Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 195 (2021) 135–144

Procedia Computer Science

www.elsevier.com/locate/procedia

XI Latin and American Algorithms, Graphs and Optimization Symposium

Instance Optimal Join Size Estimation

Mahmoud Abo-Khamis[a], Sungjin Im[b,1], Benjamin Moseley[c,2], Kirk Pruhs[d,3], Alireza Samadian[d,4]

[a] *RelationalAI, Berkeley, USA*
[b] *University of California, Merced, Merced, USA*
[c] *Carnegie Mellon University, Pittsburgh, USA*
[d] *University of Pittsburgh, Pittsburgh, USA*

Abstract

We consider the problem of efficiently estimating the size of the join of a collection of preprocessed relational tables from the perspective of instance optimality analysis. The running time of instance optimal algorithms is comparable to the minimum time needed to verify the correctness of a solution. Previously, instance optimal algorithms were only known when the size of the join was small (as one component of their running time was linear in the join size). We give an instance optimal algorithm for estimating the join size for all instances, including when the join size is large, by removing the dependency on the join size. As a byproduct, we show how to sample rows from the join uniformly at random in a comparable amount of time.

Keywords: join size estimation; instance optimality analysis; beyond worst-case analysis.

1. Introduction

1.1. The Problem Statement

We consider the problem of efficiently estimating the size, or equivalently the number of rows, in a join (natural inner join) $J = T_1 \bowtie T_2 \bowtie \cdots \bowtie T_t$ given as input the collection $\tau = \{T_1, \ldots, T_t\}$ of relational tables, and some associated data structures that were obtained by independently preprocessing the tables. Recall that the join of two

[1] Supported in part by NSF grants CCF-1409130, CCF-1617653, and CCF-1844939.
[2] Supported in part by a Google Research Award, an Infor Research Award, a Carnegie Bosch Junior Faculty Chair and NSF grants CCF-1824303, CCF-1845146, CCF-1733873 and CMMI-1938909.
[3] Supported in part by NSF grants CCF-1535755, CCF-1907673, CCF-2036077 and an IBM Faculty Award.
[4] Corresponding author: emailsamadian@cs.pitt.edu.

1877-0509
10.1016/j.procs.2021.11.019

tables is a table with a row for each pair of rows of the given tables that match the values of the common columns. We also consider the related problem of uniformly sampling a row from J without constructing J.

One application of estimating the join size is in the context of determining a physical query plan for a particular logical query [12, Chapter 15]. As an example, let us consider the logical query $T_1 \bowtie T_2 \bowtie T_3$. Assume that one knew that $T_1 \bowtie T_2$ was large, but $T_2 \bowtie T_3$ and $T_1 \bowtie T_2 \bowtie T_3$ were small. Then first joining T_2 with T_3, and then joining the result with T_1, would likely be a more efficient physical query plan than first joining T_1 and T_2 as it would avoid (unnecessarily) computing a large intermediate result. If the joined table J is large, one common method for efficiently solving some computational problem on a joined table J is to solve the problem on a uniformly sampled collection of points from J. For example, [11] shows that uniform sampling can be used to efficiently solve a wide range of common regularized loss minimization problems with arbitrary accuracy.

It is NP-hard to determine whether or not the joined table J is empty (see, for example, [7, 9]). Thus, it is NP-hard to approximate the join size within any reasonable factor. Due to this hardness result, this paper addresses join size estimation from the perspective of instance optimality analysis.

1.2. Instance Optimality Analysis

We first informally build up the underlying intuition for instance optimality analysis before giving a formal definition. In instance optimality analysis, one ideally seeks a correct algorithm A that on every instance $\mathcal{I}$ is as fast as the fastest correct algorithm A' is on instance $\mathcal{I}$. That is, $A(\mathcal{I}) \leq \min_{A' \in \mathcal{D}} A'(\mathcal{I})$, where $A(\mathcal{I})$ is the running time for algorithm A on input $\mathcal{I}$, $A'(\mathcal{I})$ is the time for algorithm A' on input $\mathcal{I}$, and $\mathcal{D}$ is the collection of all correct algorithms. Note that in order for an algorithm to be correct, it must be correct on all inputs, not just on input $\mathcal{I}$.

However, this ideal is overly ambitious and impractical for most problems for several reasons. One of these reasons is that characterizing the fastest time that an algorithm can solve a particular problem or problem instance is generally well beyond our current understanding (the P=NP problem is essentially the problem of characterizing the fastest time that instances of NP-complete problems can be solved). A standard workaround is to first observe that the nondeterministic time complexity lower bounds the deterministic time complexity. That is, the fastest time that an algorithm can solve an instance is lower bounded by the fastest time that an algorithm can verify a proof or certificate that a candidate solution is correct for that instance. Again, allowing arbitrary certificates is overly ambitious and impractical. So generally for each problem one picks a particular type C of certificate that seems natural for the particular problem under consideration. Then one might seek an algorithm A whose running time is at most the minimum certificate size. That is, $A(\mathcal{I}) = O(\min_{C \in \mathcal{C}(\mathcal{I})} |C|)$, where here $\mathcal{C}(\mathcal{I})$ is the collection of certificates of type C that prove that a particular answer is correct for the instance $\mathcal{I}$, and $|C|$ denotes the size of C. Such an algorithm A would then be instance optimal among all algorithms that produce a certificate of type C. For example, [1] gives an algorithm for finding a Pareto frontier of points in the plane that is instance optimal among comparison-based sorting algorithms.

One often needs to relax this requirement a bit more via approximation. Therefore, we seek an algorithm A where $A(\mathcal{I}) = \tilde{O}(\min_{C \in \mathcal{C}(\mathcal{I})} F(|C|))$, where the $\tilde{O}$ hides poly-log factors of the input size, and F is some function that is as close to linear as possible. That is, ignoring poly-log factors, we want the time to be a function of the certificate size, and we want this function to be as slowly growing as possible.

Instance optimality analysis is often viewed as falling under the rubric of "beyond worst-case" analysis techniques. A survey of instance optimality analysis and other "beyond worst-case" analysis techniques can be found in [10].

1.3. Previous Instance Optimality Results for Join Size

We consider a type of certificate that has been used in previous papers on join size computation using instance optimality analysis [8, 2, 3], namely a collection of gap boxes. To apply this certificate one assumes that a priori it is the case that all table entries have been mapped to integers in the range $[1, n]$. Thus, if there are a total of d attributes in the tables in τ then J can be viewed as a collection of z axis-parallel unit hypercubes in $[0, n]^d$. To accomplish this, one can view each row r of J as a point in $\mathbb{N}^d$ in the natural way, and then associate r with the axis-parallel unit hypercube h that contains r, and where r is the furthest point in h from the origin. We define a box to be an axis-parallel hyperrectangle that is the Cartesian product of intervals. Then a *gap box* b is a box that does not intersect any unit hypercube in J. A collection B of gap boxes, whose union has volume V, can be viewed as a certificate that the join size z is at most $n^d - V$. Let C be the smallest set of gap boxes that cover $[0, n]^d - J$.

(a) (b)

Fig. 1: Two points (represented by two unit hypercubes) in two dimensional space and the minimum number of gap boxes needed to cover the empty space.

As an example of these definitions, consider the instance in Figure 1 where $d = 2$. Figure 1a shows the two unit hypercubes associated with the points $(2, 4)$ and $(3, 1)$. Figure 1b shows a minimum cardinality collection of gap boxes that establishes that the volume of these points is at most two. Thus, $C = 4$ for this instance.

The algorithm Tetris is introduced in [8] for the join size problem. The algorithm Tetris and its analysis were extended in [2, 3]. The authors of [2, 3] show how to preprocess each table to construct a collection of $O(dtm \log^d n)$ gap boxes, which are the maximal dyadic gap boxes. A *dyadic gap box* is a gap box such that its size in each dimension is a power of 2 and the i^{th} coordinate of each vertex is an integer multiple of the size of the box in that dimension. Without loss of generality, we assume the whole space has dimensions that are powers of 2. It is also shown that any certificate of gap boxes can be replaced by a certificate that is a subset of maximal dyadic gap boxes without a substantial increase in the size of the certificate [2, 3]. The running time of the preprocessing is $\tilde{O}(n)$. The work in [8] explains how to store dyadic gap boxes in what is essentially a trie (prefix tree) using space $O(d \log n)$ per box. Using this data structure, Tetris computes the exact join size z in time $\tilde{O}(|C|^{d/2} + z)$ by constructing a collection B of $\tilde{O}(C)$ gap boxes. Conceptually, in each iteration, Tetris finds a point p that is not in the joined table J, and not covered by any box in B, and then adds to B all the dyadic boxes that contain p in any of the data structures associated with any of the tables. The costly operation is finding p, which conceptually is achieved by exhaustive enumeration.

1.4. Our Results

In many applications of the join size computation/estimation, such as the application of computing a physical query plan, the linear dependence on the join size z in the running time of Tetris is not desirable. In these settings one wants to be able to estimate quickly even if the join is large. Our main result is an algorithm Welltris[5] that is a variation of Tetris that estimates the join size without any dependence in the running time on the join size z. So we assume that the input to the join size problem additionally contains two parameters ϵ and δ. Welltris then computes an estimate $\tilde{z}$ to the number of rows z in the joined table J that is accurate to within a factor of $1 + \epsilon$ with probability at least $1 - \delta$. That is $\Pr\left[\frac{z}{1+\epsilon} \le \tilde{z} \le (1 + \epsilon)z\right] \ge 1 - \delta$.

The preprocessing of the tables in Welltris is identical to that of Tetris used in [2, 3]. But for completeness we cover this in Section 3. When computing the join size, Welltris, like Tetris, maintains a collection B of $\tilde{O}(C)$ dyadic boxes, and in each iteration it finds a point p not in the joined table and not covered by any box in B. But Welltris does not use exhaustive search to find p. Instead Welltris *samples* points uniformly at random from points not covered by the boxes in B. The intuition behind this is that the uniform sampling will likely lead to choosing boxes that contain a large number of points uncovered by any box in B. To accomplish this it modifies Chan's algorithm [5] for Klee's measure problem, which is the problem of computing the volume of the union of a collection of boxes. The running time for Chan's algorithm is $\tilde{O}(C^{d/2})$. We show that the running time of Welltris is $\tilde{O}(C^{d/2+1})$[6], or more precisely,

$$O\left(C^{d/2+1} \cdot \frac{1}{\epsilon} \cdot \log(1/\delta) \cdot 2^{d^2/2+d} \cdot \log^{d^2+d}(n) \cdot \log(m)\right)$$

where m is the number of rows in the largest input table.

[5] The game Welltris is a variation of the original game Tetris [6].

[6] This assumes that ϵ, δ and d are constants.

Therefore, Welltris can estimate the join size quickly for instances in which there is a modestly-sized certificate bounding the join size, and in which the number of dimensions is small, even if the join size is large. As one simple example where our bounds for Welltris improve the bound for Tetris, consider the cross product join of two one column tables with $m = n$. In this case $|C| = O(1)$. The running time of Tetris is $\Omega(m^2)$, whereas the running time of Welltris is $O(\frac{1}{\epsilon} \cdot \log(1/\delta) \cdot \log^4(m))$.

Welltris can be modified to sample q points uniformly from the joined table J in time $\tilde{O}(C^{d/2+1} + qC)$, or more precisely,

$$O\left(C^{d/2+1} \cdot \log(q) \cdot 2^{d^2/2+d} \cdot \log^{d^2/2+d}(n) \cdot \log^{d^2/2}(m) + q \cdot C \cdot d^4 2^d \cdot \log^{2d+1}(n)\right).$$

Given a universe box S, it is $W[1]$-hard with respect to d to decide if a collection of boxes covers the whole space of S [4]. That is, it is $W[1]$-hard to decide whether $z = 0$ or whether $z > 0$. Thus a running time that did not have exponential dependence on d would imply $W[1] = \text{FPT}$, which is considered unlikely. For this reason, we believe Welltris has roughly optimal efficiency for join size estimation with this type of certificate.

2. Formal Definitions

In this section we state the formal definitions of the terms used in the rest of the paper. A *table/relation* T_i with d_i columns/dimensions is a collection of points in a d_i dimensional space such that each dimension has an associated positive integer that we refer to as the attribute of the column. No attribute can be associated with more than one dimension in the table. Different tables may share attributes. The *natural inner join* of $T_1, \ldots, T_t$ denoted by $T_1 \bowtie \cdots \bowtie T_t$ is a table J such that the set of attributes of J is the union of the attributes of $T_1, \ldots, T_t$ and a point x is in J if and only if for all T_i the projection of x onto the subspace formed by the columns of T_i exists in T_i. In this paper we use the term join instead of natural inner join since it is the most common type of join. We use t to denote the number of tables in our join query, d to denote the total number of columns in J, m to be the maximum number of rows in any of the input tables, and n to be the size of the domain of each dimension (the maximum number of unique values in that dimension). Therefore, we can assume all the values in the input tables are integers in $[1, n]$, and are represented by fixed-length binary strings. Note that if we assume that the tables involved in the join are the only tables in the database, then $n \leq tm$.

Given a set of points in d dimensional space, a *gap box* is a d dimensional box that does not contain any of the points. We denote the set of all gap boxes by B. A box $b \in B$ is a *maximal gap box* if and only if there is no other box in B that covers b entirely and is not equal to b. Given a set of boxes B', we call $C(B')$ a certificate for B' if and only if it is a set of minimum number of boxes needed to cover the union of B'. It is easy to observe that if B' is the set of maximal boxes in B then $|C(B)| = |C(B')|$. If B is the collection of all the gap boxes of the tables $T_1, \ldots, T_t$, then $C(B)$ is the optimal certificate of the join query $T_1 \bowtie T_2 \bowtie \cdots \bowtie T_t$. Based on the definition of the join, a point is in J if and only if it is not covered by any of the gap boxes.

Recall that a dyadic gap box is a gap box such that its size in each dimension is a power of 2 and the i^{th} coordinate of each vertex is an integer multiple of the size of the box in that dimension. We denote the set of maximal dyadic gap boxes by $B_{\mathcal{D}}$. A dyadic gap box is maximal if no other dyadic gap box contains it. In our setting, every point is a unit size box, and it can be represented using a d dimensional vector specifying its location. Then a dyadic box can be represented by a d dimensional vector, and in each dimension its value can be seen as the binary prefix of a set of points; then such a dyadic box covers all the points that this box is a prefix of in all dimensions. Note that the binary prefix can be empty in a dimension (denoted by λ) and in that case that dyadic box spans that dimension entirely. Note that each point is also a dyadic box of size 1. For $d = 3$ and $n = 8$, a dyadic box can be $(\lambda, 1, 101)$ which means its size in the first dimension is 8, in the second one 4, and in the third dimension is 1. In this case, $(111, 100, 101)$ is a point covered by this dyadic box and $(111, 000, 101)$ is an uncovered point.

Lemma 2.1 ([8]). *Each dyadic box can be fully contained in at most* $\log^d(n)$ *other dyadic boxes.*

Proof. Based on the definition of a dyadic box, a dyadic box $A = (a_1, \ldots, a_d)$ contains all the points $x = (x_1, \ldots, x_d)$ such that a_i is a prefix of x_i for all i. Therefore, a dyadic box $A = (a_1, \ldots, a_d)$ contains a dyadic box $B = (b_1, \ldots, b_d)$ if and only if a_i is a prefix of b_i for all i. Since each dimension has at most $\log(n)$ prefixes, the lemma follows. □

3. Preprocessing

3.1. Gap box construction

The preprocessing step for our algorithm constructs a collection of dyadic gap boxes for each table, including all maximal dyadic boxes, using an algorithm from [3]. As in [8], we will also show that any certificate can be replaced by a certificate that is a subset of maximal dyadic gap boxes without a substantial increase in the size of the certificate. For completeness, we explain the algorithm and analysis from [3, 8].

Algorithm 1 Algorithm for dyadic gap box construction [3]

procedure CONSTRUCT GAP BOXES(T_i)
  Input: An input table T_i.
  Output: A set of dyadic gap boxes including all maximal dyadic gap boxes for T_i.
  $D \leftarrow \emptyset$
  $D' \leftarrow \emptyset$
  if T_i is empty **then**
    return the dyadic gap box that covers the whole space.
  for $x \in T_i$ **do**
    for all dyadic boxes b such that $x \in b$ **do**
      $D \leftarrow D \cup \{b\}$
      for all attributes a such that $b_a \neq \lambda$ **do**
        $b' \leftarrow b$
        Flip the last bit of b'_a
        $D' \leftarrow D' \cup \{b'\}$
  return $D' \setminus D$.

The algorithm picks an input table T_i and constructs a set of dyadic gap boxes that include all the maximal dyadic gap boxes in them. We assume that all attributes have a specified domain and they have been mapped to an integer in $[1, n]$. The algorithm initializes two sets D and D' to be empty. Then for all tuples $x \in T_i$ and for any dyadic box b which contains x, the algorithm adds b to D. This step can be performed by enumerating all dyadic gap boxes whose elements in different dimensions are prefixes of the ones in x. Furthermore, for every box b that is added to D and every dimension a of b that does not span the whole space (is not λ), it creates a dyadic box by flipping the last bit of a in b and adds it to D'. For example, if $b = (10, \lambda, 111)$, then the algorithm adds the dyadic boxes $(11, \lambda, 111)$ and $(10, \lambda, 110)$ to D'. At the end, the algorithm returns $D' \setminus D$. Algorithm 1 shows a pseudocode of the preprocessing step. Algorithm 1 outputs at most $d(\log(m) + 1)^d$ dyadic gap boxes which include all the maximal dyadic gap boxes.

Theorem 3.1. *Let $B^i_{\mathcal{D}}$ be the output of Algorithm 1 for table T_i and let $B_{\mathcal{D}} = \bigcup_i B^i_{\mathcal{D}}$. Furthermore, let B^i be the set of all the gap boxes for table T_i and $B = \bigcup_i B^i$. Then,*

1. $|C(B_{\mathcal{D}})| = O(2^d \log^d(n))|C(B)|$;
2. $|B_{\mathcal{D}}| = O(dt \log^d(n)m)$.

Note that, in the above theorem, $C(B)$ is the smallest possible certificate over any set of gap boxes. Therefore, Theorem 3.1 intuitively says that Algorithm 1 produces a collection of dyadic gap boxes that admit a nearly optimal certificate. Moreover, note that we have defined $C(B)$ over gap boxes that are defined on the subspace formed by the columns of J; however, for each input table T_i, Algorithm 1 returns a set of boxes that are initially defined only in the subspace formed by the columns of T_i. This mismatch of dimensions can be fixed by the implicit assumption that the boxes in B^i extend to cover the whole space in all the dimensions not presented in T_i.

(a) (b)

Fig. 2: Illustrating the simplification step of Algorithm 2 on an example. The red box in 2a spans all the dimensions except for one, and it is removed in 2b

3.2. Dyadic Trie Data Structure

For our algorithm, we need a data structure to efficiently store a set of dyadic boxes and search if a given dyadic box is included in the set or not. To store the dyadic boxes, first we consider each dyadic box as a string from the alphabet $\{0, 1, \lambda, , \}$ (the last element of the alphabet is a comma). For example, a dyadic box of the form $(00, \lambda, 1)$ can be represented as the string "$00, \lambda, 1$". Then we use a Trie (prefix tree) to store and search the strings representing the dyadic boxes. Since there is a one-to-one relation between the dyadic boxes and their string representations, the trie correctly stores the dyadic boxes and searches for any query box.

Note that the length of every string representing a dyadic box is at most $O(d \log(n))$. Thus, the time needed to construct the trie in the most straight-forward manner is $O(kd \log(n))$, where k is the number of dyadic boxes, and the time to search for a dyadic box is $O(d \log(n))$.

3.3. Chan's Algorithm for Klee's Measure Problem

In this section we explain the general outline of Chan's algorithm [5]. Given a set B of boxes in d dimensional space, Chan's algorithm returns the volume of the union of the boxes in time $O(|B|^{d/2})$. We later use this algorithm to sample an uncovered point uniformly at random. You may find the general description of the algorithm in Algorithm 2. The input of the algorithm is a set of boxes B and a boundary box S that represents the whole space. The output is the volume of the union of B intersecting with S.

Algorithm 2 Chan's algorithm [5]

```
1:  procedure MEASURE(B, S)
2:      Input: A set of boxes B and a universe box S.
3:      Output: The volume of the intersection of S with the union of boxes in B.
4:      if |B| is below a constant then
5:          return the answer directly
6:      Simplify B
7:      Cut S into two sub-cells S_1 and S_2.
8:      B_1 ← boxes in B that have intersection with S_1
9:      B_2 ← boxes in B that have intersection with S_2
10:     return measure(B_1, S_1) + measure(B_2, S_2)
```

The reason that the algorithm has a good time complexity lies in steps 6 and 7. The simplification step removes any box in B that is equivalent to slabs spanning in all dimensions except for one when restricted to S. When removing these boxes, the algorithm changes S and all the boxes in B that have intersection with the slabs and removes the intersection from them. Note that this step does not increase the number of boxes in B. Figure 2 illustrates this step on some example.

After removing the slabs and simplifying the boxes, the algorithm cuts S into two disjoint sub-cells S_1 and S_2 and recursively calls the same function on those subproblems. The cut is done in a way that balances the total weights of

the faces of the boxes in B that are present in S_1 and S_2 and it gives different weights to different faces based on the dimensions they are spanning.

The main properties of Chan's algorithm that are useful for us in order to uniformly sample a point are the following: (1) S_1 and S_2 are disjoint, and (2) there is a one-to-one relation between the uncovered points in S_1 and S_2 and the uncovered points in S.

Using the above properties, it is very easy to sample uniformly at random k points which are not covered by a given set of gap boxes B. We initialize S to be the cross product of the domains of the attributes (having all possible tuples). Then the sampling algorithm runs Chan's algorithm on (B, S) to measure the number of uncovered points in S. Then the algorithm generates a list L containing k random positive integers smaller than or equal to the number of uncovered points and runs the Retrieving algorithm which is a modification of Chan's algorithm in which the input of the algorithm also has L in a sorted array and an integer V that indicates the volume of visited uncovered points in the previous recursions of the Retrieving algorithm (initially 0). The recursion in Retrieving algorithm is similar to the ones in Chan's algorithm. The leaves of the recursive calls in Retrieving algorithm return an uncovered point uniformly at random for any of the numbers in L that is between the number of uncovered points in the prior leaves and the number of uncovered points in this leaves and the prior ones combined. Algorithms 3 and 4 describe the uniform sampling and retrieving process.

Algorithm 3 Sampling algorithm

1: **procedure** UNIFORM(B, S, k)
2: **Input:** A set of boxes B, a universe box S, and an integer k.
3: **Output:** A set of k uniformly sampled points in S that are not covered by B.
4: $V' \leftarrow$ Measure(B, S)
5: $L \leftarrow$ A sorted list of k uniformly random integers in $[1, V']$
6: **return** Retrieve($B, S, L, 0$)

Algorithm 4 Retrieving Samples Algorithm

1: **procedure** RETRIEVE(B, S, L, V)
2: **Input:** A set of boxes B, a universe box S, a list L of random integers between 1 to size of the points in S that are uncovered by B, and an auxiliary variable V that is zero in the initial call.
3: **Output:** A set of points in S that are uncovered by B and are associated with the random numbers in L.
4: **if** $|B|$ is below a constant **then**
5: $V' \leftarrow$ volume of uncovered points in S.
6: $k' \leftarrow |\{x : x \in L \text{ and } V < x \leq V + V'\}|$
7: $P \leftarrow k'$ uniformly sampled points in S uncovered by B.
8: return (P, V')
9: Simplify B and S
10: Cut S into two sub-cells S_1 and S_2.
11: $B_1 \leftarrow$ boxes in B that have intersection with S_1
12: $B_2 \leftarrow$ boxes in B that have intersection with S_2
13: $(V_1, P_1) \leftarrow$ Retrieve(B_1, S_1, L, V)
14: $(V_2, P_2) \leftarrow$ Retrieve($B_1, S_1, L, V + V_1$)
15: Map coordinates of P_1 and P_2 according to S before step 9.
16: **return** $(V_1 + V_2, P_1 \cup P_2)$

Note that since we simplify and cut S, the coordinates of the points returned by the recursive calls are not the same as their original coordinate. Therefore, we need to perform step 15 to change their coordinates.

Theorem 3.2. *The uniform sampling algorithm returns k uniformly at random selected uncovered points in time* $O(\log(k)|B|^{d/2} + kd^2 \log^2(|B|))$.

Proof. The only extra operations that Algorithm 4 is performing compared to Chan's algorithm is the sampling of the points and mapping the coordinates of the sampled points in the recursion. Step 6 is done in all of the leaves and it takes $O(\log(k))$. The sampling step happens once per sampled point and it takes $O(d)$. The only remaining step to analyse is the mapping step. Mapping can be done in $O(d \log(|B|))$ for each sampled point, because all we need to know for each dimension is the location of the slabs that are removed. The number of times each sampled point goes through the mapping phase is the depth of the recursion tree. The analysis of Chan's algorithm shows that the depth of the recursion is $O(\log(|B|)d)$. Therefore, the total time that step 15 takes in all of the recursions is $O(k \log^2(|B|)d^2)$. □

4. The Welltris Algorithm

Given a join query and the maximal dyadic gap boxes of all relations, we propose the following algorithm for estimating the row count of a join in time proportional to the optimal certificate size of that join. The proposed algorithm finds a $(1+\epsilon)$-approximation of the join size with probability at least $1-\delta$. Consider an input table T_i and a gap box b of T_i. We say b covers a point p if the projection of p onto the subspace, with the dimensions in T_i, lies inside b.

Algorithm 5 Welltris algorithm

1: **procedure** WELLTRIS($B_{\mathcal{D}}, S$)
2: **Input:** A set of dyadic gap boxes $B_{\mathcal{D}}$ in a Trie data structure and a universe box S.
3: **Output:** A $(1+\epsilon)$-approximation of the volume of the points in S that are uncovered by $B_{\mathcal{D}}$.
4: $E \leftarrow \emptyset$
5: $k \leftarrow \frac{4}{\epsilon}(\log(|B_{\mathcal{D}}|) + \log(\frac{1}{\delta}))$
6: **do**
7: $U \leftarrow \text{Uniform}(B_{\mathcal{D}}, S, k)$
8: **if** a point p in U is covered by any box in $B_{\mathcal{D}}$ **then**
9: add all the dyadic gap boxes in $B_{\mathcal{D}}$ that cover p to E.
10: **while** there is a point in U that is covered by $B_{\mathcal{D}}$
11: **return** the volume of the points not covered by E as the estimate of the join size.

In the following theorem statement recall that B is the set of all gap boxes of all tables and $B_{\mathcal{D}}$ is the set of all maximal dyadic gap boxes of all tables; see Theorem 3.1.

Theorem 4.1. *Let $C(B)$ be the smallest certificate over all choices of gap boxes and $C = |C(B)|$. Then, given the dyadic gap boxes constructed in the preprocessing algorithm (Algorithm 1), algorithm Welltris returns a $(1+\epsilon)$-estimation of the join size with probability at least $1-\delta$ in time $O(\frac{1}{\epsilon}\log(1/\delta)\cdot\log(m)\cdot 2^{d^2/2+d}\cdot\log^{d^2+d}(n)\cdot C^{d/2+1})$.*

Proof. First, we prove the approximation guarantee and then we prove the time complexity. The approximation guarantee can be proven by applying a Chernoff bound and a union bound. Let U_i be the set of uncovered points at iteration i and let $|J|$ be the actual join size. For an arbitrary iteration i, using the fact that k points are sampled independently, we will prove that if $|U_i| \geq (1+\epsilon)|J|$ then the probability that our algorithm stops in that iteration and returns $|U_i|$ is at most $\frac{\delta}{|B_{\mathcal{D}}|}$. Then using a union bound over all iterations, whose number is bounded by $|C(B_{\mathcal{D}})|$, we will show that the probability that our algorithm stops at any iteration for which $|U_i| > (1+\epsilon)|J|$ is at most δ.

Fix an iteration i. Let $p_1, p_2, \ldots, p_k$ be the points that the algorithm samples at iteration i, and let $X_1, X_2, \ldots, X_k$ be Bernoulli random variables such that X_j is 1 if and only if $p_j \notin J$ and let $X = \sum_{j=1}^{k} X_j$. Then we have

$$\Pr[X_j = 1] = \frac{|U_i \setminus J|}{|U_i|} \geq \frac{\epsilon}{1+\epsilon} \geq \frac{\epsilon}{2}$$

because $|U_i| > (1+\epsilon)|J|$. Note that the algorithm stops if $X = 0$. Using the fact that $\{X_j\}_{j\in[k]}$ are independent, we have

$$\Pr[X = 0] = \prod_{j\in[k]} \Pr[X_j = 0] \leq (1 - \frac{\epsilon}{2})^k \leq \exp(-\frac{k\epsilon}{2}).$$

Plugging in $k = \frac{4}{\epsilon}(\log(|B_{\mathcal{D}}|) + \log(\frac{1}{\delta}))$ we have $\Pr[X = 0] \leq \frac{\delta}{|B_{\mathcal{D}}|}$.

At each iteration of the algorithm, at least one of the boxes in the certificate of the dyadic boxes $C(B_{\mathcal{D}})$ is added to E, therefore the maximum number of iterations is $|C(B_{\mathcal{D}})|$ which is smaller than $|B_{\mathcal{D}}|$. Therefore, using a union bound over all iterations, we can conclude that with probability $1 - \delta$ the algorithm stops when $|U_i| \leq (1 + \epsilon)|J|$ and it returns a $(1 + \epsilon)$ approximation of the join size.

The time complexity of the algorithm depends on the time needed to sample the points, the time needed to find the boxes covering the sampled points, and the number of iterations. Let $C_{\mathcal{D}}$ denote $|C(B_{\mathcal{D}})|$ and C denote $|C(B)|$. Furthermore, let E_i denote the number of boxes in E at iteration i. At each iteration, based on Lemma 2.1, at most $O(\log^d(n))$ boxes are added to E including at least one of the boxes in $C(B_{\mathcal{D}})$; therefore, using Theorem 3.1, the number of iterations is at most $C_{\mathcal{D}}$, and for all iterations we have $E_i = O(\log^d(n)C_{\mathcal{D}})$.

Using Theorem 3.2, the sampling of k points in iteration i takes $O(\log(k)E_i^{d/2} + kd^2\log^2(E_i))$ which is at most $O(kE_i^{d/2})$. Based on Theorem 3.1, $|B_{\mathcal{D}}| = dtm\log^d(n)$. Therefore, we can replace k with the upper bound

$$k = \frac{4}{\epsilon}(\log(|B_{\mathcal{D}}|) + \log(\frac{1}{\delta})) = O(\frac{d}{\epsilon}\log(\frac{1}{\delta})\log(m)),$$

and replace E_i with $O(\log^d(n)C_{\mathcal{D}})$ to derive the following time complexity for the sampling step in each iteration:

$$O(kE_i^{d/2}) \leq O(\frac{d}{\epsilon}\log(\frac{1}{\delta})\log(m)\log^{d^2/2}(n)C_{\mathcal{D}}^{d/2}).$$

Finding out if a point is covered by any of the dyadic boxes in $B_{\mathcal{D}}$ can be done in time $O(d\log^{(d+1)}(n))$ using the Dyadic Trie data structure by looking for all possible $O(\log^d(n))$ dyadic boxes that include the point; See Section 3.2. Therefore, steps 8 and 9 can be performed in time

$$O(kd\log^{d+1}(n)) = O(\frac{d^2}{\epsilon}\log(\frac{1}{\delta})\log(m)\log^{d+1}(n)).$$

Multiplying the time complexity of all three steps by the number of iterations gives us the total time complexity of

$$O(\frac{d}{\epsilon}\log(\frac{1}{\delta})\log(m)\log^{d^2/2}(n)C_{\mathcal{D}}^{d/2+1} + \frac{d^2}{\epsilon}\log(\frac{1}{\delta})\log(m)\log^{d+1}(n)C_{\mathcal{D}}),$$

and by replacing $C_{\mathcal{D}}$ with $2^d\log^d(n)C$ we have

$$\begin{aligned} &O(\frac{d}{\epsilon}\log(\frac{1}{\delta})2^{d^2/2+d}\log^{d^2+d}(n)\log(m)C^{d/2+1} + \frac{d^2}{\epsilon}\log(\frac{1}{\delta})\log(m)2^d\log^{2d+1}(n)C) \\ &= O(\frac{1}{\epsilon}\log(\frac{1}{\delta})\log(m)2^{d^2/2+d}\log^{d^2+d}(n)C^{d/2+1}). \end{aligned}$$

□

We now explain how to modify Welltris to return q points sampled uniformly at random from the joined table J. The number k of points sampled in each iteration is set to be q. Then in step 8, every time that any of the sampled points is not in any gap box, that point is returned as one of the q sampled points. By rejection sampling, we can conclude these points are sampled uniformly at random from J.

Corollary 4.2. *This modification of Welltris samples q points uniformly at random from a join in time $O\left(\log(q)2^{d^2/2+d}\log^{d^2+d}(n)C^{d/2+1} + qd^4 2^d\log^{2d+1}(n)C\right)$.*

Proof. The time complexity of steps 7, 8, 9 in each iteration is $O(\log(q)E_i^{d/2} + qd^2\log^2(E_i) + qd\log^{d+1}(n))$. Note that $E_i = O(2^d\log^{2d}(n)C)$, and we can replace $\log(E_i)$ with $d\log(n)$ because the maximum size of the certificate is n^d. Note that every time that a sample is rejected, the algorithm adds at least a box from $C(B_{\mathcal{D}})$ to E_i; therefore, multiplying by the maximum number of iterations $C_{\mathcal{D}} = O(2^d\log^d(n)C)$, we derive the claimed worst case time complexity. □

References

[1] Afshani, P., Barbay, J., Chan, T.M., 2017. Instance-optimal geometric algorithms. J. ACM 64, 3:1–3:38.
[2] Alway, K., 2019. Domain Ordering and Box Cover Problems for Beyond Worst-Case Join Processing. Master's thesis. University of Waterloo.
[3] Alway, K., Blais, E., Salihoglu, S., 2019. Box covers and domain orderings for beyond worst-case join processing. CoRR abs/1909.12102.
[4] Chan, T.M., 2010. A (slightly) faster algorithm for Klee's measure problem. Computational Geometry 43, 243 – 250.
[5] Chan, T.M., 2013. Klee's measure problem made easy, in: IEEE Symposium on Foundations of Computer Science, pp. 410–419.
[6] Encyclopedia, W.T.F., . Welltris wikipedia page. URL: https://en.wikipedia.org/wiki/Welltris.
[7] Grohe, M., 2006. The structure of tractable constraint satisfaction problems, in: International Symposium on Mathematical Foundations of Computer Science, Springer. pp. 58–72.
[8] Khamis, M.A., Ngo, H.Q., Ré, C., Rudra, A., 2016. Joins via geometric resolutions: Worst case and beyond. ACM Transactions on Database Systems (TODS) 41, 22.
[9] Marx, D., 2013. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. J. ACM 60, 42.
[10] Roughgarden, T., 2019. Beyond worst-case analysis. Commun. ACM 62, 88–96.
[11] Samadian, A., Pruhs, K., Moseley, B., Im, S., Curtin, R.R., 2020. Unconditional coresets for regularized loss minimization, in: International Conference on Artificial Intelligence and Statistics, AISTATS, PMLR. pp. 482–492.
[12] Ullman, J., Garcia-Molina, H., Widom, J., 2001. Database Systems: The Complete Book. Prentice Hall PTR.