Figure 16: Jaccard Similarity for top NBA players

the scores of all the users for each movie. However, this method leads to movies that are widely known but not necessarily top rated candidates for many users. For instance, the original 3 Star Wars movies appeared in the top 4, however the lowest had an average rating of roughly 4/5, which is significantly lower than that of many movies appearing much later in the list. The NBA data set reveals another problem with aggregation-based ranking: different dimensions have vastly different scales, making it difficult to combine them into a single meaningful utility function.

By comparison, as the preceding experiments have shown, the $\mathcal{KDS}$-based ranking gives sensible and robust results without the need for a domain-specific and difficult to formulate utility function. When the full skyline has far too many points, the tunable parameter k of the $\mathcal{KDS}$ automatically acts as a proxy for the importance of skyline vectors: the earlier a point appears on $\mathcal{KDS}$ the more significant it is.

4 Concluding Remarks

In this work we studied three computational problems - team formation, lottery paths, and k-dominance.

Our team formation work introduces a simple and natural model for multi-dimensional multi-team formation, and shows that computing optimal teams is NP-hard even in moderate dimensions. We show that the problem of forming multiple teams optimally can be efficiently solved in two dimensions, as is the problem of forming a single team in any dimension $d = O(\log n)$. The problem of forming multiple teams in higher than two dimension, either exactly or approximately, remains an interesting open problem.

Secondly, we considered the *lottery path* problem, in which an agent wishes to find the most cost effective path to a Bernoulli prize. We presented a polynomial time algorithm for computing a constant factor approximation of the optimal path. We also showed that if the search space is a one-dimensional line, or a directed acyclic graph, then the optimal path can be computed quite efficiently.

Lastly, we made both theoretical and applied contributions to the study of k-dominance in multidimensional data. On the theoretical front, we derived an upper bound on the average case complexity of $\mathcal{KDS}$, which generalizes a classical result of Bentley et al. [8]. We also showed that while k-dominance does not satisfy transitivity, one can still compute the $\mathcal{KDS}$ in roughly the same time as the conventional skyline (worst-case sub-quadratic) as long as the dimension is $d = O(\log n / \log \log n)$. Our experiments show that the size of $\mathcal{KDS}$ in many real-world multi-dimensional data sets follows the same exponential trend of our average case analysis, suggesting that our theoretical bounds can be helpful predictors in practice. Finally, we validate the usefulness of k-dominant skylines as a tool for selecting a small set of top-ranked vectors when the full skyline contains far too many.

References

[1] P. Agarwal, N. Kumar, S. Sintos, and S. Suri. Efficient algorithms for k-regret minimizing sets. In *Proc. of 16th Int. Symp. on Experimental Algorithms*, 2017, to appear.

[2] A. Archer, M. Bateni, M. Hajiaghayi, and H. J. Karloff. Improved approximation algorithms for prize-collecting Steiner tree and TSP. *SIAM J. Comput.*, 40(2):309–332, 2011.

[3] G. Ausiello, S. Leonardi, and A. Marchetti-Spaccamela. On salesmen, repairmen, spiders, and other traveling agents. In *Proc. of CIAC*, pages 1–16, 2000.

[4] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. New approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *SIAM J. Comput.*, 28(1):254–262, 1998.

[5] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[6] Basketball Data. http://databasebasketball.com/.

[7] J. L. Bentley. Multidimensional Divide and Conquer. *Communications of the ACM*, 23(4):214–229, 1980.

[8] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the Average Number of Maxima in a Set of Vectors and Applications. *Journal of the ACM*, 25(4):536–543, 1978.

[9] A. Blum, S. Chawla, D. Karger, T. Lane, A. Meyerson, and M. Minkoff. Approximation algorithms for orienteering and discounted-reward TSP. *SIAM Journal on Computing*, pages 653–670, 2007.

[10] Bryan L. Bonner, Michael R. Baumann, Austin K. Lehn, Daisy M. Pierce, and Erin C. Wheeler. Modeling collective choice: decision-making on complex intellective tasks. *European Journal of Social Psychology*, 36(5):617–633, 2006.

[11] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering*, pages 421–430, 2001.

[12] C. Buchta. On the average number of maxima in a set of vectors. *Information Processing Letters*, 33(2):63 – 65, 1989.

[13] C. Y. Chan, H. V. Jagadish, K. L. Tan, A. K. H. Tung, and Z. Zhang. Finding K-dominant Skylines in High Dimensional Space. In *Proceedings of the ACM SIGMOD International Conference on Management*, pages 503–514, 2006.

[14] C. Y. Chan, H. V. Jagadish, K. L. Tan, A. K. H. Tung, and Z. Zhang. On High Dimensional Skylines. In *Proc. 10th International Conference on Extending Database Technology (EDBT)*, pages 478–495, 2006.

[15] C. Chekuri, N. Korula, and M. Pál. Improved algorithms for orienteering and related problems. *ACM Trans. Algorithms*, 8(3):1–27, 2012.

[16] Shi-Jie Chen and Li Lin. Modeling team member characteristics for the formation of a multifunctional team in concurrent engineering. *IEEE Transactions on Engineering Management*, 51(2):111–124, 2004.

[17] S. Chester and I. Assent. Explanations for Skyline Query Results. In *Proc. International Conference on Extending Database Technology (EDBT)*, pages 349–360, 2015.

[18] S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides. Computing k-regret Minimizing Sets. *PVLDB*, 7(5):389–400, 2014.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 2nd edition*. MIT Press, McGraw-Hill Book Company, 2000.

[20] Chad Crawford, Zenefa Rahaman, and Sandip Sen. Evaluating the efficiency of robust team formation algorithms. In *Autonomous Agents and Multiagent Systems*, pages 14–29, Cham, 2016. Springer International Publishing.

[21] UCI Data Repository. http://archive.ics.uci.edu/ml/.

[22] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.

[23] Christoph Dorn and Schahram Dustdar. Composing near-optimal expert teams: A trade-off between skills and connectivity. In *On the Move to Meaningful Internet Systems: OTM 2010*, pages 472–489, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[24] Erin L. Fitzpatrick and Ronald G. Askin. Forming effective worker teams with multi-functional skill requirements. *Computers & Industrial Engineering*, 48(3):593 – 608, 2005.

[25] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.

[26] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[27] P. Godfrey. Skyline cardinality for relational processing. In *Proc. Foundations of Information and Knowledge Systems (FoIKS)*, pages 78–97, 2004.

[28] P. Godfrey, R. Shipley, and J. Gryz. Algorithms and analyses for maximal vector computation. *VLDB J.*, 16(1):5–28, 2007.

[29] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5(4):1–19, 2015.

[30] H. K. Hwang, T. H. Tsai, and W. M. Chen. Threshold phenomena in k-dominant skylines of random samples. *SIAM Journal on Computing*, 42(2):405–441, 2013.

[31] Larson J.R., Jr. *In search of synergy: In small group performance.* Psychology Press, Taylor & Francis, New York, 2010.

[32] Jon Kleinberg and Maithra Raghu. Team performance with test scores. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, EC '15, pages 511–528, 2015.

[33] V. Koltun and C. H. Papadimitriou. Approximately Dominating Representatives. *Theoretical Computer Science*, 371(3):148–154, 2007.

[34] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB)*, pages 275–286, 2002.

[35] E. Koutsoupias, C. H. Papadimitriou, and M. Yannakakis. Searching a fixed graph. In *Proc. of ICALP*, pages 280–289, 1996.

[36] H. T. Kung, F. L., and F. P. Preparata. On Finding the Maxima of a Set of Vectors. *Journal of the ACM*, 22(4):469–476, 1975.

[37] Theodoros Lappas, Kun Liu, and Evimaria Terzi. Finding a team of experts in social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 467–476, 2009.

[38] Patrick R. Laughlin and Andrea B. Hollingshead. A theory of collective induction. *Organizational Behavior and Human Decision Processes*, 61(1):94 – 107, 1995.

[39] C. Li and M. Shan. Team formation for generalized tasks in expertise social networks. In *2010 IEEE Second International Conference on Social Computing*, pages 9–16, Aug 2010.

[40] Somchaya Liemhetcharat and Manuela Veloso. Modeling and learning synergy for team formation with heterogeneous agents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '12, 2012.

[41] Somchaya Liemhetcharat and Manuela Veloso. Weighted synergy graphs for effective team formation with heterogeneous ad hoc agents. *Artificial Intelligence*, 208:41 – 65, 2014.

[42] Yang Lu, Jiakui Zhao, Lijun Chen, Bin Cui, and Dongqing Yang. Effective skyline cardinality estimation on data streams. In *19th International Conference on Database and Expert Systems Applications*, pages 241–254, 2008.

[43] J. Matousek. Computing dominances in E^n. *Information Processing Letters*, 38(5):277–278, 1991.

[44] Movie Lens Data. http://grouplens.org/datasets/movielens/.

[45] D. Nanongkai, A. D. Sarma, A. Lall, R. J. Lipton, and J. Xu. Regret-Minimizing Representative Databases. *PVLDB*, 3(1):1114–1124, 2010.

[46] E. Nikolova and D. R. Karger. Route planning under uncertainty: The Canadian traveller problem. In *Proc. of AAAI*, pages 969–974, 2008.

[47] Scott Page. *The Difference: How the Power of Diversity Creates Better Groups, Firms, Schools, and Societies*. Princeton University Press, 2007.

[48] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *Proc. ACM SIGMOD*, pages 467–478, 2003.

[49] C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theor. Comput. Sci.*, 84(1):127–150, 1991.

[50] P. Peng and R. C-W. Wong. Geometry approach for k-regret query. In *Proc. International Conference on Data Engineering (ICDE)*, pages 772–783, 2014.

[51] F. P. Preparata and M. I. Shamos. *Computational Geometry—An Introduction*. Springer, 1985.

[52] Habibur Rahman, Senjuti Basu Roy, Saravanan Thirumuruganathan, Sihem Amer-Yahia, and Gautam Das. Optimized group formation for solving collaborative tasks. *The VLDB Journal*, 28(1):1–23, February 2019.

[53] Tuomas Sandholm, Kate Larson, Martin Andersson, Onn Shehory, and Fernando Tohmé. Coalition structure generation with worst case guarantees. *Artif. Intell.*, 111(1-2):209–238, 1999.

[54] Travis C. Service and Julie A. Adams. Coalition formation for task allocation: theory and algorithms. *Autonomous Agents and Multi-Agent Systems*, 22(2):225–248, Mar 2011.

[55] Marjorie E. Shaw. A comparison of individuals and small groups in the rational solution of complex problems. *The American Journal of Psychology*, 44(3):491–504, 1932.

[56] I. D. Steiner. *Group process and productivity*. New York: Academic Press, 1972.

[57] E. Tiakas, A. N. Papadopoulos, and Y. Manolopoulos. On estimating the maximum domination value and the skyline cardinality of multi-dimensional data sets. *Int. J. Knowledge-Based Organ.*, 3(4):61–83, 2013.

[58] M. van Ee and R. Sitters. Approximation and complexity of multi-target graph search and the Canadian traveler problem. *Theor. Comput. Sci.*, 732:14–25, 2018.

[59] Anita Williams Woolley, Christopher F. Chabris, Alex Pentland, Nada Hashmi, and Thomas W. Malone. Evidence for a collective intelligence factor in the performance of human groups. *Science*, 330(6004):686–688, 2010.