A Search-Free Intersection Algorithm

Akadej Udomchaiporn Veera Boonjing
Department of Mathematics and Computer Science, Faculty of Science
King Mongkut's Institute of Technology Ladkrabang
kuakadej@kmitl.ac.th kbveera@kmitl.ac.th

Abstract

This paper proposes a new intersection algorithm of sorted sets. The new algorithm employs a comparison-and-elimination approach to the intersection problem instead of using search algorithms as existing intersection solutions. It takes (1) O(1) times for the best case; (2) O(λ) times for the average case, where λ is an average size of sorted sets; and (3) O(kn) times for the worst case, where k is the number of sorted sets and n is the total elements of k sorted sets.

1. Introduction

The intersection problem of sorted sets is to determine a result set containing common elements appearing in all of these sets. This problem is a common problem in a context of queries in a database or a search engine. It is also known as the multiple search problem [1] since its purpose is to search for common elements in all input sets using efficient search algorithms such as binary search [2], Fibonaccian search [3], interpolation search [4-5], galloping search [6]. Existing algorithms differ on how they minimize search to reach a result set. The standard algorithm [7], called "Small versus Small", minimizes search by repeatedly intersecting the remaining two smallest sets by searching for each element in the smaller set from the larger set or searching for each element in the smaller remaining element set from the larger remaining element set. The adaptive algorithm [8] searches for an element of a particular set from all other remaining sets using galloping search and binary search. If the element being searched for is the common element, then their positions are remembered as the ending points of the search in order to minimize search. The sequential algorithm [9] is the adaptive algorithm performing one entire galloping search at a time in each next available set instead of a single galloping step. To obtain less search than the sequential algorithm, the random sequential algorithm [10] randomly chooses the remaining sets to search for the specific element. The Baeza-Yates algorithm [11-12] minimizes search by searching for median element of the smaller set from the larger set and dividing the problem into two sub-problems for each median element. The problems are then solved recursively. Each pair of the sorted result set and the remaining smallest set are then intersected using the same procedure. As mentioned above, all of these algorithms rely on a search algorithm employed and a way to minimize search. However, the nature of intersection problem of sorted sets allows us to determine a result set without relying on them.

This paper proposes a new approach to intersection problem of sorted sets using a comparison-and-elimination approach. The new algorithm determines elements of the result set by repeatedly finding a maximum value of all first elements and a minimum value of all last elements. These two values becomes a range of possible elements of result set. Hence, we use them to eliminate elements of all sets not in this range. The common minimum or maximum values become elements of result set and they are eliminated from all sets.

The rest of paper is organized as follows. Related works are given in section 2. Section 3 describes the new intersection algorithm named a Search-Free Intersection Algorithm, and section 4 proves time complexity of the algorithm. Finally, section 5 concludes the paper.

2. Related Works

In research conducted on this issue, there are many works related to intersection algorithms introduced in section 1. In this section, some of them are analyzed and described in more details. First of all, Demaine, et al. [2] propose an algorithm for finding the intersections, unions, and differences of a collection of sorted sets, and the framework for designing and evaluating adaptive algorithms in the comparison model is also presented. Advantage of this work is quickly search in a database or data warehousing environment, and is also quickly in finding unions of sorted sets. On the other hand, running time of finding intersections and differences of sorted sets is not good as unions, and it is difficult to compute because its scaled running time depends on gap cost and difficulty factors of the problem. Barbay, et al. [3-4] present finding the intersection set of sorted sets using several improved algorithms and a variant of interpolation search. They also compare the number of comparisons and searches performed among those intersection algorithms. In addition, Yehoshua, et al. [5] apply

978-0-7695-3099-4/08 $25.00 © 2008 IEEE
DOI 10.1109/ITNG.2008.225

interpolation search to alphabetic tables, but its problem is that such tables are far from uniform data distribution. Therefore, we can sum up that interpolation search is not suitable for sorted sets which their data distribution is not normal. Bentley, et al. [6] introduce galloping search, which is a problem of searching for a rank of a key in a sorted sets with unbounded size. A search begins with the first position and then increases to 2k+1 every time, where k is a position of sorted sets. This technique can solve unbounded search problem, but the number of comparisons is more than many intersection algorithms. Hwang, et al. [7] introduce a widely used algorithm called Small versus Small (SvS). It intersects the sets two at a time in increasing order by size, starting with the two smallest, and repeatedly intersecting the remaining two smallest sets by searching for each element in the smaller set from the larger set, or searching for each element in the smaller remaining element set from the larger remaining element set. This algorithm is usually solved by recursion, so it takes much more space than almost all algorithms. The adaptive algorithm proposed by Demaine, et al. [8] searches for an element of a particular set from all other remaining sets using galloping and binary search. It deals with an encoding of the shortest proof of the answer, so it is practical significance in web search engines. However, many techniques used in this algorithm makes some negative impacts such as high adaptive complexity especially in numerous sets of data. Barbay, et al. [9-10] propose sequential and random sequential algorithm. The sequential algorithm is an adaptive algorithm performing one entire galloping search at a time in each next available set instead of a single galloping step. It is optimal for a different measure of difficulty, based on the non-deterministic complexity of the instance. For random sequential algorithm, it performs comparisons less than sequential algorithm on average, because the random sequential algorithm randomly chooses the remaining sets to search for the specific element instead of performing one entire gallop search at a time. However, both sequential and random sequential algorithms take more time in a search than almost all mentioned algorithms. Baeza-Yates, et al. [11-12] present Baeza-Yates algorithm intended for the intersection of two sorted sets. It takes the median element of the smaller set and searches for it in a larger set. If the element is in a larger set, it will be added to the result set. This algorithm solves recursively the instances formed by each pair of subsets, so the memory space used is more than any other described algorithms. To evaluate efficiencies of intersection algorithms, alternation and redundancy analysis [13] can be applied to them. The alternation analysis bases on the non-deterministic complexity of an instance, while the redundancy analysis concerns a measure of the internal redundancy of the instance. Both alternation and redundancy analysis can be used as a comparison model to compare the intersection algorithms.

According to the intersection algorithms mentioned in this section, there are many disadvantages such as using a lot of memory space, high adaptation complexity, and taking much time to find the intersection set of sorted sets. Furthermore, all algorithms rely on search problems which are critical in aspect of time and other constraints. Therefore, a comparison-and-elimination approach named a Search-Free Intersection Algorithm is proposed in order to solve limitations of intersection search problems.

3. A Search-Free Intersection Algorithm

A Search-Free Intersection Algorithm is an algorithm for finding the intersection set of sorted sets using a comparison-and-elimination approach. It is described in details in this section. Firstly, some definitions relating to the algorithm are introduced in section 3.1.

3.1 Definitions

There are 4 definitions relating to the algorithm shown as follows.

Definition 3.1.1

Let k sorted sets be A_1, A_2, … , A_k. The intersection set (I) of those sorted sets, shown as $A_1 \cap A_2 \cap \ldots \cap A_k$ is the set consisting of the elements appearing in every A_1, A_2, … , A_k.

Example 3.1.1

Assume A_1, A_2 and A_3 are 3 sorted sets, and their elements are assumed as follows.

$A_1 = \{2, 4, 6, 7, 8, 10, 12\}$

$A_2 = \{1, 3, 4, 5, 6, 8, 9\}$

$A_3 = \{1, 4, 5, 7, 8, 9, 11, 13\}$

A_1:		2		4		6	7	8		10		12	
A_2:	1		3	4	5	6		8	9				
A_3:	1			4	5		7	8	9		11		13

Figure 3.1 An example of definition 3.1.1

From example 3.1.1, the intersection set (I) of the sorted sets A_1, A_2 and A_3 is {4, 8}, or $A_1 \cap A_2 \cap A_3$ is {4, 8}.

Definition 3.1.2

Let k sorted sets be A_1, A_2, … , A_k. l_i is the first index number of A_i whose element is not null, and r_i is the last index number of A_i whose element is not null. ($1 \leq i \leq k$)

Example 3.1.2

Assume A_1, A_2 and A_3 are 3 sorted sets, and their elements are assumed as follows.

A_1 = {*n*, *n*, 4, 6, 7, 8, 9, *n*, *n*, *n*}
A_2 = {1, 2, 3, 4, 6, 7, 8, 9, *n*, *n*}
A_3 = {*n*, *n*, *n*, *n*, *n*, 6, 7, 8, 9, 10}

Index	1	2	3	4	5	6	7	8	9	10
A_1:	*n*	*n*	4	6	7	8	9	*n*	*n*	*n*
A_2:	1	2	3	4	6	7	8	9	*n*	*n*
A_3:	*n*	*n*	*n*	*n*	*n*	6	7	8	9	10

Figure 3.2 An example of definition 3.1.2

From example 3.1.2, *n* is assumed as a null value, so the value of l_i and r_i, where $1 \le i \le 3$, can be shown as follows.

- l_1 is 3, and r_1 is 7.
- l_2 is 1, and r_2 is 8.
- l_3 is 6, and r_3 is 10.

Definition 3.1.3

Let k sorted sets be A_1, A_2, … , A_k. A set A_i ($1 \le i \le k$) is an empty set (∅), shown as ($A_i = \emptyset$), when it contains only null (*n*) value.

Example 3.1.3

Assume A_1, A_2 and A_3 are 3 sorted sets. Their elements are assumed as follows.
A_1 = {*n*, *n*, *n*, *n*, *n*, *n*, *n*, *n*, *n*, *n*}
A_2 = {*n*, *n*, *n*, 4, 6, 7, 8, *n*, *n*, *n*}
A_3 = {*n*, *n*, *n*, *n*, *n*, *n*, *n*, *n*, *n*, *n*}

Index	1	2	3	4	5	6	7	8	9	10
A_1:	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*
A_2:	*n*	*n*	*n*	4	6	7	8	*n*	*n*	*n*
A_3:	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*	*n*

Figure 3.3 An example of definition 3.1.3

From example 3.1.3, the empty sets are the set A_1 and A_3 because they contain only null (*n*) value.

Definition 3.1.4

Let k sorted sets be A_1, A_2, … , A_k, and all sets contain ascending order data.

$L = \text{Max}\{A_i[l_i]\}$,

$R = \text{Min}\{A_i[r_i]\}$,

I = the intersection set of k sorted sets A_i, where $A_i \ne \emptyset$, and $1 \le i \le k$.

An intersection element in the intersection set (I) must be in an interval [L, R].

Example 3.1.4

Assume A_1, A_2 and A_3 are 3 sorted sets, and their elements are assumed as follows.
A_1 = {2, 4, 6, 7, 8, 10, 12}
A_2 = {1, 3, 4, 5, 6, 8, 9}
A_3 = {1, 4, 5, 7, 8, 9, 11, 13}

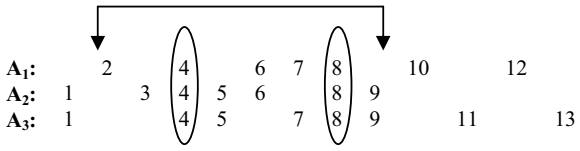

Figure 3.4 An example of definition 3.1.4

From example 3.1.4, it shows that 4 and 8 are the intersection elements of the intersection set (I), and they must be in the interval [2, 9]. The value of L and R are shown as follows.

- L = Max{1, 1, 2} = 2
- R = Min{9, 12, 13} = 9

3.2 A Search-Free Intersection Algorithm

The input of a Search-Free Intersection Algorithm is k sorted sets, where the elements in each set are unique. The output of the algorithm is the intersection set (I). The algorithm details are described as follows.

Input: k sorted sets (arrays) A_i, where $1 \le i \le k$, and the elements in a set A_i are unique.

Output: The intersection set (I) = $A_1 \cap A_2 \cap_{\ldots} \cap A_k$

Algorithm:

```
Find L and R  (from definition 3.1.4)                    // (line 1)
While ((L ≤ R) and (Every Ai ≠ ∅))                       // (line 2)
    If  (A1[l1] = A2[l2] = … = Ak[lk]) Then
        Add A1[l1] to I;
        Set every Ai[li] = null; // (1 ≤ i ≤ k)
        If ((A1[r1] = A2[r2] = … = Ak[rk]) and
        (A1[r1] ≠ null)) Then
            Add A1[r1] to I;
            Set every Ai[ri] = null; // (1 ≤ i ≤ k)
    Else
        i =1;
        While (i <= k)
            While (Ai[li] < L) Then    // (line 13)
                Set Ai[li] = null;
                Increment li; // (Move right)
            End While
            While (Ai[ri] > R) Then   // (line 17)
                Set Ai[ri] = null;
                Decrement ri; // (Move left)
            End While
            increment i;
        End While
    End If
    Find L and R
End While                                                // (line 25)
```

Figure 3.5 A Search-Free Intersection Algorithm

Example 3.2

Assume A_1, A_2 and A_3 are sorted sets, and their elements are assumed as follows.

$A_1 = \{3, 5, 6, 7, 8, 9, 11, 13\}$

$A_2 = \{2, 3, 4, 5, 6, 9\}$

$A_3 = \{1, 4, 6, 7, 8, 10, 12\}$

A_1:			3		5	6	7	8	9		11		13
A_2:		2	3	4	5	6			9				
A_3:	1			4		6	7	8		10		12	

Figure 3.6 Each element of all sorted sets in example 3.2

Iteration 1: Find L and R

L = Max{3, 2, 1} = 3

R = Min{13, 9, 12} = 9

If any elements in every sorted set (A_1, A_2, and A_3) are less than L, and are greater than R, those elements are removed. Thus,

- 1, 2 are less than 3, so they are removed.
- 10, 11, 12, 13 are greater than 9, so they are removed.

Therefore, the elements of A_1, A_2, and A_3 after iteration 1 are shown as follows.

$A_1 = \{3, 5, 6, 7, 8, 9, n, n\}$

$A_2 = \{n, 3, 4, 5, 6, 9\}$

$A_3 = \{n, 4, 6, 7, 8, n, n\}$

A_1:			3		5	6	7	8	9		n		n
A_2:		n	3	4	5	6			9				
A_3:	n			4		6	7	8		n		n	

Figure 3.7 Each element of all sorted sets after iteration 1

Iteration 2: Find L and R

L = Max{3, 3, 4} = 4

R = Min{9, 9, 8} = 8

- 3 is less than 4, so 3 is removed.
- 9 is greater than 8, so 9 is removed.

Therefore, the elements of A_1, A_2, and A_3 after iteration 2 are shown as follows.

$A_1 = \{n, 5, 6, 7, 8, n, n, n\}$

$A_2 = \{n, n, 4, 5, 6, n\}$

$A_3 = \{n, 4, 6, 7, 8, n, n\}$

A_1:			n		5	6	7	8	n		n		n
A_2:		n	n	4	5	6			n				
A_3:	n			4		6	7	8		n		n	

Figure 3.8 Each element of all sorted sets after iteration 2

Interation 3: Find L and R

L = Max{5, 4, 4} = 5

R = Min{8, 6, 8} = 6

- 4 is less than 5, so 4 is removed.
- 7, 8 are greater than 6, so they are removed.

Therefore, the elements of A_1, A_2, and A_3 after iteration 3 are shown as follows.

$A_1 = \{n, 5, 6, n, n, n, n, n\}$

$A_2 = \{n, n, n, 5, 6, n\}$

$A_3 = \{n, n, 6, n, n, n, n\}$

A_1:			n		5	6	n	n	n		n		n
A_2:		n	n	n	5	6			n				
A_3:	n			n		6	n	n		n		n	

Figure 3.9 Each element of all sorted sets after iteration 3

Iteration 4: Find L and R

L = Max{5, 5, 6} = 6

R = Min{6, 6, 6} = 6

- 5 is less than 6, so 5 is removed.

Therefore, the elements of A_1, A_2, and A_3 after iteration 4 are shown as follows.

$A_1 = \{n, n, 6, n, n, n, n, n\}$

$A_2 = \{n, n, n, n, 6, n\}$

$A_3 = \{n, n, 6, n, n, n, n\}$

A_1:			n		n	6	n	n	n		n		n
A_2:		n	n	n	n	6			n				
A_3:	n			n		6	n	n		n		n	

Figure 3.10 Each element of all sorted sets after iteration 4

Iteration 5: Find L and R

L = Max{6, 6, 6} = 6

R = Min{6, 6, 6} = 6

In this iteration, $A_1[l_1] = A_2[l_2] = A_3[l_3]$, so the element 6 is added to the intersection set (I). After that, $A_1[l_1]$, $A_2[l_2]$, and $A_3[l_3]$ will be set to be null. After iteration 5, the elements of A_1, A_2, and A_3 are shown as follows.

$A_1 = \{n, n, n, n, n, n, n, n\}$

$A_2 = \{n, n, n, n, n, n\}$

$A_3 = \{n, n, n, n, n, n, n\}$

A_1:			n		n	n	n	n	n		n		n
A_2:		n	n	n	n	n			n				
A_3:	n			n		n	n	n		n		n	

Figure 3.11 Each element of all sorted sets after iteration 5

Iteration 6: Find L and R

L = Max{n, n, n} = n

R = Min{n, n, n} = n

In this iteration, (L > R), and (A_1, A_2, and $A_3 = \varnothing$), so the algorithm is terminated. Finally, the intersection set (I) of the sorted sets A_1, A_2, and A_3 ($A_1 \cap A_2 \cap A_3$) is {6}.

However, sample sorted sets of ascending order are shown in example 3.2, but not of descending order. If the sorted sets contained descending order data, the algorithm would be changed as follows:

In the first line of the algorithm, finding L and R from definition 3.1.4 would be changed to:

$L = \text{Min}\{A_i[l_i]\}$,

$R = \text{Max}\{A_i[r_i]\}$, where $A_i \neq \varnothing$, and $1 \leq i \leq k$

The next line, the statement would be changed to:

While ((L ≤ R) and (Every $A_i \neq \varnothing$))

In the thirteenth line, it would be changed to:

While ($A_i[l_i]$ > L) Then

Finally, in the seventeenth line, it would be changed to:

While ($A_i[r_i] < R$) Then

4. Time Complexity

In this section, time complexity of a Search-Free Intersection Algorithm is evaluated. The best case, average case, and worst case of time complexity are summarized in theorem 4.1 as follows.

Theorem 4.1: the best case of a Search-Free Intersection Algorithm is performed on O(1), the average case is performed on O(λ), where λ is an average size of sorted sets, and the worst case is performed on O(kn), where k is a number of sorted sets.

Best case proof:

Suppose L = m, and R = n, and there does not exist element between m and n, or m = n. This is the best case where the algorithm can be terminated in the first iteration. Hence, its time complexity is O(1).

Average case proof:

Suppose $\lambda = \frac{1}{k}\sum_{i=1}^{k} |A_i|$. The algorithm can perform in λ iterations. Thus, its time complexity is O(λ).

Worst case proof:

Suppose L = p, R = q, n = $\frac{1}{k}\sum_{i=1}^{k} |A_i|$, and there exist n elements in A_i including p and q. Therefore, the algorithm's time complexity is O(kn).

5. Conclusions

This paper proposes an algorithm for searching the intersection set of any sorted sets. The algorithm uses a comparison-and-elimination approach to find the intersection set instead of using existing search algorithms. According to theorem 4.1, time complexity of a Search-Free Intersection Algorithm which is our approach is O(1) times for the best case, O(λ) times for the average case, where λ is an average size of sorted sets, and O(kn) times for the worst case, where k is a number of sorted sets, and n is the total elements of k sorted sets.

6. References

[1] R. Baeza-Yates and B. Ribeiro-Neto, *"Modern Information Retrieval"*, Addison-Wesley, 1999.

[2] E. D. Demaine, A. López-Ortiz and J. I. Munro, *"Adaptive Set Intersections, Unions, and Differences"*, Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA), p. 743–752, 2000.

[3] J. Barbay, A. López-Ortiz, T. Lu and A. Salinger, *"Faster Set Intersections Algorithms for Text Searching"*, ACM Journal of Experimental Algorithmics, September 2006.

[4] J. Barbay, A. López-Ortiz and T. Lu, *"Faster Adaptive Set Intersections for Text Searching"*, Proceedings volume 4007 of Lecture Notes in Computer Science (LNCS), p. 146–157, Springer Berlin / Heidelberg, 2006.

[5] P. Yehoshua and G. Loizos, *"Arithmatic Interpolation Search for Alphabetic Tables"*, IEEE Transactions on Computers, Vol. 41, No. 4, April 1992.

[6] J. L. Bentley and A.C.C. Yao, *"An Almost Optimal Algorithm for Unbounded Searching"*, Information processing letters, p. 82-87, 1976.

[7] F.K. Hwang and S. Lin, *"Optimal Merging of 2 Elements with n Elements"*, Acta Informatica, p.145-158, 1971.

[8] E. D. Demaine, A. López-Ortiz, and J. I. Munro, *"Experiments on Adaptive Set Intersections for Text Retrieval Systems"*, Proceedings of the 3rd Workshop on Algorithm Engineering and Experiments, Lecture Notes in Computer Science, p. 5–6, Washington DC, 2001.

[9] J. Barbay and C. Kenyon, *"Adaptive Intersection and t-Threshold Problems"*. Proceedings of the thirteenth ACM-SIAM Symposium On Discrete Algorithms (SODA), p. 390–399, ACM, 2002.

[10] J. Barbay, *"Optimality of Randomized Algorithms for the Intersection Problem"*, Proceedings of the Symposium on Stochastic Algorithms, Foundations and Applications (SAGA), Vol. 2827/2003, p. 26-38, Lecture Notes in Computer Science (LNCS), Springer-Verlag Heidelberg, 2003.

[11] R. Baeza-Yates, *"A Fast Set Intersection Algorithm for Sorted Sequences"*, Proceedings of the 15th AnnualSymposium on Combinatorial Pattern Matching (CPM), Vol. 3109 of *LNCS*, p. 400-408, Springer, 2004.

[12] R. Baeza-Yates and A. Salinger, *"Experimental analysis of a fast intersection algorithm for sorted sequences"*, Proceedings of 12th International Conference on String Processing and Information Retrieval, p. 13-24, 2005.

[13] J. Barbay and C. Kenyon, *"Alternation and Redundancy Analysis of the Intersection Problem"*, ACM Journal, Vol. 5, p. 1-18, 2006.