Exploring Pros and Cons of Ranked Entities with COMPETE*

Kiril Panev
TU Kaiserslautern
Kaiserslautern, Germany
panev@cs.uni-kl.de

Sebastian Michel
TU Kaiserslautern
Kaiserslautern, Germany
michel@cs.uni-kl.de

ABSTRACT

We present COMPETE, a novel approach to explore data using rankings. Utilizing rankings, which succinctly summarize the relative performance of entities, is a very intuitive way to inspecting underlying data. In this work we present an approach where users can understand the dominance of entities and interactively explore data. The approach harnesses diverse precomputed rankings, capturing many different aspects of possible user interest. For a given set of input entities, COMPETE identifies entities that are dominating or are dominated by the input, thus expanding on the relative performance of the input and giving focus on other related entities which are always better (or worse) than the input. We consider several aspects of the dominance relationship and provide different approaches that reflect these nuances. We report on the results of an experimental evaluation over data obtained from the Internet Movie Database (IMDb).

1 INTRODUCTION

In this paper we propose the usage of rankings to explore entity-centric data. Consider a user Alice who wants to buy a car. Alice does not know too much about cars, but there are a couple of models that she likes how they look, say a BMW 118i and an Audi A3. She wants to find all rankings that contain both models as one of the top-performing entities. Thus, she can learn what are the constraints and ranking criteria that make these cars as one of the best. Moreover, she is interested in which other car models outperform her (so far) favorites in all the lists where they appear, i.e., the car models that are *dominating* her input. Considering the rankings in Figure 1, COMPETE identifies Mercedes A220 as a model that outperforms the two input models across all rankings in the dataset. Thus, Alice learns that this model is always better than her favorites and that she should probably consider it as a potential one for buying. Moreover, she can examine what are the constraints and ranking criteria where the Mercedes is better, for instance, in cars of type = Hatchback and ranked by power. She also wants to see which car models are *dominated* by these two and she gets the Citroen, Ford, and Chrysler models as being dominated by the input. Since her favorites are always ranked better than these three models, she can disregard them as potential models of interest. Alice might also want to relax the conditions a bit and

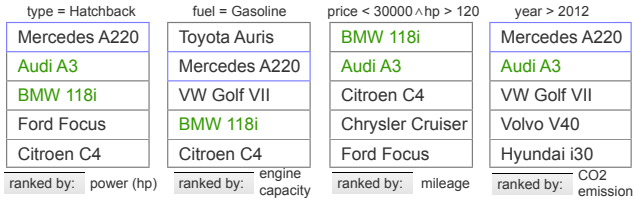


Figure 1: Example of dominance between ranked entities

is interested to find cars that are dominating and dominated by her input models in a certain fraction of the lists, say what are the models that outperform the BMW and the Audi in at least 70% of the top-k lists. These rankings are already generated and indexed, the system maintains a set of diverse rankings that cover various aspects of car categories and performance.

In this paper, we investigate the dominance relationship between ranked entities and develop algorithms to explore data summarized in rankings by computing dominant entities.

1.1 Problem Statement

We consider a set T of rankings τ_i. Each ranking has a domain D_{τ_i} of items it contains. The rankings are of fixed size k, i.e., $|D_{\tau_i}| = k$, but we investigate the impact of different choices of k on the system performance. None of the rankings contains any duplicate items and there are no ties between items. The rank of an entity e in a ranking τ is given as $\tau(e)$. For each ranking τ_i, the constraints and ranking criterion which characterize the ranking list is also maintained.

Given a set of input entities I, the objective in this paper is to efficiently and effectively determine all entities e_j that dominate, i.e., are always ranked higher than the entities in I across all lists where they co-occur together. In other words, we are looking for the set of entities:

$$\hat{D}(I) = \{e_j | \forall e \in I, \forall \tau_i \; s.t. \; (e, e_j) \in \tau_i : \; \tau_i(e_j) < \tau_i(e)\}$$

We are also interested in finding the set of entities $\check{D} = \{e_j\}$ that are dominated, i.e., ranked lower than those in the input I in all lists they appear together, i.e., where $\tau_i(e) < \tau_i(e_j)$.

The problem can be further extended by relaxing the criterion of the entities dominating the input from across all common lists to a certain portion of the lists. Thus, given a user-specified threshold θ, dominating entities are to be considered those that are ranked higher than the input in more than θ of the common rankings. We elaborate on what are the consequences and how we adjust our methods by having this relaxed criterion.

For simplicity reasons, in the remainder of the paper we will consider only finding the entities that are dominating I; however, all methods can be applied for also finding the dominated entities. We define the dominance relationship more formally in Section 4.

*This work has been supported by the German Research Foundation (DFG) under grant MI 1794/1-1 and grant MI 1794/1-2.

ExploreDB 2018 , June 15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-5847-7/18/06…$15.00
https://doi.org/10.1145/3214708.3214709

The problem of finding dominating entities interactively requires efficient computation and scans over potentially a very large number of rankings. Hence, we consider suitable index structures and develop algorithms that make use of early pruning, compare them to baseline methods and discuss on how the pruning reduces data access and affects the efficiency of computation.

1.2 Contributions and Outline

With this paper we make the following contributions:

- We present COMPETE, an efficient, effective, and extensible approach for exploring data through ranked lists that computes dominating entities.
- We define the entity dominance relationship in a set of rankings and consider the different types of dominance that this can incur given a set of input entities.
- We derive methods for finding dominating entities and consider the impact of introducing pruning and intermediate result sharing on the computation efficiency.
- We report on the results on an extensive experimental evaluation using data from the IMDb [1] dataset.

The remainder of the paper is organized as follows: Section 2 discusses related work. Section 3 presents background information on rankings. Section 4 presents the approach and the nuances of the entity dominance relationship. Section 5 describes the baseline algorithms for computing dominant entities and proposes using pruning and result sharing as optimizations. Section 6 reports on the results of the experimental evaluation and Section 7 concludes the paper.

2 RELATED WORK

Research in data exploration considers efficiently extracting knowledge from potentially unfamiliar data. Idreos et al. [8] provide an overview of past and emerging research in the area. An approach for result-driven recommendation is presented by Drosou and Pitoura [5] which proposes additional items to the users based on sets of attribute values. In [4], Dimitriadou et al. present a framework that iteratively steers users towards interesting data areas by leveraging relevance feedback on data samples. A prominent exploration technique is faceted search (e.g., [9, 15]) where the results are classified into categories (facets), and the user refines them by selecting one or multiple facets. Bast et al. [2] present a system to explore Freebase by guiding users in creating a query that retrieves a list of entities of interest, not knowing the underlying schemata. Psallidas et al. [13] and Shen et al. [14] focus on discovering queries given example tuples, while Panev et al. [11] demonstrate a system focused on list oriented querying, where example top-k lists as user input are reverse engineered into SQL queries with aggregation. The dominance relation between items is correlated and similarly defined by Khalefa et al. [10] when looking for skylines over incomplete data, however, skyline queries compute all entities that are not dominated by any other entity, which differs from our goal. In computing dominance in rankings, the reference point are the entities in the input and the ranking lists where they appear. Dynamic skyline queries [12] consider dynamic attributes of data objects to be computed by a set of dimension functions, however they consider complete data across all dimensions, and still look for the skyline using the dynamic coordinates. To the best of our knowledge, there has not been any work that consider dominance of ranked entities for investigating data.

τ_1	τ_2	τ_3	τ_4	τ_5
e_1	e_3	e_2	e_1	e_4
e_4	e_7	e_6	e_2	e_2
e_2	e_1	e_3	e_3	e_5
e_6	e_6	e_5	e_6	e_6
e_8	e_8	e_4	e_4	e_3

Figure 2: Sample set of rankings

3 BACKGROUND ON RANKINGS

Complete rankings are regarded to be permutations over a fixed domain D. Following the notation and references by Fagin et al. [6], a permutation σ is defined as a bijection from the domain $D = D_\sigma$ onto the set $[n] = \{1, \ldots, n\}$. The value $\sigma(i)$ describes the rank of an element i in the permutation σ. An element i is said to be ranked higher of an element j in σ if $\sigma(i) < \sigma(j)$.

We consider incomplete rankings called top-k lists [6]. Formally, a top-k list τ is a bijection from D_τ onto $|k| = \{1, \ldots, k\}$. It is important to note that two individual top-k lists, e.g., τ_1 and τ_2, do not necessarily share the same domain, i.e., $D_{\tau_1} \neq D_{\tau_2}$. Moreover, for each top-k list the constraints and ranking criteria that were used for creating it are also available, as shown in Figure 1. For instance, a ranking list can contain the car models of type Hatchback ranked by horse power. This is the valuable information that is shown to the user when exploring the data in the system. We consider positive rankings, higher ranked entities are regarded as better.

Rankings can be indexed with traditional inverted indices [7] that keep for each entity a list of rankings that contain the entity. The inverted index as a structure allows for efficiently identifying those rankings that have one or more entities in common with the input entities. The key point of using inverted indices is their ability to efficiently reduce the global amount of all rankings to potential candidates by eliminating the rankings that do not contain the input entities. A partial example inverted index following the sample rankings depicted in Figure 2 is shown below:

entity $e_1 \longrightarrow < (\tau_1), (\tau_2), (\tau_4) >$

entity $e_3 \longrightarrow < (\tau_2), (\tau_3), (\tau_4), (\tau_5) >$

Looking up in the inverted index can reveal that the entity e_1 appears only in the rankings τ_1, τ_2, and τ_4.

4 THE COMPETE APPROACH

Our data exploration approach of finding entity dominance is based on utilizing a set of precomputed rankings and for a given set of input entities produces as result the entities that dominate the input. We consider the subtle distinctions of defining the dominance relationship given a set of input entities I and a set of ranking lists T. Following the definition of the dominance relationship on computing skyline queries over incomplete data [10], we define:

Definition 4.1. **Dominance**
We say an entity e_a dominates an entity e_b, denoted as $e_a \prec e_b$ in a set of ranking lists T iff the following two conditions hold: (i) for all ranking lists $\tau_i \in T$ that contain both e_a and e_b, $\tau_i(e_a) < \tau_i(e_b)$; and (ii) for all other ranking lists $\tau_j \in T$, $j \neq i$, either $e_i \notin \tau_j$ or $e_j \notin \tau_j$.

We denote e_a as the dominating entity, e_b as the dominated entity, and the rankings τ_i that contain both as common rankings

of e_a and e_b. Intuitively, the dominating entity e_a is ranked higher than the dominated one e_b in all their common ranking lists. The absence of, say e_b in a ranking list which contains e_a can mean that e_b is not selected with the selection predicates that were used in creating τ_i and this is the assumption that we make in this work. Alternatively it can mean that it is selected by the same predicate but ranked lower in τ_i, i.e., $\tau_i(e_b) > k$, where k is the size of the ranking list.

Definition 4.2. **Support of Two Entities**
The support $s(e_a, e_b)$ of two entities e_a and e_b is the number of rankings τ_i where both are contained.

$$s(e_a, e_b) = |\{\tau_i \mid (e_a, e_b) \in \tau_i\}|$$

It is important to note that an entity e_a can dominate an entity e_b only if their support $s(e_a, e_b) > 0$. The incompleteness of top-k rankings entails that two entities may not co-occur in any of the lists. On the other hand, high support means that the entities are highly related, they share categorical attributes and are top-performers for various ranking criteria.

Consider the sample set of rankings presented in Figure 2. The entity e_1 is said to dominate e_2, since in their common rankings, τ_1 and τ_4, e_1 is ranked higher than e_2: $\tau_1(e_1) < \tau_1(e_2)$ and $\tau_4(e_1) < \tau_4(e_2)$. The support of these entities is $s(e_1, e_2) = 2$, since they co-occur in two ranking lists together. Similarly, e_2 dominates e_3. An important property of the dominance relation in incomplete data is the loss of the transitive dominance property [10]. Thus, when comparing e_1 and e_3 we see that neither one dominates the other, since $\tau_2(e_3) < \tau_2(e_1)$ and $\tau_4(e_1) < \tau_4(e_3)$.

Following Definition 4.1, we define two types of dominance of an entity e_a over a set of entities I, **strict** and **loose** dominance.

Definition 4.3. **Strict Dominance**
An entity e_a strictly dominates a set of entities I, denoted as $e_a \trianglelefteq I$ iff e_a dominates all $e_i \in I$, i.e., $\forall e_i \in I : \ e_a \prec e_i$.

Definition 4.4. **Loose Dominance**
An entity e_a loosely dominates a set of entities I, denoted as $e_a \triangleleft I$ iff e_a dominates at least one entity $e_i \in I$ and for the other entities $e_j \in I, \ i \neq j$, the support $s(e_a, e_j) = 0$.

Furthermore, if an entity e_a strictly dominates the set I, then it also loosely dominates I, i.e., it holds that $e_a \trianglelefteq I \Rightarrow e_a \triangleleft I$, but not the other way around. A strictly dominating entity co-occurs with each and every one of the entities in the input set and is dominating all of them. However, a loosely dominating entity does not need to co-occur with all the entities in I, but it dominates the entities from I with which it co-occurs in the rankings.

Let us consider the input set $I = \{e_5, e_6\}$ and the sample rankings in Figure 2. We find that the entity e_2 strictly dominates I, since it dominates both e_5 and e_6. On the other hand, it is said that e_1, e_2, and e_7 loosely dominate I, considering that e_1 and e_7 dominate e_6 and $s(e_5, e_1) = s(e_5, e_7) = 0$. Note that if, say $s(e_5, e_7) > 0$, but e_7 is ranked lower than e_5 in some ranking list, thus is not dominating e_5, then e_7 will not be part of the loosely dominating entities, even though it dominates e_6. Loose dominance is important when the input contains multiple entities. For instance, if $|I| = 5$ and a dominating entity, say e_a is dominating 4 of the 5 entities in the input, then e_a would probably still be an interesting result to the user, however will not qualify as a result with strict dominance.

In order to be able to quantify the dominance of an entity we introduce the following definition:

Definition 4.5. **Degree and Fraction of Dominance**
The degree of dominance $d(e_a, e_b)$ of an entity e_a over entity e_b is defined as the number of rankings where e_a dominates e_b.

$$d(e_a, e_b) = |\{\tau_i \mid \tau_i(e_a) < \tau_i(e_b)\}|$$

The fraction of dominance $f(e_a, e_b)$ of an entity e_a over another entity e_b is defined as the degree of dominance $d(e_a, e_b)$ over their support $s(e_a, e_b)$.

$$f(e_a, e_b) = \frac{d(e_a, e_b)}{s(e_a, e_b)}$$

Now, let e_a and e_b be two highly co-occuring entities with $s(e_a, e_b) = 100$ where e_a is ranked higher than e_b in 95 of the 100 rankings. Intuitively, e_a is *mostly* dominating e_b and the user would be interested to consider it as a result but according to Definition 4.1 it will not be returned as one of the dominating entities. Thus, we consider not being so stringent and allow some flexibility in looking for dominating entities. The fraction of dominance allows for further insight on how two entities relate to each other in terms of performance. The highest fraction of dominance amounts to 1 and that is when e_a is ranked higher than e_b (or vice-versa) in all lists where they co-occur. Note that, $f(e_a, e_b) = 1 - f(e_b, e_a)$. Using the fraction of dominance, we define:

Definition 4.6. **Partial Dominance**
We say an entity e_a partially dominates an entity e_b given a threshold θ, denoted as $e_a \prec_\theta e_b$ iff $f(e_a, e_b) > \theta$.

The definition for partial dominance allows for some leniency in computing dominating entities, however θ should still be chosen reasonably high. Thus, following the example above and having $\theta = 0.9$, we say $e_a \prec_\theta e_b$, since $f(e_a, e_b) = 0.95 > \theta$.

We combine the partial dominance with the previous definitions of strict and loose dominance and we get four variations of the dominance relationship given a set of input entities I, each with different characteristics: strict, loose, strict-partial, and loose-partial dominance.

5 ALGORITHMS

We developed methods that, for a given set of input entities I and a set of rankings T, compute the set of dominating $\hat{D}$ and dominated entities $\check{D}$ according to the different dominance relationships presented in Section 4. For each of the input entities $e \in I$ the system gets the list of rankings that contain e by looking up in the inverted index. These rankings are then processed and a set of dominating entities is computed as result. Note that for reasons of brevity we only show the details of finding the dominating entities; similar steps are taken to compute the set of dominated entities as well.

Finding Strict Dominance. When computing the result entities that strictly dominate the input I, each of the entities in the result needs to dominate each of the entities I. Hence, the algorithm finds the dominating entities for each input entity $e \in I$ and then intersect their dominating entities to find those who dominate all. This method is presented in Algorithm 1.

The idea behind the algorithm is to go through the rankings τ for each input entity e and put its co-occuring entities therein into two sets: those that are ranked higher in candidate dominating $\hat{C}_e$, and those that are ranked lower in candidate dominated $\check{C}_e$ (Lines 5–6 in Algorithm 1). To this purpose, we first find the position of e in τ, which is not expensive since top-k lists are by definition relatively

method: **findStrictDominance**

```
1  for each e ∈ I
2      rList_e := I.getPostingList(e)
3      for each τ ∈ rList_e
4          τ(e) = τ.getPosition(e)
5          add all e_i s.t. τ(e) > τ(e_i) to Ĉ_e
6          add all e_j s.t. τ(e) < τ(e_j) to Č_e
7      D̂_e = Ĉ_e \ Č_e
8  D̂ = ⋂_{e∈I} D̂_e
9  return D̂
```

Algorithm 1: Finding strict dominance

short. Then, the dominating entities are those who occur as ranked higher than e, but never appeared as ranked lower than e (Line 7). Finally, intersecting the dominating entities of each $e \in I$ results in the entities that dominates all input entities.

Finding Loose Dominance. The entity e_a that loosely dominates the input set I, dominates some of the entities $e_i \in I$ and does not have any common rankings with the remaining entities $e_j \in I, i \neq j$, i.e. $s(e_a, e_j) = 0$. Thus, computing loose dominance differs to strict dominance in two key steps. First, in computing the dominating entities for each individual $e \in I$, it also needs to eliminate the candidate dominated entities for the remaining input entities. Otherwise, there can be false positives in the dominating entities, for instance, consider an entity e_a that dominates $e_i \in I$, but for some of the remaining input entities, say e_j, $s(e_a, e_j) > 0$ and e_a is not dominating e_j. Thus, $\check{C} = \bigcup_{e \in I} \check{C}_e$ is created that contains all candidate dominated entities. Then, $\hat{D}_e = \hat{C}_e \setminus \check{C}$ will contain the entities that dominate a single entity e and never occur as ranked lower than any of the remaining input entities. The second difference to computing strict dominance is that the final result $\hat{D} = \bigcup_{e \in I} D_e$, since a loosely dominating entity does not need to have common lists with each input entity.

Finding Partial Dominance. Finding partial dominance allows for some leniency in computing dominating entities and a partially dominating entity needs to be ranked higher than an entity in more than θ fraction of their common lists. In **strict-partial** dominance, an entity needs to have common rankings with and dominate all entities from the input in more than a θ fraction of the lists. The approach is similar to the one presented in Algorithm 1, however, instead of classifying the appearance of being ranked higher or lower of the co-occuring entities e_j of each input entity e, some bookkeeping of counting the actual number of times $\tau(e_j) < \tau(e)$ (and vice-versa) is necessary. This bookkeeping is later used to compute the fraction of dominance of each e_j and checked whether $f(e_j, e) > \theta$. In this way, the method computes the set $\hat{D}_e$ containing the partially dominating entities for each $e \in I$. Then, the strict-partially dominating entities are computed as $\hat{D} = \bigcap_{e \in I} \hat{D}_e$.

Loose-partial dominance requires the same criterion as loose dominance regarding the number of input entities dominated, hence similar steps for avoiding false positives need to be taken. Hence, in addition of $\hat{D}_e = \{e_j \mid f(e_j, e) > \theta\}$, a set $F = \{e_i \mid f(e_i, e) < \theta\}$ containing potential false positives needs to be maintained. The entities in F are then used to filter out the false positives with $\hat{D}_e = \hat{D}_e \setminus F$, since those might appear as partially dominating for certain input entities, but do not have common lists with some of the remaining entities in I.

5.1 Efficiency Optimizations

The methods presented in Section 5 are agnostic to the input entities and do not consider sharing intermediate computation. For instance, entities that might have already been discarded as not dominating for a certain input entity $e \in I$, are still considered as candidates for the remaining ones in I. Furthermore, with strict dominance, the sets of dominating entities $\hat{D}_e$ are computed individually and these are later intersected to get the final result, even though it is necessary an entity to dominate all of the input entities. This needless processing diminishes the efficiency of the system.

Computation can be shared across all input entities in I, thus reducing the data access and computation runtime. We use the different nuances of each of the dominance relationships and propose two optimizations that improve the efficiency of the algorithms.

Prune and Blacklist. We consider introducing a prune-and-blacklist strategy that eliminates entities early and avoids their re-processing by placing them in a blacklist. Using this approach, while processing the rankings for an input entity e, the entities e_i that are observed as being ranked both higher and lower than e, i.e., cannot be part of the result, are blacklisted and pruned. The entities placed in the blacklist are skipped in the remainder of the processing. This strategy is used for all methods. Algorithm 2 shows the optimized method for computing strict dominance. The prune-and-blacklist strategy is implemented in Lines 6–12 and is used similarly in improving the method for loose dominance. In computing non-partial dominance, a single occurrence of an entity e_i as being ranked lower than e leads to pruning e_i from the results and its blacklisting.

Additional statistics are necessary in order to use pruning in the methods for finding **partial dominance**. In partial dominance, an entity e_i can still be part of the dominating result even if it is ranked lower than an input entity e in multiple rankings and the actual number of rankings depends on the input threshold θ. Thus, to compute this threshold number of rankings, the support $s(e, e_i)$ needs to be known. We compute the support of all possible entity pairs in a pre-processing step and use it for computing the threshold number of rankings where an entity e_i can be ranked lower than an input entity e in partial dominance:

$$\text{threshold}_\theta = s(e, e_i) - \theta \times s(e, e_i)$$

In computing partial dominance, an entity e_i is pruned and blacklisted once the number of rankings where $\tau(e) < \tau(e_i)$ reaches threshold_θ. Note that threshold_θ is different for each entities pair (e, e_i). Significant computational overhead is saved by the early pruning and blacklisting of these entities in all methods.

Result Sharing. The methods for finding strict and strict-partial dominance can additionally benefit from sharing the intermediate results computed in previous steps. Therefore, we find the dominating entities for one input entity e and then consider only those in each subsequent step to check whether they also dominate the remaining input entities in I, similarly to TAAT query processing [3] in search engines. Moreover, the input entities are processed in ascending order of their support, starting with the rarest entity e which occurs in the lowest number of rankings, since a small initial

method: **findStrictDominance+**

```
1   order e ∈ I in ascending order of s(e)
2   /* start with the rarest e and compute dominance */
3   rList_e := I.getPostingList(e)
4   for each τ ∈ rList_e
5       τ(e) = τ.getPosition(e)
6       for each e_i ∈ τ
7         if blacklist.contains(e_i) then skip e_i
8         if τ(e) > τ(e_i) then
9           if Ď.contains(e_i) then
10            prune and blacklist e_i
11          else if ¬D̂.contains(e_i) then add e_i to D̂
12        /* same is done for τ(e) < τ(e_i) but for Ď */
13  for each remaining e in ascending order of s(e)
14      rList_e := I.getPostingList(e)
15      for each τ ∈ rList_e
16        τ(e) = τ.getPosition(e)
17        for each e_i ∈ τ
18          skip e_i if not already a result
19          if τ(e) > τ(e_i) ∧ Ď.contains(e_i) then
20            prune e_i from D̂
21  return D̂
```

Algorithm 2: Optimized strict dominance

result-set is desirable. For each of the remaining $e \in I$, the methods consider only the result entities that were computed in the previous step, as shown in Lines 13–20 of Algorithm 2. Note that $\hat{D}$ will get smaller with the processing of each additional $e \in I$. Thus, with processing more rankings τ, the number of potential results in $\hat{D}$ decreases, which enables avoiding unnecessary computation. Once finished processing all $e \in I$, $\hat{D}$ will contain the entities that strictly dominate the input I.

The methods for loose and loose-partial dominance cannot benefit from the result-sharing optimization, since a dominating entity e_i does not need to dominate all $e \in I$. Thus, the individual result-sets $\hat{D}_e$ for each $e \in I$ need to be computed separately and cannot start from an initial result as in finding strict dominance. Nevertheless, the prune-and-blacklist strategy still manages to avoid significant computational overhead.

6 EXPERIMENTAL EVALUATION

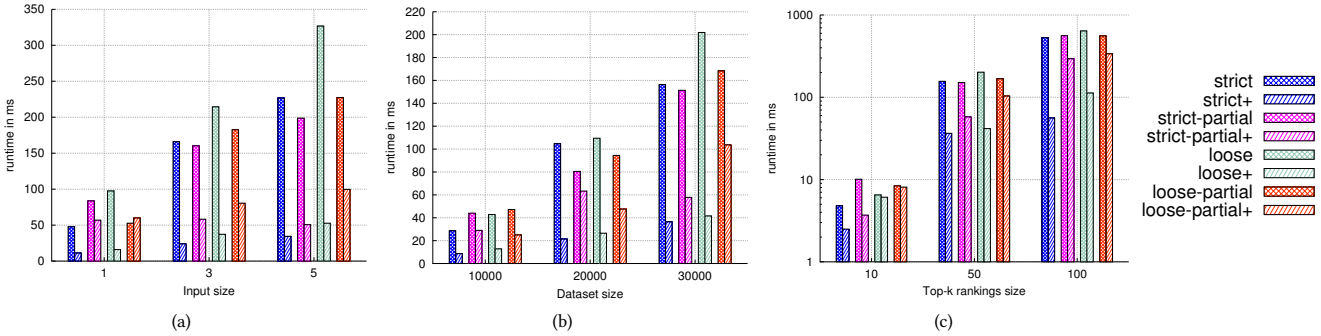
Experiments were conducted on a 2× Intel Xeon 6-core machine, with 128GB RAM, running Ubuntu 14.04 as an operating system and using Java 1.8 (limited to 8GB memory). On system start-up, the dataset, index structures, and statistics are loaded into main memory, thus we examine the performance of all methods as in-memory approaches. We focus on measuring the performance and distinction in results of the methods for computing different type of dominance and use the methods presented in Section 5 as baseline methods. Depending on the dominance relationship computed we present results from the following four methods: strict, loose, strict-partial, and loose-partial. Furthermore, we show the efficiency benefits from the optimizations techniques presented in Section 5.1. We add a plus sign '+' at the end of each optimized method, for instance strict+.

Datasets and Workload. We evaluate our approach of computing entity dominance for a given set of input entities using data from the IMDb dataset [1]. We created 30,000 rankings with top-k size 100 (using the LIMIT clause) by executing (meaningful) queries with different constraints in the WHERE clause and using a variety of aggregation functions as ranking criteria. There are 101 distinct atomic predicates which were combined to create predicates of size 1 to 5, while 13 different ranking criteria were used. For instance, year = 2015, genre = 'Drama', and max(rating) are sample predicates and ranking criterion. These 30,000 rankings contain 8,850 distinct entities, with a median entity occurrence being 147.5 rankings. In order to examine the effects of the dataset size n and top-k rankings size, by choosing subsets of the created ranking lists we created experiments with $n \in \{10000, 20000, 30000\}$ and the top-$k \in \{10, 20, 30, 50, 100\}$. Unless otherwise specified, we report on average results from the dataset with 30,000 top-k rankings of size 50, and $\theta = 0.8$ for the methods with partial dominance.

As workload, we created sets of input entities by using popular actors and actresses. In order to examine the result size and performance of our methods in regard to the number of input entities, we varied the number of the entities in a set. Thus, we created 6 sets containing 5 entities each and removed entities from them to create sets with smaller size, resulting in total of 30 input sets containing 1 to 5 entities.

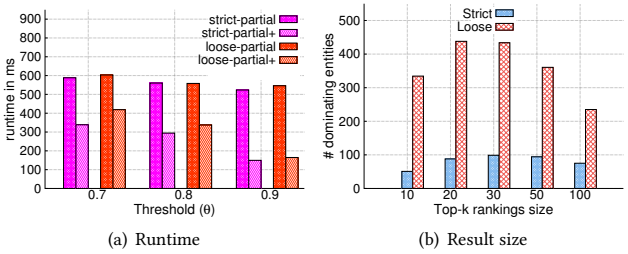
Efficiency. We study the efficiency of the different methods for the various dominance relationships and the proposed optimizations techniques. Figure 3(a) shows the average execution time for each method and its optimized counterpart with different input size. We observe that each of the optimized methods significantly outperforms its baseline equivalent. As expected, the runtime increases with larger input size, however this is much smaller with the optimized methods. For instance, the average execution time for strict with input size 1 and 5, amounts to 47.7 and 226.7 milliseconds, while with strict+ equals to 11.3 and 34.5 milliseconds. Computing loose dominance is more expensive than computing strict dominance. This is due to the larger number of entities that potentially can be part of the result with loose dominance. We observe that computing partial dominance is slower than non-partial dominance. There are more results with partial dominance and potential dominating entities take longer to be blacklisted, considering that they need to be ranked better than the input only in a fraction of their common rankings. The optimizations reduce the number of entities processed, therefore the runtime is reduced.

We examine the execution time of our methods with different dataset size and different size of the top-k rankings in the dataset. Figure 3(b) depicts the execution time when varying the number of rankings in the dataset. As expected, the execution time increases with larger dataset. However, for the optimized algorithms the increase is not as significant as with the baseline algorithms. The most significant performance gain is with the largest dataset of 30,000 rankings. The performance of our methods with the dataset with 30,000 rankings when varying the size of the top-k lists is shown in Figure 3(c). We observe that having larger top-k rankings leads to slower computation time across all methods and the largest gain from the optimization strategies are with $k = 100$. We study the runtime for computing partial dominance with different threshold size θ, as presented in Figure 4(a). We observe that smaller θ reduces the performance of all methods. Smaller θ means being



(a) (b) (c)

Figure 3: Comparing the runtime of the baseline and optimized methods with different (a) input size, (b) dataset and (c) top-k rankings size



(a) Runtime (b) Result size

Figure 4: Comparing the runtime and result size with different threshold θ and top-k size

Method	Input size $\|I\|$	#entities	#constraints	#rank-criteria
strict	1	421.8	45.8	13.0
	2	41.5	42.3	12.8
	3	5.8	17.0	9.3
	4	1.5	7.8	3.5
	5	0.3	2.7	1.3
loose	1	421.8	45.8	13.0
	2	398.0	54.3	13.0
	3	343.8	55.8	13.0
	4	319.7	57.0	13.0
	5	319.8	62.7	13.0

Table 1: Results discovered with different input size

less restrictive when computing results, thus leading to processing more entities and slower performance.

Results Discovered. Our system discovers dominating entities characterized by various constraints and ranking criteria, thus providing valuable insight about the input entities. We study the effect of the size of the input and the size of the top-k lists over the number of dominating entities discovered. Table 1 shows the average number of dominating entities, as well as the average number of distinct atomic constraints and ranking criteria found in the ranking lists which dominate the input entities. With strict dominance, as expected, we observed that the number of results is decreasing with larger input size, since each of the results need to co-occur with each of the input entities. The number of discovered dominating entities with loose dominance is also decreasing with larger input size, however since with loose dominance a dominating entity does not need to co-occur with all of the input, the decrease is very mild.

Figure 4(b) show the average number of result entities with different top-k size. We observe that for strict dominance, the number of dominating entities increases with larger k and peaks for $k = 30$ before starting to decrease with $k = 50$ and $k = 100$. The decrease of number of results is due to the higher chance of eliminating a dominating entity with larger top-k lists, considering that with longer lists the probability of occurring lower than the input is higher. We observe similar behavior with loose dominance.

7 CONCLUSION AND OUTLOOK

We described COMPETE, an approach that enables access to and understanding of outstanding entities by computing a ranking-induced dominance relationship between them. We first defined the notion of dominance in different flavors using the position of competing entities in rankings, and generalized this for sets of input entities. The core computational problem was optimized through index structures and early result eviction, demonstrating clear gains in the experimental evaluation over IMDb data. Ultimately, besides computing dominating entities, due to the sheer volume of possible outcomes, a way to order results based on user-perceived relevance is our current ongoing work.

REFERENCES

[1] Internet Movie Database. http://www.imdb.com
[2] Bast et al. Easy access to the freebase dataset. *WWW*, 2014
[3] Büttcher et al. Information Retrieval - Implementing and Evaluating Search Engines. *MIT Press*, 2010
[4] Dimitriadou et al. Explore-by-example: an automatic query steering framework for interactive data exploration. *SIGMOD*, 2014
[5] Drosou and Pitoura. YmalDB: exploring relational databases via result-driven recommendations. *VLDB J.*, 2013
[6] Fagin et al. Comparing Top k Lists. *SIAM J. Discrete Math.*, 2003
[7] Helmer and Moerkotte. A performance study of four index structures for set-valued attributes of low cardinality. *VLDB J.* pages, 2003
[8] Idreos et al. Overview of Data Exploration Techniques. *SIGMOD*, 2015
[9] Kashyap et al. FACeTOR: cost-driven exploration of faceted query results. *CIKM*, 2010
[10] Khalefa et al. Skyline Query Processing for Incomplete Data. *ICDE*, 2008
[11] Panev et al. Exploring Databases via Reverse Engineering Ranking Queries with PALEO. *PVLDB*, 2016
[12] Papadias et al. An Optimal and Progressive Algorithm for Skyline Queries. *SIGMOD*, 2003
[13] Psallidas et al. S4: Top-k Spreadsheet-Style Search for Query Discovery. *SIGMOD*, 2015
[14] Shen et al. Discovering queries based on example tuples. *SIGMOD*, 2014
[15] Tzitzikas and Papadakos. Interactive Exploration of Multi-Dimensional and Hierarchical Information Spaces with Real-Time Preference Elicitation. *Fundam. Inform.*, 2013