Skyline Queries Against Mobile Lightweight Devices in MANETs

Zhiyong Huang[1] Christian S. Jensen[2] Hua Lu[1] Beng Chin Ooi[1]
[1]School of Computing
National University of Singapore, Singapore
{huangzy, luhua, ooibc}@comp.nus.edu.sg
[2]Department of Computer Science
Aalborg University, Denmark
csj@cs.aau.dk

Abstract

Skyline queries are well suited when retrieving data according to multiple criteria. While most previous work has assumed a centralized setting this paper considers skyline querying in a mobile and distributed setting, where each mobile device is capable of holding only a portion of the whole dataset; where devices communicate through mobile ad hoc networks; and where a query issued by a mobile user is interested only in the user's local area, although a query generally involves data stored on many mobile devices due to the storage limitations. We present techniques that aim to reduce the costs of communication among mobile devices and reduce the execution time on each single mobile device. For the former, skyline query requests are forwarded among mobile devices in a deliberate way, such that the amount of data to be transferred is reduced. For the latter, specific optimization measures are proposed for resource-constrained mobile devices. We conduct extensive experiments to show that our proposal performs efficiently in real mobile devices and simulated wireless ad hoc networks.

1. Introduction

With the continued advances in electronics and wireless communications, more and more mobile handsets with computing and wireless networking capabilities are being deployed. For example, mobile handsets are being equipped with infrared, Bluetooth, or even Wi-Fi capabilities. Further, positioning capabilities are becoming available on handsets, based, e.g., on GPS, the communication infrastructure, or a combination.

In particular, handsets are now being equipped with wireless peer-to-peer (P2P) capabilities. This enables handsets to become parts of self-organizing, wireless mobile ad hoc networks (MANETs) that allow seamless, low-cost, and easily deployed communications [7, 10]. It is conceivable that ad hoc and P2P technologies will be combined to bring about wireless communications without the presence of central servers [4].

Assuming a relational setting, a skyline query [9] returns tuples from a set of tuples that are not dominated by others. A tuple tp_1 is said to *dominate* tuple tp_2, if tp_1 is no worse than tp_2 in any dimension and is better than tp_2 in at least one dimension. In different contexts, "better" can be "smaller" or "larger". For example, a skyline query may request the hotels that have high quality ratings and low room prices.

Most previous work on skyline queries [9, 12, 15, 19, 21] has assumed that the data is stored in a centralized fashion. This paper considers skyline querying in a mobile context with the following assumptions: 1) each mobile device only holds a portion of the entire dataset; 2) devices communicate through MANETs; 3) and mobile users posing skyline queries are only interested in data pertaining to a limited geographical area, although the queries involve data stored on many mobile devices due to the storage limitations of the devices.

Consider the example skyline query shown in Figure 1. Here, $M_1, ..., M_4$, are mobile devices, each storing data corresponding to different portions of geographical space. Device M_2 is interested in the region represented by the circle, the data of which is held on all four mobile devices. Thus, the skyline query of M_2 involves all four mobile devices.

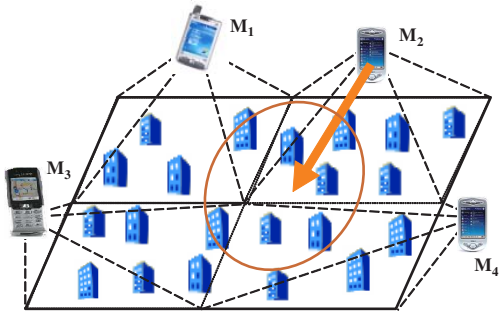


Figure 1. Skyline Query on Mobile Devices

To improve the efficiency of skyline queries, we consider the two most important costs: the cost of the communication among the mobile devices and the cost of query execution on the mobile devices. To reduce the former cost, a skyline query request is sent to the mobile devices involved in a deliberate way such that the amount of data to be transferred is

Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)
8-7695-2570-9/06 $20.00 © 2006 IEEE

minimized. To reduce the latter, specific optimization measures are proposed for resource-constrained mobile devices. Our experimental study shows that the proposed method is efficient in terms of both communication cost and response time.

The paper makes the following contributions: First, it captures the problem of skyline querying in MANETs. Second, it proposes distributed processing strategies that aim to reduce the data to be transferred during the processing of queries. Third, it proposes optimizations that aim to speed up the local query processing on resource-constrained devices. Fourth, it reports on extensive experiments that involve both real mobile devices and MANET simulators. The results show that the paper's proposals perform efficiently.

The remainder of this paper is organized as follows. Section 2 defines the problem to be solved. Section 3 presents the proposed skyline query processing strategies. Section 4 concerns optimization measures for resource-limited mobile devices. Section 5 experimentally evaluates the proposed techniques. Section 6 reviews the related work, followed by conclusions and future directions in Section 7.

2. Problem Definition

We assume a setting with m mobile devices $M = \{M_1, M_2, \ldots, M_m\}$. Each device M_i holds a database relation R_i that contains the data that pertains to sites located in a small geographical area. All R_is on all devices conform to the same schema $<x, y, p_1, p_2, \ldots, p_n>$, where (x, y) represents the location of a site and the p_i are attributes describing a site. The contents of different R_is may overlap, i.e., it is possible that $R_i \cap R_j \neq \emptyset$ for any $i \neq j$. We may also envision a global (and virtual) relation R, such that $\bigcup_{i=1}^{m} R_i = R$.

In this setting, a mobile device M_i can ask a distributed skyline query, whose result consists of all sites s in global relation R that satisfy these conditions: (a) site s is within distance d of M_i's current position; (b) site s is in the skyline of R' in terms of all attributes p_i, where R' is the set of all sites satisfying condition (a). Such a query can be defined as $Q_{ds} = (id, pos_{org}, d)$, where id is the identifier of the device M_{org} issuing the query, pos_{org} is the location of M_{org}, and d is the distance specifying the region of interest.

Such queries are meaningful in practice. For instance, a tourist may want to know about inexpensive and highly rated restaurants within a certain range, in order to find a near-by place for dinner. However, the data on the tourist's own device does not cover the region of interest. The device thus contacts other available devices to obtain additional data. As has been pointed out [17], the wireless cellular link between a mobile phone and its base station is usually much slower than IEEE 802.11-based wireless P2P links between mobile devices. This argues for the relevance of our problem definition that does not assume a fixed base station. Alternatively, we can limit our problem to the extent of a single cell [17].

The queries we consider differ from traditional skyline queries. First, a region of interest is specified in the query, making the query a constrained skyline query. However, this query differs from those obtained by placing constraints on the dimensions involved in the skyline operation [19]. In particular, we use spatial constraints that are not involved in the skyline operation. The use of spatial constraints is reasonable because usually, a mobile user is only interested in sites located in a region around the user's current position, not in the entire geographical space. Also, when within the region of interest, the specific location of a site is often not particularly important. Instead, other attributes of the sites are the important ones. Thus, a skyline query applied to those sites within the range provides the user with a relevant set of answers. Second, horizontal partitions of the global relation are distributed across different mobile devices, which is different from vertical partitioning [6]. Third, mobile devices here have limited storage and processing capabilities in comparison to the computers normally assumed in skyline querying.

Together, the above features pose several challenges to our distributed skyline queries. First, the wireless communication channels between mobile devices are slow and unreliable, in comparison to wired connections. This implies that the amounts of data transferred between devices should be reduced. Second, the devices are resource-constrained. This calls for processing and energy saving techniques for use on the mobile devices. Finally, because different R_is on different mobile devices may overlap, duplicate elimination must be applied before results are returned.

In the considered environment, both communication costs and processing costs on the mobile devices are important to the overall query performance. Indeed, these two aspects determine the performance of a query. With this in mind, we provide processing methods that can reduce the amount of data to be transferred among mobile devices and speed up the query processing on a single mobile device. Symbols to be used throughout this paper are summarized in Table 1.

3. Skyline Queries on Mobile Devices

3.1. Straightforward Strategy

The originator M_{org} of a query does not need to obtain all relations from all other mobile devices—only those tuples that may potentially appear in the final skyline are needed. Based on this observation, we can reduce the amounts of data to be transferred as follows: query originator M_{org} sends the query specification, $Q_{ds} = (id, pos_{org}, d)$, rather than simple data requests. On receiving the specification, device M_i does a skyline query on its relation R_i and then

Symbol	Description
m	Total number of mobile devices
M_i	One mobile device
n	Number of attributes of a tuple
p_i	ith non-spatial attribute of a tuple
R_i	Local relation on M_i
R	Virtual global relation, union of all R_is
M_{org}	The mobile device originating query Q_{ds}
Q_{ds}	Distributed mobile skyline query
pos_{org}	Location of query originator M_{org}
d	Distance of interest in query Q_{ds}
SK_i	Local skyline on M_i w.r.t. Q_{ds}
SK	Final skyline w.r.t. Q_{ds}
tp_{flt}	Tuple used for filtering

Table 1. Symbols Used in Discussions

sends M_{org} the result only, not the entire R_i. Device M_{org} also computes a local skyline for its relation after sending out the query specification. Later, it merges each result it receives with the previous result in an incremental fashion, while also removing the non-qualifying tuples.

This method reduces the amount of data to be transferred at the cost of local computations at each M_i. Suppose for relation R_i on mobile device M_i that the result of skyline query Q_{ds} is SK_i. Then the reduction ratio of the data transferred is $(|R_i| - |SK_i|)/(|R_i|) = 1 - |SK_i|/|R_i|$. Clearly, the more selective a local skyline query is, the smaller the communication cost.

Although the use of local skyline queries can reduce the amount of data sent back to the query originator, there is still a chance that a local result may contain tuples that do not belong to the final skyline. In other words, the union of the SK_i is a superset of SK. The set $FSK = \bigcup_{i=1}^{m} SK_i - SK$ contains all those tuples that appear in one or more local skylines SK_i, but not in the final skyline SK. Transmitting $SK_i - SK$ from mobile device M_i to query originator M_{org} is a waste because it does not affect the final query result. We use FSK_i to represent that subset $SK_i - SK$ on each mobile device. If we can reduce each FSK_i, we can also reduce the communication cost. As an extreme, if each FSK_i is empty, $\bigcup_{i=1}^{m} SK_i = SK$ will hold, and the communication cost is at its lowest. Unfortunately, this is almost impossible in practice because it implies a very special data distribution: the final skyline SK must be perfectly partitioned among all mobile devices, and every tuple in M_i not belonging to SK has at least a dominator in SK within the same local relation R_i.

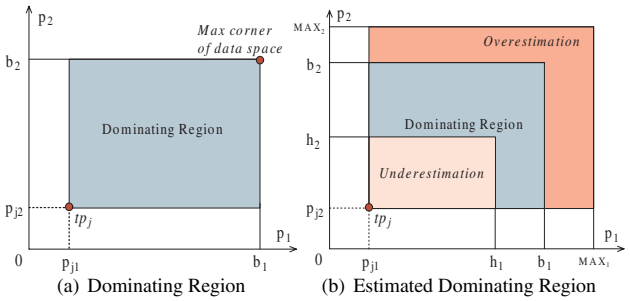
In distributed join processing, a smaller projection of one relation is first sent from one site to its peer site, where it is used to reduce the tuples to be sent to the first site, so that the total communication cost of joining the two relations is reduced [24]. In our setting, FSK and the FSK_i provide indications as to where and how to reduce the communication cost. We proceed to present a strategy that is based on the analysis above and is inspired by the join processing approach.

3.2. Filtering Tuple Based Strategy

Since in each local skyline SK_i there may exist non-qualifying tuples for the final skyline SK, it may be helpful to identify and prevent these from being transmitted. If a tuple tp_i in SK_i does not appear in SK, there must be at least one tuple tp_j in SK and not in SK_i that dominates tp_i. If we can know tp_j when doing the local skyline query on device M_i, tp_i can be removed from the result. We use SK_i' to represent SK_i from which some (maybe not all) tp_is of this kind have been removed. In this way, the number of reduced tuples for transmission is $|SK_i| - |SK_i'|$.

The problem is now how to get such tp_js for a mobile device M_i. Such a tp_js must come from somewhere else than M_i. Since we do a local skyline query on query originator M_{org}, we can pick possible tp_js from its local result. We use SK_{org} to represent the initial, local skyline on M_{org}. Which tuple to choose from SK_{org} is then of interest. To address this issue, we need to consider for a tuple tp_j in SK_{org} its ability to dominate and then remove other tuples.

For a tuple tp_j ($<p_{j1}, \ldots, p_{jn}>$), its ability to dominate other tuples is determined by its own values and the boundaries of the data space, i.e., the hyper-rectangle whose diagonal is the line segment with tp_j and the maximum corner of the data space as coordinates. Any other tuple that resides in that region is dominated by tp_j and is excluded from the skyline. For this reason, we call that hyper-rectangle the *dominating region* of tuple tp_j. Intuitively, the larger tp_js dominating region is, the more other tuples are dominated by tp_j because a larger hyper-rectangle covers more tuples in the data space, especially when the tuples are distributed independently. For simplicity, we use a 2-D illustration as shown in Figure 2(a) in our discussion.



(a) Dominating Region (b) Estimated Dominating Region

Figure 2. Example of Dominating Region

Suppose the value range on dimension p_k is $[s_k, b_k]$ in the virtual global relation R. Then the volume of tuple tp_js dominating region is $VDR_j = \prod_{k=1}^{n}(b_k - p_{jk})$. We choose the tuple, termed tp_{flt}, from SK_{org} with the maximum VDR_j value and use this tuple to filter out non-qualifying tuples. Then instead of sending only the query specification

to the mobile devices, we include also tp_{flt}. Each device M_i will then use tp_{flt} to prune non-qualifying tuples during its local skyline query processing.

Because we add a tuple to what we send to the mobile devices from query originator M_{org}, the communication savings is $|SK_i| - |SK_i'| - 1$ for one M_i. It can be seen that if tp_{flt} fails to remove any non-qualifying tuples from SK_i, the communication cost is actually increased by one tuple. However, we expect that in total, this strategy will be competitive. That is, we expect that $\sum_{i=1,i\neq org}^{m}(|SK_i| - |SK_i'| - 1) = \sum_{i=1,i\neq org}^{m}(|SK_i| - |SK_i'|) - m + 1 > 0$.

As an example, two mobile devices M_1 and M_2 hold hotel relations R_1 and R_2, respectively, as shown in Tables 2 and 3. Assume M_2 is the query originator that wants

hotel	price	rating
h_{11}	20	7
h_{12}	40	5
h_{13}	80	7
h_{14}	80	4
h_{15}	100	7
h_{16}	100	3

Table 2. Relation R_1

hotel	price	rating
h_{21}	60	3
h_{22}	90	2
h_{23}	120	1
h_{24}	140	2
h_{25}	100	4

Table 3. Relation R_2

information on cheap and good hotels (smaller value means better rating). In both relations, each hotel tuple has attributes for the price and the rating based on recommendations. The skyline on M_2 is $\{h_{21}, h_{22}, h_{23}\}$, whereas that on M_1 is $\{h_{11}, h_{12}, h_{14}, h_{16}\}$. If no filtering tuple is used, all four tuples in M_1's local skyline are transferred to M_2. To use a filtering tuple, assume the global upper bound on price and rating is 200 and 10, respectively. We need to pick a filtering tuple from M_2's local skyline $\{h_{21}, h_{22}, h_{23}\}$. Using the VDR definition from above, we have $VDR_{21} = (200 - 60)*(10-3) = 980$, $VDR_{22} = (200-90)*(10-2) = 880$, and $VDR_{23} = (200 - 120) * (10 - 1) = 720$. Because h_{21} has the largest VDR value, we choose it as the filtering tuple. This tuple eliminates h_{14} and h_{16} from M_1's local skyline. (Tuple h_{22} happens to also have this effect.) As a result, the amount of data transferred to M_2 is reduced by two, and the total savings are one tuple. As the cardinality of R_i increases, more non-qualifying tuples can be identified from the local skyline using a filtering tuple.

3.3. Estimated Dominating Region

In the above, we have assumed that the global domain range of any attribute p_j is known on mobile device M_i. This ensures exact computation of the dominating region. Sometimes, the global domain range may be unknown to M_i. In this case, we can compute over-estimated and under-estimated dominating regions for a given tuple tp_i.

Over-estimation of the dominating region for tuple tp_j is achieved using formula $VDR_o = \prod_{k=1}^{n}(\max_k - p_{jk})$, where $\max_k$ is a pre-specified value larger than the global domain upper bound b_k, or the largest possible value of the attribute value type. Underestimation is done using $VDR_u = \prod_{k=1}^{n}(h_k - p_{jk})$, where h_k is the local maximum value of attribute p_k known to M_i. The estimations of the dominating region are shown in Figure 2(b). Note that neither over- nor under-estimation affects the correctness of query results. They possibly pick different filtering tuples and therefore have different filtering abilities. Section 5 explores this aspect.

3.4. Adaptation to Wireless Ad Hoc Networks

So far, we have assumed that the query originator can directly communicate with any other mobile device. In that setting, the filtering tuple decided by the originator is used for all mobile devices. In a real mobile setting, however, communication is more likely to be accomplished using multiple hops via intermediate mobile devices if source and destination cannot contact each other directly. In such a setting, we need to adapt our strategy to reduce the communication cost.

First, tp_{flt} is dynamically updated during the procedure of query relay to increase its pruning potential. After doing a local skyline query on mobile device M_i, the tuple tp'_{flt} with the maximum VDR value is obtained from the local skyline result SK_i. Then tp'_{flt} is compared with the current tp_{flt}, and the one with the larger pruning potential, i.e., the larger VDR value, will be used as the new filtering tuple for other mobile devices to which M_i will forward this query.

As an example consider the three mobile devices M_1, M_3, and M_4, whose relations are shown in Tables 2, 4, and 5. Here, M_4 is the query originator and M_3 is

hotel	price	rating
h_{31}	60	3
h_{32}	80	5
h_{33}	120	4

Table 4. Relation R_3

hotel	price	rating
h_{41}	80	2
h_{42}	120	1
h_{43}	140	2

Table 5. Relation R_4

the intermediate in-between M_4 and M_1. The local skyline on M_4 is $\{h_{41}, h_{42}\}$ and that on M_3 is $\{h_{31}\}$. Based on the VDR values, h_{41} on M_4 is chosen as the filtering tuple and is sent to M_3. If the filtering tuple is not dynamically adjusted, h_{41} will be sent to M_1 as well, where it will eliminate h_{16} only. If the filtering tuple is dynamically adjusted, h_{31} on M_3 will be used as the new filtering tuple. When h_{31} is sent to M_1, it will eliminate both h_{16} and h_{14}. This example shows how dynamic updates of the filtering tuple can make a difference.

Second, a check is made in every mobile device to avoid processing the same query more than once. To support this check, a tag cnt is added to every query, which then becomes (id, cnt, pos_{org}, d). This tag is a local count, generated by each query originator. In other words, each mobile device maintains a count for all queries it issues, and this

count is attached to the corresponding query. When a mobile device M_i receives a query (id, cnt, pos_{org}, d), it will check its log to see if this query has been processed. If not, M_i processes this query and forwards it to others. Otherwise, the query is ignored.

To save communication cost, count cnt can be defined as a byte, allowing a device to generate 256 queries with increasing cnt value. The count can be reset at regular intervals, e.g., each day. The log on a device keeps for every device its last arriving query's count cnt, which is implemented simply by a hash table that maps the device identifier id to a count. On each mobile device, the worst case space cost of such a hash table is $O(m)$, where m is the total number of mobile devices. The time cost of the check is $O(1)$. This mechanism works well under the assumption that a mobile device is only interested in its latest query.

4. Optimizations on Mobile Devices

On the mobile devices, relations are stored in a manner that takes into account the limited storage space on the devices and that enables efficient local skyline query processing.

4.1. Dataset Storage

Because mobile devices have limited storage (sometimes both data and running programs share the same limited memory space), simply storing all tuples sequentially in so-called flat storage will either consume too much space or harm query processing performance. Several storage alternatives have been proposed for devices with limited space [5, 8, 20].

We adopt a hybrid storage scheme for each relation R_i stored on a mobile devices. Each R_i has both spatial and non-spatial attributes. Because different tuples are usually located at different geographic sites and thus have different spatial attributes, storing the spatial values separately from the tuples does not save storage space. Hence, we store for each tuple its spatial coordinate values directly in R_i. Non-spatial attribute values are likely to be shared among multiple tuples, making it beneficial to store them separately. To facilitate skyline query processing on mobile device R_i, we use ID-based storage [20] for the non-spatial attributes. To support fast spatial range checks, the maximum and minimum spatial coordinates are kept as constants in x_{max}, y_{max}, x_{min} and y_{min}. These coordinates specify the minimum bounding rectangle MBR_i of all sites in R_i.

We do not use ring storage [8], as not every tuple has pointers to values in this scheme. Instead, all tuples with the same value for an attribute are linked by internal pointers, and only one tuple has a pointer to the shared value. This causes expensive access to tuple values, since we have to traverse the internal pointer chain to reach the unique tuple with the external pointer. This hurts skyline query processing, which needs tuple values frequently in dominance comparisons.

Though domain storage [5] has tuple-to-value pointers for each attribute of each tuple, we also do not use this scheme, as it still consumes extra time to use tuple-to-value pointers to access a tuple's attribute values from separate domains. Also, domain storage does not provide the convenience given by ID-based storage.

4.2. Local Skyline Computing

In domain storage, ring storage, and ID-based storage, all domain values are stored in an array for each attribute. This arrangement can be used to facilitate local skyline query processing on a mobile device. Suppose on mobile device M_i the value range of attribute p_j is $[l_j, h_j]$. Such an l_j or h_j value can be fetched on a mobile device in $O(1)$ time, if we store all available values in each sorted. We assume that smaller values are preferred for each attribute p_i, and all values in each attribute domain are stored in ascending order. Our hybrid storage model for local relation R_i is illustrated in Figure 3.

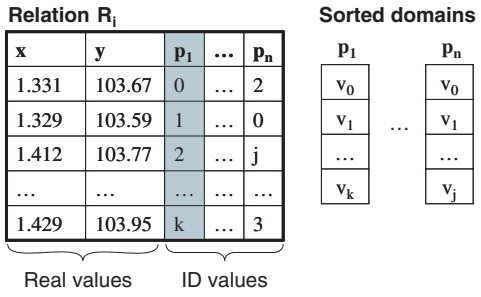


Figure 3. Hybrid Storage Model

Inspired by the SFS algorithm [12], we also sort R_i on the ID value of one attribute. This reduces the number of value comparisons in determining dominance during query processing. We choose the attribute with the largest number of distinct values as the attribute to be sorted on. For simplicity and without loss of generality, we assume that p_1 is that attribute.

With the domain information and tp_{flt} received from the query originator, we can determine efficiently whether we need to generate the local skyline on mobile device M_i. If on every attribute, we have $tp_{flt}.p_j \leq l_j$, which means that the best tuple (potentially, maybe unavailable) on M_i is dominated by tp_{flt}, no tuple in relation R_i will be in the global skyline. All these comparisons cost $O(n)$ time, where n is the number of attributes. In this case, M_i needs to do nothing, but return a correct, short message to M_{org}. Thus, local computation is saved significantly.

If the range check does not imply that all of R_i is dominated by tp_{flt}, we need to generate the local skyline query.

Here, the sorted storage of domains is also useful. Because the domain values of any dimension p_j are ordered, the IDs in relation R_i reflect the inequality between the real p_j values of different tuples. For instance, suppose two tuples tp_1 and tp_2 have id_{j1} and id_{j2}, respectively, on dimension p_j, and that all domain values are stored in ascending order. Then simply comparing each pair of id_{j1} and id_{j2}, instead of accessing and comparing the real domain values, is enough to determine the dominance between tp_1 and tp_2. This has two benefits. First, access time is saved since no offset based addressing is needed to access values. Second, comparison of simple ID integers generally costs less time than that of domain values.

The algorithm for a local skyline query on mobile device M_i is presented in Figure 4. Initially, a spatial range

Algorithm local_skyline(pos_{org}, d, tp_{flt})
Input: pos_{org} is the location of query originator
d is the distance of interest
tp_{flt} is the filtering tuple
Output: reduced local skyline, and updated filtering tuple
// Check if R_i's MBR overlaps the query region
if $(mindist(pos_{org}, MBR_i) > d)$ **return**;
// Check if R_i is dominated by the filtering tuple
$skip$ = TRUE;
for each attribute j of R_i
if $(tp_{flt}.p_j > l_j)$ $skip$ = FALSE; **break**;
if $(skip)$ **return**; **else** $SK_i = \emptyset$;
// Local ID-based SFS processing
for each tuple tp_j in R_i
// Too far away from query point
if $(dist(pos_{org}, tp_j) > d)$ **continue**;
out = FALSE;
for each skyline point sp_k in SK_i
// sp_k dominates tp_j
if $(\forall l > 1, sp_k.id_l < tp_j.id_l)$ out = TRUE; **break**;
if $(!out)$ add tp_j into SK_i
// Filtering, and picking up maximum VDR
$idx = null$; $VDR_m = 0$;
for each skyline point sp_k in SK_i
if $(\forall l, tp_{flt}.p_l < sp_k.pl)$ remove sp_k from SK_i
else if $(VDR_k > VDR_m)$ $idx = k$; $VDR_m = VDR_k$
// Update filtering tuple if necessary
if $(VDR_m > VDR_{flt})$ $tp_{flt} = tp_{idx}$;

Figure 4. Local Skyline Query on M_i

check is done to see if the spatial extent covered by mobile device M_i overlaps with the query region specified in Q_{ds}. If not, the processing stops. Otherwise, each attribute domain range's lower bound is checked against the filtering tuple tp_{flt}, to determine whether the entire local relation R_i is dominated by tp_{flt}. If not, R_i is scanned sequentially to obtain the local skyline SK_i, which takes advantage of the sorted attribute p_1 and checks the rest of the dimensions only. Two aspects make this procedure different from that of the SFS algorithm. First, attribute IDs are compared instead of real attribute values. Second, a spatial distance check is used to exclude tuples too far away from the query position. After getting the local skyline SK_i, the filtering tuple is used again to filter out non-qualifying tuples, and a more capable filtering tuple might be found in SK_i to replace the old one.

4.3. Assembly on Query Originator

When receiving results back from devices, the query originator needs to combine them with its own local skyline to obtain the correct global skyline. Assembly involves two tasks. The one is to remove non-qualifying tuples from the incoming result and M_{org}'s own local result. The other is to remove all duplicate tuples in the final skyline result. Both tasks can be done within a simple nested loop, i.e., for each tuple tp_j in an incoming result SK'_i, every tuple tp_k in the local current result SK_{org} is checked. Duplicates can be identified by checking the x and y values only, since we assume that no two tuples represent the same geographic location. If tp_j and tp_k are not the same, the dominance between them is determined by checking all their non-spatial attributes. This way, SK'_i and SK_{org} are merged together correctly to produce the updated SK_{org}.

5. Experimental Studies

We proceed to offer insight into the properties of the methods proposed in this paper based on experimental studies. We first study the efficiency of the local optimizations, then study the performance of the distributed processing strategies in a simulated MANET environment.

5.1. Studies on Local Optimizations

This set of experiments is conducted on an HP iPAQ h6365 pocket PC running MS Windows Mobile 2003 with a 200MHz TI OMAP1510 processor and 64MB SDRAM (55MB user accessible). All programs are written using SuperWaba, a Java-based open-source platform for PDA and Smartphone application development [3]. The parameters used in the experiments are listed in Table 6.

Parameter	Setting
Number of total mobile devices	$3^2, 4^2, \ldots, 10^2$
Cardinality of global relation	100K, 200K, ..., 1000K
Cardinality of local relations	10K, 20K, ..., 100K
Storage model for local relations	Flat, Hybrid
Number of non-spatial attributes	$2, 3, 4, 5$
Non-spatial attribute range	[0, 1000], [0.0, 9.9]
Spatial extent of global relation	1000×1000
Attribute distribution	Indep., Anti-Correl.
Query distance of interest	100, 250, 500

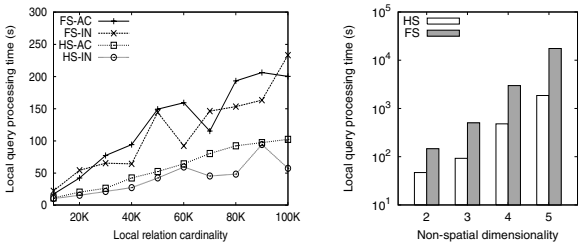
Table 6. Parameters Used in Experiments

For tests on dataset cardinality, the datasets contain 10K to 100K points with two non-spatial attributes. For tests on

non-spatial dimensionality, the datasets contain 50K points with two to five non-spatial attributes. Each non-spatial attribute is of type float and its domain is $\{0.0, 0.1, 0.2, \ldots, 9.9\}$. Since each domain contains 100 distinct values, we use byte type IDs in the hybrid storage implementation. Synthetic datasets with both independent and anti-correlated distributed attributes are used.

For the reasons stated in Section 4.1, we only compare the hybrid storage (HS) scheme that we propose with the flat storage (FS) scheme in terms of skyline query processing time. For either scheme, we assume that no extra index is used. For the HS scheme, we use the algorithm we propose to process the local skyline query. For the FS scheme, we use the simple BNL algorithm since no multi-dimensional index or sort order is assumed to be available on a mobile device.

Figure 5 shows the query processing time for each scheme, where IN and AC represents the independent and anti-correlated dataset, respectively. It is not surprising that query processing needs less time on HS than on FS. This is because in HS, most comparisons are between simple IDs of type byte, whereas in FS, all comparisons are between float raw values which consume more time. In addition, HS keeps the domains and the first attribute sorted, which is also helpful in speeding up skyline computation. In Figure 5(b) we show the average costs of both distributions because their costs are very close to each other for each dimensionality.



(a) Time vs. cardinality (b) Time vs. dimensionality

Figure 5. Local Processing Time

5.2. Performance in Simulated MANETs

In this set of experiments, we test our proposed methods in a MANET environment simulated using JiST-SWANS [1], a Java-based wireless ad hoc network simulator. We consider three main performance aspects: (1) the data reduction efficiency of the distributed query processing; (2) the overall response time; and 3) the numbers of messages used to forward a query between mobile devices. The simulation experiments have been conducted on a Pentium IV desktop PC running MS Windows XP with a 2.99GHz CPU and 1GB main memory.

5.2.1 Experiment Settings

We use global relations of 100K to 1M. In each global relation, all tuples are distributed randomly within a 1000×1000 spatial domain. Based on a uniform grid on the spatial domain, a global relation R is divided into local relations (the R_is), each containing all the tuples within its corresponding grid cell.

We use total numbers of mobile devices (m) equal to the squares of the numbers 3 to 10, i.e., $\{9, 16, 25, 36, 49, 64, 81, 100\}$, with each device containing the data of a grid cell. The number of non-spatial attributes is varied from 2 to 5. All non-spatial attributes are of integer type in the range $[1, 1000]$, and they conform to either independent or anti-correlated distributions. The total number of mobile devices indicates that we test our methods in environments of small-scale and moderate-scale MANETs, according to a recent classification [18]. All devices move within the spatial domain according to the random waypoint mobility model [10]. In that model, every device moves towards its own destination with its own speed, and when it reaches that destination it will stop there for a period of time (holding time) and then move to another destination with a new random speed. The mobility and wireless settings used are listed in Table 7. Every mobile device issues 1 to 5 queries

Parameter	Setting
Total simulation time	2h
Speed range	2m/s–10m/s
Holding time	120s
Wireless routing protocol	AODV

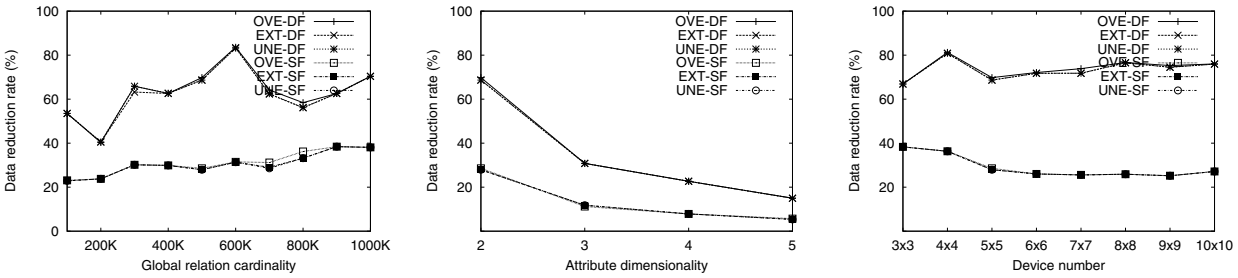
Table 7. Parameters Used in Simulations

at random times during the simulation. Queries of different devices can coexist, while a single device does not issue a new query if it has one in progress.

For query forwarding, we compared two different strategies. The first is a *breadth-first* strategy, where initially the query originator sends its query to all its neighbors. Each neighbor processes the query locally, sends the result back to the originator and then forwards the query to its own neighbors. The same procedure is repeated on every mobile device involved. The second is a *depth-first* strategy, where a query is forwarded to only one neighbor to which the query has not been sent. The query result will only be returned when no further neighbor is available or all neighbors have processed it. Then the result will be forwarded back along the reversed path. Each mobile device (including the originator) on the path merges the result with its own result and then either sends the result back or sends the query to another available neighbor.

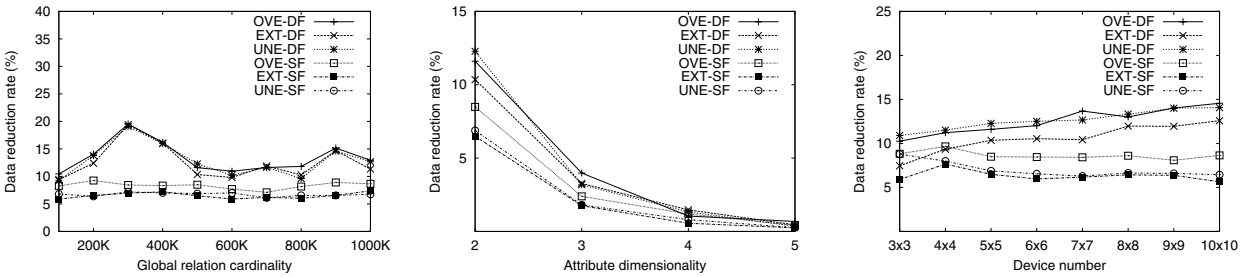
5.2.2 Data Reduction Efficiency

We proceed to study the efficiency of the distributed processing strategies in terms of their data reduction rate. The



(a) DRR vs. global cardinality (b) DRR vs. attribute dimensionality (c) DRR vs. device number

Figure 6. Data Reduction Rate on Independent Datasets in a Static Setting



(a) DRR vs. global cardinality (b) DRR vs. non-spatial dimensionality (c) DRR vs. device number

Figure 7. Data Reduction Rate on Anti-Correlated Datasets in a Static Setting

data reduction rate DRR is the proportion of tuples reduced by the filtering tuple to the number of tuples in the unreduced skyline. To be specific, recall from Section 3 that, for each mobile device M_i (except the query originator M_{org}), the unreduced skyline is SK_i and the reduced skyline is SK_i'. The data reduction rate with respect to the whole system is then defined as:

$$DRR = \frac{\sum_{i=1,i\neq org}^{m}(|SK_i| - |SK_i'| - 1)}{\sum_{i=1,i\neq org}^{m}|SK_i|} \qquad (1)$$

I. Pre-Tests in Static Setting

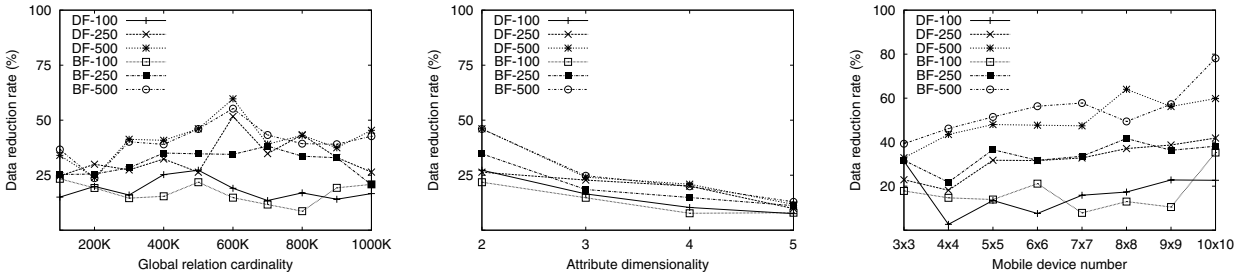
Before conducting the simulation, we test the different filtering tuple selections in a static setting where no devices move and queries are forwarded recursively from the originator to the outer neighbors in the grid. We also ignore the distance constraint and use every device M_i as the query originator once. The final result is the average of a total of $m \times m$ queries for each single experiment.

The experimental results on independent global relations are shown in Figure 6. The results show that different estimations of the dominating region (OVE for over-estimation, EXT for exact computation, and UNE for under-estimation) barely affect the filtering efficiency for uniform global relations. This justifies the use of estimation in a mobile device that does not require knowledge about the global bounds of each attribute.

In the experiment covered in Figure 6(a), all global relations have two non-spatial attributes and are partitioned among 5×5 mobile devices. For the strategy using a single filtering tuple (SF), the data reduction rate grows slowly as the global cardinality increases. A fixed-size data space D becomes increasingly dense as more tuples are added, and tuple tp_{flt} is likely to dominate more tuples. The use of a dynamic filtering tuple (DF) is attractive. The filtering tuple is dynamically changed based on its pruning capacity such that a tuple with higher pruning potential (if one exists) is picked for further processing. This dynamic adjustment also renders DF less stable.

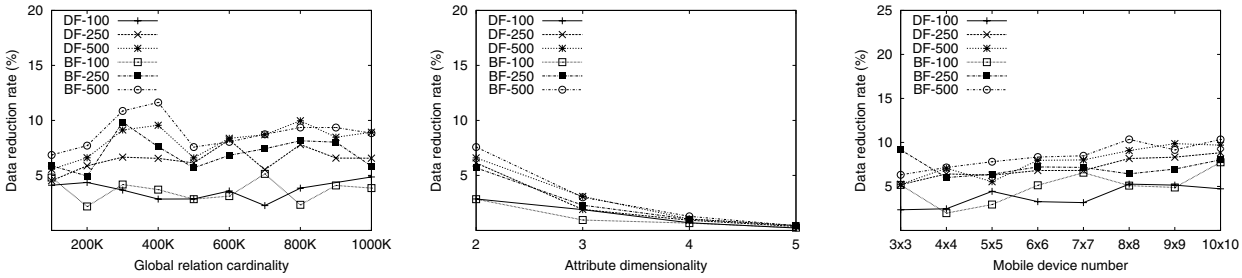
In the experiment covered by Figure 6(b), all global relations have 500K tuples and are partitioned among 5×5 mobile devices. For both filtering strategies, the data reduction rate decreases as the attribute dimensionality increases. In contrast to Figure 6(a), the fixed-cardinality data space D becomes sparser as the dimensionality increases, meaning that a given tuple tp_{flt} is likely to dominate fewer tuples.

In the experiment reported upon in Figure 6(c), all global relations have 500K tuples with two non-spatial attributes. For the SF strategy, the data reduction rate decreases slightly as the number of mobile devices increases. As the global relation is partitioned among more mobile devices, the denominator $\sum_{i=1,i\neq org}^{m}|SK_i|$ in Formula 1 possibly increases while the single filtering tuple strategy cannot prune additional tuples, which leads to a smaller DRR value. The pruning capacity of the dynamic strategy is not affected, as the filtering tuple is dynamically changed according to the local skyline on every mobile device.



(a) DRR vs. global cardinality (b) DRR vs. attribute dimensionality (c) DRR vs. mobile network scale

Figure 8. Data Reduction Rate on Independent Datasets in MANET Simulation



(a) DRR vs. global cardinality (b) DRR vs. non-spatial dimensionality (c) DRR vs. mobile network scale

Figure 9. Data Reduction Rate on Anti-Correlated Datasets in MANET Simulation

The experimental results on anti-correlated global relations are shown in Figure 7. For this data, we can see that for the SF strategy, over-estimation of the dominating region exhibits the best filtering efficiency in almost all cases. Also, for every single experiment, the filtering efficiency is lower when compared to that of its counterpart for independent global relations. This is not surprising because filtering tuples are chosen based on the assumption of an independent distribution.

II. Tests in MANET Simulation

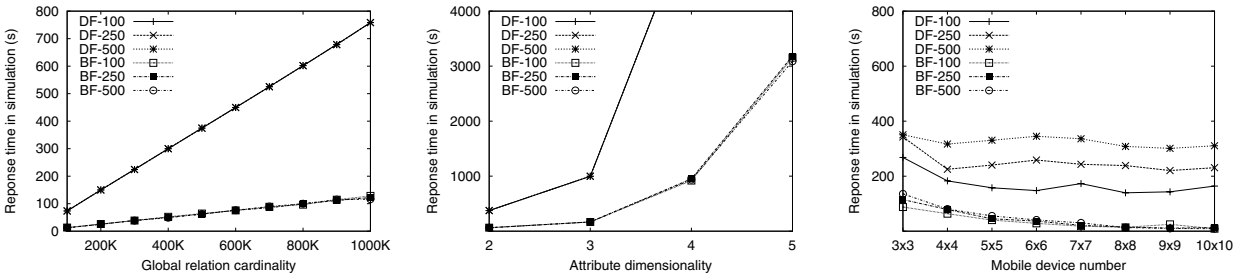
The pre-tests in the static setting suggest that the use of estimated versus exact selections of filtering tuples leads only to slight differences, especially for uniform datasets. It is also seen that dynamic filtering of tuples yields better DRRs. Thus, we use only under-estimation of dominating regions when selecting filtering tuples in the simulation, and dynamically update them between mobile devices, if possible. The same series of datasets are used in the simulation as in the pre-tests. The DRR results are shown in Figures 8 and 9, where DF (BF) is for the depth-first (breadth-first) query forwarding strategy and the integers are the distances of interest in queries.

For both distributions, DRRs are lower compared to those in the static setting. The is attributed to the MANET setting, where not all devices always participate in the query processing, thus decreasing the data reduction. The mobile characteristic also makes the DRR changes for increasing global cardinalities less stable. This is because the part(s) of the global relation that do not participate in the query processing vary. The DRR change in terms of attribute dimensionality is still pronounced, which indicates that dimensionality still plays an important role in the query processing performance in MANETs.

5.2.3 Response Time

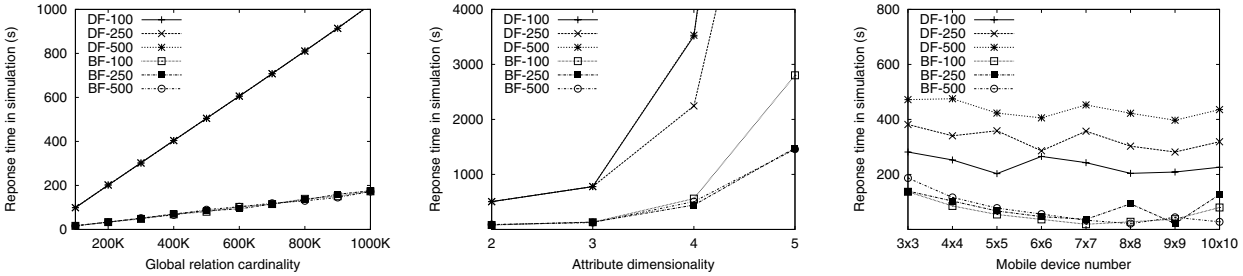
For the BF strategy, the response time is defined as the elapsed time from the moment that a query is issued at a mobile device M_{org} to the moment that 80% of the other devices in the network have sent back results, since in a wireless ad hoc network, it is not ensured that all devices are always reachable and available. For the DF strategy, the response time is defined a bit differently. Here, a query ends when the originator receives the result and finds that all its neighbors have processed the query. The simulation results are shown in Figures 10 and 11, covering independent and anti-correlated datasets, respectively. The response time consists of both the wireless transfer time and the local processing time on each device. Based on the results from Section 5.1, we estimated the local processing costs in the simulation and added them to the communication delays gained in the MANET simulator to obtain the total response time.

From the figures, we see that BF exhibits shorter response times than does DF. The most important explanation is that the BF query forwarding strategy enables parallel query processing among the mobile devices, while the DF strategy only allows each query to be processed serially along all devices involved. Another reason that leads to a



(a) RT vs. global cardinality (b) RT vs. attribute dimensionality (c) RT vs. mobile scale

Figure 10. Response Time on Independent Datasets in MANET Simulation



(a) RT vs. global cardinality (b) RT vs. non-spatial dimensionality (c) RT vs. mobile network scale

Figure 11. Response Time on Anti-Correlated Datasets in MANET Simulation

marginal difference is that we only count 80% results back in computing the response time for BF, whereas DF needs to wait longer before a query stops.

DF deteriorates much more quickly than does BF when the dimensionality increases, as shown in Figures 10(b) and 11(b). Local skyline processing over multi-dimensional datasets is time consuming on resource-constrained mobile devices. BF offsets that effect through parallelism; in contrast, DF is only hurt by that effect because of serialization.

BF improves as the number of mobile devices increases, as shown in Figures 10(c) and 11(c). This is because more devices increases the degree of parallelism of BF. The distance constraints make a more obvious difference to DF than to BF. This is also attributed to DF's serialization, which is more sensitive than parallelism to distance constraints, as larger search ranges usually involve more devices and data.

5.2.4 Query Message Count

In the simulation we found that the cardinality, the dimensionality, and the distribution have little impact on the message count. Therefore, we only show in Figure 12 how the message count varies as the number of mobile devices increases. Although BF shows better performance than does DF in terms of response time, this gain is not for free. Parallelism generates and forwards more messages in the wireless network, which in turn consumes more wireless communication bandwidth. As a result of this effect, the improvement of response time slows down in our simulation

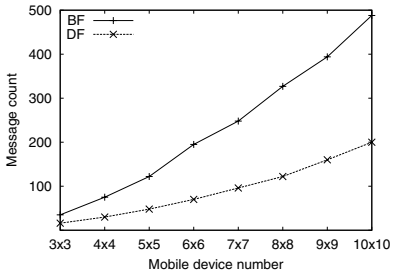


Figure 12. Query Message Count

(shown in Figures 10(c) and 11(c)) as the number of mobile devices increases.

6. Related Work

One area with related work is that of skyline querying. Borzonyi et al. [9] introduce the skyline operator into database systems with algorithms *Block Nested Loop* (BNL) and *Divide-and-Conquer* (D&C). Chomicki et al. [12] propose a *Sort-Filter-Skyline* (SFS) algorithm as a variant of BNL. Tan et al. [21] propose two progressive algorithms: *Bitmap* and *Index*. The former represents points in bit vectors and employs bit-wise operations, while the latter utilizes data transformation and B^+-tree indexing. Kossmann et al. [15] propose a *Nearest Neighbor* (NN) method. It identifies skyline points by recursively invoking R^*-tree based depth-first NN search over different data portions.

Papadias et al. [19] propose a *Branch-and-Bound Skyline* (BBS) method based on the best-first nearest neighbor algorithm [13]. All works above assume centralized data storage. In contrast, we assume that data is distributed horizontally among multiple mobile devices. This also differs from [6] where different dimensions are stored at different web sites.

Another area with related work concerns mobile P2P, where P2P and MANETs are combined into a new and interesting area. Kortuem et al. [14] describe scenarios where mobile devices can exchange data when they encounter each other. Budiarto et al. [11] mainly discuss strategies for data replication in a mobile P2P environment. Xu et al. [23] cover systemic topics on data management in mobile P2P networks. The same ideas have been applied to disseminate spatio-temporal resource information in mobile P2P networks [22]. Lindemann et al. [16] propose a distributed document search service for applications in mobile ad hoc networks. With controlled message forwarding and local caching, their method avoids flooding messages throughout the network. For a quality presentation of the state of the art on mobile ad hoc networking, readers are referred to a recent book [7].

7. Conclusion and Future Work

Assuming a setting with mobile devices communicating via an ad hoc network, this paper studies skyline queries that involve spatial constraints. Each mobile device contains some portion of the data against which the queries are issued. To reduce the communication cost, a distributed query processing strategy is proposed that takes advantage of the skyline dominance relationship to eliminate non-qualifying intermediate tuples, thus reducing the amount of data transmitted. On each mobile device involved, the local query processing is optimized using a hybrid storage model for the tuples, which have both spatial and non-spatial attributes. Extensive experimental studies demonstrate the efficiency of the methods, in terms of both communication cost savings and response time.

One research direction is to generalize the filtering idea, using more than one filtering tuple. Important questions include how many, and which, tuples should be used as filters, to achieve the best data reduction rate. Another direction is to extend the current strategies to retain good performance while incorporating the redistribution of local relations due to device mobility.

Acknowledgments This work is supported by the *SpADE* (A SPatio-temporal Autonomic Database Engine for location-aware services) [2] project, which studies the provisioning of mobile users with location-based data, e.g., traffic data, maps, and points of interest.

Christian S. Jensen is also an adjunct professor at Agder University College, Norway.

References

[1] JiST/SWANS. http://jist.ece.cornell.edu.

[2] SpADE. http://spade.ddns.comp.nus.edu.sg/spade.

[3] SuperWaba. http://www.superwaba.com.

[4] Fusing ad hoc and P2P. *Pictures of the Future (Siemens Magazine for Research and Innovation)*, Spring, 2005.

[5] A. Ammann, M. Hanrahan, and R. Krishnamurthy. Design of a memory resident DBMS. In *Proc. IEEE COMPCON*, pp. 54–57, 1985.

[6] W.-T. Balke, U. Güntzer, and J. X. Zheng. Efficient distributed skylining for web information systems. In *Proc. EDBT*, pp. 256–273, 2004.

[7] S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic, editors. *Mobile Ad Hoc Networking*. Wiley-IEEE Press, New Jersey, 2004.

[8] C. Bobineau, L. Bouganim, P. Pucheral, and P. Valduriez. PicoDBMS: Scaling down database techniques for the smartcard. In *Proc. VLDB*, pp. 11–20, 2000.

[9] S. Borzonyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. IEEE ICDE*, pp. 421–430, 2001.

[10] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proc. MOBICOM*, pp. 85–97, 1998.

[11] Budiarto, S. Nishio, and M. Tsukamoto. Data management issues in mobile and peer-to-peer environments. *Data Knowl. Eng.*, 41(2-3): 183–204, 2002.

[12] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *Proc. IEEE ICDE*, pp. 717–719, 2003.

[13] G. Hjaltason and H. Samet. Distance browsing in spatial database. *ACM TODS*, 24(2): 265–318, 1999.

[14] G. Kortuem, J. Schneider, D. Preuitt, T. G. C. Thompson, S. Fickas, and Z. Segall. When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad hoc networks. In *Proc. P2P*, pp. 75–94, 2001.

[15] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *Proc. VLDB*, pp. 275–286, 2002.

[16] C. Lindemann and O. P. Waldhorst. A distributed search service for peer-to-peer file sharing in mobile applications. In *Proc. P2P*, pp. 73–80, 2002.

[17] H. Luo, R. Ramjee, P. Sinha, L. E. Li, and S. Lu. UCAN: a unified cellular and ad-hoc network architecture. In *Proc. MOBICOM*, pp. 353–367, 2003.

[18] J. Macker and M. Corson. *Mobile Ad Hoc Networking*, chapter Mobile Ad Hoc Networks (MANETs): Routing Technology for Dynamic, Wireless Networking. Wiley-IEEE Press, New Jersey, 2004.

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

[20] R. Sen and K. Ramamritham. Efficient data management on lightweight computing device. In *Proc. IEEE ICDE*, pp. 419–420, 2005.

[21] K. Tan, P. Eng, and B. Ooi. Efficient progressive skyline computation. In *Proc. VLDB*, pp. 301–310, 2001.

[22] O. Wolfson and B. Xu. Opportunistic dissemination of spatio-temporal resource information in mobile peer-to-peer networks. In *Proc. PDMST*, pp. 954–958, 2004.

[23] B. Xu and O. Wolfson. Data management in mobile peer-to-peer networks. In *Proc. DBISP2P*, pp. 1–15, 2004.

[24] C. T. Yu and C. C. Chang. Distributed query processing. *ACM Computer Surveys*, 16(4): 399–433, 1984.