Randomized Multi-pass Streaming Skyline Algorithms*

Atish Das Sarma† Ashwin Lall Danupon Nanongkai‡ Jun Xu§
Georgia Institute of Technology
{atish, alall, danupon, jx}@cc.gatech.edu

ABSTRACT

We consider external algorithms for skyline computation without pre-processing. Our goal is to develop an algorithm with a good worst case guarantee while performing well on average. Due to the nature of disks, it is desirable that such algorithms access the input as a stream (even if in multiple passes). Using the tools of randomness, proved to be useful in many applications, we present an efficient multi-pass streaming algorithm, RAND, for skyline computation. As far as we are aware, RAND is the first randomized skyline algorithm in the literature.

RAND is near-optimal for the streaming model, which we prove via a simple lower bound. Additionally, our algorithm is distributable and can handle partially ordered domains on each attribute. Finally, we demonstrate the robustness of RAND via extensive experiments on both real and synthetic datasets. RAND is comparable to the existing algorithms in average case and additionally tolerant to simple modifications of the data, while other algorithms degrade considerably with such variation.

1. INTRODUCTION

The skyline of a d-dimensional dataset is the set of points (tuples) that are not *dominated* by any other point, where we say that a point p dominates another point p' if the coordinate of p on each dimension is not smaller than that of p', and strictly larger on at least one dimension. A popular example is a hotel reservation system. Consider a hypothetical scenario in which a tourist is searching for a hotel that both is cheap and has high quality. Although most hotels that have higher quality tend to be more expensive, there could be real-life instances in which a hotel A is more expensive than a hotel B but B has better quality than A. Clearly, in this scenario, inferior hotels such as A should not be shown to this tourist. For example, Figure 1 shows the first 5 query results of hotels in Lyon from August 24 to August 28, 2009, when we search `hotels.com` in March 2009. Should `hotels.com` support skyline query and someone execute the query `[Select *, From Hotels, Skyline of Price min, Guest_Rating max]`, then hotels 2, 4, 5 should not be shown as they are dominated by hotel 1 (the latter has better guest rating and lower price); Only 1 and 3 should be on the skyline, as one has a better guest rating and the other has a lower price.

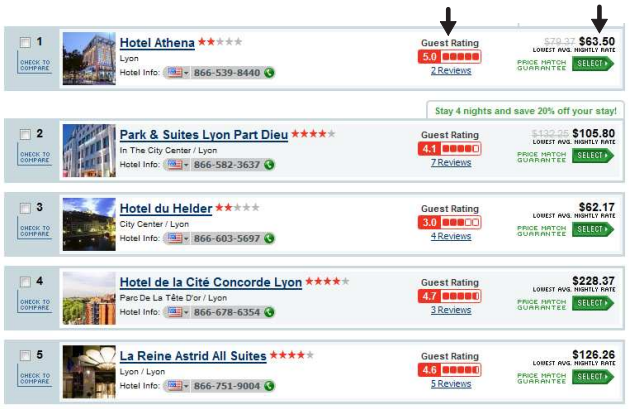


Figure 1: Search result from `hotels.com`.

Since a database is usually too large to be stored in the main memory, skyline algorithms are external [4] in the sense that data resides on the external memory (i.e., disk) while its processing happens in the main memory. Skyline algorithms can be classified into two categories: with and without pre-processing. Pre-processing such as indexing [18] and sorting [8] helps speed up the skyline computation, but maintaining indices over a large number of dimensions could be computationally intensive. Without pre-processing, skyline algorithms have to take at least one pass over the database for computing the skyline and therefore are often slower than those with pre-processing. However, their flexibility and wider applicability makes them attractive alternatives in many application scenarios [10]. In this paper, we focus on the skyline algorithms without pre-processing.

In this work, we appeal to the concept of optimizing the

*Author names are in alphabetical order.

†This work is supported in part by Georgia Tech ARC Fellowship and by Richard J. Lipton
‡This work is supported in part by Georgia Tech ACO fellowship and by Richard J. Lipton.
§This work is supported in part by NSF grants CNS-0626979, CNS-0716423, CNS-0905169, and CNS-0910592

VLDB '09, August 24-28, 2009, Lyon, France

worst case behavior of skyline algorithms. There are many real-time applications in which one would be interested in upper-bounding the times taken in skyline computation and minimizing its variance. Consider a flight booking site where the site searches for results, when presented with a query, and displays a skyline set of options. If the time taken for this page to load is longer than usual, the user is likely to abandon the search and try one of many other such sites. Another example is skyline queries on stock markets, where prices change very rapidly. If a user (or automatic trading software on behalf of the user) wants to perform a transaction based on the results from the skyline query, presenting the results reliably fast is essential.

In this work, we propose a set of multi-pass data streaming algorithms that can compute the skyline of a massive database with strong worst-case performance guarantees (motivated above). The data stream in this context refers to the data items in the massive database (residing on disks) that are read into and processed through the main memory in a stream fashion. Our algorithms have a salient feature: They are not sensitive (performance wise) to the order in which the data items show up in the stream. Therefore, we can simply read the data items out from the database in the "physically sequential order" in the sense that disk heads only need to move in one direction during each stream pass. Since the seek and rotational delays of a disk are orders of magnitude larger than the transmission delay, this feature may translate into significant performance improvements in real-world systems.

Even with physically sequential accesses of disks, however, each pass over the database is still quite time-consuming. Therefore, our skyline algorithms need to minimize the number of such passes in order to have good performance. This objective is achieved using randomization[1], which turns out to be a powerful tool in quickly eliminating a large number of non-skyline points from consideration after each pass.

Our Contributions

Our contributions can be summarized as follows.

- We formalize the multi-pass streaming model (implicitly with physically sequential accesses of disks) for the skyline problem. We are interested in worst case analysis, in terms of random and sequential I/O's and comparison operations. We prove a simple yet instructive performance lower bound under this model. (Section 3)

- The key contribution of this paper is a randomized multi-pass streaming algorithm, RAND. We present two versions, one with and one without fixed window, and prove their theoretical worst-case performance guarantees. These performance guarantees, combined with the aforementioned lower bound, shows that RAND algorithms are near-optimal. (Section 4)

- RAND can be extended to many other settings: (1) It can be adapted to the distributed model, where the goal is to minimize the communication when different sites have different parts of the input. (2) It extends to a deterministic variant that works for the two-dimensional case. (3) It works even for partially-ordered domains. (Section 5)

- We perform extensive experiments on real and synthetic data, which show that RAND is comparable to the state-of-the-art skyline algorithms in various performance metrics. We also show that, with certain perturbations of the data orders, the performance of the other algorithms degrade considerably while RAND is robust to such changes. (Section 6)

[1]Our technique uses randomness in the algorithm but does not assume any distribution on the input.

2. RELATED WORK

Skyline computation, previously known as Pareto sets, admissible points, and maximal vectors, has been extensively studied in the theory and mathematics community since 1960s (see, e.g., [3, 14, 16]). However, they assume that the whole input data can be fit in the internal memory. Therefore, these algorithms do not scale well to large databases. In particular, any scalable algorithm should be *external*. The problem in this setting, and the name *skyline*, were introduced to the database community by Börzsönyi et al. [4] and has been studied extensively since then.

As mentioned in the introduction, external skyline algorithms can be classified into two categories: with and without pre-processing. Each category has its own advantages and disadvantages. Algorithms presented in this paper are in the latter category. We review some state-of-the-art algorithms in this category that we compare against: BNL and LESS. Both algorithms serve as good benchmarks for comparison as they perform well with respect to different parameters.

BNL [4] operates with a memory window of size w. It makes several passes over the data, each time storing the first w points that are undominated. While reading the disk, any point that is dominated is eliminated so that it is never read in the future. BNL has a timestamping mechanism that is used to determine when a point is in the skyline and when all points it dominates have been eliminated. It continues to make passes over the data until all skyline points have been obtained. BNL remains to be a classic algorithm in that several other algorithms use BNL or a modification of it as a subroutine.

LESS [10] is an extension of the SFS algorithm [8]. SFS assumes that the data is pre-processed by sorted according to a scoring function. Once sorted, a BNL-like filtering procedure can then be performed on the data to get the skyline points. Sorting the data gives the desirable property that, as soon as a point gets added to the buffer, it can be output as being in the skyline and does not need any timestamping overhead. The authors of [8] suggest the use of the entropy function to efficiently eliminate many tuples. LESS eliminate some more points while sorting and integrates the final pass of the external sorting phase with the first filter pass.

Another category of skyline algorithms that we do not consider here are those with pre-processing. The main feature of these algorithms is that they can compute skyline without going through the whole input data; thus, they are *progressive*. Most of the algorithms in this category are index-based and exploit R-tree and its variations to obtain good performances. The first algorithm in this category is the nearest neighbor (NN) algorithm in [13] and the state-of-the-art algorithm in this category is BBS [18] which is I/O-optimal. As mentioned in the introduction, we do not

consider this category in this paper.

Several other variants have been considered. Since it is not possible to list a complete survey of all papers, we mention a few here. Algorithms using the bitmap method [25] and for partially-ordered attributes to capture dynamic user preferences [6, 5, 32, 22] , computing cardinality or exploiting low cardinality domains [7, 17], sliding window or time-series skyline queries [15, 27, 12, 35], distributed and super-peer architectures [2, 31, 36, 19], representative skylines [26], probabilistic skylines on uncertain data [20] have been studied. We mention the streaming and distributed related works again in Sections 3 and 5.1 respectively.

3. MULTI-PASS STREAMING MODEL FOR SKYLINE ALGORITHMS

In this section, we define the streaming model for skyline computation and present a lower bound for the problem.

Since most databases today are too large to fit in main (internal) memory, they are typically stored in external memory on one or more magnetic disks. In this type of memory, sequential disk access is preferable to random disk access for several reasons. First, sequential disk access is considerably faster than the random disk access as the latter involves a number of seek operations. For example, algorithms for spatial join that access pre-existing index structures (and thus do random I/O) can often be slower in practice than algorithms that access substantially more data but in a sequential order (as in streaming) [1]. Second, sequential access has the advantage of using modern caching architectures optimally, making the algorithm independent of the block size (i.e., *cache-oblivious*) [9]. For these reasons, the models designed to capture magnetic disks have to distinguish the two types of memory access.

There have been many practical models proposed in the literature. Well known models include the *parallel disk model* (PDM) and the *multi-pass streaming model* [30, 21]. In this paper, we aim at exploring the power and limitation of the latter model in the context of skyline algorithms. In this model, the data is streamed sequentially through internal memory. Moreover, it is possible that the data is streamed multiple times; we refer to each scan through the dataset as a *pass*. Additionally, we allow our algorithms to produce a new stream while they are reading the current stream data. This is a common method used in many skyline algorithms: Read data from one file and write a new file if necessary. We note that this can be implemented with only sequential disk access when there are at least two disks available. Such a model has recently been defined as a Read/Write streaming model [23, 24] and has received considerable attention from the theory community.

There has been work on computing skylines in a streaming setting [15, 27, 12, 35]. However, these works look at single-pass streams under the sliding window model, whereas we are interested in multi-pass algorithms. A single pass is too restrictive for computing skylines as we demonstrate in the following section.

3.1 How hard is it to stream?

In this section, we show that it is impossible to design an efficient algorithm that reads each point *exactly once*. We note, however, that our algorithms described in the next section can complete in very few passes.

THEOREM 1. *Consider any $m \geq 3$ and any $n \geq m$ and consider any deterministic or randomized single pass streaming algorithm $\mathcal{A}$ that always stores fewer than $n/2$ points. There exists an input on $n+m$ points with m skyline points such that $\mathcal{A}$ fails with probability at least $1/2$.*

Notice that the theorem asserts a lower bound for every m, in particular, even when m is constant and much less than n.

PROOF. We prove the theorem for $m = 3$ and it naturally generalizes to larger m. Construct a distribution over sets of points from $\mathbb{R}^2$, $X_1, X_2, \ldots, X_n$. Each of them contains the following n points $\{(1,n),(2,n-1),(3,n-2),\ldots,(n-1,2),(n,1)\}$. In addition, each X_i contains two more points $\{(i-1,n+1),(n+1,n-i)\}$. Algorithm $\mathcal{A}$ is presented with X_j where j is chosen uniformly at random from $\{1,2,\ldots,n\}$.

Notice that the skyline of X_i is precisely the set of three points $\{(i-1,n+1),(i,n-i+1),(n+1,n-i)\}$. If $\mathcal{A}$ stores only s points, it is able to store only s points of the first n. Since the last two points determine which of the first n is in the skyline, and there is an equal probability of any one of the first n points being in the skyline, $\mathcal{A}$ fails with probability at least $\frac{n-s}{n}$. The result follows for deterministic algorithms.

We now use Yao's minimax principle [34] to prove the same lower bound for randomized algorithms. Yao's principle states that the expected cost of any randomized algorithm for solving a given problem, on the worst case input for that algorithm can be no better than the expected cost for a worst-case random probability distribution on the inputs of the deterministic algorithm that performs best against that distribution. In the construction above, we had a probability distribution of inputs such that *every* deterministic algorithm (and therefore the best deterministic algorithm) failed with probability at least 1/2. Therefore, for any randomized algorithm, there is a worst-case input such that in expectation it fails at least half the time. □

In the next section, we present our main algorithms for computing skylines under the streaming and related models.

4. ALGORITHMS

In the rest of this paper, we use n for the number of points, m for the number of skyline points, and d for the number of dimensions of each point. We measure the performance of algorithms in terms of random I/O's, sequential I/O's, and comparisons performed. The random I/O's is simply the number of passes performed by an algorithm in the streaming model. In this section, we present randomized algorithms for the following different settings:

- STREAMING-RAND algorithm: This algorithm gives an efficient tradeoff between the memory space and the number of passes (random I/O's). In the worst case, it uses $O(m \log n)$ space, $O(\log n)$ passes (random I/O's), $O(n \log m)$ sequential I/O's and $O(dmn \log m \log n)$ comparisons with high probability.
- FIXEDWINDOW-RAND algorithm: This algorithm runs using a fixed memory space; i.e., for a predetermined window size w, it always stores at most w points. We use this algorithm to compare performance with previous algorithms in the literature in Section 6. With window size w, this algorithm uses $O(m \log n/w)$ random

I/O's, $O(mn/w)$ sequential I/O's and $O(dmn)$ comparisons in expectation. The high probability bounds are only $O(\log n)$ more than the expected bounds presented above. We also present them in this section.

Notice that the worst case is over all inputs, and the randomization (and therefore high probability or expectation bounds) is for the algorithms' coin tosses. In other words, our algorithms have the guarantees for *any* kind of input.

Outline

The main idea is to quickly find skyline points that dominate many points in a few of passes. In particular, we present an algorithm that finds a set of skyline points which dominate about *half* of the input points in three passes, using memory space $O(m)$. The main idea is that such skyline points are easy to find by sampling: If we pick one input point uniformly at random and find a skyline point that dominates this point, then we are likely to find a skyline point that dominates more points than another. The main goal of our sampling technique is to be able to sample skyline points. However, there are two difficulties: it is not obvious how to sample a skyline point from all skyline points (as these are not known to the algorithm). Further, sampling skyline points uniformly does not necessarily ensure dominating a lot of points. We wish to sample skyline points in proportion to the number of points they dominate. It turns out that sampling roughly $24m$ points is sufficient to find skyline points that dominate at least $\frac{3n}{4}$ points, in expectation. We now describe the algorithms and their analysis in details.

4.1 Key idea: Eliminating points in a few passes

We start with the following simple question: If m is known, how many points can we dominate using a constant-pass streaming algorithm with about m space? In this section we answer this question. We present a simple three-pass, $24m$-space algorithm, ELIMINATE-POINTS (cf. Algorithm 4.1), that guarantees to eliminate at least $\frac{3n}{4}$ elements in expectation. The tricky part here is analyzing its performance. We later extend this algorithm to other cases.

Algorithm 1 ELIMINATE-POINTS (m)

Input: $p_1, p_2, ..., p_{n'}$ (in order) where n' is the number of points in the stream.
Output: Skyline points S'

1: Let $x = 24m$.
2: **Pass 1:** For $j = 1, 2, ..., x$, let p'_j be a point picked uniformly at random from the stream. Let S be the set of such points.
3: **Pass 2:**
4: **for** $i = 1..n'$ **do**
5: For any p'_j, if p_i dominates p'_j then $p'_j := p_i$
6: **end for**
7: Let $S' = \{p'_1, p'_2, ...p'_x\}$.
8: **Pass 3:** Delete from stream all points in S' and all points dominated by any point in S'.
9: **return** S'

The main idea of the ELIMINATE-POINTS algorithm is to sample $24m$ points from the stream. (This sampling can be done in one pass using *reservoir sampling* [29].) Next, we spend one more pass to replace these sampled points by points that dominate them. We note that it is important to do this in order (as it is crucial in the analysis of Lemma 2). Points obtained at the end of this pass are skyline points (although not necessarily distinct). In the last pass, we delete points we obtain from the second pass and any points dominated by them.

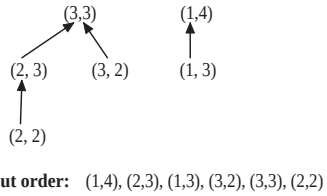


Figure 2: Example from the proof of Lemma 2.

Now we analyze the algorithm. First, we analyze the performance in expectation. To simplify the analysis, we assume that the sampling is done *with replacement*; that is one point may be sampled more than once. The performance of the algorithm will be better if we sample without replacement.

LEMMA 2. *After* ELIMINATE-POINTS *algorithm with parameter m, the expected number of points left in the stream is at most $n'/4$, where n' is the number of points in the input stream.*

PROOF. First, we construct the following directed graph, denoted by G. The goal here is to allocate each point to a unique skyline point that dominates it (note that there could be multiple skyline points that dominate a given point). This graph consists of n' vertices corresponding to points in the stream. We abuse notation and denote the vertices by $p_1, p_2, ..., p_{n'}$. It will be clear from the context whether we refer p_i as a point in the stream or a vertex. For each i, we draw an edge from p_i to p_j if and only if p_j is the first point (leftmost) in the stream that dominates p_i. Figure 2 shows an example. (In the example, notice that $(2, 2)$ points to $(2, 3)$ but not $(3, 2)$ since $(2, 3)$ is the first point in the stream that dominates $(2, 2)$. Also note that if $(2, 2)$ is sampled then it will be replaced by $(2, 3)$ and $(2, 3)$ will be later replaced by $(3, 3)$. Therefore, we get $(3, 3)$ as a product.)

Now, let $q_1, q_2, ..., q_m$ be the skyline points. For $i = 1, 2, ..., m$, define $S_i = \{p_j \mid$ There is a path from p_j to $q_i\}$. (These sets are disjoint and their union is the set of all points in the stream.)

CLAIM 3. *If a point p is sampled in Pass 1 then the skyline point q_i is in S' after the iteration ends where i is such that $p \in S_i$.*

Proof of this claim is placed in appendix. The main observation to make is that the sampled point p is in fact replaced, one by one as the stream progresses, by points on the path from p to q_i.

By Claim 3, a point will remain in the stream after the algorithm finishes only if none of the points in the set S_i containing it are picked in Pass 1. In other words, the number of points left in the stream is at most $\sum_{i:\ S_i \cap S = \emptyset} |S_i|$ (recall that S is the set of points sampled in Pass 1). We now bound the expected value of this quantity.

CLAIM 4. $\mathbb{E}[\sum_{i:\ S_i \cap S = \emptyset} |S_i|] \leq (n'/4)$.

We mention the intuition behind the proof here and reserve the technical details for the appendix. First, notice that the bigger S_i is, the more likely a point in S_i will be sampled. Consequently, it is more likely that we end up with a skyline point q_i as compared to q_j if $|S_i| > |S_j|$. This is the key insight in the algorithm, as the random sampling biases the algorithm towards skyline points that are likely to dominate a larger number of points.

However, can we argue that at most $\frac{n'}{4}$ are left? Consider all S_i such that $|S_i| \geq \frac{n'}{8m}$, i.e., the *large* S_i's. For any of these large sets, its size is $1/8m$ fraction of all the points. Therefore, sampling at least $8m$ points will get us at least one point in this set in expectation. However, we may not find samples from all large sets. A clever analysis can be used to show that we eliminate a large fraction of points in these large sets. Finally, a counting argument shows that the total number of points in these sets combined is a large fraction of all the points (to be precise, it is at least $\frac{7n'}{8}$).

This completes the intuition of the proof. The lemma follows immediately from Claim 4. □

4.2 Streaming algorithm

Now we develop a time/space-efficient streaming algorithm for finding all skyline points. We first give a high level idea of the algorithm. Let us focus on the number of passes for now. The basic idea is to apply the ELIMINATE-POINTS algorithm repeatedly until no points are left in the stream. If m (the number of skyline points) is known ahead of time then this process is likely to finish in $O(\log n)$ steps, where n is the number of input points. This is because we are likely to delete half of the points in each application of ELIMINATE-POINTS.

However, the difficulty is that m is not known. One way to get over this is to use the standard doubling trick to find m: Start with $m' = 1$ (as the *guess* for the number of skyline points), run ELIMINATE-POINTS for $O(\log n)$ steps assuming that $m = m'$. If the problem is not solved, double the size of m' and repeat. Since we will be done (with high probability) when $m' \geq m$ we have to repeat only $\log m$ steps (with high probability). Therefore, this algorithm is likely to stop in $O(\log n \log m)$.

To get the number of passes to $O(\log n + \log m)$, we exploit Lemma 5 further. By such Lemma, if $m' \geq m$ then we are likely to eliminate half of the points in every pass. Therefore, if we find out that the algorithm eliminates less than such expected fraction, we double m' immediately instead of waiting for another $\log n$ passes. The algorithm is stated as the STREAMING-RAND algorithm (cf. Algorithm 2).

We now give a formal analysis of this algorithm. The efficiency of the algorithm relies on the fact that we are likely to stop before m' gets too big, as shown in the following two lemmas.

LEMMA 5. *After* ELIMINATE-POINTS *algorithm with parameter m, at most $n'/2$ points are left in the stream with probability at least $1/2$, where n' is the number of points in the input stream.*

PROOF. Let X be the number of point left in the stream after ELIMINATE-POINTS algorithm. Recall from Lemma 2 that $\mathbb{E}[X] \leq \frac{n'}{4}$. By the Markov's inequality, $Pr[X \geq \frac{n'}{2}] \geq Pr[X \geq 2\mathbb{E}[X]] \leq 1/2$. □

Algorithm 2 STREAMING-RAND

1: Let n be the number of points in the input stream. Let $m' = 1$.
2: **while** the input stream is not empty **do**
3: Let n' be the current number of points in the stream
4: Call ELIMINATE-POINTS($m' \log(n \log n)$)
5: If more than $n'/2$ points are left in the stream, $m' = 2m'$.
6: **end while**

Remark: In case the stream cannot be changed, we do not have to actually delete points from stream. We only keep the skyline points found so far and consider only points in the stream that is not dominated by any found skyline points.

LEMMA 6. *The probability that the algorithm repeats until $m' \geq 2m$ is at most $1/n$.*

PROOF. Let τ be the first iteration that m' is at least m; that is, $m \leq m' < 2m$. Consider any iteration after τ. Recall that by Lemma 5, each run of ELIMINATE-POINTS(m), which samples $24m$ points, halves the stream with probability at least $1/2$. It follows that each run of ELIMINATE-POINTS($m \log(n \log n)$), which samples $24m \log(n \log n)$ points, halves the stream with probability at least $1 - 1/(n \log n)$. By union bound, the probability that all $\log n$ iterations after τ delete at least half the points is at least $1 - 1/n$. This implies that, with probability $1 - 1/n$, the stream will be empty before m' is increased again. □

Now we analyze the algorithm in all aspects.

THEOREM 7. STREAMING-RAND *algorithm (cf. Algorithm 2) uses with probability at least $1 - 1/n$,*

1. *$O(m \log n)$ space*
2. *$O(\log n)$ random I/O's (passes)*
3. *$O(n \log m)$ sequential I/O's, and*
4. *$O(dmn \log n \log m)$ comparisons.*

PROOF. By Lemma 6, $m' < 2m$ with probability at least $1 - 1/n$. We show that the theorem holds when this happens.

For the first claim, when $m' < 2m$ the space becomes $O(m \log(mn \log n)) = O(m \log n)$ as claimed.

For the second claim, observe that the algorithm in each iteration either scales m' up twice or scales n' down by half. It can scale down n' for only $\lceil \log n \rceil$ times and it scales m' up for only $\lceil \log m \rceil$ times (before $m' \geq 2m$). We thus prove the second claim.

For the third claim, first we count the number of sequential I/O's made by iterations that increases m'. There are $\lceil \log m \rceil$ such iterations (if $m' < 2m$) and each iteration reads through the stream three times. So, the total number of sequential I/O's used by these iterations is $O(n \log m)$. For the number of sequential I/O's used by the remaining iterations, observe that the size of the stream is at most $n/2^{i-1}$ in the i-th such iteration (because such iterations scale the size of the stream down by half). Therefore, the number of sequential I/O's for this m' is $O((1 + 1/2 + 1/2^2 + ...)n) = O(n)$. We thus prove the third claim.

For the last claim, observe that when the algorithm reads a new element from the stream, it compares this element with all elements in the memory; i.e., it compares $O(m \log n)$

pairs of points per sequential I/O. Further, comparison of any pair of points requires comparing d different dimensions in worst case. The claim thus follows from the third claim that there are $O(n \log m)$ sequential I/O's. □

Notice that this is a near-optimal trade-off between space and passes of any streaming algorithm, since our lower bound shows that with one pass, any algorithm requires $\Omega(n)$ space. We increases the passes by only a logarithmic factor and get almost optimal space bounds.

4.3 Fixed-window algorithm

Many of the previous skyline algorithms are presented with predetermined window size w. That is, the algorithm is allowed to store only $O(w)$ points in the stream; this might be a constrained due to the memory specifications of the machine. The goal is to analyze a variant of the algorithm for the numbers of random I/O's (passes), sequential I/O's, and comparisons under this setting. We show that the following very simple algorithm is efficient:

FixedWindow-RAND: While the stream is not empty, call Eliminate-Points($\lfloor w/24 \rfloor$).

Now we analyze the algorithm. We first state the theorem in terms of expectation, and then the high probability bound.

Theorem 8. *In expectation,* FixedWindow-RAND *algorithm uses* $O(\frac{m}{w} \log n)$ *random I/O's,* $O(\frac{nm}{w})$ *sequential I/O's, and* $O(dnm)$ *comparisons.*

Proof. Consider running Eliminate-Points($\lfloor w/24 \rfloor$) for $\lceil 24m/w \rceil$ times. We claim that this process is as efficient as running Eliminate-Points(m). (In other words, the distribution of the running time of the former process stochastically dominates that of the latter one.) Intuitively, this is because both processes get the same number of samples but the former process does not sample points that are dominated by the previous samples. We now analyze the process of running Eliminate-Points(m).

The key idea is to consider the runs of Eliminate-Points(m) that reduces the stream by at least half. Let us call these runs "success" runs and the rest runs "fail". Recall from Lemma 5 that each run succeeds with probability at least 1/2.

For the expected number of times we have to run Eliminate-Points(m), observe that we have to run the algorithm until we see $\log n$ success runs. Since each run succeeds with probability 1/2, the expected number of times we have to run Eliminate-Points(m) is at most $2 \log n$. Multiplying this number by $\lceil 24m/w \rceil$ gives the number of random I/O's claimed in the theorem statement.

For sequential I/O's, we claim that the expected number of fails between each pair of consecutive successes is 1. To be precise, for $i = 0, 1, ..., \log n$, let X_i be the number of fail runs between the i-th and $(i+1)$-th success runs. Since each run succeeds with probability at least 1/2, $\mathbb{E}[X_i] \leq 1$ for all i, as claimed. Now, observe that the number of sequential I/O's is at most $2n(X_0 + X_1/2 + X_2/4 + ...)$ since each success run halves the stream. Therefore, by the linearity of expectation, the expected number of sequential I/O's is $2n(\mathbb{E}[X_0] + \mathbb{E}[X_1]/2 + \mathbb{E}[X_2]/4 + ...) \leq 4n$. The expected number of sequential I/O's claimed in the theorem statement follows by multiplying the number by $\lceil 24m/w \rceil$.

Algorithm	Random I/O's	Sequential I/O's	comparisons
BNL(w)	$\Theta(\min\{m, \frac{n}{w}\})$	$\Theta(\min\{mn, \frac{n^2}{w}\})$	$\Theta(d \min(wmn, n^2))$
LESS(w)	$\Theta(n \log_w \frac{n}{w})$	$\Theta(\frac{mn}{w})$	$\Theta(dmn + n \log n)$
RAND(w)	$O(\frac{m \log n}{w})$	$O(\frac{mn}{w})$	$O(dmn)$

Table 1: Comparison of algorithms

For the number of comparisons, observe that we compare each element read from the stream with w elements in the window. Moreover, the number of elements read from the stream is bounded by the number of sequential I/O's which is $O(nm/w)$. Therefore, there are $O(nm)$ comparisons. Each vector comparison needs d comparisons. The theorem is thus completed. □

Theorem 9. *With high probability,* FixedWindow-RAND *algorithm uses* $O(\frac{m \log n}{w})$ *random I/O's,* $O(\frac{nm \log n}{w})$ *sequential I/O's, and* $O(dnm \log n)$ *comparisons.*

Proof. The random I/O's can be proven by Chernoff bounds (essentially arguing that in $2m \log n/w$ executions with w space with high probability, at least $m \log n/w$ executions result in eliminating half the points). Observe that in $O((m \ln(mn \log n))/w)$ passes we get samples equal to one round of eliminate-points. The lemma follows immediately from the previous section. An important point to note is that, in expectation, the number of sequential I/Os (and consequently comparisons) save a $\log n$ factor as there is no need to perform the doubling trick. □

4.4 Algorithm Comparisons

We compare performance of the FixedWindow-RAND algorithm (referred to as RAND from now on) in the worst case against BNL and LESS, two previously proposed non-preprocessing fixed-window algorithms. The asymptotic performance bounds are summarized in Table 1. Recall that n denotes the number of input points, m denotes the number of points in the skyline, and d denotes the dimension.

We analyze three parameters that affect the algorithms' performance: random I/O's, sequential I/O's, and the number of comparisons. Although, intuitively, random I/O's are those I/O's that need seek operations, this definition could be confusing sometimes as sequential I/O's can be sometimes counted as random I/O's. To avoid this confusion, we define a random I/O to be I/O that reverses the hard disk head: the previously read element appears after the newly read element on the disk.

The expected bound of RAND is shown in Theorem 8. For BNL and LESS, the upper bounds are already shown many times in the literature and the the formal proof of the lower bounds can be constructed by creating fairly simple tight examples. We omit them here for brevity.

As can be seen from the comparisons table, in all the algorithms, the number of sequential I/O's is significantly more than the number of random I/O's. However, the cost of a random I/O can be a lot more than the cost of a sequential I/O. Therefore, it is important to analyze these separately. In terms of random I/O's, LESS scales linearly with n, so is worse than BNL and RAND. Further, RAND is better than BNL with a saving of almost a w factor (the random I/O's of RAND are actually the minimum of what is written in the table for BNL and for RAND). In terms of sequential

I/O's, LESS and RAND are asymptotically the same, and better than BNL by a w-factor.

Apart from I/O's, comparisons made by algorithms is a key contributor to the overall performance. LESS has a sorting phase because of which it incurs $\Theta(n \log n)$ more comparisons than RAND. Further, BNL incurs roughly a w-factor extra comparison cost as compared to RAND (which is why BNL's cost increases when the window size is increased beyond a point).

These interpretations are corroborated by the results in our evaluations section.

5. EXTENSIONS

In this section, we discuss about extending our algorithms to various settings. First, we show that it can be extended to efficient distributed algorithms. Secondly, we show a deterministic variant of the algorithms for the case of 2-dimensions. Finally, we show that the algorithms can solve a more general problem on posets.

5.1 Extending to Distributed Algorithm

We extend our algorithm to the distributed setting, where the database is horizontally partitioned, i.e., the set of points is partitioned across several machines. The model assumes that there is a central machine (coordinator) that queries the other machines and outputs the final answer. While executing the algorithm, there is point to point communication between the coordinator and every other machine. In particular, the central machine may broadcast information to all other machines. A key objective, however, is to minimize such communication. See Zhu et al. [36] for an excellent survey and description of a similar distributed model.

We start by making ELIMINATE-POINTS algorithm (cf. 4.1) suitable for the distributed setting. The main difficulty is that Pass 2 relies on the input order. We modify Pass 2 slightly to make it *order-independent*, as follows. (Note, however, that this version loses its generality. For example, it cannot solve the problem on posets discussed in Section 5.3.)

Pass 2: For each p'_j, pick p_i in the stream that dominates p'_j and is *lexicographically* maximum (break tie by picking the first such points in the stream).

We call this algorithm LEX-ELIMINATE-POINTS. We claim that this algorithm also has the same bound guarantee as the ELIMINATE-POINTS algorithm (as in Lemma 5). Since the proof is essentially the same (with the definitions of "big sets" slightly modified), we omit the details.

Remark: The lexicographic order can be replaced by any *total* ordering that preserves domination. For example, if the input data is a set of vectors, we can pick p_i that dominates p'_j and has *maximum summation of values over all coordinates*, or *maximum entropy*, instead.

Now, let us consider the following distributed version of the LEX-ELIMINATE-POINTS algorithm.

To analyze the DISTRIBUTED-LEX-ELIMINATE-POINTS algorithm, consider a stream obtained by concatenating the streams from all machines (order the streams by the corresponding machines' IDs). It can be easily observed that the DISTRIBUTED-LEX-ELIMINATE-POINTS algorithm gives the same result as the LEX-ELIMINATE-POINTS algorithm on the stream defined above. We get the following theorem.

Algorithm 3 DISTRIBUTED-LEX-ELIMINATE-POINTS(m)

1: Let k machines be denoted by $\mathcal{M}_1, \mathcal{M}_2, ..., \mathcal{M}_k$. Let n'_i be the current number of points in $\mathcal{M}_i$, and denote points by $p^i_1, p^i_2, ..., p^i_{n'_i}$ (in order). Let $x = \lceil m \ln(mn \log n) \rceil$. Let $\mathcal{C}$ denote the central machine.
2: **Pass 1:** $\mathcal{C}$ finds x random numbers, in the range 1 to $\sum_{i=1}^{k} n'_i$, denoted by $s_1, s_2, ..., s_x$. For each $j = 1, 2, ..., x$, let i_j and s'_j be such that $s'_j + \sum_{z=1}^{i_j - 1} n'_z = s_j$. For each j, $\mathcal{C}$ sends s'_j to $\mathcal{M}_{i_j}$ and $\mathcal{M}_{i_j}$ responds with $p^{i_j}_{s'_j}$. Let $p_1, p_2, ..., p_x$ be the returned points.
3: **Pass 2:** $\mathcal{C}$ broadcasts $p_1, p_2, ..., p_x$ to all machines. For each $j = 1, 2, ..., x$, each machine returns the *lexicographically* maximum point in the stream that dominates p_j. For each j, $\mathcal{C}$ keeps the returned point that is lexicographically maximum (breaking ties by prioritizing machines with smaller ID). Let the set $S' = \{p'_1, p'_2, ..., p'_x\}$ denote points kept by $\mathcal{C}$.
4: **Pass 3:** $\mathcal{C}$ broadcasts S' . Each machine deletes from its stream all points in or dominated by S'.
5: return S'

LEMMA 10. *The* DISTRIBUTED-LEX-ELIMINATE-POINTS *algorithm uses $O(1)$ rounds of communication. In each round of communication, the central machine broadcasts $O(x)$ points and each of other machines sends $O(x)$ points to the central machine, where $x = \lceil m \ln(mn \log n) \rceil$. Moreover, after the algorithm finishes, at least half of the input data will be eliminated with probability at least $1 - 1/(n \log n)$.*

Using the above lemma, one can modify algorithms STREAMING-RAND and FIXEDWINDOW-RAND to get distributed algorithms. The number of random I/O's becomes the number of rounds and the memory space becomes the communication per round. We get the following results from Theorem 7 and Theorem 9.

THEOREM 11. *There is a distributed streaming skyline algorithm such that the central machine broadcasts and each of other machines sends $O(m \log n)$ points per round and, with high probability, finishes in $O(\log n)$ rounds with probability at least $1 - 1/n$.*

THEOREM 12. *For any w, there is a distributed streaming skyline algorithm such that the central machine broadcasts and each of other machines sends $O(w)$ points per round and, with high probability, finishes in $O(\frac{m}{w} \log n)$ rounds.*

5.2 Deterministic 2D algorithm

Among deterministic and randomized algorithms with the same performance, the former is preferable. We show that when the points have only two dimensions, there is an efficient deterministic skyline streaming algorithm. We note that previous algorithm [14] could not be adapted for sequential access since it requires sorting.

The main idea is to replace the ELIMINATE-POINTS by the deterministic algorithm called 2D-ELIMINATE-POINTS (cf. Algorithm 5.2).

The main component of the 2D-ELIMINATE-POINTS algorithm is *Greenwald-Khanna's approximate quantile summary* [11]: Suppose we read n numbers from stream $a_1, a_2, ..., a_n$. Let π be a sorting permutation of data; i.e., $a_{\pi(1)} \leq a_{\pi(2)} \leq$

Algorithm 4 2D-ELIMINATE-POINTS

1: Let n be the size of the stream
2: **Pass 1:** Compute ϵ-approximate quantile summary of the first coordinate using Greenwald-Khanna's algorithm [11] where $\epsilon = 1/4m$. From this summary, find $v_0, v_1, ..., v_m$, which are the 0-th, $(1/m)$-th, $(2/m)$-th, ..., (m/m)-th approximate quantiles respectively.
3: **Pass 2:** For any $0 \leq i \leq m$, let p'_i be a point such that $v_i \leq p'_i[1] < v_{i+1}$. Let $S = \{p'_0, p'_1, ..., p'_m\}$
4: **Pass 3:** For any p_i: **(1)** If p_i is dominates some points in S then delete those dominated points from S. **(2)** If (1) holds and p_i is not dominated by any points in S then add p_i to S.
5: **Pass 4:** Delete all points in S and all points dominated by any point in S from the stream.
6: **return** S

$... \leq a_{\pi(n)}$. For any $0 \leq \phi \leq 1$, the approximate ϕ-quantile is a number t that is not too far from $a_{\lceil \phi n \rceil}$; i.e., $a_{\pi(\lceil \phi n - \epsilon n \rceil)} \leq t \leq a_{\pi(\lceil \phi n + \epsilon n \rceil)}$. In particular, the value of v_i has rank between $(ni)/m - \epsilon n = (ni)/m - n/(4m)$ and $(ni)/m + \epsilon n = (ni)/m + n/(4m)$. As a consequence, for any i the number of element of value (of the first coordinate) between v_i and v_{i+1} is between $n/(2m)$ and $6n/(4m)$. (Note that v_0 is always the minimum value and v_m is the maximum value.)

LEMMA 13. *After the* 2D-ELIMINATE-POINTS *algorithm finishes, either* $|S| \geq m/2$ *or the stream size is at most* $n/4$.

PROOF. Let S' be the set returned by the algorithm. Suppose that $|S'| < m/2$. We partition the points $p'_0, p'_1, ..., p'_m$ obtained from Pass 2 into three sets. Let S_0 be the set of p'_i's that are in S after the algorithm ends. Let S_1 be the set of p'_i such that p'_i is dominated by a point in S_0, or there is a point p in S' that dominates p'_i and p'_j for any $j > i$. Let S_2 be the rest of the points. Note that every point in S_1 and S_2 must be dominated by some point in S'. We present two simple claims whose proofs are placed in the appendix.

CLAIM 14. $|S_1| > m/2$.

CLAIM 15. *For any* i, *if* $p'_i \in S_1$ *then every point* p *in the stream such that* $v_i \leq p[1] < v_{i+1}$ *is dominated by some point in* S'.

It follows, as before, that for any i there are at least $n/(2m)$ points whose first coordinate lies between v_i and v_{i+1}. Therefore, there are at least $|S_1| \cdot \frac{n}{2m} > n/4$ points deleted from the stream. □

We obtain deterministic algorithms by using 2D-ELIMINATE-POINTS as a subroutine of the STREAMING-SKYLINE and FIXEDWINDOW-STREAMING-SKYLINE algorithms.

5.3 Posets

Although the algorithms we described so far are developed primarily to compute skylines where the input is assumed to be a set of vectors, they can in fact solve more general problems on mathematical objects called *posets*.

We begin by defining posets formally. A partial order is a binary relation $\preceq$ that is reflexive, antisymmetric and transitive. (That is, for any element a, $a \preceq a$. For any a and b, if $a \preceq b$ and $b \preceq a$ then $a = b$. And, for any a, b and c, if $a \preceq b$ and $b \preceq c$ then $a \preceq c$.) A set with a partial order is called a *partially ordered set* (poset). It is easy to see that the following forms a poset: a set of vectors where we say that $p \preceq q$ if and only if p dominates q, for any vector p and q.

The problem of finding minimal elements on posts is to find all element p such that there is no element $q \preceq p$. In other words, we want to find all elements that are not "dominated" by any other elements. It follows that skyline computation is a special case of such problem when the elements are vectors and "$\preceq$" is equivalent to "dominate".

One of the results in Daskalakis et al. is a randomized algorithm that solves the problem using $O(\omega n + \omega^2 \log n)$ "queries" in expectation where ω is the "width" of the poset. Since our algorithms presented in section 4 do not need any structure of vectors, it can be used to find minimal elements on posets as well. In particular, the STREAMING-SKYLINE is a streaming algorithm that uses $O(mn) = O(\omega n)$ queries in expectation. Moreover, it uses only $O(m \log n)$ space, $O(\log n)$ passes. Thus, it is asymptotically as good as the previous algorithm in terms of the number of queries. Moreover, to the best of our knowledge, it is the first non-trivial streaming algorithm.

This version of our algorithm on posets can be adapted to compute skylines with partially-ordered domains (see, e.g., [6, 32, 33, 22] and references therein).

Dataset	n	dimensions	Skyline size
House	127931	6	5774
NBA	17264	5	495
Island	63383	9	467
Color	68040	9	1533

Figure 3: Comparison of Real Datasets

6. EXPERIMENTAL EVALUATION

In this section we present results from an experimental study comparing our external randomized algorithm (FIXEDWINDOW-RAND or RAND) with the best known non-indexing algorithms. All our implementations were done in Java. To remove the effects of caching on our I/O times, we padded each record up to 104 bytes, similar to [4, 10]. Furthermore, we decreased the memory size of our machine to just 256MB so that very little memory would be available for caching. All our experiments were performed on a dual-core 1.7GHz Intel Xeon running Linux 2.6.9. The machine had two hard disks and we implemented all algorithms to use both of them.

We performed experiments on both synthetically generated and real datasets. We generated independent, correlated, and anti-correlated data using the dataset generator that was provided by the authors of [4]. The real data sets that we used are summarized in Table 3 [2]. For all these datasets, we computed the skyline using *min* in all dimensions.

We compared the performance of our algorithm against the state of the art external non-indexed algorithms : BNL [4] and LESS [10]. In particular, we did not compare against indexed algorithms such as BBS [18] as they are not com-

[2]All these datasets are available for download from Dr. Yufei Tao's homepage: http://www.cse.cuhk.edu.hk/∼taoyf/. We note that Island is in fact a synthetic data set [26].

parable in this setting. We also evaluated the performance of some other algorithms, including SFS [8] and BEST [28], but they were considerably slower (as noted earlier in [10]) and we do not show the results for them here.

In our experiments, we compare the wall-clock running time, number of comparisons and I/O's. Unless otherwise stated, each experiment used as default a window size equal to 1% of the number of points (as in [4, 10]). Therefore, in our implementation, for any comparison, all three algorithms use same amount of memory. The synthetic data sets each had one million data points and five dimensional data.

Since our algorithm is randomized we measured the variance in its performance for a single data set. We repeatedly ran RAND on the House data 500 times and measured the standard deviation of the running time to be less than 1% of the mean. Furthermore, in 95% of the runs the running time of RAND exceeded the mean running time by at most 2%, and in all the runs it never exceeded the mean by more than 4%. Hence, the performance of RAND does not vary much due to randomization.

6.1 Baseline Comparisons

We compared our algorithm with BNL and LESS for all the datasets described above. For brevity, though, we only show the results for the House and anti-correlated datasets, and comment on how the results were different in other cases.

We do not wish to assert any quantitative comparisons between these algorithms since we believe that the time taken by each algorithm is dependent on several variables such as the machine specifications, implementation details, and data type. Instead, we present these results as a qualitative analysis to show the relative performance of each algorithm as some parameter is varied.

In Figures 4 and 5, we show the performance of the different algorithms when the window size w (i.e., the number of points that can fit in main memory) is varied between 0.1% and 10% of the number of points n in the datasets. We show the graphs of I/O's and comparisons only for the anti-correlated case. The trends are similar for the house data. Since the number of I/O's (Figure 5(b)) decreases rapidly and the number of comparisons (Figure 5(c)) increases very slowly, RAND and LESS benefit from the larger window size. However, beyond some point, there is only marginal benefit. The running time of BNL increases as the window is made larger beyond some point. This is because, as the window size increases, the number of comparisons increases and BNL has book-keeping costs associated with each comparisons.

In Figure 6, we varied the number of data points when the window size is fixed to 1%. We observe that the performance in all three metrics of all three algorithms is roughly linear in the number of points. We only show the plots for time for brevity. Whereas BNL is the best in the House dataset and LESS does well for the anti-correlated data, our algorithm performs comparably with both of them as n is increased.

In Figure 7, we studied the variation of the performance in terms of dimensions. For House dataset, we pruned it down to the first two, three, four, five, and finally all six dimensions and compared the performance of the algorithms. All algorithms' performance in all metrics sharply increases, with different rates for each algorithm, when the number of dimensions goes beyond 5. This is because the number

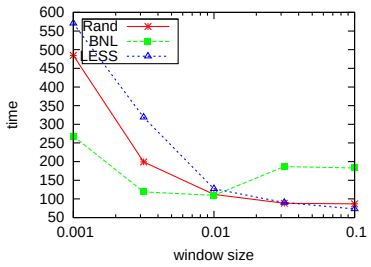
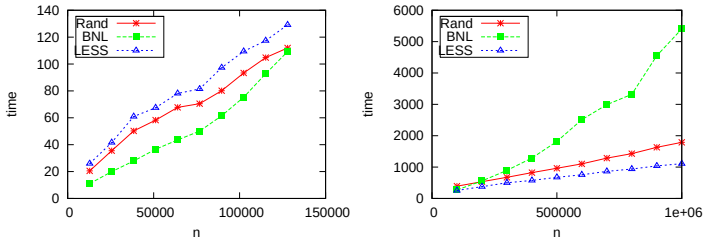


Figure 4: Varying w as a fraction of n for House



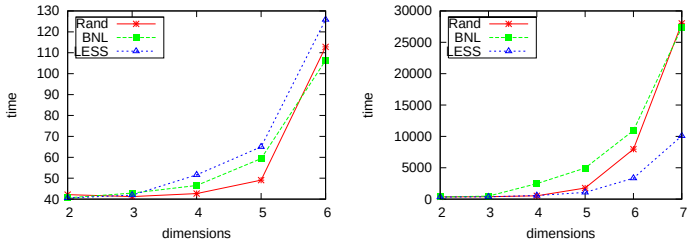
(a) House (b) Anti-correlated

Figure 6: Varying the number of points n

of skyline points increases rapidly as the number of dimensions increases. This is similar to results observed in [10] for the independent dataset. The fact that LESS becomes better than BNL and RAND as the dimensions are increased (equivalently the number of skyline points increased), but BNL and RAND beat LESS when the number of points are scaled can also be predicted from the comparisons table in Section 4.4.

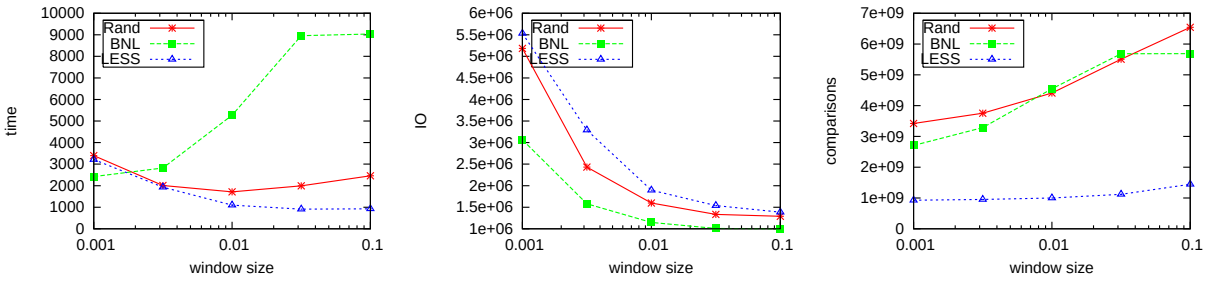
From these results it is apparent that the performance of RAND is comparable to BNL and LESS. We observed similar performance on the other real and synthetic data sets.

The running time of all algorithms in all datasets is summarized in Figure 8. Roughly speaking, LESS performs very well in the synthetic datasets (Figure 8 (a)) while BNL's performance can be sometimes very poor. However, BNL performs very well in all real datasets (Figure 8 (b)) while LESS' performance is always the worst and sometimes very poor. It could be concluded from these results that the performance of BNL and LESS relies heavily on the structure of input data. However, RAND is always close to the best



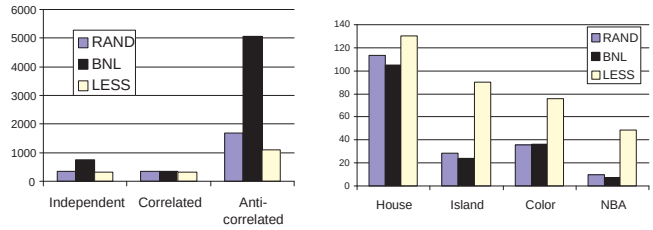
(a) House (b) Anti-correlated

Figure 7: Varying the number of dimensions d



(a) Time (b) I/O's (c) Comparisons

Figure 5: Varying w as a fraction of n for Anti-correlated



(a) Synthetic (b) Real

Figure 8: Running time on various datasets

algorithm. This could be partly because it has a good worst case guarantee.

6.2 Order Dependence

Next, we show how the performance of the other algorithms degrade considerably with a simple re-ordering of the data, whereas our algorithm's performance remains robust against these changes. We perform the following simple modifications of the data sets described above: in each case, we sort the data in decreasing order of entropy. Defined in [8], entropy is a scoring function that determines the likelihood with which a point is in the skyline. Additionally, we plot graphs where we sort the data in decreasing order of the entry in the first dimension.

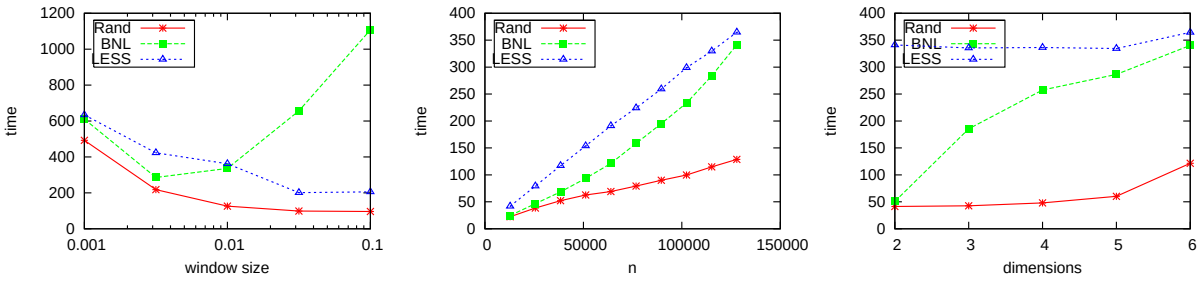
Figure 10 shows the comparative times for all three algorithms when the data is unsorted and sorted in decreasing entropy order. In almost all the cases, our algorithm shows little to no variance in performance. On the other hand, BNL and LESS both degrade in performance upto a factor of 2 or 3 times of their original running times for the real datasets and almost an order of magnitude difference for the larger synthetic datasets! This behavior can be explained as follows. By sorting in decreasing entropy order, we invalidate the effect of the entropy-based elimination filter of LESS. This makes LESS equivalent to SFS whose sorting phase incurs a lot of random I/O's and makes the algorithm much slower. In the case of BNL, the points which dominate fewer points are more likely to be found first in the stream and hence make their way in to the window. As a result, not many points are deleted from the stream in each pass. RAND, on the other hand, is still able to eliminate many points quickly as the skyline points in each phase are found starting from randomly sampled points.

In Figure 9, we vary w as a fraction of n, and vary d and n on the House data and test the performance of all three algorithms when sorted in decreasing entropy. The point of this experiment is to see whether the trends reflected in Figure 10 are an exception or whether this trend is expected for various settings (in terms of sizes and choice of parameters). As can be seen, in Figure 9 (b) and (c), as n or d is increased, the performance of BNL and LESS remain poor while RAND continues to perform a few times better. Figure 9 (a) also shows the same trend but is a little confusing to interpret. The reason BNL climbs up rapidly after a point is because of what we had seen in Figure 4 due to the bookkeeping cost increasing. However, the message here is that even if this factor is ignored, RAND is consistently better than both BNL and LESS, and therefore more resistant to perturbations in the input.

A similar variation is seen in the case of sorting in decreasing order by first dimension, see Figure 10 (c) and (d). While in the real data the variation of all three algorithms is similar to the entropy sorted case, for synthetic data, the variation is slightly less. Regardless, RAND remains the most robust to such minor alteration while the performance of both BNL and LESS degrade by a factor between about 2 and 10, depending on the case. We therefore believe that RAND is a good choice when real time performance requirements are stringent. It is hard to imagine a situation where a database sorts all records based on their entropy, however, many databases indeed store the records sorted by an attribute value. While these experiments only show trends on sorting by simple rules, it is conceivable that by changing the data, the performance fluctuates even more.

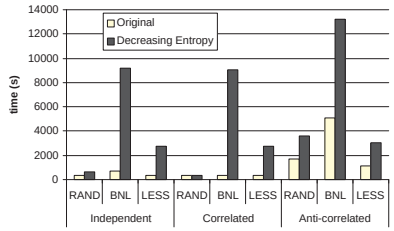
7. CONCLUSIONS

We present the first randomized streaming algorithm, RAND, for skyline computation. RAND has provable worst case guarantees on the number of sequential and randomized I/O's and number of pairwise comparisons. We show that it is optimal to within a logarithmic factor in terms of the space and passes used. We present a distributed version of RAND and a deterministic version for the 2-dimensional case. Finally, we experimentally evaluate the performance of RAND to LESS and BNL and show that it is comparable in the average case. Further, RAND is robust to minor variations in the input while the performance of LESS and BNL deteriorate significantly. We believe that for applications where running time guarantees are desirable on skyline queries, RAND is the best choice.

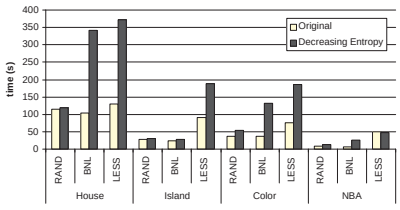


(a) Vary w (b) Vary n (c) Vary d

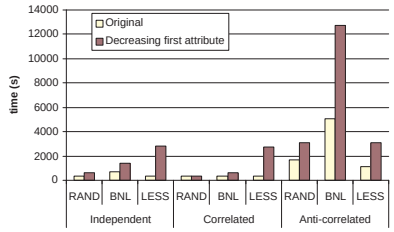
Figure 9: House, sorted by decreasing entropy. Varying w as a fraction of n, and varying d, n.



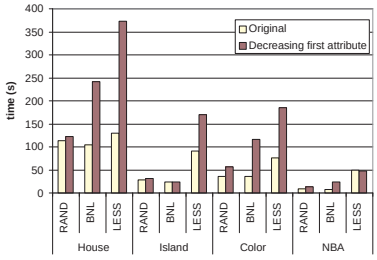
(a) Synthetic



(b) Real



(c) Synthetic



(d) Real

Figure 10: Variation in times for real and synthetic data sets (entropy sorted)

We present the most simple version of RAND, however, variations of the algorithm may be more suitable for specific scenarios such as low-cardinality domains or specific input distributions etc. Further, a good question is whether a modification of RAND can be made to handle real-time user preferences. Finally, handling dynamic streams (additions and deletions of points) is an interesting setting that RAND does not yet adapt to.

Acknowledgement

We would like to thank Richard J. Lipton for useful discussions, Donald Kossman, Michael Morse, Bernard Seeger, and Yufei Tao for sharing code and data sets, and Sungwoo Park for sharing a manuscript of his work.

8. REFERENCES

[1] L. Arge and J. S. Vitter. Optimal dynamic interval management in external memory. In *FOCS*, pages 560–569, 1996.
[2] W.-T. Balke, U. Güntzer, and J. X. Zheng. Efficient distributed skylining for web information systems. In *EDBT*, pages 256–273, 2004.
[3] O. Barndorff-Nielsen and M. Sobel. On the distribution of the number of admissible points in a vector random sample. *Theory of Probability and its Applications*, 11(2):249–269, 1966.
[4] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.
[5] C.-Y. Chan, P.-K. Eng, and K.-L. Tan. Efficient processing of skyline queries with partially-ordered domains. In *ICDE'05*, pages 190–191, Washington, DC, USA, 2005. IEEE Computer Society.
[6] C. Y. Chan, P.-K. Eng, and K.-L. Tan. Stratified computation of skylines with partially-ordered domains. In *SIGMOD Conference*, pages 203–214, 2005.
[7] S. Chaudhuri, N. Dalvi, and R. Kaushik. Robust cardinality and cost estimation for skyline operator. In *ICDE'06*, page 64, Washington, DC, USA, 2006. IEEE Computer Society.
[8] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting: Theory and optimizations. In *Intelligent Information Systems*, pages 595–604, 2005. Also appeared in ICDE'03.
[9] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *FOCS*, pages 285–298, 1999.
[10] P. Godfrey, R. Shipley, and J. Gryz. Algorithms and analyses for maximal vector computation. *VLDB J.*, 16(1):5–28, 2007. Also appeared in VLDB'05.
[11] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD '01*,

pages 58–66, New York, NY, USA, 2001. ACM.
[12] B. Jiang and J. Pei. Online interval skyline queries on time series. In *ICDE*, 2009.
[13] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: an online algorithm for skyline queries. In *VLDB'02*, pages 275–286. VLDB Endowment, 2002.
[14] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, 1975.
[15] X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the sky: Efficient skyline computation over sliding windows. In *ICDE*, pages 502–513, 2005.
[16] J. Matousek. Computing dominances in e^n. *Inf. Process. Lett.*, 38(5):277–278, 1991.
[17] M. D. Morse, J. M. Patel, and H. V. Jagadish. Efficient skyline computation over low-cardinality domains. In *VLDB*, pages 267–278, 2007.
[18] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005. Also appeard in SIGMOD'03.
[19] S. Park, T. Kim, J. Park, J. Kim, and H. Im. Parallel skyline computation on multicore architectures. In *ICDE*, 2009.
[20] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, pages 15–26, 2007.
[21] J. M. Ruhl. *Efficient algorithms for new computational models*. PhD thesis, 2003. Supervisor-David R. Karger.
[22] D. Sacharidis, S. Papadopoulos, and D. Papadias. Topologically sorted skylines for partially ordered domains. In *ICDE*, 2009.
[23] N. Schweikardt. Machine models and lower bounds for query processing. In *PODS*, pages 41–52, 2007.
[24] N. Schweikardt. Lower bounds for multi-pass processing of multiple data streams. In *STACS*, pages 51–61, 2009.
[25] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB '01*, pages 301–310, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
[26] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE*, 2009.
[27] Y. Tao and D. Papadias. Maintaining sliding window skylines on data streams. *IEEE Trans. Knowl. Data Eng.*, 18(2):377–391, 2006.
[28] R. Torlone, P. Ciaccia, and U. Romatre. Which are my preferred items. In *In Workshop on Recommendation and Personalization in E-Commerce*, pages 1–9, 2002.
[29] J. S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985. Also appeared in FOCS'83.
[30] J. S. Vitter. Algorithms and data structures for external memory. *Foundations and Trends in Theoretical Computer Science*, 2(4):305–474, 2006.
[31] A. Vlachou, C. Doulkeridis, Y. Kotidis, and M. Vazirgiannis. Skypeer: Efficient subspace skyline computation over distributed data. In *ICDE*, pages 416–425, 2007.
[32] R. C.-W. Wong, A. W.-C. Fu, J. Pei, Y. S. Ho, T. Wong, and Y. Liu. Efficient skyline querying with variable user preferences on nominal attributes. *PVLDB*, 1(1):1032–1043, 2008.
[33] R. C.-W. Wong, J. Pei, A. W.-C. Fu, and K. Wang. Online skyline analysis with dynamic preferences on nominal attributes. *IEEE Trans. Knowl. Data Eng.*, 21(1):35–49, 2009.
[34] A. C.-C. Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *FOCS*, pages 222–227, 1977.
[35] W. Zhang, X. Lin, Y. Zhang, W. Wang, and J. X. Yu. Probabilistic skyline operator over sliding windows. In *ICDE*, 2009.
[36] L. Zhu, Y. Tao, and S. Zhou. Distributed skyline retrieval with low bandwidth consumption. *IEEE Transactions on Knowledge and Data Engineering*, 21(3):384–400, 2009.

APPENDIX

A. PROOF OF CLAIMS IN SECTION WITH ELIMINATE-POINTS

PROOF OF CLAIM 3. Let $p = x_0, x_1, x_2, ..., x_t = q_i$ be the path from p to q_i. By definition, p is replaced by x_1. Observe that x_2 appears after x_1 in the stream (since x_2 also dominates p). Therefore, x_1 is replaced by x_2. For the same reason, we conclude that x_j is replaced by x_{j+1} for all $j = 0, 1, ..., t$. In the end, q_i is in S as claimed. □

PROOF OF CLAIM 4. Without loss of generality, assume that $|S_1| \geq |S_2| \geq ... \geq |S_m|$ (by permuting). For any set S_i, the probability that no elements in the set S_i are picked is $(1 - |S_i|/n')^{-24m} \leq e^{-24m|S_i|/n'}$. Therefore,

$$\mathbb{E}[\sum_{i:\ S_i \cap S = \emptyset} |S_i|] = \sum_{i=1}^{m} |S_i| e^{-24m|S_i|/n'}.$$

by union bound.

Now we break the quantity above into two terms with the sets of size at least and at most $n/(8m)$, respectively. That is, consider $S_{k+1}, S_{k+2}, ..., S_m$ where k is such that $|S_k| \geq n'/(8m)$ and either $k = m$ or $|S_{k+1}| < n'/(8m)$. We rewrite the previous quantity as

$$\sum_{i=1}^{k} |S_i| e^{-24m|S_i|/n'} + \sum_{i=k+1}^{m} |S_i| e^{-24m|S_i|/n'}.$$

We now bound the first term. When $|S_i| \geq n'/(8m)$,

$$e^{-24m|S_i|/n'} \leq e^{-3} \leq 1/(8).$$

Therefore, the first term becomes

$$\sum_{i=1}^{k} |S_i| e^{-24m|S_i|/n'} \leq \frac{1}{8} \sum_{i=1}^{k} |S_i| \leq \frac{n'}{8}.$$

For the second term, note that $|S_i| e^{-24m|S_i|/n'} \leq |S_i| \leq n'/(8m)$. Therefore, the second term becomes

$$\sum_{i=k+1}^{m} |S_i| e^{-24m|S_i|/n'} \leq \sum_{i=k+1}^{m} |S_i| \leq m \frac{n'}{8m} \leq \frac{n'}{8}.$$

Summing the two terms together, give the claimed bound $\mathbb{E}[\sum_{i:\ S_i \cap S = \emptyset} |S_i|] \leq n'/4$. □

B. PROOF OF CLAIMS IN DETERMINISTC 2D ALGORITHM

PROOF OF CLAIM 14. Draw a line from every point in S_0 to itself in S' and from every point in S_1 and S_2 to any point in S' that dominates them. Since those points in S_0 and S_2 share no points in S' with other points and $|S'| < m/2$, $|S_0| + |S_2| < m/2$. The claim thus follows. □

PROOF OF CLAIM 15. Observe that in both cases that makes p'_i in S_1, there is a point q and an integer $j > i$ such that $q[1] \geq v_j$. Moreover, since q dominates p'_i, $q[2] > p'_i[2]$. However, $p'_i[2] \geq p[2]$ for any point p such that $v_i \leq p[1] < v_{i+1}$ (by Pass 2). The claim follows. □