Efficient Routing of Subspace Skyline Queries over Highly Distributed Data

Akrivi Vlachou, Christos Doulkeridis, Yannis Kotidis, and Michalis Vazirgiannis

Abstract—Data generation increases at highly dynamic rates, making its storage, processing and update costs at one central location excessive. The P2P paradigm emerges as a powerful model for organizing and searching large data repositories distributed over independent sources. Advanced query operators, such as skyline queries, are necessary in order to help users handle the huge amount of available data. A skyline query retrieves the set of non-dominated data points in a multi-dimensional dataset. Skyline query processing in P2P networks poses inherent challenges and demands non-traditional techniques, due to the distribution of content and the lack of global knowledge. Relying on a super-peer architecture, we propose a threshold-based algorithm, called SKYPEER and its variants, for efficient computation of skyline points in arbitrary subspaces, while reducing both computational time and volume of transmitted data. Furthermore, we address the problem of routing skyline queries over the super-peer network and we propose an efficient routing mechanism, namely SKYPEER$^+$, which further improves the performance by reducing the number of contacted super-peers. Finally, we provide an extensive experimental evaluation showing that our approach performs efficiently and provides a viable solution when a large degree of distribution is required.

Index Terms—skyline queries, peer-to-peer systems, routing indexes.

1 Introduction

Skyline queries help users make intelligent decisions over complex data, where different and often conflicting criteria are considered. Such queries return a set of interesting data points – from the user's perspective – that are not dominated by any other point on all dimensions [1]. Skyline queries have been studied in centralized systems [1], but also in distributed environments such as web information systems [2] and peer-to-peer (P2P) networks [3], [4], [5], [6], [7], [8]. The P2P paradigm emerges as a powerful model for organizing and searching large data repositories distributed over independent sources [9], [10]. As an example consider a global-scale web-based hotel reservation system, consisting of a large set of independent servers geographically dispersed around the world. Servers accept subscriptions by travel agencies in order to provide booking services over the universal hotel database, without requiring from each travel agency to register with each server. The challenge is to enable users to pose queries that capture the individual user's interests and preferences over this network of servers, and retrieve those results that match a possibly different (each time) set of user-defined criteria.

In this work we explore how subspace skyline queries can be computed efficiently over a distributed and large-scale web information system modeled as a P2P network. Our approach relies on a super-peer architecture, where each peer holds only a portion of the entire dataset, hence horizontal data distribution is assumed. There exist several approaches for distributed skyline computation. Nevertheless, [5], [6] and [7] assume a constrained horizontal partitioning (as also mentioned in [8]) in which a structured P2P overlay network determines for each data point a server that must store it. These techniques have an expected advantage in query processing performance due to the data relocation when peer data enters the network. In this work, we make no specific assumptions and our techniques work with arbitrary horizontal partitioning of the data. In [8] and [3] the authors also employ an arbitrary horizontal partitioning, but these approaches assume the absence of an overlay network. Furthermore, in our approach we assume a super-peer architecture that reduces the degree of decentralization compared to other P2P networks [5], but facilitates efficient query processing as also demonstrated in our experimental evaluation.

In previous work [4], SKYPEER, a distributed framework for subspace skyline processing was proposed. SKYPEER propagates the subspace skyline query to all super-peers and gathers the local skyline results sets using an efficient scheme based on threshold usage and intermediate result merging. Contacting all super-peers during query processing is practically unavoidable in the case of uniform data distribution among the super-peers, where all super-peers may store part of the skyline result set. But in many real-life applications, data is clustered and then it is imperative to avoid contacting all super-peers. In this paper, we address the problem of routing the skyline queries over a super-peer network, aiming to reduce the number of contacted super-peers. Each

- *A. Vlachou and C. Doulkeridis are with the Department of Computer Science, Norwegian University of Science and Technology, Norway. E-mails: {vlachou, cdoulk}@idi.ntnu.no*
- *Y. Kotidis and M. Vazirgiannis are with the Department of Informatics, Athens University of Economics and Business, Greece. E-mails: {kotidis, mvazirg}@aueb.gr*

super-peer clusters its local data, in order to provide a summary description and to make its contents searchable by other super-peers. The cluster descriptions are published to the other super-peers and routing indexes are built. Therefore, we propose an appropriate indexing and routing mechanism for subspace skyline queries over a super-peer network. The main contributions of our work are:

- We explore the implications of processing and routing subspace skyline queries in large scale P2P networks.
- We present SKYPEER which utilizes a thresholding scheme in order to facilitate pruning of dominated data across the peers. We explore different strategies for (i) threshold propagation and (ii) result merging over the P2P network.
- We introduce SKYPEER$^+$, an efficient routing algorithm, which improves performance in the case of clustered distributions of data among super-peers. By using the routing indexes, SKYPEER+ further improves the thresholding scheme and reduces drastically the transferred data.
- We present a set of novel techniques for efficient management of cluster information. We propose the iSUBSKY mapping for local indexing of clusters, threshold refinement for individual clusters and cluster-driven query routing.
- We provide an extensive experimental evaluation of our algorithms and show their comparative benefits over a baseline skyline computation algorithm adapted to work in P2P networks. In addition, we provide an experimental comparison with existing algorithms SSP [5] and FDS [8] for various setups.

The rest of this paper is organized as follows: Section 2 provides an overview of related work. In Section 3, we present the necessary definitions and we introduce the extended skyline. In Section 4, we give an overview of the P2P system model and sketch our approach. In Section 5, we describe the pre-processing phase. Then, in Section 6, we present the basic SKYPEER algorithm and in Section 7 we introduce the efficient routing algorithm SKYPEER$^+$. In Section 8, we provide a comparative performance analysis and we discuss issues related to maintenance. Our experimental results are presented in Section 9. Finally, in Section 10 we conclude the paper.

2 Related Work

Computing skylines was first investigated in computational geometry [11]. Börzsönyi et al. [1] first investigate the skyline computation problem in the context of databases. Thereafter, several techniques are proposed in the relevant research literature for the efficient skyline computation in a centralized setting [12], [13], [14], [15], [16]. Motivated by the different preferences of users, recent papers focus on algorithms to support *subspace skyline* retrieval. Pei et al. [17] and Yuan et al. [18] independently propose the SKYCUBE, which consists of the skylines in all possible subspaces. In [19], [20] SUBSKY algorithm is presented, which transforms the multi-dimensional data to one-dimensional values, and then indexes the dataset with a B-Tree. Similar, in [16] the idea of limiting the amount of data to be read by exploiting the value of a monotone function was studied. However, [16] does not study subspace skyline queries.

The skyline operator has been also studied in decentralized and distributed environments. In [2] an algorithm for distributed Web-accessible sources was proposed. Unfortunately, their assumptions are hardly applicable to large-scale P2P systems. [21] focuses on P2P skyline computation, while providing probabilistic guarantees for the result's correctness. In comparison, our algorithms provably return *exact* answers to arbitrary subspace skyline computations. In [22], Huang et al. assume a setting with mobile devices communicating via an ad hoc network, and study skyline queries that involve spatial constraints. The problem of parallelizing progressive constrained skyline queries [14] in a shared-nothing architecture is addressed in [6]. Although non-constrained skyline queries could probably be supported, this approach suffers from poor load balancing and additional mechanisms, such as replication, need to be used.

An approach for P2P skyline query processing using a tree-structured overlay is presented in [5] and later extended in [23]. In this approach, due to the employed space partitioning scheme, the problem of load balancing is quite important. In [7], an approach for P2P skyline retrieval is described that uses Chord as the underlying infrastructure and minimizes the bandwidth consumption as well as the number of visited nodes. In [24], an approach for continuous subspace skylines in a distributed setting that is based on bitmap representation is presented. In recent work [3], the *PaDSkyline* algorithm is proposed for constrained skyline query processing in distributed environments. However, in contrast to our approach, the authors do not address the issue of communication among sites (peers), as they assume that no overlay exists. In [8], the FDS algorithm is also proposed for skyline query processing in distributed environments where no overlay exists. The aim is to minimize the number of transferred objects, however this may result in several round-trips until the correct result is computed, thereby incurring a high total response time.

Routing indices for efficient search in P2P systems have been previously proposed in the research literature [25], [26]. In this paper, we capitalize on such techniques, in order to build an efficient routing mechanism at super-peer level.

3 Preliminaries and Definitions

We assume a P2P network of N_p peers, where some peers have special roles, due to enhanced features, such as availability, storage capability and bandwidth capacity. These peers are called super-peers SP_i ($i = 1..N_{sp}$), and

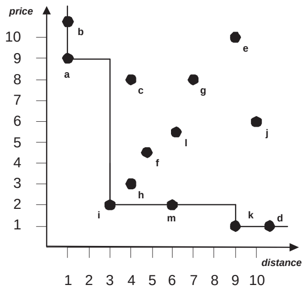


Fig. 1. Skyline example.

they constitute only a small fraction of the peers in the network, i.e. $N_{sp} << N_p$. Peers that join the network directly connect to one of the super-peers. Each super-peer maintains links to simple peers, based on the value of its degree parameter DEG_p. In addition, a super-peer is connected to a limited set of at most DEG_{sp} other super-peers ($DEG_{sp} < DEG_p$).

Each peer P_i holds n_i d-dimensional points, denoted as a set S_i ($i = 1..N_p$). Obviously the size of the complete set of points is $n = \sum_{i=1}^{N_p} n_i$ and the dataset S is the union of all peers' datasets S_i: $S = \cup S_i$. Given a space D defined by a set of d dimensions $\{d_1, d_2, .., d_d\}$ and a dataset S on D, a point $p \in S$ can be represented as $p = \{p[1], p[2], ..., p[d]\}$ where $p[i]$, is a value on dimension d_i.

Definition 3.1: **Skyline:** A point $p \in S$ is said to *dominate* another point $q \in S$, denoted as $p \prec q$, if (1) on every dimension $d_i \in D$, $p[i] \leq q[i]$; and (2) on at least one dimension $d_j \in D$, $p[j] < q[j]$. The *skyline* is a set of points $SKY \subseteq S$ which are not dominated by any other point. The points in SKY are called skyline points.

Without loss of generality, we assume that skylines are computed with respect to min conditions on all dimensions and that all values are non-negative. The notion of skyline can be extended to subspaces. Each non-empty subset U of D ($U \subseteq D$) is referred to as a *subspace* of D. The data space D is also referred to as full space of the dataset S.

Definition 3.2: **Subspace Skyline:** A point $p \in S$ is said to *dominate* another point $q \in S$ on subspace $U \subseteq D$, denoted as $p \prec_U q$, if (1) on every dimension $d_i \in U$, $p[i] \leq q[i]$; and (2) on at least one dimension $d_j \in U$, $p[j] < q[j]$. The *skyline* of a subspace $U \subseteq D$ is a set of points $SKY_U \subseteq S$ which are not dominated by any other point on subspace U. The points in SKY_U are called *skyline points* on subspace U.

Consider for example the dataset depicted in Figure 1. The skyline points are $SKY = \{a, i, k\}$, while for the (non-empty) subspace $U = \{y\}$ the skyline points on U are $SKY_U = \{k, d\}$. Notice that the point d is a skyline point on the subspace $\{y\}$ but it is dominated by the point k in the space $\{x, y\}$. As shown in [18], [4], the skyline set of the full space does not contain all the skyline points for any subspace. A skyline point q in SKY_U is either a skyline point in SKY_V (assuming $U \subset V$) or there is another data point p, such that $p[i] = q[i]$ ($\forall d_i \in U$), that dominates q on the dimension set $V - U$. Thus, a super-set of the union of all subspace skylines is the skyline set of the full space enriched with all points p for which there exists a point $q \in SKY_D$ such that $q[i] = p[i]$ for at least one dimension d_i. Consider for example Figure 1 where e and k have the same x-value but k is a subspace skyline point in contrast to point e which is not a skyline point in any subspace.

To reduce the computation overhead we define a subset of the aforementioned super-set that is able to answer any subspace skyline query. Therefore, we adjust the dominance definition to compute all necessary values co-instantaneously during the skyline calculation. We define [4] the *extended-skyline (ext-skyline)* based on the *extended domination (ext-domination)* definition.

Definition 3.3: **Extended Skyline:** For any dimension set U, where $U \subseteq D$, p ext-dominates q if on each dimension $d_i \in U$, $p[i] < q[i]$. The ext-skyline ($ext\text{-}SKY_U$) is set of all points that are not ext-dominated by any other.

The following lemmas show that $ext\text{-}SKY_D$ is sufficient to answer any subspace skyline query correctly.

Lemma 3.4: Every point that belongs to the skyline of U belongs also to the ext-skyline of U, i.e. $SKY_U \subseteq ext\text{-}SKY_U$.

Lemma 3.5: Every point that belongs to the skyline of a subspace $V \subseteq U$ belongs to the ext-skyline of U, i.e. $SKY_V \subseteq ext\text{-}SKY_U$, $V \subseteq U$.

For example, in Figure 1, points b, m and d belong to the ext-skyline, which is not the case with point e, since e is globally dominated by i, which in turn does not have any value equal to the attribute values of e. Notice that neither e nor m belong to any subspace skyline, in contrast to b and d.

4 System Overview

In the following, we first describe a baseline approach, henceforth referred to as *naive*. Then we describe our thresholding approach, which is more efficient than the naive approach. Finally, we outline an extension of our algorithm with enhanced routing capabilities and particularly suitable for clustered data distributions.

4.1 A Baseline Algorithm

In a pre-processing phase (Section 5.2), each peer P_i computes the local ext-skyline of its dataset S_i. Each super-peer SP_i, collects the ext-skylines of its associated peers and merges them by discarding all ext-dominated points. Thereafter, given a subspace skyline query, each super-peer is able to answer the query based on its peers' data without actually contacting any peer. The querying peer P_{init} is usually a simple peer that submits a query to the system. However, in the rest of this paper, we use P_{init} to refer to the super-peer responsible for the simple peer. In order to answer a subspace skyline query over the entire super-peer network, the initiator super-peer P_{init} has to contact all other super-peers through the neighboring super-peers. More detailed, each super-peer that receives a subspace skyline query, individually processes the request based on the locally stored ext-skylines and routes the results back to the query initiator through its neighbors. The query initiator collects the results from all super-peers and merges them by discarding dominated points. This approach is referred to as naive. In what follows, we describe our approach, which aims at improving the performance of the naive approach, by drastically reducing communication costs and the overall processing time.

4.2 Threshold Usage for Skyline Queries

Subspace skyline points computed at any super-peer may dominate – and thus prune – points of the current super-peer. Therefore, in our algorithm a threshold value is defined based on already computed subspace skyline points, and the threshold is attached to the query before it is propagated in the network. The threshold can be updated at any super-peer in the network during query processing. In our work, we explore different strategies for threshold propagation and result merging through the P2P network (Section 6). In order to support threshold-based query processing, data is transformed into one dimensional values in a pre-processing phase (Section 5).

4.3 Routing of Skyline Queries

Contacting all super-peers during query processing is acceptable for the case of uniform data distribution among super-peers, since all super-peers probably store some skyline points and have to be contacted, in order to retrieve the exact skyline result set. But in the case of clustered data distribution, it is desirable to avoid contacting super-peers that do not contribute to the result set.

Instead of flooding the network, we propose a routing mechanism to contact only those super-peers that may contribute to the overall subspace skyline result set, while reducing the number of contacted super-peers and transferred data. More detailed, in the pre-processing phase, each super-peer SP_A additionally applies a clustering algorithm on its ext-skyline points. The cluster descriptions are broadcast over the super-peer network. Each super-peer collects the cluster information of all super-peers and builds routing indexes based on them (Section 7.3). During query processing the routing indexes are used to define which super-peers may contribute to the overall subspace skyline set (Section 7.4), thus avoiding flooding.

5 Mapping and Pre-processing

At super-peer level there exists a pre-processing phase that allows subsequent query processing over the entire network of peers. In order to enable the threshold-based skyline computation, the multi-dimensional data is transformed into one-dimensional values. This mapping is the topic of the next subsection. Thereafter, we discuss the pre-processing phase in detail.

Algorithm 1 Subspace skyline computation

Input: U: query dimensions
    L: sorted list of data points
$SKY_U \leftarrow \{\emptyset\}$
$threshold \leftarrow$ MAX_INT
$p \leftarrow L$.pop()
while ($f(p) < threshold$) **do**
    if ($\nexists q \in SKY_U : q \prec_U p$) **then**
        $SKY_U \leftarrow SKY_U - \{q\}, p \prec_U q$
        $SKY_U \leftarrow SKY_U \cup \{p\}$
        $threshold \leftarrow min_{p_i \in SKY_U}(dist_U(p_i))$
    end if
    $p \leftarrow L$.pop()
end while
return SKY_U

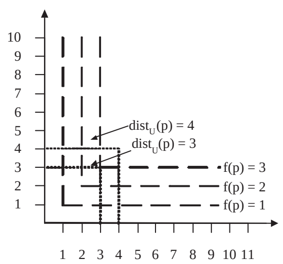


Fig. 2. Mapping example.

5.1 Mapping

Recall that each peer maintains a portion of the dataset. A query refers to a non-empty subset U of D. Inspired by [19], [16], each d-dimensional point p is transformed to a one-dimensional value $f(p)$ based on the formula:

$$f(p) = \min_{i=1}^{d}(p[i]) \quad (1)$$

Let $dist_U(p)$ denote the L_∞-distance of point p from the origin based on the dimension set U, i.e. $dist_U(p) = \max_{i \in U}(p[i])$.

Observation 5.1: Let p_{sky} be a skyline point in a subspace U. A point p for which the following inequality holds can not be a skyline point in subspace U.

$$f(p) > dist_U(p_{sky}) \quad (2)$$

Algorithm 1 uses the proposed mapping for efficient local subspace skyline computation at a peer. Data points are accessed in an ascending order of their $f(p)$ values. Note that the $f(p)$ value is computed once on D independently from the queried subspace U. In contrast, $dist_U(p)$ refers to U and it is calculated during the

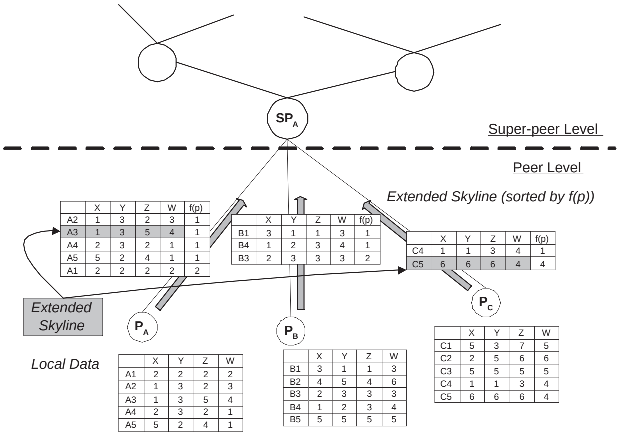


Fig. 3. Peer pre-processing example.

skyline computation. The non-dominated points among the data already examined are added into the current subspace skyline set SKY_U. The algorithm uses as $threshold$ the minimum value of the $dist_U(p)$ of all points in SKY_U. Based on Observation 5.1, the algorithm terminates when $threshold$ is smaller than the $f(p)$ value of the next point p.

Figure 2 depicts a mapping example for a two-dimensional dataset. For sake of simplicity, in our examples we assume that the query space and the data space are identical. In Figure 2, the dashed lines correspond to the points with $f(p)$ values of 1, 2 and 3 respectively. Actually, we examine the data space in a way that is equivalent to the dashed line being shifted from the origin towards the right corner of the data space. On the other hand, the dotted line shows the points that cause a threshold value ($dist_U(p_{sky})$) of 3 and 4 respectively. Notice that if there exists a point that sets the threshold equal to 3 (lies on the corresponding dotted line), this point dominates any point on the dashed line with $f(p)$ equal to 3 independently where the points are exactly (Observation 5.1). Therefore, the threshold-based algorithm manages to return the correct skyline set.

5.2 Pre-processing Phase

First, each peer P_i computes the local ext-skyline of its dataset S_i and sends it to the associated super-peer. Then, the super-peer gathers the local ext-skylines of the individual peers and merges them by pruning out those points of a peer P_i that are ext-dominated by points of another peer P_j, resulting in one set that constitutes the ext-skyline of space D with respect to the dataset on the super-peer and its associated peers. Based on Lemma 3.4 the local ext-skyline is sufficient for a super-peer to determine if any of its peers P_i contributes to the results of any subspace skyline query.

We illustrate the details of peer pre-processing by means of an example. In Figure 3, three peers (P_A, P_B, P_C) are assigned to super-peer SP_A and their local datasets are shown. The dimensionality of the dataset is 4. Each peer computes its local ext-skyline in the original space. The points added to the result set due to the ext-skyline definition are grey shaded. For instance, four of the five points of P_A are skyline points, while $A3$ is included as an ext-skyline point.

6 THE BASIC SKYPEER ALGORITHM

Our algorithm utilizes a thresholding scheme, in order to facilitate pruning of dominated data across the peers. A querying super-peer hands on the query to its neighboring super-peers along the super-peer backbone, which in turn forward the query to their adjacent super-peers. The super-peers execute the query over locally stored ext-skylines and retrieve local results, which are sent back through the query routing path. In the sequel we present all relevant steps in detail.

6.1 Threshold-based Skyline Processing

Let t be the threshold value at the end of the local skyline computation (i.e. Algorithm 1) at the initiator. Based on Observation 5.1, the threshold value indicates that there is a local skyline point p that dominates all points with $f(p)$ values larger than t. At the end of the local skyline computation, t corresponds to the point with the minimum threshold value, i.e. the highest pruning capability. Since data is horizontally partitioned over the super-peers, the local skyline point p dominates all points of any super-peer with $f(p)$ larger than t. Therefore, t is attached to the query and propagated to the super-peers. The threshold value is used as an initial threshold for the local subspace skyline computation to further reduce the computation and communication cost.

An alternative threshold propagation strategy is to compute and refine the threshold on each super-peer that is processing the query, instead of forwarding P_{init}'s fixed threshold value. Intuitively, by progressively lowering the threshold value, the pruning capability of the query increases at each forwarding step. However, this approach requires that the query is propagated after the super-peer has finished the local skyline computation.

6.2 Merging Phase

During query processing, the local subspace skyline sets have to be merged into one overall result set. We explore two different strategies for the merging phase.

The simplest strategy is that the initiator super-peer P_{init} collects the local subspace skyline result sets of all super-peers through its neighbors and merges the local result sets of the individual super-peers to one global result set. Even though the use of the threshold reduces the amount of data transmitted to the query initiator P_{init}, there is still a chance that a local result may contain points that do not belong to the overall skyline. Therefore, a second strategy for the merging phase, is to merge progressively the local skyline sets during query evaluation. Instead of forwarding all results back to P_{init},

Fixed Threshold Fixed Merging (FTFM)
Fixed Threshold Progressive Merging (FTPM)
Refined Threshold Fixed Merging (RTFM)
Refined Threshold Progressive Merging (RTPM)

TABLE 1
Variants of the basic SKYPEER.

each super-peer merges the results of its neighbors, and forwards the merged result back to P_{init}. The benefit is twofold. The transferred data is reduced and a time-consuming centralized merging phase is avoided. On the other hand, the computation cost for each super-peer increases due to the additional merging phase. However, we note that the input and therefore the cost of each subsequent merging phase is much smaller.

6.3 SKYPEER

SKYPEER propagates the subspace skyline query to all super-peers and gathers the local subspace skyline results sets using an efficient thresholding scheme. Thereafter, these local subspace skyline results have to be merged into a global result set, for example by the initiator super-peer. In Table 1, we present our variants of the basic SKYPEER based on two different optimization criteria for distributed subspace skyline computation:

1) Threshold propagation: *(i) Fixed Threshold:* P_{init} calculates its threshold t for $q(U,t)$ and forwards the threshold value to all super-peers. *(ii) Refined Threshold:* P_{init} calculates and sends its threshold to its neighboring super-peers, which do not forward it immediately to other super-peers, but rather they first compute the subspace skyline, calculate the new threshold t', and then forward $q(U,t')$.
2) Merging strategy: *(i) Fixed Merging at* P_{init}*:* In this approach, all super-peers forward their computed subspace skyline back to P_{init}, and P_{init} is responsible for merging the results and computing the resulting subspace skyline for $q(U)$. *(ii) Progressive Merging:* Each super-peer merges the results it receives with its locally computed subspace skyline, before sending the results back to the super-peer from which it originally received the query.

We now introduce the SKYPEER algorithm (see Algorithm 2). A subspace skyline query $q(U)$ is posed by the initiator super-peer P_{init}. The initiator super-peer first computes the skyline on its local ext-skyline resulting in a threshold value t, which based on Observation 5.1 can be used to prune out points that can not belong to the result of the skyline query. The threshold value is attached to the query $q(U)$ that becomes $q(U,t)$ before it is forwarded to P_{init}'s neighbors at super-peer level. Each super-peer SP_i receiving the query, forwards it to its neighboring super-peers and executes a local *threshold-based* subspace skyline computation on its local ext-skyline points. If the RT*M variants are employed, then before forwarding the query, SP_i computes the skyline on its local ext-skyline which results in a refined threshold value t' (or in worst case $t' = t$) that is attached to the query before it is sent to the neighboring super-peers. The results are routed back to P_{init}. If one of the *TPM variants is employed, SP_i first merges the results it receives, before forwarding them to the super-peer from which it received the query (querying super-peer).

Algorithm 2 SKYPEER on super-peer SP_i

Input: mode (FTFM, FTPM, RTFM, RTPM)
  Query ($q(U,t)$)
  Querying super-peer (SP_q)
$L_N \leftarrow$ list of all neighbors except SP_q
if (mode $\in$ {RTFM,RTPM}) or ($SP_i = P_{init}$) **then**
  $SKY_{U_0} \leftarrow$ compute local skyline
  $t \leftarrow refineThreshold(t)$
end if
for $n_i \in L_N$ **do**
  send(n_i, $q(U,t)$)
end for
if mode $\in$ {FTFM,FTPM} **then**
  $SKY_{U_0} \leftarrow$ compute local skyline
end if
for $j = 1$ **to** $|L_N| - 1$ **do**
  receive SKY_{U_j}
  if mode $\in$ {FTPM,RTPM} **then**
    $SKY_{U_0} \leftarrow$mergeResults(SKY_{U_j}, SKY_{U_0})
  else
    send(SP_q, SKY_{U_j})
  end if
end for
send(SP_q, SKY_{U_0})

7 THE SKYPEER$^+$ ALGORITHM

In this Section, we describe SKYPEER$^+$, an algorithm for the case of non-uniform data distribution on peers, that aims at efficient and deliberate routing of skyline queries. In order to achieve this goal, a mechanism for pruning super-peers that can not contribute to the result is necessary. In our approach, super-peers exchange compact data descriptions, thereby enabling pruning of dominated super-peers at query time. The exchanged data descriptions enable the construction of routing indexes for intentional query forwarding to specific super-peers.

Each super-peer clusters the extended skyline points using a standard clustering algorithm. During the preprocessing phase, each super-peer publishes only the cluster descriptions C_i to all other super-peers, as a summarization of its data, while the extended skyline is stored by the super-peer itself. The remaining challenge is to answer skyline queries over the entire super-peer network. Instead of flooding queries at super-peer level, we build routing indexes based on the cluster descriptions that enable selective query routing only to super-peers that may actually be responsible for peers with relevant results.

In Section 7.1 we introduce the one-dimensional mapping used by SKYPEER$^+$, while in Section 7.2, we discuss how local query processing is performed on a super-peer. In Section 7.3 we describe the construction of the routing indexes. Then, in Section 7.4, we present the cluster dominance relationships and we introduce the skyline routing algorithm.

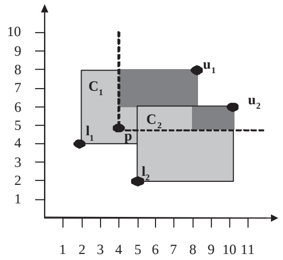


Fig. 4. Threshold example.

7.1 Mapping and Representation

Recall that each super-peer stores the extended skylines of its associated peers, henceforth also called data points of the super-peer.

For non-uniform datasets, we follow the same intuition of [19] where the dataset is partitioned in N_C clusters. Each super-peer first clusters the extended skyline points using a standard clustering algorithm (like K-Means) or using a application specific clustering method [19]. Thus, each data object is assigned to the nearest cluster based on the distance to the cluster's centroid. The super-peer determines for each cluster C_i the minimum bounding rectangle (MBR), which is represented by two reference points l_i and u_i. Point l_i is defined as $l_i[j] = min_{\forall p \in C_i}(p[j])$ and dominates all points in C_i, while point u_i is defined as $u_i[j] = max_{\forall p \in C_i}(p[j])$ and is dominated by all points in C_i. For each data point $p \in C_i$, a one-dimensional mapping is applied according to the distance of p to l_i:

$$f(p, C_i) = \min_{j=1}^{d}(p[j] - l_i[j]) \tag{3}$$

The one-dimensional mapping combined with the clustering information allows us to determine which data points can not belong to the subspace skyline set and can therefore safely be pruned. Let $dist_U(p, C_i)$ denote the L_∞-distance of point p that belongs to the cluster C_i from the lower corner l_i of the MBR_i based on the dimension set U, i.e.

$$dist_U(p, C_i) = \max_{j \in U}(p[j] - l_i[j]). \tag{4}$$

Observation 7.1: Let p_{sky} be a skyline point in a subspace U. A point $p \in C_i$ for which the following inequality holds can not be a skyline point in subspace U.

$$f(p, C_i) > dist_U(p_{sky}, C_i) \tag{5}$$

Based on Observation 7.1, already examined points define a threshold $t(C_i)$ for each cluster C_i, which indicates the region within the cluster that is pruned. Notice that even a point p that does not belong to cluster C_i can dominate points that belong to C_i, as long as p dominates u_i, and thus p can refine the threshold $t(C_i)$.

Consider for example Figure 4. Point p for which $f(p, C_1) = 1$ sets the threshold $t(C_1)$ based on Equation 4 equal to 2. Therefore the pruned area of the cluster C_1 is the dark area within C_1. The dominance area of p partially covers cluster C_2, since p dominates u_2 and the threshold value $t(C_2)$ may be refined by p. Hence, the threshold $t(C_2)$ is set to 3, i.e. $dist_U(p, C_2)$, indicating that point p prunes the dashed area in cluster C_2. In this way, we can prune points in cluster C_2 using the threshold, before we even examine any point of cluster C_2.

7.2 Indexing and Local Query Processing

In the following, we describe the indexing method, namely iSUBSKY (Section 7.2.1), that is employed by the super-peer to index its data, in order to answer efficiently subspace skyline queries (Section 7.2.2) during local query processing. The iSUBSKY method indexes the extended skyline points at each super-peer, since these points are sufficient for a super-peer to answer a subspace skyline query locally. During subspace skyline query processing, iSUBSKY enables early pruning of clusters that do not contribute to the final result set, which reduces the processing cost significantly.

7.2.1 Indexing

Our approach uses a thresholding scheme (Observation 7.1) that relies on a one-dimensional mapping, in order to efficiently prune points and clusters that do not belong to the result set. The one-dimensional values $f(p, C_i)$ refer to the full space (original data space) and do not depend on the queried subspace, therefore they can be computed in a pre-processing phase and stored on disk. To maintain efficiently the one-dimensional values, the values have to be stored to a one-dimensional index, such as B$^+$-tree. In addition, we need a representation of the values where each cluster corresponds to a separate interval in the B$^+$-tree.

Inspired by [27], [9], [19], we propose iSUBSKY, which is an indexing method suitable for subspace skyline query processing. Given a set of N_C clusters and a constant c, each data object p that belongs to a cluster C_i is assigned a one-dimensional iSUBSKY value according to its $f(p, C_i)$ value:

$$iSUBSKY(p) = i * c + f(p, C_i) \tag{6}$$

Using a sufficiently high c value, all objects in the i-th cluster C_i are mapped to the interval $[i*c, i*c+f(u_i, C_i)]$, which is non-overlapping with other cluster intervals. Figure 5 depicts an example of the iSUBSKY mapping.

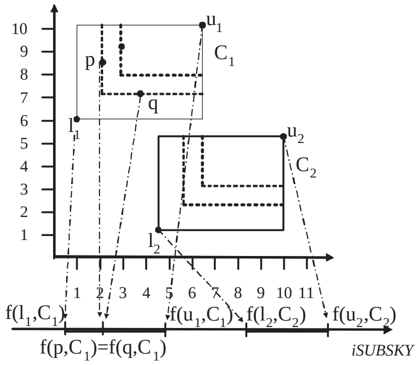


Fig. 5. Example of iSUBSKY.

Notice that points are mapped into one-dimensional values, whereas clusters correspond to intervals. The actual data points can now be efficiently stored in a B$^+$-tree using the iSUBSKY values as keys. Additionally, the clusters $C_i = \{l_i, u_i\}$ are kept in a main memory list C. In this way, we can process a subspace skyline query by examining only some specific intervals of the B$^+$-tree, as described in the following paragraphs.

7.2.2 Subspace Skyline Query Processing

During local query processing on a super-peer, the cluster information is accessed from the main memory list C and the points that belong to each cluster are retrieved using the index. For each cluster C_i accessed at query time, we have to examine at most the interval $[i*c, i*c+f(u_i, C_i)]$. Similar to Algorithm 1, the points for each cluster C_i are examined in an ascending order of $f(p, C_i)$ values. We also keep a threshold value $t(C_i)$ for each cluster C_i. Notice that most probably we do not have to retrieve all data points in the searched interval, as the processing may stop earlier based on the threshold condition (Observation 7.1).

Algorithm 3 describes the pseudocode in detail. For a subspace skyline query $q(U)$, the cluster descriptions C_i are kept in a main memory list C of clusters sorted in ascending order based on the values $\min_{j\in U}(l_i[j])$ (Line 3). In each iteration, we examine the next cluster C_m (Line 7) and we retrieve the point p that belongs to C_m with the minimum iSUBSKY ($f(p, C_m)$) value (Lines 9-10). We repeatedly retrieve the next point based on the iSUBSKY value, until the threshold condition holds (Line 11), since we can safely ignore the remaining points. Each time a new point p is retrieved, we examine if this point is dominated by any point in SKY_U or by any cluster $C_i \in C$ (Line 12)[1]. If this is not the case, we remove all points q retrieved so far that are dominated by p (Line 13), as well as all clusters C_i that are dominated by p (Line 14). Then, p is added to SKY_U as a candidate skyline (Line 15). We also refine the current threshold (Lines 16-18) of all clusters C_i for which p dominates u_i. The threshold $t(C_i)$ is set as the minimum value

1. A cluster $C_i(l_i, u_i)$ dominates a point p in U, if $u_i \prec_U p$.

Algorithm 3 Super-peer subspace skyline computation

1: **Input:** U: query dimensions
   C: list of clusters $\{C_i\}$
2: **Output:** SKY_U
3: Sort C in ascending order according to the value $\min_{j\in U}(l_i[j])$ for each C_i
4: $SKY_U \leftarrow \{\emptyset\}$
5: $t(C_i) \leftarrow$ MAX_INT , $\forall\ C_i \in C$
6: **while** ($C \neq \{\emptyset\}$) **do**
7:   $C_m \leftarrow C$.pop()
8:   $C \leftarrow C - \{C_i\}, C_m \prec_U C_i$
9:   $cursor \leftarrow$ search($[m*c, m*c+f(u_m, C_m)]$)
10:   $p \leftarrow cursor$.pop()
11:   **while** ($f(p, C_m) < t(C_m)$) and ($p \neq$ null) **do**
12:     **if** ($\nexists q \in SKY_U : q \prec_U p$) and
     ($\nexists C_i \in C : C_i \prec_U p$) **then**
13:       $SKY_U \leftarrow SKY_U - \{q\}, p \prec_U q$
14:       $C \leftarrow C - \{C_i\}, p \prec_U C_i$
15:       $SKY_U \leftarrow SKY_U \cup \{p\}$
16:       **for** ($\forall C_i \in C : p \prec_U u_i$) **do**
17:         $t \leftarrow \max_{j\in U}(p[j] - l_i[j])$
18:         $t(C_i) \leftarrow \min(t(C_i), t)$
19:       **end for**
20:     **end if**
21:     $p \leftarrow cursor$.pop()
22:   **end while**
23: **end while**
24: **return** SKY_U

of the previous threshold $t(C_i)$ and the new distance $dist_U(p, C_i)$. In the next iteration, if $f(p, C_m) > t(C_m)$ we discard C_m without retrieving the remaining points of the cluster from the B$^+$-tree (Line 11). Our algorithm returns the subspace skyline set SKY_U (Line 24), after examining all clusters that are not dominated by any skyline point retrieved so far.

Instead of iSUBSKY, the method proposed in [19] could be used that stores tuples of cluster identifiers and $f(p)$ values in a B$^+$-tree. This method applies a heuristic to decide which cluster to examine in each step and to reduce the threshold of all clusters, while exploring all clusters in parallel by using multiple iterators. Our proposed method, examines the clusters in a serialized way, so that the threshold is reduced only by already examined clusters. The main advantage of iSUBSKY is that it allows us to discard entire clusters without accessing any of their data points.

7.3 Routing Indexes Construction

During the pre-processing phase (Section 5), each super-peer additionally broadcasts its cluster descriptions $C_i(l_i, u_i)$ to the network, in order to build routing indexes at super-peer level. Notice that this is a tolerable cost, since it is a one-time cost (when no updates occur) and it does not saturate the network.

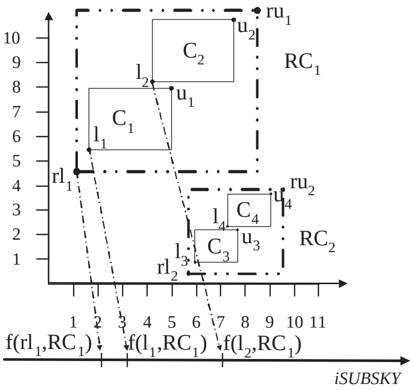


Fig. 6. Routing indexes construction.

After the exchange, each super-peer holds a list of clusters received through each neighbor. In order to efficiently maintain and process the collected cluster information, we use the iSUBSKY method for indexing the cluster information. Each super-peer applies a clustering algorithm[2] on all received clusters, resulting in N_{RC} routing clusters $RC_i(rl_i, ru_i)$, which are represented as MBRs defined by the left lower corner rl_i and the right upper corner ru_i of the MBR. Figure 6 depicts an example where two routing clusters RC_1, RC_2 are created that summarize four clusters C_1-C_4. The routing clusters RC_i are kept in main memory. For each cluster C_j that belongs to a routing cluster RC_i, we index its lower corner (l_j) using the iSUBSKY value $f(l_j, RC_i)$, and we store for each cluster a triple $\{f(l_j, RC_i), l_j, u_j\}$. The triples are stored in a B$^+$-tree using the iSUBSKY values as keys. By accessing the clusters C_j in ascending order of the $f(l_j, RC_i)$ value and setting the threshold based on the maximum distance to the upper corner u_j, it is easy to extend Algorithm 3 for subspace skyline computation, where instead of data points, the dominance relationships between MBRs are considered. In the next section, we will further discuss, how efficient query routing can be established, while computing additionally a threshold for each non-dominated cluster. Using the routing clusters, a super-peer can determine to which neighbors a query $q(U)$ should be forwarded, and thus prune network paths represented by dominated clusters.

Query routing through a super-peer network that contains cycles leads to redundant messages. Therefore, to improve the performance of our routing indexes, each super-peer keeps a list L_i for each super-peer SP_i, which contains super-peer identifiers. A super-peer identifier $SP_i \in L_s$ at super-peer SP_j can be interpreted as: clusters of SP_s are reached by SP_i through a path that contains SP_j. Thus, SP_j forwards queries that refer to clusters of SP_s, only if the query initiator belongs to L_s.

2. Merging bounding boxes of two clusters into a new cluster is a well-studied problem from multidimensional indexing methods. Optimization of the clustering algorithm is out of the scope of this paper, therefore in our experiments we apply k-Means on the MBRs centers.

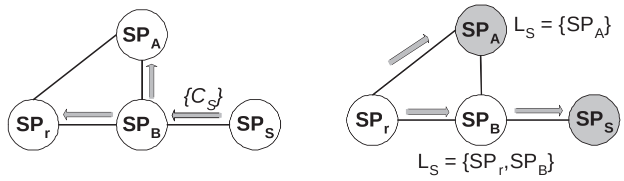


Fig. 7. Cycles example.

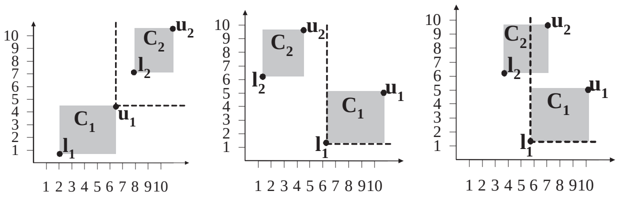
During the construction of routing indexes, each super-peer SP_r that receives the clusters $\{C_S\}$ of another super-peer SP_s for the first time sends back a *verification* message to SP_s through the path. The verification message contains the identifiers of the SP_s and SP_r super-peers. Each time an intermediate super-peer SP_i in the path receives a verification message, it adds SP_r to the corresponding list L_s and propagates the message back to SP_s. For example consider Figure 7 (left). The figure depicts (part of) a super-peer network and let us assume when SP_s propagates its clusters, the clusters are sent to SP_r through SP_B. After SP_r receives the cluster descriptions from SP_s, it sends back to SP_B a verification message, informing SP_B that the clusters of SP_s are searchable from SP_r through SP_B.

This extra information is useful during query routing, as it avoids redundant forwarding of queries in the network and can be efficiently managed using centralized techniques, like hash tables. The larger the number and the size of cycles, the higher the gain of saved messages. If super-peer SP_i receives a query for SP_s's clusters, the query is forwarded only if the querying super-peer belongs to the list L_s, otherwise the query is not forwarded further. For example, consider a subspace skyline query (Figure 7 right) at SP_r that has to retrieve data from super-peer SP_A and SP_s. Then, SP_r propagates the query to SP_A in order to retrieve the necessary data points, and also to SP_B in order to retrieve the points from SP_s. Super-peer SP_A based on its routing information would also forward the query to SP_B in order to retrieve points from SP_s. As depicted in Figure 7, the list L_s of SP_A does not contain SP_r, therefore SP_A knows that SP_r has contacted SP_s through another path.

The advantage of using this technique is more efficient query routing, avoiding duplicate messages when the super-peer topology contains cycles. The disadvantage is the maintenance cost, imposed in the case of super-peer failures, for updating the extra (routing) information. In our setup, super-peer failures are infrequent, as super-peers are mainly stable, therefore our technique is acceptable and provides important gains in performance.

7.4 Query Routing based on Clusters

The aim of the routing indices is to identify a set of clusters (and the corresponding super-peers) that may enclose points that belong to the subspace skyline result set. Super-peers are only aware of the cluster descriptions of other super-peers. Therefore, it is necessary



(a) $C_1 \prec_U C_2$ (b) $C_1 \prec\succ_U C_2$ (c) $C_1 \preceq_U C_2$

Fig. 8. Clusters domination examples.

Algorithm 4 Threshold update (C_n,$\{C_i\}$)

Input: C_n: cluster
C: list of clusters $\{C_i\}$
Output: updated thresholds
for $(\forall C_i \in SKY_U : C_n \preceq_U C_i)$ **do**
  $t \leftarrow \max_{j \in U}(u_n[j] - l_i[j])$
  $t_i \leftarrow \min(t_i, t)$
end for
for $(\forall C_i \in SKY_U : C_i \preceq_U C_n)$ **do**
  $t \leftarrow \max_{j \in U}(u_i[j] - l_n[j])$
  $t_n \leftarrow \min(t_n, t)$
end for

to process this information at each super-peer and to retrieve the clusters that may contain non-dominated points. In the following, we propose an algorithm (Section 7.4.2) that takes as input the cluster information (instead of points), prunes clusters that can not provide any results and returns only those clusters that may contain non-dominated points. By knowing only the cluster information, it is feasible to prune parts of other clusters by setting a threshold. Therefore, we assume that a threshold $t(C_i)$ is attached to each cluster C_i. In the following, we first discuss (Section 7.4.1) the different cluster relationships and the conditions that may lead to updating a cluster's threshold.

7.4.1 Cluster Dominance Relationships

Based on the employed cluster representations, we can straightforwardly derive dominance relationships between clusters. Given two clusters $C_1(l_1, u_1)$ and $C_2(l_2, u_2)$ we define the following dominance relationships (also depicted in Figure 8).

Definition 7.2: Cluster C_1 dominates C_2 in U, denoted as $C_1 \prec_U C_2$, if u_1 dominates l_2.

If C_1 dominates C_2, this means that there exists at least one point in C_1 that dominates all possible points in C_2. Cluster C_2 can not contribute to the subspace skyline result and can be safely pruned during query processing.

Definition 7.3: Clusters C_1 and C_2 are incomparable in U, denoted as $C_1 \prec\succ_U C_2$, if l_1 does not dominate u_2 and l_2 does not dominate u_1.

In this case, no point of C_1 can be dominated nor dominate any point of C_2, thus both clusters have to be examined during query processing.

Definition 7.4: Cluster C_1 partially dominates C_2 in U, denoted as $C_1 \preceq_U C_2$, if l_1 dominates u_2 but u_1 does not dominate l_2.

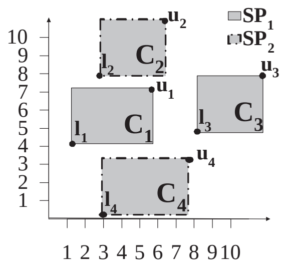


Fig. 9. Clusters of super-peers SP_1 and SP_2.

Therefore if C_1 partially dominates C_2 some points of C_1 may dominate some points of C_2. In this case, we have to examine both clusters in order to retrieve the entire subspace skyline set. However, we can set the threshold of C_2 corresponding to the intersection of the pruning area of C_1 with C_2.

Observation 7.5: Given two clusters C_1, C_2 where $C_1 \preceq_U C_2$, if $u_1 \prec_U u_2$ then the threshold $t(C_2)$ can be set as $dist_U(u_1, C_2)$.

In the case where C_1 partially dominates C_2 in U, we can adjust the threshold of C_2, if u_1 dominates u_2. Algorithm 4 describes the procedure of updating the thresholds. Each time a threshold is updated, we keep the minimum value of the current threshold and the previously defined threshold by another dominance relationship.

The dominance relationships between clusters are essential for our routing algorithm for either pruning entire clusters or restricting the number of points in a cluster that should be accessed, as will be shown shortly.

7.4.2 Super-Peer Query Routing

During query processing, each super-peer that receives a subspace skyline query: 1) uses the query routing algorithm (Algorithm 5) to identify local and non-local candidate clusters, 2) propagates the query to the neighboring super-peers responsible for the non-local clusters, and 3) processes the query locally for the local candidate clusters using Algorithm 3. In the following we describe the query routing process in detail.

First, let us consider a simple example depicted in Figure 9. Both super-peers SP_1 and SP_2 eventually store the cluster descriptions of all four clusters. For better readability, we depict only the clusters and not the routing clusters. Let us assume that SP_1 and SP_2 store incomparable clusters (namely C_1 and C_4), but also some dominated or partially dominated clusters by other super-peers. If SP_1 receives a 2-d skyline query, SP_1 concludes based on the routing clusters that the query has to be propagated to SP_2, because of the clusters C_2 and C_4. Also SP_1 starts a local skyline search only for the cluster C_1, while cluster C_3 is discarded since it is dominated by cluster C_4 of SP_2. When super-peer SP_2 receives the query, it first examines the routing information, and sets

Algorithm 5 Super-peer query routing

1: **Input:** U: query dimensions
RC: list of routing clusters RC_i
2: **Output:** C_{SKY}: list of non-dominated clusters
3: Sort RC in ascending order according to the value $\min_{j \in U}(rl_i[j])$ for each RC_i
4: $C_{SKY} \leftarrow \{\emptyset\}$
5: $t(RC_i) \leftarrow$ MAX_INT , $\forall RC_i \in RC$
6: **while** $(RC \neq \{\emptyset\})$ **do**
7: $RC_m \leftarrow RC$.pop()
8: $RC \leftarrow RC - \{RC_i\}, RC_m \prec_U RC_i$
9: $cursor \leftarrow$ search($[m * c, m * c + f(ru_m, RC_m)]$)
10: $C_n \leftarrow cursor$.pop()
11: **while** $(f(l_n, RC_m) < t(RC_m))$ and ($C_n \neq$null) **do**
12: **if** $(\nexists C_i \in C_{SKY} : C_i \prec_U C_n)$ and $(\nexists RC_i \in RC : RC_i \prec_U C_n)$ **then**
13: $C_{SKY} \leftarrow C_{SKY} - \{C_i\}, C_n \prec_U C_i$
14: $RC \leftarrow RC - \{RC_i\}, C_n \prec_U RC_i$
15: updateThreshold(C_n, C_{SKY})
16: $C_{SKY} \leftarrow C_{SKY} \cup C_n$
17: updateThreshold(C_n, RC)
18: **end if**
19: $C_n \leftarrow cursor$.pop()
20: **end while**
21: **end while**
22: **return** C_{SKY}

a threshold for cluster C_2. The threshold is set based on cluster C_1 of SP_1 and corresponds to the pruning area of point u_1. Thereafter, a skyline search on the clusters C_2 and C_4 is processed. Notice that the threshold for cluster C_2 can not be refined based on cluster C_4, even though C_2 is partially dominated by C_4, since point u_4 does not dominate point u_2. In this way, each super-peer selectively processes only those clusters, that can affect the result set, using appropriate threshold values.

Algorithm 5 describes the proposed routing algorithm. As in iSUBSKY method, the routing clusters $RC_i = [rl_i, ru_i]$ are examined in ascending order of the $\min_{j \in U}(rl_i[j])$ values (Lines 3-7). During processing the clusters (Line 13) and routing clusters (Line 14) that are dominated by the currently examined cluster C_n are discarded. For clusters and routing clusters that are partially dominated by or partially dominate C_n, we compute new threshold values (Line 15 and 17 respectively). At the end of Algorithm 5, C_{SKY} contains all non-dominated clusters that should be processed.

A cluster $C_i \in C_{SKY}$ may be a local cluster and then we add C_i to the list C ($C = C \cup C_i$) with clusters that should be processed locally and set the initial threshold of C_i based on Algorithm 4. Otherwise, the subspace skyline query has to be posed to the corresponding super-peer $SP(C_i)$, therefore we add $SP(C_i)$ to the list SP ($SP = SP \cup SP(C_i)$) of super-peers that should be contacted. Then the super-peer propagates the query to all $SP_i \in SP$ and processes the query locally on the list C by using Algorithm 3.

Notice that in contrast to SKYPEER, we do not need to propagate any threshold among super-peers during query processing. Instead the threshold is refined at each super-peer based on the routing information, i.e. the cluster descriptions. Similar to SKYPEER, the intermediate results have to be merged as described in Section 6.2, which leads to two different variations of SKYPEER+. In the case of fixed merging at P_{init} we have the RFM (Routing-based Fixed Merging) variation, while progressive merging leads to RPM (Routing-based Progressive Merging) variation.

8 MAINTENANCE & COST ANALYSIS

In this Section, we present how maintenance is performed with respect to data updates as well as peer joins and failures. Then, we provide a comparative performance analysis for SKYPEER with respect to other existing approaches [5], [6].

8.1 Maintenance

In a dynamic P2P network, it is necessary to provide support both for data changes and peer churn. The most common maintenance operation are data insertions and deletions (notice that a data update can be treated as a deletion followed by an insertion). Data insertion at a peer may change the peer's extended skyline, therefore this leads to an update to the responsible super-peer. In some cases, the insertion causes also the super-peer's extended skyline to change. Data deletions at a peer may change the peer's extended skyline. In those cases, the change needs to be sent to the super-peer, and the super-peer needs to recompute its extended skyline by requesting its peers to send their extended skyline sets.

In the case of SKYPEER, changes at super-peer level do not need to be propagated further, whereas, in the case of SKYPEER+, updates are propagated to the rest of the super-peer network, only if they change the super-peer's MBRs. Therefore, assuming that the data distribution does not change, only a small percentage of data insertions and deletions actually cause communication at super-peer level.

When a peer joins the P2P network by connecting to a super-peer, its extended skyline is transferred to the super-peer and then the same procedure as in data insertions takes place. Peer failures are more complicated, because they resemble data deletion, which means that the responsible super-peer needs to recompute its extended skyline.

8.2 Performance Analysis

In the following, we provide a performance analysis of the SKYPEER framework and an analytical cost comparison with other existing approaches [5], [6] for skyline query processing in P2P networks. We emphasize that the aim is not to present a direct comparison, which is not necessarily meaningful as they rely on different underlying architectures and assumptions, but rather to serve as a comparative performance analysis. Our evaluation uses three important factors that comprise the total cost, namely construction cost, search cost and update cost.

SKYPEER takes advantage of the two-level architecture of the super-peer system. Therefore, SKYPEER adopts a pre-computation phase at super-peer level, which is similar to materializing results. Other competitor approaches [5], [6] proposed for skyline computation in peer-to-peer systems assume a structured P2P overlay, with distinguishing feature that communication between any two peers is $O(logN_P)$.

In terms of construction cost, SKYPEER requires that each peer sends its extended skyline points to the corresponding super-peer. Therefore, the pre-processing cost of SKYPEER in terms of network traffic, is $O(|ext\text{-}SKY(S_i)| * N_P)$, where $|ext\text{-}SKY(S_i)|$ represents the cardinality of the local extended skyline of a peer's dataset S_i. On the other hand, structured P2P overlays require for each peer's data point to determine the appropriate peer that indexes the point, which leads to a construction cost of $O(|S_i| * N_P * logN_P)$. The cardinality of the local extended skyline set ($|ext\text{-}SKY(S_i)|$) is smaller than the cardinality of the dataset ($|S_i|$), therefore the construction cost of SKYPEER is smaller and it incurs only on local peer to super-peer links.

Regarding search, the authors of [5] report an average number of steps equal to: $O((1 + 2d(1 - \frac{1}{\sqrt[d]{N_P}}))logN_P)$ for uniform query load. SKYPEER contacts all super-peers during query processing leading to $O(N_{sp})$ steps (worst-case scenario). Therefore, the SKYPEER search cost depends on the number of super-peers (N_{sp}), while the cost of [5] depends on the logarithm of the number of peers (N_P) in the network. As also indicated by our experiments, these numbers may differ in as much as 1-2 orders of magnitude.

Data insertions or updates have a cost of $O(logN_P)$ in structured P2P overlay networks, as in [5]. In SKYPEER, super-peers are informed about data insertions, only if the peer's extended skyline is modified. In those cases, the cost incurred is $O(1)$ in terms of messages, since the required communication is between a super-peer and its peer.

	SKYPEER	Structured P2P
Construction	$O(\|ext\text{-}SKY(S_i)\|N_P)$	$O(\|S_i\|N_P logN_P)$
Search	$O(N_{sp})$	$O((1+2d(1-\frac{1}{\sqrt[d]{N_P}}))logN_P)$
Updates	O(1)	$O(logN_P)$

TABLE 2
Performance analysis.

Concluding, the suitability of each approach clearly depends on the application scenario, as well as on the requirements and characteristics of the P2P system. We summarize the results of our analysis in Table 2.

9 EXPERIMENTAL EVALUATION

We evaluate the performance of our algorithms, implemented in Java, using simulations that ran on 3GHz Pentium IV PCs and locally stored data. In order to be able to test the algorithms with realistic network sizes, we ran multiple peer instances on the same machine and simulated the network interconnection. The P2P network topology used in the experiments consists of N_{sp} interconnected super-peers in a random graph topology. We used the GT-ITM topology generator[3] to create well-connected random graphs of N_{sp} peers with a user-specified average connectivity (DEG_{sp}). In our experiments we vary the network size (N_p) from 4000 to 80000 peers, while the number of super-peers is $N_{sp} = 5\% \times N_p$ (for $N_p \geq 20000$ we used $N_{sp} = 1\% \times N_p$).

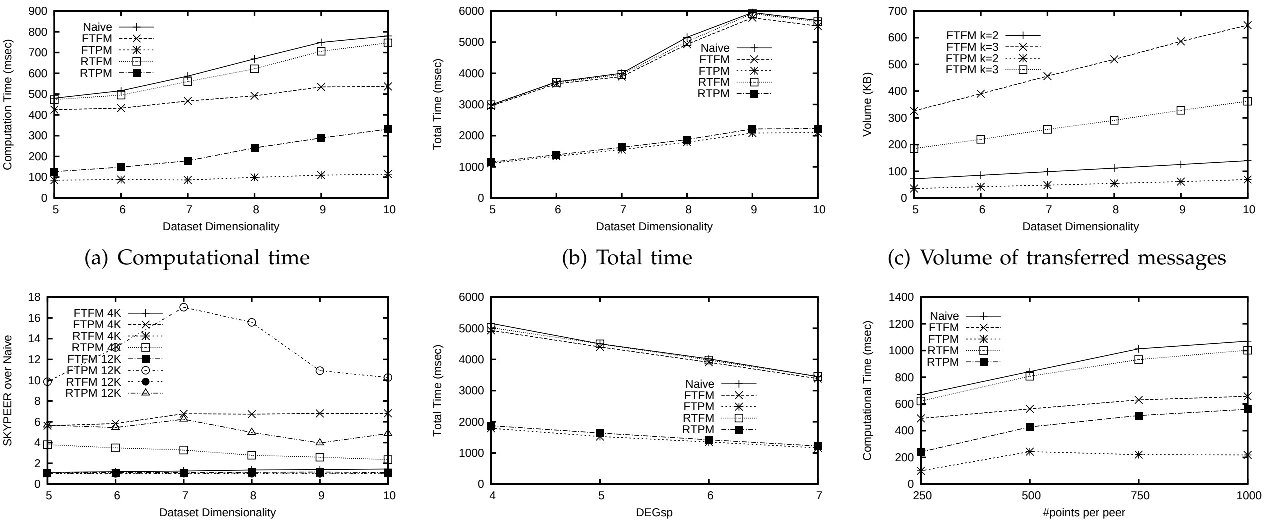
We used two different datasets: uniform and clustered. The uniform dataset includes random points in a unit space. For the clustered dataset, each super-peer picks randomly a point and thereafter $k = 5$ cluster centroids are generated that follow a Gaussian distribution on each axis with variance 0.025, and a mean equal to the corresponding coordinate of the random point. Then, all associated peers obtain points, the coordinates of which follow a Gaussian distribution on each axis with variance 0.005 around the cluster centroids of the super-peer. Given a query dimensionality, all subspaces have uniform probability to be requested. We generate 100 queries, having a randomly selected super-peer initiator for each query, and we present the average values of our results.

9.1 Scalability Study for Uniform Data

In the first series of experiments we examine the proposed method's scaling features with regards to dataset dimensionality. Unless mentioned explicitly, the default values are: $d = 8$, $k = 3$, $DEG_{sp} = 4$, $N_p = 4000$, each peer holds 250 uniformly distributed data points, thus the cardinality of the dataset is n=1M data points.

Figure 10(a) depicts the performance of SKYPEER in terms of computational time, neglecting network delays. The refined threshold variants (RT*M) are more costly than the fixed threshold variants (FT*M) respectively, however they are still more efficient than the naive one, because of threshold usage. Further, progressive merging (*TPM) is faster than fixed merging (*TFM) at the initiator, because the fixed merging at P_{init} is very costly due to the high number of elements, in contrast to the total merging cost incurring at intermediate super-peers. The plot in Figure 10(b) illustrates the total response time, which depends on the size of transmitted data, taking into account the network delay. We assume a modest 4KB/sec as the network transfer bandwidth on each connection. The four variants of

3. Available at: http://www.cc.gatech.edu/projects/gtitm/



(a) Computational time (b) Total time (c) Volume of transferred messages

(d) SKYPEER's relative performance (e) Total time for varying DEG_{sp} (f) Comp.time for varying #points/peer

Fig. 10. Comparison of SKYPEER variants for uniform data distribution.

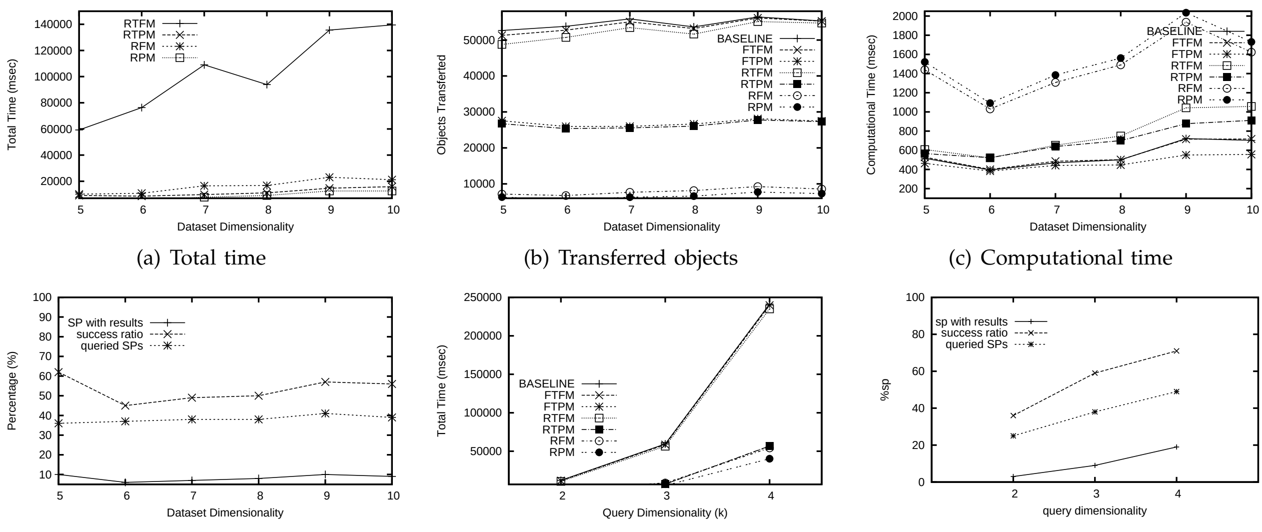
SKYPEER constantly outperform the naive algorithm. Progressive merging reduces communication costs, as several candidate skyline points are pruned out earlier. In Figure 10(c), the volume of the messages (in KB) is presented for various dataset dimensionality values, and for query dimensionality 2 and 3. Since the volume of the messages exchanged to retrieve the skyline result depends mainly on the merging strategy, we present the results of FTFM and FTPM as representative ones. Obviously progressive merging reduces the volume of transferred data.

In order to study SKYPEER's behavior for larger networks, we increased the network size from 4000 to 12000 peers. In Figure 10(d), we present a performance comparison between SKYPEER and the naive algorithm for different network sizes. The vertical axis represents the relative performance of SKYPEER as compared to the naive approach. It is clear that SKYPEER always outperforms naive, and for large networks (12000 peers) the FTPM variant is 17 times faster than naive. We also study how different DEG_{sp} values affect the performance of SKYPEER. In Figure 10(e), we experiment with a network of 4000 peers with DEG_{sp} varying from 4 to 7. We conclude that even though the computational time is not affected by DEG_{sp}, the total time is reduced when DEG_{sp} increases. This is because higher DEG_{sp} values, result in smaller routing paths, hence smaller network transfer costs. Finally, in Figure 10(f), we increased the number of points per peer (n/N_p) from 250 to 1000. We notice that the progressive merging variants clearly outperform the fixed merging ones, as the number of points per peer increases.

9.2 Scalability Study for Clustered Data

In the next experiments, we evaluate the routing ability of SKYPEER$^+$ using a clustered dataset distributed over $N_p = 4000$ peers. Each peer holds 250 points. In Figure 11(a) we depict the response time for the best variants, namely refine threshold (RT*M) and routing-based (R*M). We notice that the response time for the progressive merging methods is slightly higher in the case of RTPM than for RPM. However, the fixed merging approach RFM performs much better than RTFM. This is mainly caused by the very small number of objects that are transferred (Figure 11(b)). For the routing-based approach, the threshold is refined for each super-peer based on the routing indexes that summarize the information available on the whole network. In Figure 11(b) we notice that the number of transferred objects are very low, especially in the case of RFM that has only few more objects more than RPM but much less (order of magnitude) than the other fixed merging approaches. On the other hand, as depicted in Figure 11(c), the fixed threshold variants need less computational time than the refined variants and even less than the routing variants. This additional cost is caused by the processing cost induced by the routing indexes, however it is negligible compared to the total response time. Notice that the merging phase does not influence significantly the computational time, since the merging time is low.

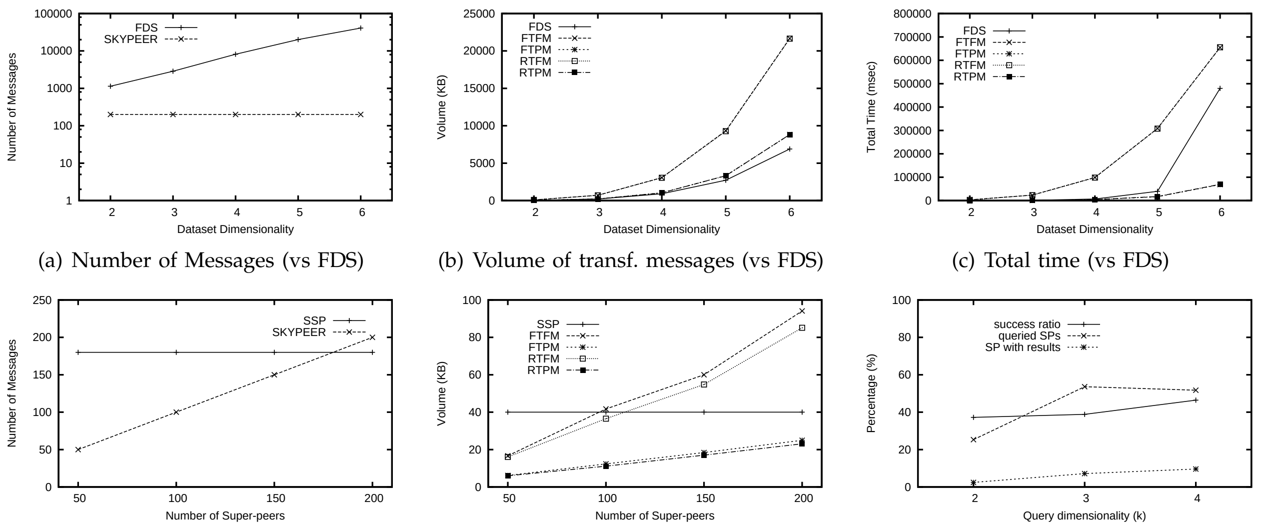
Figure 11(d) depicts: 1) the percentage of super-peers that store at least one point that belongs to the overall subspace skyline set, 2) the percentage of super-peers that are contacted when the subspace skyline query is routed, and 3) the percentage of the contacted super-peers that could not have been avoided to be queried, either because they store some subspace skyline points or because they have to propagate the query to some super-peer that stores some skyline points. Notice that we contact approximate 40% of the super-peers and 55% of them return some skyline points. In Figure 11(e), we vary the query dimensionality from 2 to 4 and we measure the response time for all variations. For



(a) Total time (b) Transferred objects (c) Computational time

(d) Success ratio (e) Total time (f) Success ratio

Fig. 11. Comparison of SKYPEER$^+$ with SKYPEER variants for clustered data distribution.



(a) Number of Messages (vs FDS) (b) Volume of transf. messages (vs FDS) (c) Total time (vs FDS)

(d) Number of Messages (vs SSP) (e) Volume of transf. messages (vs SSP) (f) Success ratio for real data

Fig. 12. Comparison of SKYPEER with other algorithms (FDS [8], SSP [5]) and experiment with real data.

any query dimensionality, SKYPEER$^+$ performs better than the SKYPEER variants, and the gain increases with the query dimensionality. Analogously to Figure 11(d), Figure 11(f) shows for varying query dimensionality that the success ratio increases up to 70%, while the queried super-peers are less than 50%.

9.3 Comparison to Existing Algorithms

In the following, we provide a comparative study against two approaches (FDS algorithm [8] and SSP [5]) for distributed skyline computation, even though they assume different distributed architectures. First, we compare SKYPEER against FDS[4], which assumes a set of servers and the absence of a network overlay. In our scenario, each server is a super-peer and even if SKYPEER assumes an overlay network, for the FDS approach we assume that each server communicates directly with any other server. We emphasize that this experimental setup favors FDS, as any two servers communicate with minimum cost without the need for efficient query routing. The Figures 12(a), 12(b) and 12(c) show the number of search messages, their volume and the response time respectively, for SKYPEER and FDS for increasing dimensionality. We employ a network of 2000 peers (with DEG_p=10 in the case of SKYPEER), n=1M, we use the uniform dataset and we evaluate full space skyline queries. Notice that SKYPEER requires fewer messages than FDS (Figure 12(a)), because FDS uses several communication phases to compute the result. In terms of transferred volume (Figure 12(b)), FDS is cheaper

4. Notice that the algorithm mentioned as ALS in [8] is the baseline variant of SKYPEER.

than SKYPEER, since FDS is designed to minimize the number of transferred objects. However, notice that the *TPM variants of SKYPEER achieve comparable results to FDS, due to the efficiency of in-network progressive merging. Furthermore, the transferred volume of FDS would increase rapidly, if FDS would follow the existing paths for transferring data, instead of direct communication between super-peers. Moreover, in Figure 12(c), the *TPM variants of SKYPEER outperform FDS in terms of total response time, when the dimensionality increases. This is because the total time includes the processing costs, which in the case of FDS are much higher than *TPM, especially for higher dimensions.

In the next experiment, we compare against the SSP [5] algorithm, that relies on a structured P2P overlay network. Structured P2P overlays relocate data on the peers, which is beneficial for query processing. On the other hand, super-peer networks gain in query processing performance due to degree of centralization introduced by the super-peers. In Figures 12(d) and 12(e), we compare our approach with SSP [5] by varying the degree of centralization of SKYPEER. We use a network of 2K peers, n=1.6M and a 2-dimensional uniform dataset. We gradually increase the number of super-peers (N_{sp}) employed by SKYPEER. In Figure 12(d), the number of search messages required by SSP is constant as it does not depend on N_{sp}. However, depending to the number of super-peers employed, SKYPEER performs better or worse than SSP. For small numbers of super-peers, SKYPEER needs fewer messages than SSP, but when the number of super-peers increases, the performance of SKYPEER degrades. Nevertheless, when studying the transferred volume (Figure 12(e)), the *TPM variants of SKYPEER are always better than SSP. Again, this is because the in-network progressive merging discards several results before they reach the querying peer.

In summary, the conclusions of the comparative study are:

- SKYPEER is more efficient than FDS in terms of number of required messages. Furthermore, scales better than FDS with dimensionality. FDS transfers marginally fewer data, however the *TPM variants of SKYPEER have similar performance.
- The *TPM variants of SKYPEER achieve smaller total response time than FDS and this gain increases as dimensionality increases.
- Compared to SSP, SKYPEER requires fewer messages for query processing, when the number of super-peers is smaller than 9% of the number of peers. Moreover, the *TPM variants of SKYPEER induce smaller volume of transferred objects, for all tested setups.

9.4 Experimental Evaluation with Real Data

We also evaluate SKYPEER$^+$ using real-life data. We crawled data (from www.zillow.com) containing information about real estate all over the United States. We obtained a 6-dimensional dataset containing 1M entries. The dataset contains 6 attributes namely number of bathrooms, number of bedrooms, living area, price, year built and lot area. We distributed the dataset randomly to 4000 peers and we use 200 super-peers. Figure 12(f) depicts the results for varying query dimensionality k. The success ratio increases up to 46.5% and the queried super-peers are less than 50%. These results verify the performance of SKYPEER$^+$ also for real-life data.

10 Conclusions

In this paper, we addressed the problem of efficient skyline query routing over a P2P network. We presented a threshold-based algorithm, called SKYPEER, which forwards the skyline query requests among peers, in such a way that the amount of data to be transferred is significantly reduced. Thereafter, we proposed SKYPEER$^+$, an advanced routing algorithm, which improves performance in the case of clustered data distribution at super-peers. SKYPEER$^+$ employs an appropriate indexing technique, in order to maintain routing information at super-peer level. Finally, in our experimental evaluation, we demonstrate the efficiency of our approach both in terms of computational and communication costs.

Acknowledgments

This work was carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme that supported Akrivi Vlachou and Christos Doulkeridis.

References

[1] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *Proceedigns of ICDE*, 2001, pp. 421–430.
[2] W.-T. Balke, U. Gunzer, and J. Zheng, "Efficient distributed skylining for web information systems," in *Proceedings of EDBT*, 2004, pp. 256–273.
[3] B. Cui, H. Lu, Q. Xu, L. Chen, Y. Dai, and Y. Zhou, "Parallel distributed processing of constrained skyline queries by filtering," in *Proceedigns of ICDE*, 2008, pp. 546–555.
[4] A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis, "SKYPEER: Efficient subspace skyline computation over distributed data," in *Proceedings of ICDE*, 2007.
[5] S. Wang, B. C. Ooi, A. K. H. Tung, and L. Xu, "Efficient skyline query processing on peer-to-peer networks," in *Proceedings of ICDE*, 2007, pp. 1126–1135.
[6] P. Wu, C. Zhang, Y. Feng, B. Y. Zhao, D. Agrawal, and A. E. Abbadi, "Parallelizing skyline queries for scalable distribution," in *Proceedings of EDBT*, 2006, pp. 112–130.
[7] L. Zhu, S. Zhou, and J. Guan, "Efficient skyline retrieval on peer-to-peer networks," *Future Generation Communications and Networking*, vol. 1, pp. 309–314, 2007.
[8] L. Zhu, Y. Tao, and S. Zhou, "Distributed skyline retrieval with low bandwidth consumption," *IEEE Trans. on Knowl. and Data Eng.*, vol. 21, no. 3, pp. 384–400, 2009.
[9] C. Doulkeridis, A. Vlachou, Y. Kotidis, and M. Vazirgiannis, "Peer-to-peer similarity search in metric spaces," in *Proceedings of VLDB*, 2007, pp. 986–997.
[10] C. Doulkeridis, K. Nørvåg, and M. Vazirgiannis, "DESENT: Decentralized and distributed semantic overlay generation in P2P networks," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 25, no. 1, pp. 25–34, January 2007.
[11] H. T. Kung, F. Luccio, and F. P. Preparata, "On finding the maxima of a set of vectors," *Journal of the ACM*, vol. 22, no. 4, pp. 469–476, 1975.

[12] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: an online algorithm for skyline queries," in *Proceedings of VLDB*, 2002, pp. 275–286.

[13] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *Proceedings of ICDE*, 2003, pp. 717–719.

[14] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Transactions on Database Systems*, vol. 30, no. 1, pp. 41–82, 2005.

[15] P. Godfrey, R. Shipley, and J. Gryz, "Maximal vector computation in large data sets," in *Proceedings of VLDB*, 2005, pp. 229–240.

[16] I. Bartolini, P. Ciaccia, and M. Patella, "SaLSa: Computing the skyline without scanning the whole sky," in *Proceedings of CIKM*, 2006, pp. 405–414.

[17] J. Pei, W. Jin, M. Ester, and Y. Tao, "Catching the best views of skyline: A semantic approach based on decisive subspaces," in *Procedings of VLDB*, 2005, pp. 253–264.

[18] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. Yu, and Q. Zhang, "Efficient computation of the skyline cube," in *Proceedings of VLDB*, 2005, pp. 241–252.

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

[20] ——, "Efficient skyline and top-k retrieval in subspaces," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 8, pp. 1072–1088, 2007.

[21] K. Hose, C. Lemke, and K.-U. Sattler, "Processing relaxed skylines in PDMS using distributed data summaries," in *Proceedings of CIKM*, 2006, pp. 425–434.

[22] Z. Huang, C. Jensen, H. Lu, and B.-C. Ooi, "Skyline queries against mobile lightweight devices in MANETs," in *Proceedings of ICDE*, 2006.

[23] S. Wang, Q. H. Vu, B. C. Ooi, A. K. H. Tung, and L. Xu, "Skyframe: a framework for skyline query processing in peer-to-peer systems," *VLDB Journal*, vol. 18, no. 1, pp. 345–362, 2009.

[24] K. Fotiadou and E. Pitoura, "Bitpeer: Continuous subspace skyline computation with distributed bitmap indexes," in *Proceedigns of DAMAP*, 2008.

[25] A. Crespo and H. Garcia-Molina, "Routing indices for peer-to-peer systems," in *Proceedings of ICDCS*, 2002, pp. 23–.

[26] B. Yang and H. Garcia-Molina, "Improving search in peer-to-peer networks," in *Proceedings of ICDCS*, 2002, pp. 5–14.

[27] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang, "iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search," *ACM Trans. on Database Systems*, vol. 30, no. 2, pp. 364–397, 2005.

Akrivi Vlachou is currently a post-doctoral researcher at the Norwegian University of Science and Technology (NTNU). She was recently awarded with an ERCIM "Alain Bensoussan" fellowship. She received her Ph.D. in 2008 from the Athens University of Economics and Business (AUEB). She received her MSc degree and her B.Sc. degree from the Department of Computer Science and Telecommunications of University of Athens in 2001 and 2003 respectively. Her research interests include query processing and data management in large-scale distributed systems. Contact her at: Dept. of Computer and Information Science (IDI), Norwegian University of Science and Technology (NTNU), Sem Sælandsv. 7-9, N-7491 Trondheim, Norway, e-mail: vlachou@idi.ntnu.no.

Christos Doulkeridis is currently a post-doctoral researcher with an ERCIM "Alain Bensoussan" fellowship at the Norwegian University of Science and Technology. He holds a Ph.D. and a MSc in Information Systems from the Department of Informatics of Athens University of Economics and Business and a B.Sc. degree in Electrical Engineering and Computer Science from the National Technical University of Athens. His research interests include data management in P2P networks, distributed knowledge discovery, mobile and context-aware computing. Contact him at: Dept. of Computer and Information Science (IDI), Norwegian University of Science and Technology (NTNU), Sem Sælandsv. 7-9, N-7491 Trondheim, Norway, e-mail: cdoulk@idi.ntnu.no.

Yannis Kotidis is an Assistant Professor in the Department of Informatics of Athens University of Economics and Business. He holds a B.Sc. degree in Electrical Engineering and Computer Science from the National Technical University of Athens, an MSc and a Ph.D. in Computer Science from the University of Maryland (USA). Between 2000 and 2006 he was a Senior Technical Specialist at the Database Research Department of AT&T Labs-Research in Florham Park, New Jersey. His main research areas include large scale data management systems, data warehousing and sensor networks. Contact him at: Department of Informatics, Athens University of Economics and Business, 76 Patission Street, GR 10434, Athens, Greece, e-mail: kotidis@aueb.gr.

Michalis Vazirgiannis is an Associate Professor in the Department of Informatics of Athens University of Economics and Business. He holds a degree in Physics (1986), a MSc. in Robotics (1988), both from U. Athens, and a MSc. in Knowledge Based Systems from Heriot Watt University (in Edinburgh, UK). He acquired a Ph.D. degree in 1994 (Dept. of Informatics, University of Athens, Greece). Since then, he has conducted research in GMD-IPSI, Max Planck MPI (Germany), in INRIA/FUTURS (Paris). He has been a visiting professor in Ecole Polytechnique (France) and in Deusto University (Spain). His research interests and work range from Interactive Multimedia Systems, to Spatiotemporal databases, Distributed Data/Web Mining. Contact him at: Department of Informatics, Athens University of Economics and Business, 76 Patission Street, GR 10434, Athens, Greece, e-mail: mvazirg@aueb.gr.