Adaptive Set Intersections, Unions, and Differences

Erik D. Demaine, Alejandro López-Ortiz, J. Ian Munro
SODA '00

Find intersection of two sorted sets

Extreme Case 1

If the sets interleave perfectly $\Omega(n)$ comparisons are required.

Extreme Case 2

if all elements of one set are known to fall between a pair of consecutive elements in the other set, the problem simply reduces to a search in a sorted array and so $\log_2 n+O(1)$ upper and lower bound apply.

Adaptive Algorithm

Adaptive Algorithm

- We cannot use the worst-case running time as our metric, because that will only reflect the case in which the instance is difficult to solve.
- We can think of a ratio as a scaled running time, which allows the running time to be large for difficult instances, but enforce it to be small for easy instances.
- Difficulty could be defined as an information theoretic measure of the difficulty of the instance.
- An algorithm that minimizes the worst case scaled running time is a natural definition of an optimal adaptive algorithm.
- It should make no a priori assumptions, but determine the kind of instance it faces as the computation proceeds.

Terminology

- Instance
- Arguement
- Set Problem
- Proof

Instance

an *instance* of the problem is a collection of k sets $A_1, \ldots, A_k$, each presented in sorted order. Hence

$$A_s = \{A_s[1], \ldots, A_s[n_s]\}$$

implies that $A_s[i] < A_s[j]$ for all s and $i < j$. Some basic terminology that we will use throughout this paper is as follows. An *element* is one of the $A_s[i]$'s; a *value* is a member of the universe, which may occur as an *element* in several sets. An element $A_s[i]$ *precedes* [*weakly precedes*] $A_s[j]$ if $i < j$ [$i \leq j$]. Successors and weak successors are defined similarly; note that they only involve elements in the same set.

Arguement

Formally, an *argument* is a finite set of symbolic equalities and inequalities, or *comparisons*, of the form $(A_s[i] < A_t[j])$ or $(A_s[i] = A_t[j])$ for $s, t, i, j \geq 1$.

Set Problem

Consider the following *set problems*:

1. *Intersection*: Compute $A_1 \cap \cdots \cap A_k$.
2. *Union*: Compute $A_1 \cup \cdots \cup A_k$.
3. *Difference*: Compute $A_1 - (A_2 \cap \cdots \cap A_k)$.[2]

Proof

The most interesting classes of arguments are those that prove that the answer to one of the three problems is a particular set. Formally, an argument P is called a *proof* for a particular set problem if all of the instances satisfying P have the same solution to that problem. If the answer is always the set A of elements, we call P an A-proof.

Intersection Proofs

Intersection Proof

Intersection Proofs. An intersection proof must show precisely which elements are contained in all sets:

An argument P is a B-proof for the intersection problem precisely if there are elements $b_1, \ldots, b_k$ for each $b \in B$, where b_i is an element of A_i and has the same value as b, such that

1. *for each $b \in B$, there is a tree on k vertices, every edge (i, j) of which satisfies $(b_i = b_j) \in P$; and*

2. *for consecutive values $b, c \in B \cup \{+\infty, -\infty\}$, the subargument involving the following elements is a $\emptyset$-proof for that subinstance: from each A_i, take the elements strictly between b_i and c_i.*

A Greedy Algorithm

Method Fewest-Comparisons

1. Initialize the eliminator e to the maximum element $A_s[1]$ over all $1 \leq s \leq k$.
2. Until e becomes infinity:
   (a) Add the comparison $(a < e)$ to the proof, where the element a is chosen so that its immediate successor e' is maximized, subject to the constraint that $a < e$.
   (b) If $e \neq e'$, set e to e'.
   (c) Otherwise, e is present in all sets:
      i. Remove the just-added comparison $(a < e)$ from the proof.
      ii. Add the comparisons $(e_s = e_{s+1})$ to the proof, where e_s is the occurrence of e in A_s, for each $1 \leq s < k$.
      iii. Reinitialize e to the maximum immediate successor of e_i over all $1 \leq s \leq k$.

Problem Formulation

Given positive integers k, p, and g ($p \leq g$), and given an algorithm for finding $\emptyset$-proofs for the intersection problem, there is a collection of k sets having a p-comparison $\emptyset$-proof with cost $O(p \log_2 k + g)$, such that every $\emptyset$-proof has gap cost $\Omega(g)$, and the algorithm takes $\Omega(kg)$ time on this input. In particular, $\mathcal{D} = O(p \log_2 k + g)$ and $\mathcal{G} = \Omega(g)$, so the scaled running time is $\Omega(k\mathcal{G}/\mathcal{D})$ for this instance.

Algorithm Empty-Intersect Part1

Algorithm Empty-Intersect

- Initialize low-jump(s) and high-jump(s) to 1, and done(s) to 0, for each $s \in \{1, \ldots, k\}$.
- Initialize elim-set to 1 and eliminator to $A_1[1]$.
- For s ranging through $\{1, \ldots, k\}$ cyclicly:
  - Skip this step if s = elim-set.
  - Low step:
    1. Let p = done(s) + low-jump(s).
    2. If $A_s[p] \geq$ eliminator (we overshot),
       (a) Binary search in the interval [done(s) + $1, p$] to find the smallest p' with $A_s[p'] \geq$ eliminator.
       (b) If $p' - 1 >$ done(s), add ($A_s[p' - 1] <$ eliminator) to the proof.
       (c) Set done(s) to $p' - 1$, and low-jump(s) to 1.
       (d) Set elim-set to s, and eliminator to $A_s[p']$.
    3. Otherwise, double low-jump(s) and set done(s) to p.

Algorithm Empty-Intersect Part2

- High step:
  1. Let $p = n_s + 1 - \text{high-jump}(s)$.
  2. If $A_s[p] <$ eliminator (we overshot),
     (a) Binary search in the interval $[p, n_s]$ to find the largest p' with $A_s[p'] <$ eliminator.
     (b) If $p' >$ done(s), add $(A_s[p'] <$ eliminator$)$ to the proof.
     (c) If $p' = n_s$, stop.
     (d) Set done(s) to p', and low-jump(s) to 1.
     (e) Set elim-set to s, and eliminator to $A_s[p'+1]$.
     (f) Reset high-jump(s) to one.
  3. Otherwise, double high-jump(s).

Note that at any point in time, $A_s[i]$ is eliminated exactly when $i \leq$ done(s).

Now we claim that the algorithm matches the lower bound from Section 4, that is, has scaled running time $O(k\mathcal{G}/\mathcal{D})$.

Extend to B-Proof

This follows from a simple modification to Algorithm Empty-Intersect above, namely whenever a comparison with the eliminator returns "equal," stop galloping in that set and increase the occurrence count of the eliminator. If the occurrence count reaches k, output the eliminator as part of the intersection, add $k - 1$ appropriate comparisons to the proof, and take the eliminator's successor as the new eliminator.

Conclusion

The intersection of k sorted sets can be computed in $O(k\mathcal{G})$ time and at most $8k\mathcal{G}$ comparisons.

Thanks!

Any questions ?

You can find me at

- qw376@nyu.edu
- www.qi-wang.com