Skyline Queries over Knowledge Graphs

Ilkcan Keles[0000−0002−1424−5223] and Katja Hose[0000−0001−7025−8099]

Aalborg University, Aalborg, Denmark
{ilkcan,khose}@cs.aau.dk

Abstract. With the continuously growing amount of data offered in the form of knowledge graphs, users are often overwhelmed by the amount of potentially relevant information and entities. Hence, helping users find relevant data is a problem that becomes more and more important. Skyline queries are typically used in multi-criteria decision making applications to find a set of objects that are of interest to a user. This type of queries has been extensively studied over relational data in the database community. But only little attention has yet been paid to investigating if and how the skyline principle can help identifying sets of interesting entities in knowledge graphs. In this paper, we therefore show how the skyline principle can be applied to RDF knowledge graphs and help the user find interesting entities. In particular, we present algorithms using commonly used standard interfaces for accessing RDF data and a lightweight extension of existing interfaces (SkyTPF) to process skyline queries. Our experiments show that the proposed algorithms enable efficient and scalable skyline query processing over knowledge graphs.

1 Introduction

More and more knowledge graphs are becoming available in different fields. Whereas some knowledge graphs, such as DBpedia [15] and YAGO [9], offer factual information about real-world entities, others are used by private companies. For querying publicly accessible knowledge graphs, there are two widely accepted interfaces: SPARQL endpoints and Triple Pattern Fragments (TPF) [24]. SPARQL endpoints typically use indexes and advanced query planning techniques to enable efficient query processing. On the other hand, the drawback is that the server running the SPARQL endpoint handles all the query processing load whereas the client that sends the query simply waits for the result. Hence, when many clients access an endpoint concurrently, it is likely to have problems regarding performance and throughput. TPF [24] has been proposed to reduce the server load by assigning more tasks to the client. To achieve this, a TPF server is only capable of processing queries consisting of a single triple pattern instead of complex SPARQL queries. In order to process SPARQL queries, the client has to take care of the remaining query processing tasks, such as query planning, filtering, and joins.

Even though these interfaces provide means of querying knowledge graphs, it is often very difficult for users to find the entities that they are interested in.

For instance, assume that a user is doing research on planets, consults DBpedia, and wants to know about planets that have high densities ($?d$) and high average speeds ($?as$). The user is unable to provide a precise scoring function, such as $0.5 * ?d + 0.5 * ?as$, that could be used to rank the planets because he/she has neither any information about the attributes' domains and ranges nor a clear understanding of whether one criterion is more important than the other.

In such use cases, skyline queries [3] can help users find interesting entities without the need to provide a specific weight for each criterion. In other words, a skyline query allows users to simply provide a set of preferences on the attributes of interest and returns a set of "interesting" entities with respect to these preferences. More formally, a skyline set consists of all entities that are not dominated by any other entity; entity e_i dominates entity e_j if e_i is at least as good in all attributes and better in at least one attribute.

Assuming the above mentioned skyline query for planets with preferences on high density and high average speed is evaluated on the dataset illustrated in Fig. 1 (subset of planets in DBpedia), the skyline set consists of Earth and Venus (highlighted in red). Fig. 1 also illustrates the dominance region of the planets in the skyline. Intuitively, with the given preferences a skyline point is clearly "preferable" over any planet contained in the dominance region – we therefore refer to the latter as being dominated.

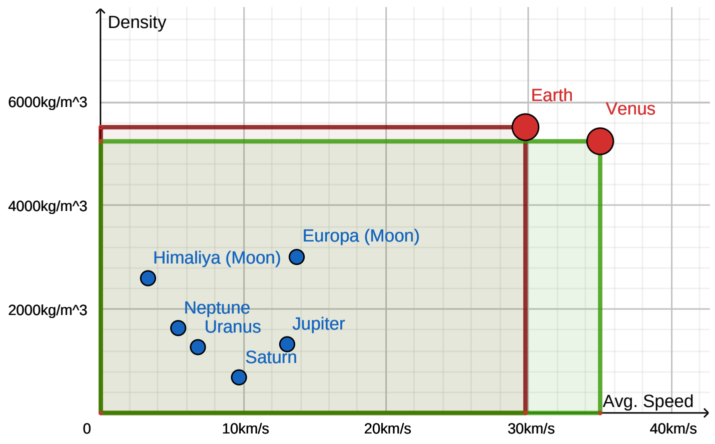

Fig. 1. Skyline of Planets

Efficient processing of skyline queries over relational databases [1–3, 6, 13, 18, 19, 22, 27] has been extensively studied. However, there is only very little related work [5, 29] on processing skyline queries over knowledge graphs. These works assume direct control over the data source and how the data is stored. We, on the other hand, aim at supporting skyline computation over standard and lightweight interfaces that we do not control and that are not restricted to skyline queries only.

In this paper, we propose methods for skyline query computation over standard interfaces, such as SPARQL endpoints and TPF. In addition, we propose the SkyTPF interface, which extends TPF and aims at making skyline query processing more efficient. Finally, we provide an extensive experimental evaluation of the proposed methods. The experiments show that the proposed methods enable efficient skyline query processing over knowledge graphs. In summary, our main contributions are: (i) methods to enable skyline query processing over standard RDF interfaces (SPARQL endpoints and TPF), (ii) SkyTPF; a TPF-like interface on the server-side and a client-side algorithm to optimize and process skyline queries, (iii) an extensive experimental evaluation of these methods.

The remainder of this paper is organized as follows. Section 2 discusses related work, and Section 3 introduces background and definitions. While Section 4 presents how to process skylines over standard interfaces (SPARQL endpoints and TPF), Section 5 presents the SkyTPF interface as well as a client-side algorithm for skyline computation. Finally, Section 6 presents our evaluation, and Section 7 concludes the paper.

2 Related Work

The general problem of computing skyline queries was first introduced by Kung et al. [14] as the maximum vector problem in the field of computational geometry. The term skyline was first introduced by Börzsönyi et al. for relational databases [3]. Since then skyline queries have attracted significant interest in the database community as they enable finding interesting data objects in consideration of multiple preference criteria. To compute skyline queries over relational databases, Börzsönyi et al. [3] propose the Block Nested Loops (BNL) and Divide and Conquer (D&C) algorithms. BNL scans the database in several rounds and compares each data object with the current set of candidate skyline objects. If a data object p is not dominated by any of the current skyline candidates, p is added to that set and, if necessary, all data objects dominated by p are removed. The D&C algorithm is based on the divide and conquer principle. So, it first partitions a dataset into several subsets, computes the skyline in each of the subsets, and then combines the partial results by checking them for mutual dominance. The SFS algorithm [6] improves BNL by presorting the data objects with respect to a monotone function. Bartolini et al. [2] propose the SaLSa algorithm to eliminate the need to scan all the data objects in the database. All these algorithms target skyline computation over relational data, where all attributes of a data item are combined in a single relational tuple. In knowledge graphs, however, joins are required to obtain the attribute values. Whereas the BNL principle of comparing all data items against each other is of course applicable in arbitrary setups, the proposed optimizations are tailored towards relational data that is available locally and cannot be applied to our problem scenario.

The literature proposes a variety of algorithms [1,13,18,19,22,27] that focus on skyline computation over vertically partitioned data and for multi-relational settings where joins are required. Balke et al. [1] propose algorithms (BDS and IDS) to compute skylines over vertically partitioned data where each server hosts a single skyline attribute. The BDS algorithm uses sorted access until a pivot data object is reached at all servers. Then, other attributes of the data objects that are better than the pivot object in at least one attribute are obtained by random access from the servers. The IDS algorithm improves BDS by using a combination of sorted and random access to reach the pivot object. Jin et al. [13] introduce the multi-relation skyline operator. Vlachou et al. [27] propose the sort-first skyline join (SFSJ) algorithm that provides an early termination condition and makes it possible to compute skyline objects progressively. Trimponias et al. [22] address the skyline computation problem in a setting where each server has a disjoint set of attributes together with a record ID. Their algorithm is able to support any decomposition of the attributes among the servers. Nagendra et al. [18, 19] improve the state-of-the-art algorithms by applying region based pruning before skyline computation. In our case, we assume that we access the knowledge graph via standard interfaces – potentially on remote servers. Unfortunately, these interfaces do not provide sorted access and random access at the same time, which renders the above mentioned techniques inapplicable. There are also numerous approaches on horizontally partitioned data in P2P systems [10–12, 26, 28]. These systems differ substantially from our setup as the data is typically located on remote peers in a distance of several hops without knowing which peers exactly provide relevant data for a given query. In this paper, however, we assume a direct connection to the relevant server.

There are a couple of studies [5, 20, 21, 23, 29] on incorporating preference-based querying over knowledge graphs. Some studies [20, 21, 23] propose ways of extending SPARQL for expressing preferences. Siberski et al. [21] provide a proof-of-concept implementation based on BNL. We also use the BNL algorithm as a straightforward solution for skyline computation over existing interfaces. Troumpoukis et al. [23] compare the performance of NL, BNL, and query rewriting. Schneider et al. [20] claim the extension for preferences can be efficiently integrated into SPARQL endpoints by using union-find algorithms. We do not focus on extending SPARQL in this paper. Chen et al. [5] focus on skyline computation on knowledge graphs that are stored as vertically partitioned relations. They use a header point (pivot object) to prune non-skyline entities. However, as our goal is to enable skyline query processing using existing interfaces, we cannot make assumptions on how the data is stored. For this reason, the algorithms proposed in [5] are not applicable to our case. Zheng et al. [29] propose subgraph skyline analysis over knowledge graphs. However, their methods require precomputing additional bit-string encodings for each vertex and edge. Again, as we do not assume that we have control over how the knowledge graph is stored, this approach is not applicable to our problem scenario. In addition, we do not want to increase the load on the server too much by pushing more work to the server for skyline computation since that would reduce throughput and therefore counteract the idea of TPF.

In addition to skyline querying over knowledge graphs, Lukasiewicz et al. [16] extend Datalog+/- with preferences and propose algorithms to execute preference queries over ontologies [16]. These queries are defined using first order logic formulas that allow including more general preferences than skyline queries.

However, preference queries on ontologies are out of scope of this paper.

3 Preliminaries

RDF [4] is a standard data format that is widely used to represent information on the Web; its basic building block is a triple. A triple is defined as a 3-tuple $t = \langle s, p, o \rangle$, where s, p, and o correspond to *subject*, *predicate* and *object*, respectively. The subject of a triple identifies an entity and is either an IRI or a blank node. The predicate is an IRI representing the relation between subject and object. And the object can be an IRI, a literal, or a blank node. A knowledge graph is then defined as a set of triples. In this paper, we assume that the knowledge graph does not contain blank nodes because blank nodes do not represent any entities.

Definition 1 (Triple Pattern and Basic Graph Pattern). *Let I, L, and V be the pairwise disjoint sets of IRIs, literals, and variables. A triple pattern then is an element of $(I \cup L \cup V) \times (I \cup V) \times (I \cup L \cup V)$. We say that a triple $t = \langle s, p, o \rangle$ is a matching triple for a triple pattern $tp = \langle s_{tp}, p_{tp}, o_{tp} \rangle$ or t satisfies tp if $(s_{tp} = s \vee s_{tp} \in V) \wedge (p_{tp} = p \vee p_{tp} \in V) \wedge (o_{tp} = o \vee o_{tp} \in V)$. The solution to a triple pattern is a mapping μ from V to $I \cup L$, i.e., the possible mappings for each variable in the triple pattern. A basic graph pattern (BGP) is a set of triple patterns. The solution to a BGP is the mappings obtained by joining the solution mappings of each triple pattern included in the BGP.*

After defining the basics of knowledge graphs, let us now define skyline sets and skyline queries.

Definition 2 (Skyline Set and Dominance). *Given a dataset D of n-dimensional data objects, $o_i \in D$ dominates $o_j \in D$ $(o_i \succ o_j)$ if o_i is better than or equal to o_j in all n dimensions and is strictly better than o_j in at least one dimension with respect to user-defined preference functions. The skyline set $S = \{o \mid o \in D \wedge \nexists e (e \in D \wedge e \succ o)\}$ consists of the objects that are not dominated by any other object. If two objects o_i and o_j do not have a dominance relation between them, we say that o_i and o_j are not comparable.*

In knowledge graphs, data objects correspond to real-life entities, such as people, cities, and countries. A skyline query over a knowledge graph is then defined as follows.

Definition 3 (Skyline Query over Knowledge Graphs). *Given a knowledge graph K, a skyline query is defined as a pair $q = \langle BGP, SV \rangle$, where BGP is a basic graph pattern and SV is a set of pairs, each of which is a skyline variable with its corresponding preference function (`MIN`, `MAX`). Skyline variables are a subset of variables included in the basic graph pattern. The result of a skyline query is the skyline set of the solutions to BGP computed with respect to the skyline variables and the preference functions. The variables of the skyline query that the preferences are defined on are called dimension variables. A query with two dimension variables is then called a 2-dimensional skyline query.*

For instance, $SV = \{\langle ?v_1, \texttt{MIN}\rangle, \langle ?v_2, \texttt{MAX}\rangle\}$ means that the user is interested in a skyline over $?v_1$ and $?v_2$, and `MIN` and `MAX` will be used to determine dominance between the solution mappings.

In line with [5], we use an extended version of SPARQL to express a skyline query. The extension contains a **SKYLINE** keyword together with **MIN** and **MAX** keywords to be able to express preference functions. The query for the motivational example is illustrated in Listing 1.1.

```
SELECT ?planet ?as ?d
WHERE
{ ?planet rdf:type dbpedia:Planet .                    #11826 triples
  ?planet dbpedia:averageSpeed ?as .                     #739 triples
  ?planet dbpedia:density ?d .                           #146 triples
}
SKYLINE OF ?as MAX, ?d MAX
```

Listing 1.1. Example Skyline Query

4 Skylines over Standard Interfaces

In this section, we propose client-side algorithms for processing skyline queries over knowledge graphs using standard interfaces (SPARQL endpoints and Triple Pattern Fragments) that are commonly used to provide access to knowledge graphs on the Web.

4.1 SPARQL Endpoint

SPARQL endpoints are widely used to query knowledge graphs on the Web using the SPARQL query language [7]. As SPARQL does not cover skyline queries, such servers do not directly support them. Hence, we propose a client-side algorithm that computes skylines while only sending standard SPARQL queries to the server (SPARQL endpoint).

The client-side algorithm first retrieves the solution mappings μ for the basic graph pattern of the query ($q.BGP$) from the SPARQL endpoint. Then, we compute the skyline over μ in a block-nested-loop fashion, i.e., we maintain a skyline set S and scan μ sequentially. When a mapping m is read from μ, the algorithm checks whether m is dominated by any mapping currently in S. If so, the algorithm continues with the next mapping. Otherwise, the mappings that are dominated by m are removed from S, and m is added to S. After iterating over all mappings in μ, S corresponds to the skyline set and it represents the output.

Example. Let us assume that Table 1 corresponds to the output mappings for the BGP of the example query from Listing 1.1. Saturn is read first and added to the skyline set S. Europa (Moon) is considered next. Since it dominates Saturn, it is added to S and Saturn is removed from S. Neptune, Jupiter, and Uranus are considered next consecutively but they are not added to S since they are dominated by Europa (Moon). Next, Earth is added to S since it dominates Europa (Moon) and the latter is removed from S. Venus is also added to S since

Table 1. Output Mappings for our Example Query

?planet	?**as**(km/s)	?**d**(kg/m^3)
Saturn	9.69	687
Europa (Moon)	13.74	3010
Neptune	5.43	1638
Jupiter	13.07	1326
Uranus	6.81	1270
Earth	29.78	5515
Venus	35.02	5243
Himalia (Moon)	3.312	2600

it is incomparable with Earth: Venus has a higher average speed but Earth has a higher density. When Himalia (Moon) is read, it is not added to S since it is dominated by Earth. The final skyline set is then $S = \{\texttt{Earth}, \texttt{Venus}\}$.

4.2 Triple Pattern Fragments (TPF)

The TPF interface [25] was proposed to increase availability and throughput of the servers by reducing their computational load. A server hosting a SPARQL endpoint has to perform all the tasks that are related to query processing such as query planning, executing joins, and filtering operations included in the query. However, a TPF server is designed only to process triple pattern requests in a paged manner. A TPF request contains a single triple pattern and a page number. The response to a TPF request is a page containing a set of matching triples together with metadata containing an estimation of the total number of matching triples. Hence, a TPF server only returns the matching triples for an input triple pattern without having to invest extensive resources on query processing. For this reason, the workload of TPF servers is much lower than of SPARQL endpoints. On the other hand, the workload at the client is considerably higher.

Our client-side algorithm builds upon the query processing algorithm proposed by Verborgh et al. [25] to retrieve the solution mappings for the basic graph pattern of a skyline query ($q.BGP$). The algorithm first iterates over the triple patterns of $q.BGP$. At each iteration, the algorithm chooses the most selective triple pattern as the next triple pattern to process. In order to find the most selective triple pattern, the algorithm retrieves the first pages of each triple pattern in $q.BGP$ to obtain the number of matching triples for each triple pattern.

The algorithm then retrieves all matching triples for the most selective triple pattern. The remaining triple patterns are updated with respect to the obtained bindings.

For instance, to process the basic graph pattern of the query given in Listing 1.1, the algorithm first processes the third triple pattern since it has the least number of matching triples (146). If the number of triples per page is 100[1],

[1] recommended page size according to [25]

we need to send two requests for this triple pattern with page numbers set to 1 and 2. After retrieving the triples and initializing the mappings for ?`planet`, the algorithm instantiates 146 triple patterns for each remaining triple pattern by replacing ?`planet` with these mappings. So, for processing the remaining two triple patterns, we have 292 instantiated triple patterns (requests). Once all requests have been successfully processed, the skyline is computed over the output mappings as described in Section 4.1.

4.3 Bindings-restricted Triple Pattern Fragments

As explained in the previous section, the TPF interface was designed to reduce the workload of servers when querying knowledge graphs. However, it leads to a higher client-side workload since clients are responsible for processing joins. Moreover, it also creates a higher network load since a TPF client has to send a high number of requests to process a query. In order to address these drawbacks, Hartig et al. [8] proposed the Bindings-restricted Triple Pattern Fragments (brTPF) interface. The key consideration is that the request sent to the server does not only contain a triple pattern but also a set of mappings originating from intermediate results computed at the client, which are used to prune the matches of the triple pattern. The number of mappings sent together is determined by a parameter (*maxMpR*). In the original paper, this parameter is set to values between 5 to 50 since more than 50 bindings result in "414 (URI too long)" response due to being included in HTTP GET requests [8]. In this paper we set *maxMpR* to 30[2].

Processing skyline queries over standard brTPF interfaces in principle works the same as with TPF interfaces. The only difference is that instead of sending separate requests for each mapping obtained from the first triple pattern, the algorithm sends mappings in groups of 30 in a single request to the brTPF server. This reduces the number of requests to 10 (5 for each remaining triple pattern) instead of 292 for the example query given in Listing 1.1. Once all requests have been successfully processed, the skyline is computed over the retrieved mappings as described in Section 4.1.

5 Skylines over SkyTPF Interface

In this section, we propose an extension of the brTPF interface for skyline query processing and a client-server hybrid algorithm to compute skylines efficiently.

5.1 SkyTPF Interface

The client-side algorithms presented in Section 4 do not take the skyline properties into account while processing the skyline query and therefore return all matching triples regardless of whether the corresponding entities can be part of the skyline or not. However, by taking skyline properties into account, the server can prune the search space and increase efficiency.

[2] recommended value according to [8]

The main idea behind SkyTPF is to use a pivot entity to prune the set of skyline candidates. It is a mapping for the variables in triple patterns processed so far. We add (i) a pivot entity, (ii) a skyline flag, (iii) a skyline variable, and (iv) a skyline preference function to the request sent to the server. If the skyline flag is not set, it is identical to brTPF. Otherwise, the server returns the triples whose corresponding entities are better than or equal to the pivot entity with respect to the given skyline variable and preference function. Since the same pivot entity is used for all the skyline triple patterns, i.e., the triple patterns containing skyline variables, any entity that is not returned from the server cannot be part of the skyline since it is guaranteed that any such entity is dominated by the pivot entity. We therefore propose a minimal extension to the brTPF interface to retain the characteristics of TPF (shifting load to the clients) while still improving skyline query performance.

In line with the definitions of bindings-restricted triple pattern selector and bindings restricted triple pattern fragment collection (Definitions 1 and 2, [8]), we define skyline binding-restricted triple pattern selector and skyline triple pattern fragment as follows:

Definition 4 (Skyline Binding-Restricted Triple Pattern Selector). *A selector is a function that selects triples from a knowledge graph according to the provided input. Given a triple pattern tp, a finite sequence of mappings Ω, a skyline flag sf that is either* 0 *or* 1*, a skyline preference function sp that is either* `MIN` *or* `MAX`*, a pivot entity pe that satisfies tp, a skyline triple pattern selector for tp, Ω, sf, sp and pe, denoted by $s_G(tp, \Omega, sf, sp, pe)$ for a knowledge graph G is defined by Equation 1. In this equation, $m(tp, G, \Omega)$ denotes the matching triples for tp that are compatible with the mappings included in Ω. If $\Omega = \emptyset$, then $m(tp, G, \Omega)$ is simply the set of matching triples.*

$$s_G(tp,\Omega,sf,sp,pe)=\begin{cases} m(tp, G, \Omega) & \text{if } sf=0 \\ \{t \mid t \in m(tp, G, \Omega) \wedge (t \succeq_{sp} pe)\} & \text{if } sf=1 \wedge pe \text{ is set} \end{cases} \quad (1)$$

In Equation 1, $t \succeq_{sp} pe$ denotes that corresponding entity of t either dominates or is equal to pe according to the skyline preference function sp and the skyline variable in tp. As shown in Equation 1, when the skyline flag sf is 0*, the selector function is just a bindings-restricted triple pattern selector.*

Definition 5 (Skyline Triple Pattern Fragment). *A Linked Data Fragment (LDF) is defined as a 5-tuple $f = \langle u, s, \Gamma, M, C\rangle$, where u is a URI that hosts the fragment f, s is a selector function, Γ is the set of triples that are selected with respect to s, M is a finite set of RDF triples that contains metadata regarding f, and C is a finite set of hypermedia controls (Definition* 2*, [25]). A skyline triple pattern fragment (SkyTPF) collection is defined for a given hypermedia control c, and a maxMpR value. A specific LDF collection F is called a SkyTPF collection if there exists one LDF $\langle u, s, \Gamma, M, C\rangle \in F$ for any possible triple pattern tp, any finite sequence of solution mappings Ω with at most maxMpR mappings, any possible skyline flag sf, any possible skyline preference function sp, and any* *possible pivot entity pe with the following conditions: (i) The selector function s is a skyline bindings-restricted triple pattern selector for tp, Ω, sf, sp, pe, (ii) there is a triple* $\langle \texttt{u}, \texttt{void:triples}, \texttt{cnt} \rangle \in M$*, where* `cnt` *represents a cardinality estimate for* Γ *(if* $\Gamma = \emptyset$*, then* $\texttt{cnt} = 0$*), and (iii)* $c \in C$*.*

Implementation The SkyTPF server is implemented using Java and extends the brTPF server implementation provided by Hartig et al. [8]. The implementation is available online[3] and uses RDF-HDT data sources [17].

A SkyTPF server is able to serve multiple data sources as TPF and brTPF. For each data source, the server creates an HDT index file, together with a dictionary-based index that holds the rank of each subject (i.e. entity) for each skyline preference function and for each predicate with a numeric value. To put differently, given a subject URI of an entity, it is possible to get the rank of the entity for a specific predicate and for a specific skyline preference function using the index. When a request is received, the server iterates over the mappings provided in the request to construct the set of triple patterns that will be used to query the data source. For each mapping, the server updates the variables of the input triple pattern according to the mapping and adds the triple pattern to this set. Then, the HDT backend is queried using these triple patterns. The resulting triples are added to the output set if their corresponding entity has a rank at least equal to the pivot entity according to the skyline preference function.

Client-side Query Processing Algorithm The complete algorithm to process skyline queries over SkyTPF interfaces is sketched in Algorithm 1.

The client first decomposes the skyline query into skyline and non-skyline subqueries (lines 1 and 2); the skyline subquery consists of the triple patterns containing skyline attributes and the remaining triple patterns constitute the non-skyline subquery. Then, the non-skyline subquery is processed using the SkyTPF endpoint (line 3); the skyline flag is set to 0 to process this subquery in a brTPF fashion.

Next, the algorithm determines the pivot using the skyline subquery (line 4) – we explain how to choose the pivot entity later. The set of candidate skyline results is populated by sending SkyTPF requests for each triple pattern in the skyline subquery (lines 6–15). Afterwards, the missing mappings for the skyline variables are retrieved for the incomplete mappings included in the candidate set. Finally, the algorithm computes the skyline in a block-nested-loop fashion and returns the output (lines 16 and 17).

We extend our SkyTPF server to return triples in a paged manner when the skyline flag is set and no pivot mapping is provided. The output triples match with the input triple pattern and their corresponding entities are better than remaining entities that have a matching triple according to a skyline attribute and a skyline preference function. The motivation behind this extension is twofold. First, we need to guarantee that the pivot entity has a matching triple for each

[3] https://github.com/ilkcan/skyTPF-server

Algorithm 1 Skyline query processing over SkyTPF interface

```
Input: $q = \langle BGP, SV \rangle$ - a skyline query, $url$ - a SkyTPF endpoint URL
Output: $S$ - the set of skyline entities
 1: $sTPs \leftarrow$ triple patterns that contain a skyline attribute
 2: $nsTPs \leftarrow q.BGP \setminus sTPs$
 3: Process the non-skyline subquery and initialize the set of mappings $\mu$ by sending
    brTPF requests to $url$
 4: $pe$ = DeterminePivot($url, \mu, sTPs$)
 5: $candSkylines = \emptyset$
 6: for all $tp$ in $sTPs$ do
 7:     Retrieve the matching triples by sending a skyTPF request for $tp$ with $pe$ and
    the corresponding skyline preference function and initialize the mappings $\mu_{tp}$ wrt
    the triples
 8:     for all $m$ in $\mu_{tp}$ do
 9:         if A mapping $m_s$ contains the subject mapping of $m$ exists in $candSkylines$
    then
10:             Extend $m_s$ with $m$
11:         else
12:             Add $m$ to $candSkylines$
13:         end if
14:     end for
15: end for
16: Retrieve missing skyline variables of $candSkylines$ by sending brTPF requests
17: Compute the set of skyline entities $S$ by applying BNL algorithm
18: return $S$
```

skyline triple pattern. Second, if a pivot entity is better than most of the entities that have matching triples for all skyline triple patterns, it provides a higher pruning power.

To determine the pivot, we first retrieve the first pages of all skyline triple patterns using this extension. If there is an entity that is present in the outputs of all skyline triple patterns and if this entity is included in the output of the non-skyline subquery, this entity is chosen as the pivot entity. Otherwise, the algorithm continues with the next page.

Example. The complete example for processing a 2-dimensional skyline query using the SkyTPF interface is given in Figure 2. The query is defined as $q.BGP = \{\langle ?id, a_1, ?A_1\rangle, \langle ?id, a_2, ?A_2\rangle\}$ and $q.SV = \{\langle ?A_1, \texttt{MAX}\rangle\ \langle ?A_2, \texttt{MAX}\rangle\}$. For ease of presentation, we leave the non-skyline triple patterns out and assume that the non-skyline output is $O = \{A, B, C, D, E, F, G\}$. In order to determine the pivot entity, the algorithm requests first pages of the triple patterns that include the skyline attributes A_1 and A_2. Since D is part of both pages and is also included in the non-skyline output, it is selected as the pivot entity. Then, the algorithm determines the set of candidates $S_c = \{D, E, F\}$ by sending a SkyTPF request for each attribute and combine the responses. The algorithm then obtains the A_1 values for E and F by sending a brTPF request. Finally, we compute the skyline in a block-nested-loop fashion as described in Section 4

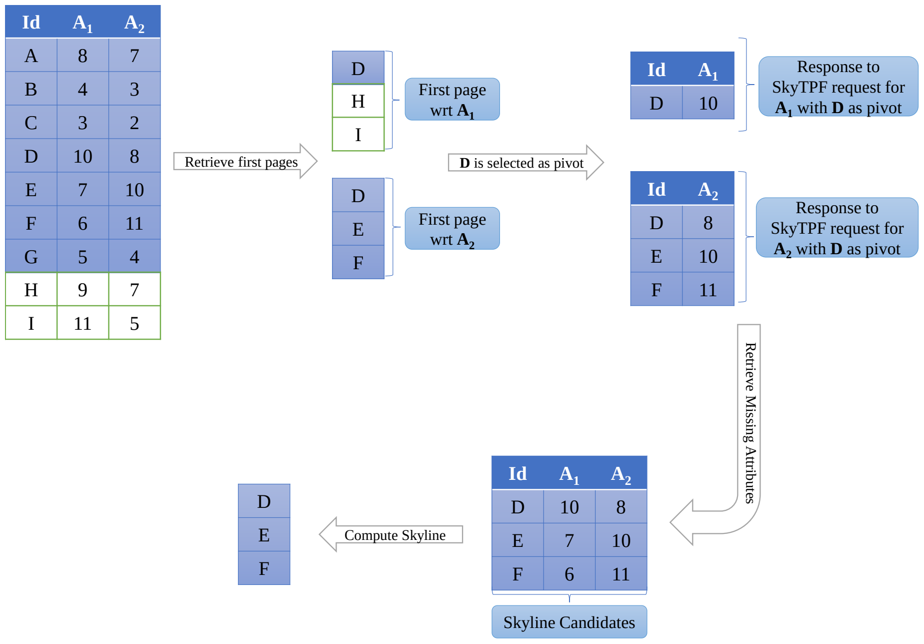

Fig. 2. Example Skyline Computation

and obtain $\{D, E, F\}$ as the output.

5.2 Extensions

The proposed algorithms can be extended to cases where data is horizontally or vertically partitioned. Horizontal partitioning occurs when each server stores all triples of a set of entities and vertical partitioning occurs when each server stores a set of predicates. In the case of horizontal partitioning, the algorithm can be applied on each server to compute the skylines for the server and then a second iteration is needed to compute the global skylines. In the case of vertical partitioning, the client needs to send the requests to the server that contains the predicate included in the triple pattern.

6 Experimental Evaluation

This section discusses the results of our evaluation. We first explain the experimental setup in Section 6.1 and we discuss the evaluation results in Section 6.2.

6.1 Experimental Setup

For the experimental evaluation, we have implemented both single-threaded and multi-threaded versions of the proposed algorithms for TPF, brTPF, and SkyTPF. The multi-threaded versions send HTTP requests in parallel and the number of threads is set to the number of CPUs in the machine. We did not include the SPARQL endpoint based algorithm since it is shown to increase the load on the server significantly compared to TPF and brTPF [8, 25].

We considered comparing our work against a client-side skyline query processing algorithm based on SPARQL query rewriting [23]. However, it has already been shown that BNL outperforms such an algorithm because of expensive not exists and filter clauses [23]; query re-writing performs better in only 1 out of 7 queries. Since we use the BNL algorithm in our client-side query processing algorithms, we do not include a comparison in our evaluation.

Datasets and Queries. We evaluate the proposed methods on synthetic datasets in line with the literature [2, 3, 5, 6] due to a number of reasons. First, when we evaluate the methods on synthetic data, we know the underlying distribution and we are able to see the effect of the underlying distribution on the performance of the methods. Second, we might introduce a bias in favor of one algorithm due to the query selection procedure and due to missing information when we use real datasets. We generated synthetic datasets with independent, correlated, and anti-correlated distributions using the data generator provided by Börzsönyi et al. [3]. In the correlated distribution, if an entity is good in one dimension, it is highly likely that it is good in other dimensions as well. In the anti-correlated distribution, the opposite holds; good in one dimension, bad in another. In the independent distribution, the dimensions follow uniform distribution, so that the probability that an entity is better than another entity with respect to a skyline dimension is independent from their relationship with respect to another skyline dimension. The anti-correlated distribution is the worst scenario for any skyline algorithm since it means that almost every entity is part of the skyline and it is therefore quite difficult to prune the search space. The number of entities in the generated datasets is between $10,000$ and $50,000$, and the number of skyline dimensions is between 2 and 6. A skyline query in our setup contains all the skyline dimensions included in the dataset. If not explicitly mentioned otherwise, the default number of entities is $10,000$ and the default number of skyline variables (dimensions) is 4.

Metrics. To evaluate and compare these approaches, we have measured the number of HTTP requests sent to the server, the number of candidates that the client computes the skylines on, and the query processing time. The number of HTTP requests provides a measure to assess the network load introduced by an algorithm. We present a single value for each method for this metric since multi-threading does not have an effect on it. We decided to include the number of skyline candidates metric in our experimental evaluation to assess the effect of pruning for the SkyTPF-based method.

Configuration. The server is hosted on a virtual machine with 4 2.29 GHZ CPUs and 8GB of main memory and the client is hosted on a virtual machine with 2 2.29 GHZ CPUs and 2GB of main memory.

6.2 Evaluation Results

Figure 3 shows the pruning power of SkyTPF's client algorithm for different dataset distributions. As expected, SkyTPF's pivot based pruning is quite ef-

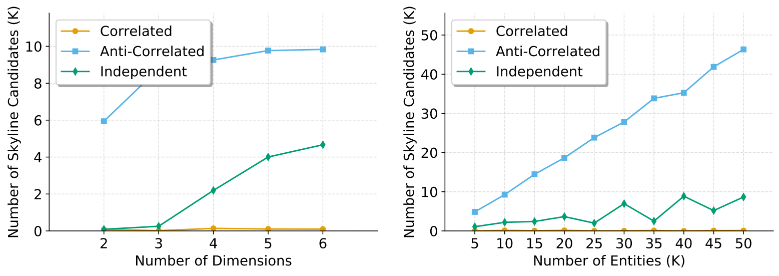

Fig. 3. Pruning power of SkyTPF's client algorithm

fective for the correlated dataset. The number of skyline candidates for the independent dataset is below 50% of the number of entities even for 6 dimensions. Moreover, the number of skyline candidates for the dataset including 50K entities is less than 10K which means that our algorithm manages to prune 80% of the entities. As expected, the figure also shows that the algorithm based on SkyTPF has very low pruning power when the dataset is anti-correlated.

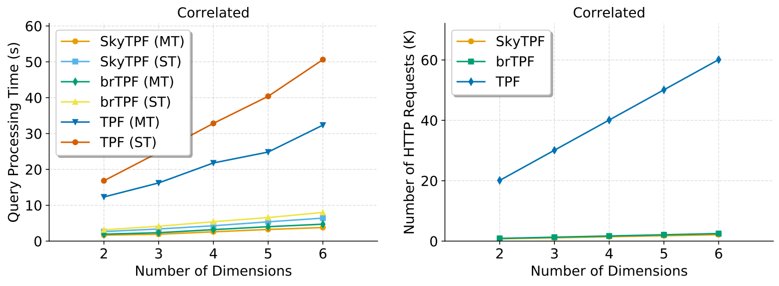

Fig. 4. Effect of the number of dimensions (correlated dataset)

Figure 4 illustrates the effect of the number of dimensions on the proposed algorithms for the correlated dataset. The TPF-based client-side algorithm performs significantly worse than the brTPF-based and SkyTPF-based algorithms. The TPF-based algorithm is between 6–8 times slower and needs at least an order of magnitude more HTTP requests to process skyline queries. To present the performance difference between algorithms more clearly and to demonstrate the results in a finer granularity, we omit our results for TPF.

Figure 5 shows the effect of the number of entities and the number of skyline dimensions on the proposed methods for different data distributions. The performance of the proposed methods becomes worse as the number of entities and the number of skyline variables increase for all distributions. This is expected since the number of dominance checks and the number of variables that the dominance will be checked on increases, respectively. The experimental results also show that the proposed methods benefit from multi-threading since the algorithm can send HTTP requests and parse the responses in parallel.

As shown in Figure 5, SkyTPF performs better than brTPF for the correlated dataset. This is due to the fact that when data is correlated, a good pivot entity provides an effective pruning power. On the other hand, the figure also

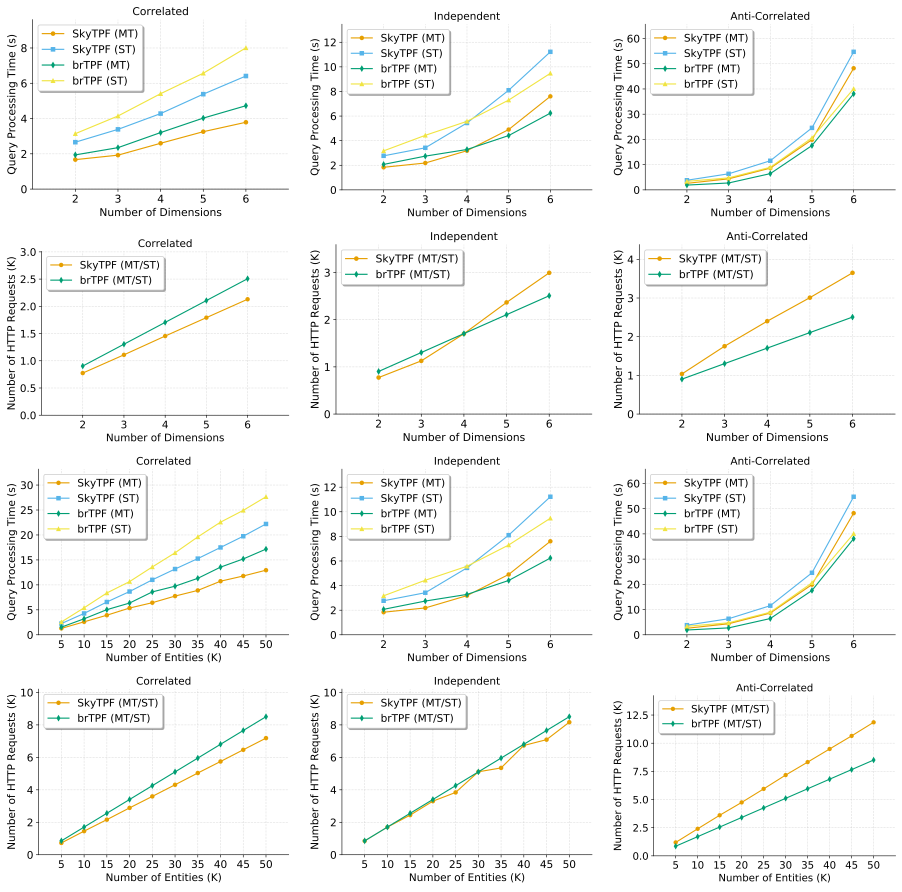

Fig. 5. Effect of the number of dimensions and the number of entities on brTPF-based and SkyTPF-based methods

suggests that the brTPF-based methods are slightly better than SkyTPF-based methods for anti-correlated datasets. This is due to the fact no matter what pivot entity is used, basically nothing can be pruned from consideration for such a data distribution and SkyTPF has some optimization overhead. Finally, SkyTPF-methods achieve a comparable performance to brTPF-methods for the independent data. Figure 5 also illustrates the number of HTTP requests to the server. As expected, the number of HTTP requests also follows a similar trend with the query processing time.

In summary, the experimental results on synthetic datasets show that SkyTPF, especially its multi-threaded implementation, is well suited for correlated and independent datasets and that it has a slight performance overhead when dealing with anti-correlated datasets. brTPF-based methods also perform well without requiring any extensions to the standard interfaces, which enables skyline query processing over knowledge graphs using standard interfaces. The

SkyTPF-based algorithm should be preferred for skyline query processing on real datasets as long as the underlying dataset is not expected to be anti-correlated. However, if one knows that the underlying dataset is anti-correlated with respect to the skyline attributes, the brTPF-based method should be preferred.

7 Conclusion

In this paper we have studied the problem of computing skyline queries over knowledge graphs and proposed solutions that exploit standard interfaces to help the user find interesting entities. Furthermore, we propose, SkyTPF, a lightweight extension of standard interfaces to process skyline queries more efficiently by pruning the search space. The experimental evaluation shows that the proposed methods are capable of computing skylines with reasonable response times. The evaluation also suggests that one should use SkyTPF-based method for skyline query processing unless the distribution of the data is expected to be anti-correlated. In our future work, we plan to increase efficiency of skyline computation by using index structures and support skyline query processing over federations of endpoints/knowledge graphs.

Acknowledgments. This research was partially funded by the Danish Council for Independent Research (DFF) under grant agreement no. DFF-4093-00301B and Aalborg University's Talent Programme.

References

1. W.-T. Balke, U. Güntzer, and J. X. Zheng. Efficient distributed skylining for web information systems. In *EDBT 2004*, pages 256–273, 2004.
2. I. Bartolini, P. Ciaccia, and M. Patella. Salsa: Computing the skyline without scanning the whole sky. In *CIKM 2006*, pages 405–414, 2006.
3. S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE 2001*, pages 421–430, 2001.
4. K. S. Candan, H. Liu, and R. Suvarna. Resource description framework: Metadata and its applications. *SIGKDD Explor. Newsl.*, 3(1):6–19, July 2001.
5. L. Chen, S. Gao, and K. Anyanwu. Efficiently evaluating skyline queries on rdf databases. In *ESWC 2011*, pages 123–138. Springer, 2011.
6. J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE 2003*, pages 717–719, 2003.
7. K. Clark, L. Feigenbaum, G. Williams, and E. Torres. SPARQL 1.1 protocol. W3C recommendation, W3C, Mar. 2013. http://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/.
8. O. Hartig and C. Buil-Aranda. Bindings-restricted triple pattern fragments. In *OTM 2016*, pages 762–779. Springer, 2016.
9. J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194:28 – 61, 2013.
10. K. Hose, C. Lemke, and K.-U. Sattler. Processing Relaxed Skylines in PDMS Using Distributed Data Summaries. In *CIKM '06*, pages 425–434, 2006.

11. K. Hose, C. Lemke, K.-U. Sattler, and D. Zinn. A Relaxed But Not Necessarily Constrained Way from the Top to the Sky. In *CoopIS'07*, pages 399–407, 2007.
12. K. Hose and A. Vlachou. A survey of skyline processing in highly distributed environments. *VLDB J.*, 21(3):359–384, 2012.
13. W. Jin, M. Ester, Z. Hu, and J. Han. The multi-relational skyline operator. In *ICDE 2007*, pages 1276–1280, 2007.
14. H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, Oct. 1975.
15. J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. Dbpedia - A large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.
16. T. Lukasiewicz, M. V. Martinez, and G. I. Simari. Preference-based query answering in datalog+/- ontologies. In *IJCAI 2013*, pages 1017–1023, 2013.
17. M. A. Martínez-Prieto, M. Arias Gallego, and J. D. Fernández. Exchange and consumption of huge rdf data. In *ESWC 2012*, pages 437–452. Springer, 2012.
18. M. Nagendra and K. S. Candan. Skyline-sensitive joins with lr-pruning. In *EDBT 2012*, pages 252–263, 2012.
19. M. Nagendra and K. S. Candan. Efficient processing of skyline-join queries over multiple data sources. *ACM TODS*, 40(2):10:1–10:46, June 2015.
20. P. F. Patel-Schneider, A. Polleres, and D. Martin. Comparative preferences in sparql. In *EKAW 2018*, pages 289–305. Springer, 2018.
21. W. Siberski, J. Z. Pan, and U. Thaden. Querying the semantic web with preferences. In *ISWC 2006*, pages 612–624. Springer, 2006.
22. G. Trimponias, I. Bartolini, D. Papadias, and Y. Yang. Skyline processing on distributed vertical decompositions. *IEEE TKDE*, 25(4):850–862, April 2013.
23. A. Troumpoukis, S. Konstantopoulos, and A. Charalambidis. An extension of sparql for expressing qualitative preferences. In *ISWC 2017*, pages 711–727. Springer, 2017.
24. R. Verborgh, O. Hartig, B. De Meester, G. Haesendonck, L. De Vocht, M. Vander Sande, R. Cyganiak, P. Colpaert, E. Mannens, and R. Van de Walle. Querying datasets on the web with high availability. In *ISWC 2014*, pages 180–196. Springer, 2014.
25. R. Verborgh, M. V. Sande, O. Hartig, J. V. Herwegen, L. D. Vocht, B. D. Meester, G. Haesendonck, and P. Colpaert. Triple pattern fragments: A low-cost knowledge graph interface for the web. *Journal of Web Semantics*, 37-38:184 – 206, 2016.
26. A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis. SKYPEER: Efficient Subspace Skyline Computation over Distributed Data. In *ICDE*, pages 416–425, 2007.
27. A. Vlachou, C. Doulkeridis, and N. Polyzotis. Skyline query processing over joins. In *SIGMOD 2011*, pages 73–84, 2011.
28. S. Wang, B. C. Ooi, A. K. H. Tung, and L. Xu. Efficient Skyline Query Processing on Peer-to-Peer Networks. In *ICDE*, pages 1126–1135. IEEE Computer Society, 2007.
29. W. Zheng, X. Lian, L. Zou, L. Hong, and D. Zhao. Online subgraph skyline analysis over knowledge graphs. *IEEE TKDE*, 28(7):1805–1819, July 2016.