Monitoring Continuous Band-Join Queries over Dynamic Data *

Pankaj K. Agarwal, Junyi Xie, Jun Yang, and Hai Yu

Department of Computer Science, Duke University
Durham, NC 27708-0129, USA
{pankaj,junyi,junyang,fishhai}@cs.duke.edu

Abstract. A *continuous query* is a standing query over a dynamic data set whose query result needs to be constantly updated as new data arrive. We consider the problem of constructing a data structure on a set of continuous *band-join* queries over two data sets R and S, where each band-join query asks for reporting the set $\{(r, s) \in R \times S \mid a \leq r - s \leq b\}$ for some parameters a and b, so that given a data update in R or S, one can quickly identify the subset of continuous queries whose results are affected by the update, and compute changes to these results.

We present the first nontrivial data structure for this problem that simultaneously achieves subquadratic space and sublinear query time. This is achieved by first decomposing the original problem into two independent subproblems, and then carefully designing data structures suitable for each case, by exploiting the particular structure in each subproblem.

A key step in the above construction is a data structure whose performance increases with the degree of *clusteredness* of the band-joins being indexed. We believe that this structure is of independent interest and should have broad impact in practice. We present the details in [1].

1 Introduction

In contrast to traditional queries, where each query is executed once on a given set of items, a *continuous query* is a standing query over a set of items that, once issued by the user, needs to keep generating new results (or changes to old results) subject to the same query condition, as new items continue to arrive in a stream. Continuous query processing has recently attracted much interest from the database community because of its wide range of traditional and emerging applications, especially in data streaming settings; see [3, 4, 6, 13] and the references therein.

One of the main challenges in continuous query processing is how to monitor a *large* number of continuous queries over dynamically changing data. For each

* Research by P.A. and H.Y. is supported by NSF under grants CCR-00-86013, EIA-01-31905, CCR-02-04118, and DEB-04-25465, by ARO grants W911NF-04-1-0278 and DAAD19-03-1-0352, and by a grant from the U.S.–Israel Binational Science Foundation. Research by J.X. and J.Y. is supported by NSF CAREER award under grant IIS-0238386.

incoming data update, one needs to identify the subset of continuous queries whose results are affected by the data update, and compute changes to these results. If there are many continuous queries, then a brute-force approach that processes each of them in turn will be too inefficient to meet the response-time requirement of most target applications, and faster techniques are needed.

An interesting aspect of continuous query processing is the interchangeable roles played by queries and data: Continuous queries can be treated as data, while each data update can be treated as a query requesting the subset of continuous queries affected by the update. Thus, to preprocess simple continuous queries, it is natural to apply indexing and query processing techniques which were traditionally intended for preprocessing data. For example, one can simply use R-trees to monitor a large number of continuous rectangular range queries over a dynamic point set in $\mathbb{R}^2$. However, for more complex continuous queries, especially those involving *joins* between multiple dynamic data sets, designing both space- and query-efficient data structures turns out to be interesting.

Problem Statement Let $R, S \subseteq \mathbb{R}$ be two different data sets. A *band-join* query on R and S, denoted by $J(a, b)$ for some parameters $a, b \in \mathbb{R}$, asks for reporting the set $J(a,b) = \big\{(r,s) \in R \times S \mid a \leq r - s \leq b\big\}$. Band-join queries naturally arise in many continuous query applications, and form the basis of more complex join queries [5, 6, 9]. In this paper we are interested in monitoring a set of continuous band-join queries when data are inserted or deleted from R or S. More precisely, let $\mathcal{Q} = \{Q_1, \cdots, Q_n\}$ be a set of continuous band-join queries on R and S, where $Q_i = J(a_i, b_i)$ and $a_i \leq b_i$, $a_i, b_i \in \mathbb{R}$. We are interested in supporting the following two functions when an element s is inserted into or deleted from S (the case where R is updated is entirely symmetric):

(Maintaining affected joins) Report all pairs (r, Q_i), where $r \in R$ and $Q_i \in \mathcal{Q}$, such that $a_i \leq r - s \leq b_i$. Each reported pair (r, Q_i) corresponds to an element $(r, s) \in R \times S$ to be inserted into or deleted from the result of Q_i.

(Reporting affected joins) Report all $Q_i \in \mathcal{Q}$ such that $a_i \leq r - s \leq b_i$ for some $r \in R$. They are the subset of queries in $\mathcal{Q}$ whose results have been affected by the insertion or deletion of s. This functionality is useful in settings in which it suffices to detect and report that Q_i has been affected, while the actual result of Q_i may be computed later on demand.

These two functionalities can be reduced to the following abstract problem. We say that an interval $\gamma \subset \mathbb{R}$ is *stabbed* by a point $x \in \mathbb{R}$ if $x \in \gamma$. Let $X = \{x_1, \ldots, x_m\}$ be a set of m points in $\mathbb{R}$, and $I = \{[a_1, b_1], \ldots, [a_n, b_n]\}$ be a set of n (closed) intervals in $\mathbb{R}$. Consider in the following queries on X and I: **(Q1)** Given a displacement $\Delta x \in \mathbb{R}$, report the set of all point-interval pairs (x, γ), where $x \in X$ and $\gamma \in I$, so that $x + \Delta x$ stabs γ. **(Q2)** Given a displacement $\Delta x \in \mathbb{R}$, report the set of all intervals in I that are stabbed by at least one point in $X + \Delta x$.

We are also interested in the following update operations: **(U1)** Add/remove a point in X. **(U2)** Add/remove an interval in I.

In the context of monitoring continuous band-joins, note that if we set $X_R = R$ and $I_{\mathcal{Q}} = \{[a_i, b_i] \mid Q_i \in \mathcal{Q}\}$, then maintaining affected joins corresponds to a (Q1) query on X_R and $I_{\mathcal{Q}}$ with $\Delta x = -s$, and reporting affected joins corresponds to a (Q2) query on X_R and $I_{\mathcal{Q}}$ with the same Δx. Also note that when an element is inserted into or deleted from R, or when a query is added into or deleted from $\mathcal{Q}$, we need to update the data structure for X_R and $I_{\mathcal{Q}}$; this corresponds to the operations (U1) and (U2).

Our Results As the problem of monitoring continuous band-joins is equivalent to (Q1) and (Q2) queries, we shall focus our attention on these two queries from now on. For either query, a straightforward data structure with $O(m + n)$ size and $O(m+n)$ query time has been known and actually used in practice for some time in the database community [5]. However, the query time of this approach is quite unsatisfactory. In the other extreme, a data structure with $O(mn)$ size and $O(\log(mn))$ query time is not hard to design either (see Lemma 1). But in this case, the size of the structure becomes problematic.

A natural open problem is whether there exists a data structure that fits between the above two extremes. We answer this question in the affirmative. In Section 2, we present the first nontrivial data structure for (Q1) and (Q2) that simultaneously achieves $o(mn)$ size and $o(m+n)$ query time. This demonstrates an intriguing tradeoff between the space and query time for both queries. Our construction proceeds by decomposing the original problem into two independent subproblems, and then carefully designing data structures suitable for each case, by exploiting the particular structure in each subproblem.

We should point out that the above structure is mostly of theoretical interest. However, in solving one of the subproblems, we have devised a data structure whose performance increases with the degree of *clusteredness* of the input intervals in I. Because in practice the intervals are often naturally clustered, we believe that this structure is of independent interest and can lead to a practically more efficient approach. We further comment on this structure in Section 2.2; details are given in [1].

Related Work Since dynamic data is ubiquitous in the real world, there has been a lot of work on designing efficient dynamic data structures for answering various queries; see, for example, the monograph by Overmars [15] and the survey by Chiang and Tamassia [7]. Motivated by more recent applications such as sensor networks, there has also been much research on maintaining various synopsis structures of the data in the data stream model [14], such as ε-approximations [2, 16], histograms [12] and so on, so that various statistical queries can be answered quickly. However, most of the above work only deals with non-continuous queries (i.e., each query is executed once on the current data set). Recently, motivated by a wide range of emerging applications, there has be a flurry of activity on studying continuous queries in the database community (see e.g., [3, 4]). But to the best of our knowledge, the problem of monitoring continuous queries has not yet received much attention from algorithms research community so far.

2 The Construction

The main result of this section is a data structure of size $o(mn)$ that answers (Q1) and (Q2) queries in $o(m+n)$ time. We first decompose the original problem into two subproblems, and then solve the two subproblems respectively (Sections 2.1 and 2.2). By putting pieces together and choosing appropriate parameters, we obtain the desired bounds on the size and query time (Section 2.3).

Let $r \in \mathbb{N}$ be a parameter to be fixed later. We first partition I into $O(n/r)$ groups in the following manner (see Figure 1). We sort the $2n$ endpoints of I and pick every $2r$-th endpoint. Thus a set of n/r points are picked. Let I_0 be the subset of intervals in I that are stabbed by any of these picked points. Note that each of the remaining intervals in $I \setminus I_0$ lies entirely between two consecutive picked points. We let $I_i \subset I$ denote the subset of intervals lying entirely between the $(i-1)$-th and the i-th picked points, for $1 \le i \le n/r+1$, where the 0-th and $(n/r+1)$-th picked points are defined to be $-\infty$ and $+\infty$. In this way, the set I is partitioned into a collection of disjoint subsets $I_0, I_1, \cdots, I_{n/r+1}$, where $|I_i| \le r$ for all $i \ge 1$.

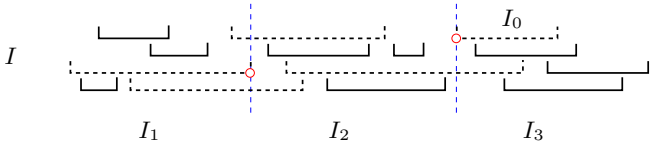

Fig. 1. The partition of I. Dashed Intervals belong to I_0, and solid intervals belong to $I_1, I_2, \ldots$, respectively.

Next, assume that the elements in $X = \{x_1, x_2, \cdots, x_m\}$ are in sorted order. We also partition X into m/r subsets $X_1, X_2, \cdots$, each of size at most r, where $X_i = \{x_{(i-1)r+1}, x_{(i-1)r+2}, \ldots, x_{ir}\}$, for $1 \le i \le m/r$.

The overall data structure $\mathcal{D}(X, I)$ for answering (Q1) and (Q2) queries on X and I consists of two separate structures: $\mathcal{D}(X, I_0)$ for answering the same queries on X and I_0, and $\mathcal{D}(X, I \setminus I_0)$ for answering the same queries on X and $I \setminus I_0$. The structure $\mathcal{D}(X, I \setminus I_0)$ further consists of a collection of data structures $\mathcal{D}(X_i, I_j)$ on X_i and I_j, for every $i \ge 1$ and $j \ge 1$. We now describe each of these components.

2.1 Structure $\mathcal{D}(X, I \setminus I_0)$

The structure $\mathcal{D}(X, I \setminus I_0)$ consists of a collection of data structures $\mathcal{D}(X_i, I_j)$ for each $i, j \ge 1$. We first describe a general data structure of quadratic size and logarithmic query time for (Q1) and (Q2) queries, and then apply it to each $\mathcal{D}(X_i, I_j)$. In the subsequent discussions, we assume that the reader is familiar with interval trees [8].

Clearly, an interval $[a_i, b_i] \in I$ is stabbed by $x_j + \Delta x$ if and only if $\Delta x \in [a_i - x_j, b_i - x_j]$. Therefore, to answer (Q1), we build an interval tree on the set of intervals $I - X = \big\{[a_i - x_j, b_i - x_j] \mid [a_i, b_i] \in I, x_j \in X\big\}$. The size of the tree is bounded by $O(|I - X|) = O(mn)$, and the preprocessing time is $O(mn \log(mn))$.

Given a query $\Delta x \in \mathbb{R}$, we find the set of intervals in $I - X$ stabbed by Δx. For each such interval $[a_i - x_j, b_i - x_j]$, we report the pair $(x_j, [a_i, b_i]) \in X \times I$. The query time is bounded by $O(\log(mn) + k)$, where k is the output size. The data structure can be made dynamic by using a dynamic interval tree [7]. Note that in our context, insertion or deletion of a point in X or an interval in I involves inserting or deleting n or m intervals, respectively, in the interval tree.

Similarly, to answer (Q2), observe that an interval $[a_i, b_i] \in I$ is stabbed by any point of $X + \Delta x$ if and only if $\Delta x \in \bigcup_{x_j \in X} [a_i - x_j, b_i - x_j]$. The set $\bigcup_{x_j \in X} [a_i - x_j, b_i - x_j]$ can be written as the union of a set I_i of at most m mutually *disjoint* intervals. We build an interval tree on intervals in $\bigcup_{i=1}^{n} I_i$. Given a query $\Delta x \in \mathbb{R}$, we find the set of intervals in $\bigcup_{i=1}^{n} I_i$ that are stabbed by Δx. For each such interval γ, if $\gamma \in I_i$, we report the interval $[a_i, b_i] \in I$. Note that each interval in I is reported at most once, and hence the query time is $O(\log(mn) + k)$, where k is the number of intervals reported. The data structure can also be made dynamic easily.

Lemma 1. *A data structure of size $O(mn)$ can be constructed in $O(mn \log(mn))$ time so that (Q1) and (Q2) can be answered in $O(\log(mn) + k)$ time, where k is the output size. Each (U1) operation takes $O(n \log(mn))$ time, and each (U2) operation takes $O(m \log(mn))$ time.*

We use Lemma 1 to preprocess each pair X_i and I_j $(i, j \geq 1)$ into a data structure $\mathcal{D}(X_i, I_j)$ of size $O(r^2)$ so that (Q1) and (Q2) on X_i and I_j can be answered in $O(\log r + k)$ time, where k is the output size. Given a displacement Δx, we can find all stabbing pairs in $(X + \Delta x) \times (I \setminus I_0)$ by scanning $X_1, X_2, \ldots$ and $I_1, I_2, \ldots$, and querying at most $O((m + n)/r)$ data structures $\mathcal{D}(X_i, I_j)$ as follows. For convenience, we denote the leftmost point in X_i by $l(X_i)$, and the rightmost point in X_i by $r(X_i)$. Similarly, we denote the leftmost endpoint in I_i by $l(I_i)$ and the rightmost endpoint in I_i by $r(I_i)$. Notice that $r(X_i) \leq l(X_{i+1})$ and $r(I_i) \leq l(I_{i+1})$. A (Q1) or (Q2) query on X and $I \setminus I_0$ can be answered in a way similar to the merge procedure of the merge-sort algorithm. Initially we let $Y = X_1$ and $J = I_1$. Suppose at a certain stage $Y = X_i$ and $J = I_j$. We first check whether $[l(Y) + \Delta x, r(Y) + \Delta x] \cap [l(J), r(J)] \neq \emptyset$. If so, there is a potential stabbing between a point in Y and an interval in J, and we use $\mathcal{D}(Y, J)$ to report all such stabbings; otherwise Y does not stab J, and we report nothing. Then, if $r(Y) + \Delta x > r(J)$, we let $J = I_{j+1}$; otherwise, we let $Y = X_{i+1}$. The procedure is repeated until all X_i and I_j are processed. Clearly, the above procedure probes at most $O((m + n)/r)$ data structures $\mathcal{D}(X_i, I_j)$, each requiring $O(\log r)$ time plus the time for reporting. Therefore the total time spent is $O(\frac{m+n}{r} \cdot \log r + k)$.

There is one problem: If we rely on Lemma 1 to construct $\mathcal{D}(X_i, I_j)$ for each $i, j \geq 1$, we need $O(r^2)$ space for each $\mathcal{D}(X_i, I_j)$, leading to an overall data structure of size $O(mn)$. To fix this problem, we store all $\mathcal{D}(X_i, I_j)$ compactly, using a method from Dumitrescu and Steiger [10]. The observation is that, for the data structure described in Lemma 1 on a point set Y and an interval set J, the combinatorial description of this structure is fully determined by the ordering of all the endpoints of $J - Y$. Using this combinatorial description of the

structure and the values of J and Y, (Q1) and (Q2) can be answered as before. If the sizes of Y and J are at most r, then the number of possible different orderings of the endpoints of $J - Y$ is $r^{O(r)}$ [11]. For each possible ordering, we construct a complete combinatorial description of the data structure with respect to this ordering. For each pair of X_i and I_j, we determine the ordering of the endpoints of $I_j - X_i$, and map this pair to the combinatorial description of $\mathcal{D}(X_i, I_j)$ according to this ordering. Therefore, the total space needed is $O(mn/r^2 + r^{O(r)} \cdot r^2) = O(mn/r^2 + r^{O(r)})$.

Lemma 2. *The data structure $\mathcal{D}(X, I \setminus I_0)$ has size $O(mn/r^2 + r^{O(r)})$, and answers (Q1) or (Q2) queries on X and $I \setminus I_0$ in $O(\frac{m+n}{r} \cdot \log r + k)$ time.*

2.2 Structure $\mathcal{D}(X, I_0)$

Given a set I of n intervals, a *stabbing set* of I is a set $P \subseteq \mathbb{R}$ of points so that each interval in I is stabbed by at least one point of P. We denote by $\tau(I)$ the size of the smallest stabbing set of I. It is well known that a stabbing set of I of size $\tau(I)$ can be computed in $O(n \log n)$ time by the following greedy algorithm: first sort the intervals in I by their left endpoints; then examine each of them from left to right, and find a maximal number of leftmost intervals so that they have a nonempty common intersection, stab these intervals by an arbitrary point from their common intersection, and repeat this step for the remaining intervals.

We first present a general data structure for answering (Q1) and (Q2) queries on X and I whose size and query time depend on the parameter $\tau = \tau(I)$. Then $\mathcal{D}(X, I_0)$ is constructed by simply applying this data structure on X and I_0. We treat (Q1) and (Q2) separately; in particular, the data structure for (Q1) is simpler than that for (Q2).

(Q1) queries. The data structure relies on the following observation.

Lemma 3. *A data structure of size $O(n)$ can be built in $O(n \log n)$ time so that the stabbing query on a set I of n intervals (i.e., reporting the subset of intervals in I that are stabbed by a query point x) can be answered in $O(\log \tau + k)$ time, where k is the number of reported intervals.*

Proof. We first compute a stabbing set $P = \{p_1, p_2, \ldots, p_\tau\}$ of size τ using the greedy algorithm described above. In fact, this algorithm also returns $\mathcal{I} = \{I_i \mid 1 \le i \le \tau\}$, a partition of I into τ subsets, so that every interval in I_i is stabbed by p_i. Let $l_i = \min_{[a_j,b_j] \in I_i} a_j$ and $r_i = \max_{[a_j,b_j] \in I_i} b_j$; clearly we have $l_i \le p_i \le r_i$. Let $\mathcal{H} = \{[l_i, r_i] \mid 1 \le i \le \tau\}$ be the *covering interval set* of I with respect to P; intuitively, each *covering interval* $[l_i, r_i]$ in $\mathcal{H}$ is the union of all intervals in I_i. We construct an interval tree [8] $\mathcal{T}$ on the set $\mathcal{H}$. In addition, for each I_i, we store two copies of I_i: (i) I_i^l, which sorts the intervals in I_i in increasing order of their left endpoints, and (ii) I_i^r, which sorts the intervals in I_i in decreasing order of their right endpoints. The size of the data structure is $O(\tau + n) = O(n)$, and the preprocessing time is $O(n \log n)$.

Given a query point x, the stabbing query is answered as follows. With the interval tree $\mathcal{T}$, we find in $O(\log \tau + k')$ time the k' covering intervals in $\mathcal{H}$ that are stabbed by x. If a covering interval $[l_i, r_i] \in \mathcal{H}$ contains x, we compare p_i and x. If $x \leq p_i$, then we scan I_i^l and report each interval until we encounter an interval whose left endpoint lies to the right of x. Symmetrically, if $x > p_i$, then we scan I_i^r and report each interval until we encounter an interval whose right endpoint lies to the left of x. The correctness of this procedure is easy to verify. Note that if $[l_i, r_i]$ is stabbed by x, then at least one interval in I_i is stabbed by x, and therefore $k' \leq k$, where k is the number of reported intervals. Hence, the total query time can be bounded by $O(\log \tau + k)$. □

Returning to our original problem, note that a (Q1) query is equivalent to a stabbing query on $I - X$ with the query point Δx, as shown in Section 2.1. Thus naturally we want to apply Lemma 3. Observe that if $P = \{p_1, p_2, \ldots, p_\tau\}$ is a stabbing set for I, then $P - X = \{p_i - x_j \mid p_i \in P, x_j \in X\}$ is a stabbing set of size at most $m\tau$ for $I - X$, and if $\mathcal{I} = \{I_1, \ldots, I_\tau\}$ is a partition of I with respect to P, then $\mathcal{I} - X = \{I_i - x_j \mid 1 \leq i \leq \tau, 1 \leq j \leq m\}$ is a partition of $I - X$ with respect to $P - X$. If we simply apply Lemma 3 directly, we would get a structure of size $O(|I - X|) = O(mn)$ for (Q1) queries. Hence, we need to be more careful. Observe that for any $1 \leq i \leq \tau$ and $1 \leq j \leq m$, I_i and $I_i - x_j$ are congruent in the sense that the sequence $I_i - x_j$ is obtained from I_i by translating each interval by $-x_j$. Thus, it suffices to store I_i. In more detail, we proceed as follows.

Let $\mathcal{H}$ be the covering interval set of I with respect to P, as defined in the proof of Lemma 3. We preprocess the intervals in $\mathcal{H} - X$ for stabbing queries, using an interval tree, as in Lemma 3. In addition, we store the sequences I_i^l and I_i^r for $1 \leq i \leq \tau$. Let Δx be a query displacement. We first report all intervals in $\mathcal{H} - X$ stabbed by Δx. Suppose an interval $[l_i - x_j, r_i - x_j]$ is stabbed by Δx. We then wish to report all intervals in $I_i - x_j$ that contain Δx, which is equivalent to reporting the intervals in I_i stabbed by $x_j + \Delta x$. This reporting can be done by the procedure described in Lemma 3. The overall query time is $O(\log(m\tau) + k)$, where k is the output size. The data structure uses $O(m\tau + n)$ storage, and can be constructed in $O(n \log n + m\tau \log(m\tau))$ time.

(Q2) queries. Similar bounds for (Q2) can also be obtained. The major difference is that, we have to make sure each stabbed interval is reported only once. This is achieved by using the notion of (i, j)-*intervals* introduced below. Again, we first compute a stabbing set $P = \{p_1, p_2, \ldots, p_\tau\}$, the corresponding partition $\mathcal{I} = \{I_i \mid 1 \leq i \leq \tau\}$ of I, and the covering interval set $\mathcal{H} = \{[l_i, r_i] \mid 1 \leq i \leq \tau\}$. For simplicity, let us assume $-\infty = x_0 < x_1 < x_2 < \cdots < x_m < x_{m+1} = +\infty$.

Definition 1. *Given $1 \leq i \leq \tau$ and $0 \leq j \leq m$, an* (i, j)-interval *is a maximal interval $\gamma \subset [p_i - x_{j+1}, p_i - x_j)$ so that for any $\Delta x \in \gamma$, $(X + \Delta x) \cap [l_i, r_i] \neq \emptyset$.*

We remark that there may be more than one (but at most two) (i, j)-interval for a fixed i and j. Intuitively, each (i, j)-interval corresponds to a contiguous range of displacement values Δx for which some interval in I_i is stabbed by some

point in $X+\Delta x$. Furthermore, as illustrated in Figure 2 (a), given a displacement value Δx in an (i,j)-interval, the stabbed intervals in I_i are precisely those stabbed by either $x_j+\Delta x$ or $x_{j+1}+\Delta x$. This is because p_i, the stabbing point of I_i, lies in between $x_j+\Delta x$ and $x_{j+1}+\Delta x$.

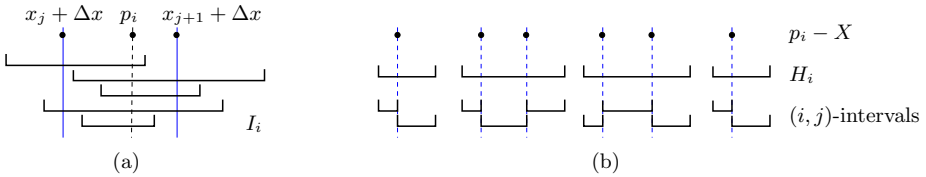

Fig. 2. (a) For any displacement Δx in an (i,j)-interval, an interval of I_i is stabbed by $X+\Delta x$ if and only if it is stabbed by either $x_j+\Delta x$ or $x_{j+1}+\Delta x$. (b) Computing the set of all (i,j)-intervals for a fixed i.

Lemma 4. *Let Π be the set of all (i,j)-intervals. Then $|\Pi| = O(m\tau)$, and Π can be computed in $O(m\log m + m\tau)$ time.*

Proof. For a fixed i, the set of all (i,j)-intervals, for $0 \le j \le m$, can be found as follows. In order for $X+\Delta x$ to stab $[l_i, r_i]$, Δx has to lie within the range $H_i = \bigcup_{x_j \in X}[l_i - x_j, r_i - x_j]$. Regard H_i as the union of a set of mutually disjoint intervals. We throw the m points from $p_i - X$ into H_i. These points further divide H_i into a set of atomic intervals, each of which is a maximal interval whose interior does not contain any point from $p_i - X$ (see Figure 2 (b)). It can be easily verified that each such atomic interval is an (i,j)-interval, if it lies between $p_i - x_{j+1}$ and $p_i - x_j$. Clearly, there are $O(m)$ such atomic intervals, all of which can be computed in $O(m)$ time, once X is sorted. Thus, the total time for computing all these intervals, for $i = 1, \cdots, \tau$, is $O(m\log m + m\tau)$. □

We preprocess Π into an interval tree, and store the sequence I_i^l and I_i^r for each $I_i \in \mathcal{I}$. By Lemma 4, the size of the structure is $O(m\tau+n)$. To answer (Q2) for a displacement Δx, we first report in $O(\log(m\tau)+k')$ time all k' intervals of Π that are stabbed by Δx. Suppose an (i,j)-interval of Π is reported. We scan the sequence I_i^l and report all of its intervals whose left endpoints lie to the left of $x_j+\Delta x$. We also report all intervals in the sequence I_i^r whose right endpoints lie to the right of $x_{j+1}+\Delta x$ but left endpoints lie to the right of $x_j+\Delta x$ (as otherwise they would have already been reported when we scan I_i^l); see Figure 2 (a).

Clearly, the overall query time is $O(\log(m\tau)+k'+k)$, where k' is the number of intervals in Π stabbed by Δx, and k is the output size. Note that for a fixed i, an (i,j_1)-interval is disjoint from an (i,j_2)-interval if $j_1 \neq j_2$. Therefore at most one (i,j)-interval is reported for any fixed i. Moreover, for each reported (i,j)-interval, at least one interval of I_i is stabbed by $X+\Delta x$. This implies that $k' \le k$. Hence the overall query time is in fact $O(\log(m\tau)+k)$.

We thus have the following lemma.

Lemma 5. *A data structure of size $O(m\tau+n)$ can be constructed in total time $O(m\tau\log(m\tau)+n\log n)$ so that (Q1) and (Q2) can be answered in $O(\log(m\tau)+k)$ time, where k is the output size.*

Note that $\tau(I_0) = O(n/r)$. Applying Lemma 5 on X and I_0, we thus obtain:

Lemma 6. *The data structure $\mathcal{D}(X, I_0)$ has size $O(mn/r + n)$, and answers (Q1) or (Q2) queries on X and I_0 in $O(\log(mn/r) + k)$ time.*

Remark. As shown by Lemma 5 above, the size and query time size and query time of this data structure depend on the parameter $\tau(I)$ of the input I. We call this an *input-sensitive* data structure. For small values of $\tau(I)$, i.e., when input intervals are highly clustered, this data structure almost achieves both linear size and logarithmic query time.

Recall that in (Q1) and (Q2) queries, the input intervals come from the continuous band-joins being indexed. These intervals naturally reflect users' interests on the data, and it is plausible to assume that in practice most of the users' interests would gather around a small number of "hotspots." In other words, in practice the intervals involved in continuous band-joins often admit a small stabbing set.[1] To exploit this pleasant property further, an alternative is to apply Lemma 5 to the entire set of intervals to be indexed (as opposed to just I_0). The result, described in detail in [1], is a dynamic practical structure whose performance increases with the degree of clusteredness of the input intervals. We also show in [1] how to dynamize that structure to support (U1) and (U2) operations, which is nontrivial because the smallest stabbing set of a set of intervals can completely change after every constant number of update operations.

2.3 Putting It Together

Overall, a query on X and I is answered by processing the query on X and I_0 as well as on X and $I \setminus I_0$ as described in the previous two sections. By Lemmas 2 and 6, the total size of the data structure is bounded by

$$S = O(mn/r + n + mn/r^2 + r^{O(r)}) = O(mn/r + r^{O(r)}),$$

and the query time is bounded by

$$T = O(\log(mn/r) + (m/r + n/r)\log r + k) = O((m/r + n/r)\log r + k).$$

Finally, we choose $r = c\log(mn)/\log\log(mn)$, where $c > 0$ is a sufficiently small constant, so that $S = O(mn\log\log(mn)/\log(mn)) = o(mn)$ and $T = O((m+n)\log^2\log(mn)/\log(mn) + k) = o(m+n) + O(k)$, as desired. We also note that the total preprocessing time can be bounded by

$$O(m\log m + n\log n + (mn/r)\log(mn/r) + mn\log r + r^{O(r)}) = O(mn\log\log(mn)).$$

Theorem 1. *A data structure of size $o(mn)$ can be constructed in total time $O(mn\log\log(mn))$ so that (Q1) and (Q2) can be answered in $o(m+n) + O(k)$ time, where k is the output size.*

[1] For example, in practice most band-join queries are only interested in the absolute difference of the data (i.e, query of the form $J(-a, a) = \{(r, s) \in R \times S \mid |r - s| \leq a\}$), or the one-sided difference of the data (i.e., query of the form $J(-\infty, a)$ or $J(a, +\infty)$). These query intervals can be all stabbed by one of the three points: 0, $-\infty$ and $+\infty$.

Remark. (1) The data structure supports each (U1) or (U2) operation in near-linear time. We omit the details here. Note that although the near-linear bound appears horrible, we show in [1] that it is essentially the best possible.

(2) Although we have been careful in the construction, the size of the data structure is only slightly subquadratic, and the query time is only slightly sub-linear. We leave finding the best tradeoff between the space and query time for (Q1) and (Q2) queries as an interesting open problem.

References

1. P. K. Agarwal, J. Xie, J. Yang, and H. Yu. Monitoring continuous band-join queries over dynamic data. Technical report, Department of Computer Science, Duke University, Durham, North Carolina, USA, Sept. 2005. Available at `http://www.cs.duke.edu/dbgroup/papers/2005-axyy-joinidx.pdf`.
2. A. Bagchi, A. Chaudhary, D. Eppstein, and M. Goodrich. Deterministic sampling and range counting in geometry data streams. In *Proc. 20th ACM Sympos. Comput. Geom.*, pages 144–151, 2004.
3. D. Carney et al. Monitoring streams: A new class of data management applications. In *Proc. 28th Intl. Conf. on Very Large Data Bases*, pages 215–226, 2002.
4. S. Chandrasekaran and M. J. Franklin. Streaming queries over streaming data. In *Proc. 28th Intl. Conf. on Very Large Data Bases*, pages 203–214, 2002.
5. S. Chandrasekaran and M. J. Franklin. PSoup: a system for streaming queries over streaming data. *The VLDB Journal*, 12(2):140–156, 2003.
6. J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagraCQ: A scalable continuous query system for internet databases. In *Proc. 19th ACM SIGMOD Intl. Conf. on Management of Data*, pages 379–390, 2000.
7. Y.-J. Chiang and R. Tamassia. Dynamic algorithms in computational geometry. *Computational Geometry: Theory & Applications*, 80(9):1412–1434, 1992.
8. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2nd edition, 2000.
9. D. DeWitt, J. Naughton, and D. Schneider. An evaluation of nonequijoin algorithms. In *Proc. 17th Intl. Conf. on Very Large Data Bases*, pages 443–452, 1991.
10. A. Dumitrescu and W. Steiger. Space-time tradeoffs for some ranking and searching queries. *Inform. Process. Lett.*, 79(5):237–241, 2001.
11. M. Fredman. How good is the information theory bound on sorting? *Theoret. Comput. Sci.*, 1:355–361, 1976.
12. S. Guha, N. Koudas, and K. Shim. Data streams and histograms. In *Proc. 33rd ACM Sympos. Theory of Computing*, pages 471–475, 2001.
13. L. Liu, C. Pu, and W. Tang. Continual queries for internet scale event-driven information delivery. *IEEE Trans. on Knowledge and Data Engineering*, 11(4):610–628, 1999.
14. S. Muthukrishnan. Data streams: algorithms and applications. Available at `http://www.cs.rutgers.edu/~muthu`.
15. M. H. Overmars. *The Design of Dynamic Data Structures*. Lecture Notes in Computer Science. Springer-Verlag, NY, 1987.
16. S. Suri, C. Toth, and Y. Zhou. Range counting over multidimensional data streams. In *Proc. 20th ACM Sympos. Comput. Geom.*, pages 160–169, 2004.