Set-Difference Range Queries

David Eppstein * Michael T. Goodrich † Joseph A. Simons ‡

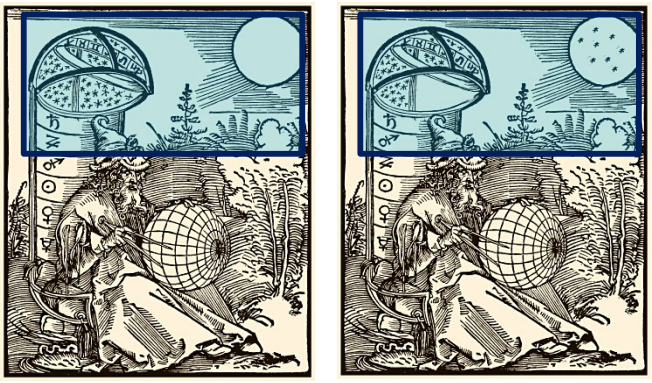
Abstract

We introduce the problem of performing set-difference range queries, where answers to queries are set-theoretic symmetric differences between sets of items in two geometric ranges. We describe a general framework for answering such queries based on a novel use of data-streaming sketches we call *signed symmetric-difference* sketches. We show that such sketches can be realized using invertible Bloom filters (IBFs), which can be composed, differenced, and searched so as to solve set-difference range queries in a wide range of scenarios.

1 Introduction

Efficiently identifying or quantifying the differences between two sets is a problem that arises frequently in applications, for example, when clients synchronize the calendars on their smart phones with their calendars at work, when databases reconcile their contents after a network partition, or when a backup service queries a file system for any changes that have occurred since the last backup. Such queries can be global, for instance, in a request for the differences across all data values for a pair of databases, or they can be localized, requesting differences for a specific range of values of particular interest. For example, clients might only need to synchronize their calendars for a certain range of dates or a pair of databases may need only to reconcile their contents for a certain range of transactions. We formalize this task by a novel type of range searching problem, which we call *set-difference range queries*.

We assume a collection $\mathcal{X}$ of sets $\{X_1, X_2, \ldots, X_N\}$, containing *data items* that are each associated with a geometric point and with a member of universe, $\mathcal{U}$, of size $U = |\mathcal{U}|$. A set-difference range query is specified by the indices of a pair of data sets, X_i and X_j, and by a pair of *ranges*, R_1 and R_2, which are each a constant-size description of a set of points such as a hyper-rectangle, simplex, or half-space. The answer to this set-difference range query consists of the elements of X_i and X_j whose associated points belong to the ranges R_1 and R_2 respectively, and whose associated elements in U are contained in one of the two data sets but not both. Thus, we preprocess $\mathcal{X}$ so that given ranges, R_1 and R_2, and two sets, X_1 and X_2, we can quickly report (or count) the universe elements in the set-theoretic symmetric difference $(R_1 \cap X_1) \triangle (R_2 \cap X_2)$. The performance goal in answering such queries is to design data structures having low space and preprocessing requirements that support fast set-difference range queries whose time depends primarily on the size of the difference, not the number of items in the range (see Figure 1). Examples of such scenarios include the following.



(a) (b)

Figure 1: Illustrating the set-difference range query problem. The images in (a) and (b) have four major differences, three of which are inside the common query range. The image (a) is a public-domain engraving of an astronomer by Albrecht Dürer, from the title page of *Messahalah, De scientia motus orbis* (1504).

- Each set contains readings from a group of sensors in a given time quantum (e.g., see [3]). Researchers may be interested in determining which sensor values have changed between two time quanta in a given region.
- Each set is a catalog of astronomical objects in a digital sky survey. Astronomers are often interested in objects that appear or disappear in a given rectangular region between pairs of nightly observations. (E.g., see [22].)
- Each set is an image taken at a certain time and place. Various applications may be interested in pinpointing changes that occur between pairs of images (e.g., see [18]), which is something that might be done, for instance, via two-dimensional binary search and repeated set-difference range queries.

*Dept. of Comp. Sci., U. of CA, Irvine, eppstein(at)uci.edu
†Dept. of Comp. Sci., U. of CA, Irvine, goodrich(at)uci.edu
‡Dept. of Comp. Sci., U. of CA, Irvine, jsimons(at)uci.edu

Query Type		Query Time	Space
Orthogonal:	Standard [5]	$O(\log^{d-1} n)$	$O(n \log^{d-1} n)$
	SD fixed m	$O(m \cdot \log^{d} n)$	$O(m \cdot n \log^{d-1} n)$
	SD variable m	$O(m \cdot \log^{d} n)$	$O(n \log^{d} n)$
	SD size est.	$O(\log^{d+1} n \log U)$	$O(n \log^{d} n \log U)$
Simplex:	Standard [17]	$O(n^{1-1/d}(\log n)^{O(1)})$	$O(n)$
	SD fixed m	$O(m \cdot n^{1-1/d}(\log n)^{O(1)})$	$O(m \cdot n)$
	SD variable m	$O(m \cdot n^{1-1/d}(\log n)^{O(1)})$	$O(n \log \log n)$
	SD size est.	$O(n^{1-1/d}(\log n)^{O(1)} \log U)$	$O(n \log n \log U)$
Stabbing:	Standard [4]	$O(\log n)$	$O(n \log n)$
	SD fixed m	$O(m \cdot \log n)$	$O(m \cdot n \log n)$
	SD variable m	$O(m \cdot \log n)$	$O(n \log n)$
	SD size est.	$O(\log^2 n \log U)$	$O(n \log n \log U)$
Partial Sum:	Standard[1]	$O(1)$	$O(n)$
	SD fixed m	$O(m)$	$O(m \cdot n)$
	SD variable m	$O(m)$	$O(n^2)$
	SD size est.	$O(\log n \log U)$	$O(n \log n \log U)$

Table 1: The results labeled "Standard" are previously known results for each data structure. Results labeled "SD" indicate bounds for set-difference range queries. Here d is the dimension of the query, m is the output size, and we assume the approximation factor $(1 \pm \epsilon)$ and failure probability δ are fixed.

1.1 Related Work

We are not aware of prior work on set-difference range queries. However, Suri *et al.* [20] consider approximate range counting in data streams. Shi and JaJa [19] present a data structure for range queries indexed at a specific time for data that is changing over time, achieving polylogarithmic query times. If used for set-difference range queries, however, their scheme would not produce answers in time proportional to the output size. For a survey of general schemes for range searching data structures, see Agarwal [1].

1.2 Our Results

We provide general methods for supporting a wide class of set-difference range queries by combining signed-symmetric difference sketches with any canonical group or semigroup range searching structure. Our methods solve range difference queries where the two sets being compared may be drawn from the same or different data sets, and may be defined by the same or different ranges. In our data structures, sets are combined in the *multiset model*: two data items may be associated with the same element of U, and if they belong to the same query range they are considered to have multiplicity two, while if they belong to the two different query ranges defining the set difference problem then their cardinalities cancel. The result of a set difference query is the set of all elements whose total cardinality defined in this way is nonzero.

Our data structures are probabilistic and return the correct results with high probability. Our running times depend on the size of the output, but only weakly depend on the size of the original sets. In particular, we derive the results shown in Table 1 for the following range-query problems (see the full version of this article [11]for details):

- Orthogonal: Preprocess a set of points in R^d such that given a query range defined by an axis-parallel hyper rectangle, we can efficiently report or estimate the size of the set of points contained in the hyper-rectangle
- Simplex: Preprocess a set of points in R^d such that given a query range defined by a simplex in R^d, we can efficiently report or estimate the size of set of points contained in the simplex
- Stabbing: Preprocess a set of intervals such that given a query point x, we can efficiently report or estimate the size of the subset which intersects x.
- Partial Sum: Preprocess a grid of total size $O(n)$ (e.g. an image with n pixels, or a d dimensional array with $O(n)$ entries for any constant d) such that given a query range defined by a hyper-rectangle on the grid, we can efficiently report or estimate the sum of the values contained in that hyper-rectangle.

2 The Abstract Range Searching Framework

In order to state our results in their full generality it is necessary to provide some general definitions from the theory of range searching (e.g., see [1]).

A *range space* is a pair (X, R) where X is a universe of objects that may appear as data in a range searching

[1] Using a standard solution to the partial sum problem.

problem (such as the set of all points in the Euclidean plane) and R is a family of subsets of X that may be used as queries (such as the family of all disks in the Euclidean plane). A *range searching problem* is defined by a range space together with an *aggregation function* f that maps subsets of X to the values that should be returned by a query. For instance, in a *range reporting problem*, the aggregation function is the identity; in a *range counting problem*, the aggregation function maps a subset of X to its cardinality. A data structure for the range searching problem must maintain a finite set $Y \subset X$ of objects, and must answer queries that take a range $r \in R$ as an argument and return the value $f(r \cap Y)$. The goal of much research in range searching is to design data structures that are efficient in terms of their space usage, preprocessing time, update time (if changes to Y such as insertions and deletions are allowed), and query time. In particular, it is generally straightforward to answer a query in time $O(|Y|)$, by testing each member of Y for membership in the query range r; the goal is to answer queries in an amount of time that may depend on the output size but that does not depend so strongly on the size of the data set.

A range searching problem is said to be *decomposable* if there exists a commutative and associative binary operation $\oplus$ such that, for any two disjoint subsets A and B of X, $f(A \cup B) = f(A) \oplus f(B)$. In this case, the operation $\oplus$ defines a *semigroup*.

Many range searching data structures have the following form, which we call a *canonical semigroup range searching data structure*: the data structure stores the values of the aggregation function on some family F of sets of data values, called *canonical sets*, with the property that every data set $r \cap Y$ that could arise in a query can be represented as the disjoint union of a small number of canonical sets $r_1, r_2, \ldots, r_K$. To answer a query, a data structure of this type performs this decomposition of the query range into canonical sets, looks up the values of the aggregation function on these sets, and combines them by the $\oplus$ operation to yield the query result. Note that this is sometimes called a *decomposition scheme*. For instance, the order-statistic tree is a binary search tree over an ordered universe in which each node stores the number of its descendants (including itself) in the tree, and it can be used to quickly answer queries that ask for the number of data values within any interval of the ordered universe. This data structure can be seen as a canonical semigroup range searching data structure in which the aggregation function is the cardinality, the combination operation $\oplus$ is integer addition, and the canonical sets are the sets of elements descending from nodes of the binary search tree. Every intersection of the data with a query interval can be decomposed into $O(\log n)$ canonical sets, and so interval range counting queries can be answered with this data structure in logarithmic time.

When the combination operation $\oplus$ has additional properties, they may sometimes be used to obtain greater efficiency. In particular, if $\oplus$ has the structure of a group, then we may form a *canonical group range searching data structure*. Again, such a data structure stores the values of the aggregation function on a family of sets of data values, but in this case it represents the query value as an expression $(\pm f(r_1)) \oplus (\pm f(r_2)) \oplus \cdots$, where the canonical sets r_i and their signs are chosen with the property that each element of the query range r belongs to exactly one more positive set than negative set. Again, in the interval range searching problem, one may store with each element its *rank*, the number of elements in the range from $-\infty$ to that element, and answer a range counting query by subtracting the rank of the right endpoint from the rank of the left endpoint. In this example, the ranks are not easy to maintain if elements are inserted and deleted, but they allow interval range queries to be answered by combining only two canonical sets instead of logarithmically many.

2.1 Signed Symmetric-Difference Sketches

Suppose that we want to represent an input set S in space sub-linear in the size of S such that we can compute some function $f(S)$ on our compressed representation. This problem often comes up in the streaming literature, and common solutions include dimension reduction (e.g. by a Johnson-Lindenstrauss transform [15]) and computing a *sketch* of S (e.g. Count-Min Sketch [9]).

A *sketch* σ_S of a set S is a randomized compressed representation of S such that we can approximately and probabilistically compute $f(S)$ by evaluating an appropriate function $f'(\sigma_S)$ on the sketch σ_S. This construct also comes up when handling massive data data sets, and in this context the compressed representation is sometimes called a synopsis [14].

A sketch algorithm σ is called *linear* if it has a group structure. That is, there exist two operators $\oplus$ and $\ominus$ on sketches σ such that given two multi-sets S and T,

$$\sigma_{S \uplus T} = \sigma_S \oplus \sigma_T \text{ and } \sigma_{S \setminus T} = \sigma_S \ominus \sigma_T$$

where $\uplus$ and $\setminus$ are the multi-set addition and subtraction operators respectively.

For our results, we define two different types of linear sketches. A *Signed Symmetric-Difference Reporting* (SDR) sketch is a linear sketch that supports a function `report`: given a pair of sketches σ_S and σ_T for two sets S and T respectively, probabilistically compute $S \setminus T$ and $T \setminus S$ using only information stored in the sketches σ_S and σ_T in $O(1+m)$ time, where m is the cardinality of the output. A *Signed Symmetric-Difference Cardinality* (SDC) sketch is a linear sketch that supports

a function `count`: given a pair of sketches σ_S and σ_T for two sets S and T respectively, probabilistically approximate $|S \triangle T|$ using only information stored in the sketches σ_S and σ_T in time linear in the size of the sketches.

3 Main Results

The main idea of our results is to represent each canonical set by an SDR or an SDC sketch. We implement our signed symmetric-difference counting sketches using a linear sketch based on the frequency moment estimation techniques of Thorup and Zhang [21]. We implement our signed symmetric-difference reporting sketches via an *invertible Bloom filter* (IBF), a data structure introduced for straggler detection in data streams [10]. IBFs can be added and subtracted, giving them a group structure and allowing an IBF for a query range to be constructed from the IBFs for its constituent canonical sets. The difference of the IBFs for two query ranges is itself an IBF that allows the difference elements to be reported when the difference is small. To handle set differences of varying sizes we use a hierarchy of IBFs of exponentially growing sizes, together with some special handling for the case that the final set difference size is larger than the size of some individual canonical set.

Further details of the SDR and SDC sketches are given in later sections. In this section, we assume the existence of SDR and SDC sketches as defined above in order to prove the following three theorems which are the crux of our results listed in Table 1

Theorem 1: *Suppose that a fixed limit m on the cardinality of the returned set differences is known in advance of constructing the data structure, and our queries must either report the difference if it has cardinality at most m, or otherwise report that it is too large. In this case, we can answer set-difference range queries with probability at least $1 - \epsilon$ for any range space that can be modeled by a canonical group or semigroup range searching data structure. Our solution stores $O(m)$ words of aggregate information per canonical set, uses a combination operation $\oplus$ that takes time $O(m)$ per combination, and allows the result of this combination to be decoded to yield the query results in $O(m)$ time. If the data structure is updated, the aggregate information associated with each changed canonical set can itself be updated in constant time per change.*

Proof. See [11] □

Theorem 2: *Suppose that we wish to report set differences that may be large or small, without the fixed bound m, in a time bound that depends on the size of the difference but that may depend only weakly on the size of the total data set. In this case, we can answer range difference queries with probability at least $1 - \epsilon$ for any range space that can be modeled by a canonical group or semigroup range searching data structure. Our solution stores a number of words of aggregate information per canonical set that is $O(1)$ per element of the set, uses a combination operation $\oplus$ that takes time $O(m)$ (where m is the cardinality of the final set-theoretic difference) and allows the result of this combination to be decoded to yield the query results in $O(m)$ time. If the data structure is updated, the aggregate information associated with each changed canonical set can itself be updated in logarithmic time per change.*

Proof. See [11]. □

Theorem 3: *Suppose that we wish to report the cardinality of the set difference rather than its elements, and further that we allow this cardinality to be reported approximately, within a fixed approximation ratio that may be arbitrarily close to one. In this case, we can answer range difference queries with probability at least $1 - \epsilon$ for any range space that can be modeled by a canonical group or semigroup range searching data structure. Our solution stores a number of words of aggregate information per canonical set that has size $O(\log n \log U)$, uses a combination operation $\oplus$ that takes time $O(\log n \log U)$ and allows the result of this combination to be decoded to yield the query results in $O(\log n \log U)$ time.*

Proof. See [11]. □

4 Invertible Bloom Filters

We implement our SDR sketches using the invertible Bloom filter (IBF) [10], a variant of the Bloom filter [6] for maintaining sets of items that extends it in three crucial ways that are central to our application. First, like the counting Bloom filter [7, 13], the IBF allows both insertions and deletions, and it allows the number of inserted elements to far exceed the capacity of the data structure as long as most of the inserted elements are deleted later. Second, unlike the counting Bloom filter, the IBF allows the elements of the set to be listed back out. And third, again unlike the counting Bloom filter, the IBF allows *false deletions*, deletions of elements that were never inserted, and again it allows the elements involved in false deletion operations to be listed back out as long as their number is small. These properties allow us to represent large sets as small IBFs, and to quickly determine the elements in the symmetric difference of the sets, as we now detail.

For the remainder of this paper, we assume without loss of generality that each element x is an integer. An

IBF supports several simple algorithms for item insertion, deletion, and membership queries; for a review of these basic details of the IBF see [11].

In addition, we can take the difference of one IBF, A, with a table T_A, and another one, B, with table T_B, to produce an IBF, C, with table T_C, representing their signed difference, with the items in $A \setminus B$ having positive signs for their cell fields in C and items in $B \setminus A$ having negative signs for their cell fields in C (we assume that C is initially empty). This simple method is also shown in [11].

Finally, given an IBF, which may have been produced either through insertions and deletions or through a subtract operation, we can list out its contents by repeatedly looking for cells with counts of $+1$ or -1 and removing the items for those cells if they pass a test for consistency. This method therefore produces a list of items that had positive signs and a list of items that had negative signs, and is shown in [11]In the case of an IBF, C, that is the result of a $\mathsf{subtract}(A, B, C)$ operation, the positive-signed elements belong to $A \setminus B$ and the negative-signed elements belong to $B \setminus A$.

4.1 Analysis

In this section, we extend previous analyses [10, 12] to bound the failure probability for the functioning of an invertible Bloom filter to be less than a given parameter, $\epsilon > 0$, which need not be a constant (e.g., its value could be a function of other parameters).

Theorem 4: *Suppose X and Y are sets with m elements in their symmetric difference, i.e., $m = |X \triangle Y|$, and let $\epsilon > 0$ be an arbitrary real number. Let A and B be invertible Bloom filters built from X and Y, respectively, such that each IBF has $\lambda \geq k + \lceil \log k \rceil$ bits in its gSum field, i.e., the range of g is $[1, 2^\lambda)$, and each IBF has at least $2km$ cells, where $k > \lceil \log(m/\epsilon) \rceil + 1$ is the number of hash functions used. Then the listItems method for the IBF C resulting from the $\mathsf{subtract}(A, B, C)$ method will list all m elements of $X \triangle Y$ and identify which belong to $X \setminus Y$ and which belong to $Y \setminus X$ with probability at least $1 - \epsilon$.*

Proof. [11]. □

To avoid infinite loops, we make a small change to listItems, forcing it to stop decoding after m items have been decoded regardless of whether there remain any decodable cells. This change does not affect the failure probability and with it the running time is always $O(mk)$.

5 Frequency Moment Estimation

We implement our SDC sketches using frequency moment estimation techniques. Let x be a vector of length U, and suppose we have a data stream of length m, consisting of a sequence of updates to x of the form $(i_1, v_1), \ldots, (i_m, v_m) \in [U] \times [-M, M]$ for some $M > 0$. That is, each update is a pair (i, v) which updates the ith coordinate of x such that $x_i \to x_i + v$.

The frequency moment of a data stream is given by

$$F_p = \sum_{i \in [U]} x_i^p = \|x\|_p^p.$$

Since the seminal paper by Alon *et al.* [2], frequency moment estimation has been an area of significant research interest. Indeed the full literature on the subject is too rich to survey here. Instead, see e.g. the recent work by Kane *et al.* [16] and the references therein. Kane *et al.* [16] gave algorithms for estimating F_p, $p \in (0, 2)$. Their algorithm requires $O(\log^2(1/\delta))$ time per update and $O(\delta^{-2} \log(mM))$ space. However, faster results are known for estimating the second frequency moment F_2 with constant probability. Thorup and Zhang [21] and Charikar*et al.* [8] independently improve upon the original result of Alon *et al.* [2], to achieve an optimal $O(1)$ update time using $O(\delta^{-2} \log(mM))$ space.

Given a sparse vector X of length m with coordinates bounded by $[-M, M]$, we can estimate $\|X\|_2^2$ by treating it as a data stream and running the algorithm of Thorup and Zhang. The algorithm computes a *sketch* S_X of X, of size $O(\delta^{-2} \log mM)$ such that $\|S_X\|_2^2$ is within a factor of $O(1 \pm \delta)$ of $\|X\|_2^2$ with constant probability. We can improve the probability bound to any arbitrary $O(1 - \epsilon)$ by running the algorithm $O(\log(1/\epsilon))$ times independently to produce $O(\log(1/\epsilon))$ independent sketches S_{Xi}, and taking the median of $\|S_{Xi}\|_p^p$. This strategy takes $O(\log(1/\epsilon))$ time to process each non-zero element in X, and the space required is $O(\delta^{-2} \log(1/\epsilon) \log(mM))$.

Furthermore, each sketch is linear, and therefore we can estimate the frequency moment of the difference of two sparse vectors X and Y by subtracting their sketches; $\|S_X - S_Y\|_2^2$ is within a $O(1 \pm \delta)$ factor of $\|X - Y\|_2^2$ with constant probability, and we can maintain $O(\log(1/\epsilon))$ independent sketches to achieve probability $O(1 - \epsilon)$.

Now, suppose we want to estimate the Hamming distance between two sets. We treat each set as a sparse bit-vector and use the fact that the Hamming distance between two bit vectors is equivalent to the squared Euclidean distance, which is just the second frequency moment of the difference of the vectors. Then we can apply the above strategy for sparse vectors to produce $O(\log(1/\epsilon))$ sketches for each set in $O(\log(1/\epsilon))$ time per element, and we subtract all $O(\log 1/\epsilon)$ pairs of sketches in $O(\delta^{-2} \log(1/\epsilon) \log U)$ time. Finally we compute the second frequency moment of each sketch and take the median over all estimations in $O(\log 1/\epsilon)$ time.

We summarize this strategy in the following theorem.

Theorem 5: *Let $0 < \epsilon < 1$ and $0 < \delta < 1$ be arbitrary real numbers. Given two sets, X and Y, taken from a universe of size U, we can compute an estimate $\hat{m}$ such that*

$$(1-\delta)|X \triangle Y| \leq \hat{m} \leq (1+\delta)|X \triangle Y|,$$

with probability $1-\epsilon$, using a sketch of size $O(\delta^{-2}\log(1/\epsilon)\log U)$ The preprocessing time, including the time required to initialize the sketch is $O(\delta^{-2}\log(1/\epsilon)\log U + (|Y|+|X|)\log(1/\epsilon))$ and the time to compute the estimate $\hat{m}$ is $O(\delta^{-2}\log(1/\epsilon)\log U)$.

References

[1] P. K. Agarwal. Range Searching. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 575–598. CRC Press, Inc., 1997.

[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[3] M. Basseville. Detecting changes in signals and systems—A survey. *Automatica*, 24(3):309–326, 1988.

[4] J. L. Bentley. Solution to Klee's Rectangle Problem. Unpublished manuscript, 1977.

[5] J. L. Bentley and J. B. Saxe. Decomposable searching problems I. Static-to-dynamic transformation. *J. Algorithms*, 1(4):301–358, 1980.

[6] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[7] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese. An improved construction for counting Bloom filters. In *Proc. 14th Eur. Symp. on Algorithms*, volume 4168 of *LNCS*, pages 684–695. Springer-Verlag, 2006.

[8] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In P. Widmayer, F. T. Ruiz, R. M. Bueno, M. Hennessy, S. Eidenbenz, and R. Conejo, editors, *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 693–703. Springer, 2002.

[9] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, April 2005.

[10] D. Eppstein and M. T. Goodrich. Straggler Identification in Round-Trip Data Streams via Newton's Identities and Invertible Bloom Filters. *IEEE Trans. on Knowledge and Data Engineering*, 23:297–306, 2011.

[11] D. Eppstein, M. T. Goodrich, and J. A. Simons. Set-difference range queries. Arxiv report, arXiv:1306.3482 [cs.DS], June 2013.

[12] D. Eppstein, M. T. Goodrich, F. Uyeda, and G. Varghese. What's the difference? Efficient set reconciliation without prior context. In *Proc. SIGCOMM*, 2011.

[13] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: a scalable wide-area web cache sharing protocol. *IEEE/ACM Trans. Netw.*, 8(3):281–293, 2000.

[14] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. In R. E. Tarjan and T. Warnow, editors, *SODA*, pages 909–910. ACM/SIAM, 1999.

[15] D. M. Kane and J. Nelson. Sparser johnson-lindenstrauss transforms. In D. Randall, editor, *SODA*, pages 1195–1206. SIAM, 2012.

[16] D. M. Kane, J. Nelson, E. Porat, and D. P. Woodruff. Fast moment estimation in data streams in optimal space. In L. Fortnow and S. P. Vadhan, editors, *43rd ACM Symp. on Theory of Computing (STOC)*, pages 745–754, 2011.

[17] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8(3):315–334, October 1992.

[18] R. Radke, S. Andra, O. Al-Kofahi, and B. Roysam. Image change detection algorithms: a systematic survey. *IEEE Trans. Image Processing*, 14(3):294–307, March 2005.

[19] Q. Shi and J. JaJa. A new framework for addressing temporal range queries and some preliminary results. *Theor. Comput. Sci.*, 332(1-3):109–121, February 2005.

[20] S. Suri, C. D. Tóth, and Y. Zhou. Range counting over multidimensional data streams. *Discrete Comput. Geom.*, 36(4):633–655, 2006.

[21] M. Thorup and Y. Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In J. I. Munro, editor, *SODA*, pages 615–624. SIAM, 2004.

[22] D. G. York, J. Adelman, J. John E. Anderson, S. F. Anderson, et al. The Sloan digital sky survey: technical summary. *The Astronomical Journal*, 120(3):1579, 2000.