Adaptive Intersection and t-Threshold Problems

Jérémy Barbay* Claire Kenyon*

Abstract

Consider the problem of computing the intersection of k sorted sets. In the comparison model, we prove a new lower bound which depends on the non-deterministic complexity of the instance, and implies that the algorithm of Demaine, López-Ortiz and Munro [2] is usually optimal in this "adaptive" sense. We extend the lower bound and the algorithm to the *t-Threshold Problem*, which consists in finding the elements which are in at least t of the k sets. These problems are motivated by boolean queries in text database systems.

1 Introduction

It is easy to compute in $O(n)$ the intersection of two sorted sets of size n, and this complexity is optimal in the worst case. However, in some cases the intersection might be much easier to compute, for example if all elements of the first set are smaller than all elements of the second set: then the intersection is empty, and a single well-chosen comparison is sufficient to certify it. One then wishes to design an "adaptive" algorithm, which is faster when the instance is "easy". This follows the same general philosophy as the search for output-sensitive algorithms in Computational Geometry, or for adaptive sorting algorithms as in [4].

Adaptive algorithms for the intersection or the union of k sorted sets were studied by Demaine, López-Ortiz and Munro in the context of Internet information queries and text database systems [2, 3]. If queries are composed of words, and for each keyword a sorted set of references to entries in the database is pre-computed, then the set of data entries in the database matching all the keywords of a query is the intersection of the sorted sets corresponding to each keyword: this is the application of the Intersection Problem to search engines, as defined in [3]. The present work was originally inspired by [2], where the authors propose a measure of difficulty of instances of the Intersection or Union problems based on the encoding size of the proof of the output, and design ingenious algorithms for these two problems; they prove that their algorithm for computing the union of k sets is optimal in the adaptive sens, and that their algorithm for computing the intersection has complexity at most $O(k)$ times their lower bound.

In this paper, we define a different, simpler and perhaps more natural measure of difficulty for an instance. Any (even non-deterministic) algorithm for the Intersection Problem must certify that the output is correct: first, it must certify that all the elements of the output are indeed elements of all the sets; second, it must certify that no element of the intersection has been omitted by exhibiting some inequalities which imply that there can be no other element in the intersection. In the comparison model, the first goal is reached by verifying $(k-1)\#I$ equalities, and the second goal is reached by verifying a certain set P of strict inequalities. We define the difficulty of an instance as $\delta = \#I + \min_P \#P$, and, in Theorem 3.2, our main result in this part of the paper, prove that any deterministic or randomized algorithm for the Intersection Problem must make at least $\Omega(\delta \sum_i \log(n_i/\delta))$ comparisons on instances with k sorted sets of sizes $n_1, \ldots, n_k$ respectively, an intersection size $\#I$ and difficulty δ. The proof of this lower bound is in 4 steps:

1. Reduction of the Intersection Problem to a set of $O(\delta)$ instances of the Intersection Element Problem in Theorem 3.2.
2. Definition of the distributions D_- and D_+ on the Intersection Element Problem in Theorems 2.1 and 2.2.
3. Analysis of the decision tree of any algorithm on D_- and D_+. (Lemma 2.3 is the keypoint.)
4. Use of the Yao-von Neumann Minimax principle [7, 6] to deduce a lower bound on randomized algorithms in Corollary 3.1.

We study first the Intersection Element Problem in section 2, and then use it in section 3 to prove results on the Intersection Problem. Then we observe in Theorem 3.3 that the algorithm of [2] has complexity $O(k\delta\log(n/\delta))$, hence is optimal when $\max_i \log n_i = O(\min_i \log n_i)$ and $k = o(n)$. Along the way, in Theorem 3.1 we provide a nice characterization of certificates.

*{jeremy,kenyon}@lri.fr, LRI, CNRS UMR 8623, Université Paris-Sud, 91405 ORSAY Cedex, FRANCE.

In the second part of the paper, we study a more general problem, the t-Threshold Problem: given k sorted sets, find the elements which are in at least t of the k sets. We extend our lower bound to prove in Theorem 4.2 a lower bound of $\Omega(\delta \sum_{i=1}^{t} \log(n_i/\delta))$ for this problem if $n_1 \geq \ldots \geq n_k$; we design an algorithm for the t-Threshold Problem, which is an appropriate generalization of the algorithm for the Intersection Problem; we analyze it and prove in Theorem 3.3 that its complexity is $O(t\delta \log(k) \log(n))$. We do not currently know whether this factor of $\log k$ can be removed from the analysis. (Note that in the applications mentioned in [2, 3], k is usually quite small.)

We conclude with some perspectives and suggestions of related questions.

2 The Intersection Element Problem

Definition 2.1. *The* insertion rank *of an element x in a set A is the rank of x in $A \cup \{x\}$.*

(Note that if $x \in A$ then the insertion rank is just the rank).

Definition 2.2. *Given k sorted sets $(A_1, A_2, \ldots, A_k)$ and an element x, the* Intersection Element Problem *consists in deciding whether $x \in A_1 \cap A_2 \cap \ldots \cap A_k$.*

The quantities (l, x, r) of the following theorems will be useful for the reduction from the Intersection Problem.

Theorem 2.1. *For any positive integers k, $n_1, n_2, \ldots, n_k$, and any real numbers $l < x < r$, there exists a distribution $D_+(l, x, r)$ on instances with k sorted sets, where set A_i has size n_i, and all the elements of every set A_i are in (l, r), such that x is in the intersection, and such that any deterministic algorithm for the Intersection Element Problem for x performs on average $\Omega(\sum_i \log(n_i))$ comparisons.*

Proof. The proof is a simple counting argument. Let us define the following distribution $D_+(l, x, r)$, which will also be useful in the lower bound for the Intersection Problem.

- For each i, a rank p_i is chosen uniformly at random from $\{1, \ldots, n_i\}$;
- Let $A_i[p_i] = x$; all the other elements are in (l, r);

To certify that x is in the intersection, the algorithm must find the rank of x in every A_i, so to each leaf of the decision tree one can associate a k-tuple $(p_1, p_2, \ldots, p_k)$; the number of such k-tuples is $\prod_i n_i$, and $D_+(l, x, r)$ gives a uniform distribution over these k-tuples, which immediately implies a lower bound of $\sum_i \log_2 n_i$ on the expected performance of any deterministic algorithm. $\square$

Theorem 2.2. *For any positive integers $k, n_1, n_2, \ldots, n_k$, and any real numbers $l < x < r$, there exists a distribution $D_-(l, x, r)$ on instances with k sorted sets, where set A_i has size n_i, and all the elements of every set A_i are in the open interval (l, r), such that x is not in the intersection, and such that any deterministic algorithm for the Intersection Element Problem for x performs on average at least $\frac{1}{4} \sum_i \log_4(n_i) - k$ comparisons.*

Remark that a simple adversary argument yields a $\Omega(\sum_i \log(n_i))$ lower bound *in the worst case*. This is in fact the proof of Lemma 4.4 in [2].

Also remark that this is more difficult than Theorem 2.1, since a leaf of the decision tree will certify that $x \notin A_1 \cap \ldots \cap A_k$ simply by exhibiting a set A_i and an index p such that $A_i[p] < x < A_i[p+1]$; there are only n such pairs (i, p), so the straightforward counting argument would only yield a lower bound of $\log_2 n$. To improve on that lower bound, one must argue that in order to find the certificate (i, p), along the way the algorithm must have determined the rank of x in many other sets as well.

Proof. Consider the following distribution $D_-(l, x, r)$:

- For each i, a rank p_i is chosen uniformly at random from $\{1, \ldots, n_i\}$;
- One set A_i is chosen at random to be A_{i_0} with probability $\alpha_i = \log_4 n_i / \sum_\ell \log_4 n_\ell$;
- Define $y = x + \varepsilon$. Let $A_{i_0}[p_{i_0}] = y$, and for $i \neq i_0$, let $A_i[p_i] = x$; all other elements are either in (l, x) or in (y, r);
- Complete the sets so that the following property holds. For each $i < i'$, let $a \in A_i$ and $a' \in A_{i'}$. If a and a' are both less than x, then $a < a'$. Similarly, if a and a' are both greater than y, then $a < a'$.

In the rest of the proof we will note D_- the distribution $D_-(l, x, r)$.

Lemma 2.1. *Let A be an algorithm for the Intersection Element Problem in the comparison model. There exists an algorithm B for the Intersection Element Problem on D_- which, instead of comparisons, performs queries of the form "How does $A_i[p]$ compares to x and y?", with four possible outcomes: $A_i[p] < x$, $A_i[p] = x$, $A_i[p] = y$ and $A_i[p] > y$; the number of queries performed by algorithm B is at most twice the number of comparisons performed by A.*

Proof. We use the "little birdy principle" and assume that B knows that the instance is drawn from D_-.

Whenever A would compare $A_i[p]$ to $A_{i'}[p']$, B does two queries instead: one for $A_i[p]$, and one for $A_{i'}[p']$. The results, together with the knowledge of D_-, determine what the outcome of the comparison $A_i[p] : A_{i'}[p']$ would have been:

- if $i = i'$ or if x or y is equal to $A_i[p]$ or to $A_{i'}[p']$, this is obvious;
- if x and y separate $A_i[p]$ from $A_{i'}[p']$, use transitivity;
- if $A_i[p]$ and $A_{i'}[p']$ are on the same side of x and y, then $A_i[p] < A_{i'}[p'] \Leftrightarrow i < i'$. □

Henceforth, we restrict ourselves to algorithms which only do queries as described in Lemma 2.1. We fix an arbitrary deterministic algorithm A. Let A' denote an algorithm which continues performing queries after A ends and until the insertion ranks p_i of x are determined in all the sets.

For a random instance from D_-, we define the following random variables:

- X_i: the number of queries performed by A in A_i.
- Y_i: the number of queries performed by A' in A_i.
- ξ_i: the indicator variable which equals 1 if and only if the insertion rank p_i of x in A_i has been determined by A.

The number of comparisons performed by A is $C = \sum_i X_i$. Restricting ourselves to sets in which the insertion rank has been determined, we can write $C \geq \sum_i X_i \xi_i = \sum_i Y_i \xi_i$.

$$\begin{aligned} \Pr\{Y_i\xi_i \geq a_i\} &= \Pr\{Y_i \geq a_i \text{ and } \xi_i = 1\} \\ &= 1 - \Pr\{Y_i < a_i \text{ or } \xi_i = 0\} \\ &\geq 1 - \Pr\{Y_i < a_i\} - \Pr\{\xi_i = 0\} \\ &= \Pr\{\xi_i = 1\} - \Pr\{Y_i < a_i\} \end{aligned} \tag{2.1}$$

LEMMA 2.2.

$$\Pr\{Y_i < a_i\} \leq 4^{a_i}/n_i.$$

Proof. This is the usual decision tree lower bound: if we consider the queries performed by A' in set A_i, there are at most 4^{a_i} leaves at depth less than a_i, and since the insertion rank of x in A_i is uniform, these leaves all have the same probability and have total probability at most $4^{a_i}/n_i$. □

Then $E(C) \geq \sum_i E(Y_i\xi_i)$

$$\begin{aligned} &\geq \sum_i (\log_4 n_i - 1) \Pr\{Y_i\xi_i \geq (\log_4 n_i - 1)\} \\ &\geq \sum_i \left(\log_4 n_i \left(\Pr\{\xi_i = 1\} - \frac{1}{4}\right) - 1\right) \\ &= \sum_i \log_4 n_i \Pr\{\xi_i = 1\} - \frac{1}{4}\sum_i \log_4 n_i - k, \end{aligned} \tag{2.2}$$

where the first inequality comes from the linearity of expectation, the second from Markov's inequality applied to $Y_i\xi_i$, and the third one come from Equation (2.1) and Lemma 2.2 with $a_i = (\log_4 n_i) - 1$.

LEMMA 2.3. *Given algorithm A, the probability $\Pr\{i_0 = j\} = \alpha_j$ and the positions $p = (p_1, p_2, \ldots, p_k)$, there exists a permutation σ of $\{1, 2, \ldots, k\}$ such that*

$$\Pr\{\xi_i = 1 | p\} = \sum_{j:\sigma_j \geq \sigma_i} \alpha_j.$$

Proof. Given $p = (p_1, p_2, \ldots, p_k)$ there are only k instances $I_1, I_2, \ldots, I_k$ corresponding to the k possible choices for i_0. Algorithm A can only differentiate between two such instances when it discovers that $A_i[p_i]$ is y for one instance and x for the other instance. The algorithm is finished as soon as it finds y, thus the restriction of A to those instances gives the decision tree in form of a branch (see Figure 1, where the decision tree is drawn from left to right). Let σ denote the order in which these instances are dealt with by A. Then $\xi_i = 1$ if and only if $\sigma_i \leq \sigma_{i_0}$, and so

$$\Pr\{\xi_i = 1 | p\} = \sum_j \alpha_j \mathbb{I}(\sigma_i \leq \sigma_j) = \sum_{j:\sigma_j \geq \sigma_i} \alpha_j \quad .\square$$

From Lemma 2.3 and the definition of α_j (recall that $\alpha_j = \log_4 n_j / \sum_l \log_4 n_l$), we obtain

$$\begin{aligned} &\log_4 n_i \Pr\{\xi_i = 1\} \\ &\quad = \sum_p \log_4 n_i \Pr\{\xi_i = 1 | p\} \Pr\{p\} \\ &\quad = \sum_p \sum_{j:\sigma_j \geq \sigma_i} \log_4 n_i \log_4 n_j \frac{\Pr\{p\}}{\sum_l \log_4 n_l}. \end{aligned}$$

Summing over i to calculate the first term in Equation (2.2), we get

$$\begin{aligned} &\sum_i \log_4 n_i \Pr\{\xi_i = 1\} \\ &\quad = \sum_p \sum_{i,j:\sigma_j \geq \sigma_i} \log_4 n_j \log_4 n_i \frac{\Pr\{p\}}{\sum_l \log_4 n_l} \\ &\quad = \sum_p \Pr\{p\} \sum_{i,j:\sigma_j \geq \sigma_i} \frac{\log_4 n_j \log_4 n_i}{\sum_l \log_4 n_l}. \end{aligned}$$

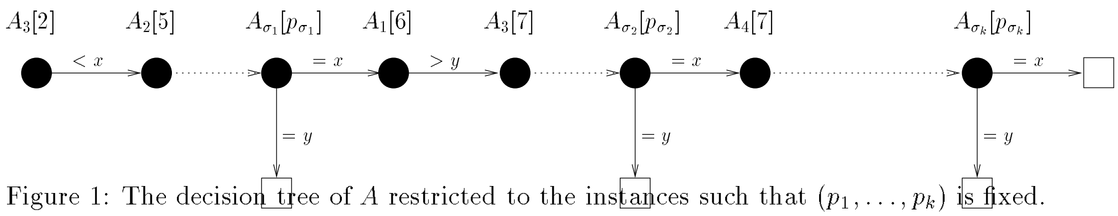

Figure 1: The decision tree of A restricted to the instances such that $(p_1, \ldots, p_k)$ is fixed.

In the sum, each term $\log_4 n_j \log_4 n_i$ appears exactly once, hence

$$\sum_{i,j:\sigma_j \geq \sigma_i} \log_4 n_j \log_4 n_i = \frac{1}{2}((\sum_i \log_4 n_i)^2 + \sum_i \log_4^2 n_i),$$

which is independent of p. Then we can conclude:

$$\begin{aligned} \sum_i \log_4 n_i \Pr\{\xi_i = 1\} &\geq \frac{1}{2} \sum_p \frac{(\sum_i \log_4 n_i)^2 \Pr\{p\}}{\sum_l \log_4 n_l} \\ &\geq \frac{1}{2} \sum_i \log_4 n_i. \end{aligned}$$

Plugging this into Equation (2.2), we obtain the bound of Theorem 2.2. □

Remark: We stated Theorem 2.1 and 2.2 for real numbers. However, since we work in the comparison model, instances over the reals could easily be replaced by integer instances over a range of size $\sum_i n_i$ at most.

3 The Intersection Problem

3.1 Definition and basic properties

DEFINITION 3.1. *Given k sorted sets $A_1, A_2, \ldots, A_k$ of sizes $n_1, \ldots, n_k$ summing to n, the* Intersection Problem *consists in computing the intersection $I = A_1 \cap A_2 \cap \ldots \cap A_k$.*

A non-deterministic algorithm would only need to exhibit a certificate for the intersection. This is similar to the notion of "proof" used in [2].

DEFINITION 3.2. *A* certificate $(\mathcal{I}, P)$ *consists of:*

1. *a set $\mathcal{I}$ of k-tuples specifying the ranks of elements of I, and*
2. *a set P of inequalities of the form $A_i[p] < A_{i'}[p']$, such that if all are satisfied then the intersection is restricted to the elements given in $\mathcal{I}$.*

For instance, the k-tuple $(p_1, \ldots, p_k)$ means that $A_1[p_1] = A_2[p_2] = \ldots = A_k[p_k]$ belongs to I. And the inequality $A_1[1] > A_2[5]$ implies that no element of rank strictly less than 6 in set A_2 is in the intersection. One instance can have many different certificates, with very different sizes. For example, if $A = \{0\}$, $B = \{1, 3, 5, 7\}$ and $C = \{2, 4, 6, 8\}$, the intersection $\mathcal{I}$ is empty, and a possible set of inequalities certifying it might consist simply of one inequality $A[1] < B[1]$, or it might be much longer, for example the six inequalities $B[1] < C[1] < B[2] < C[2] < B[3] < C[4] < B[4]$.

DEFINITION 3.3. *The* difficulty δ *of an instance of the Intersection Problem is the minimum value of $\#\mathcal{I} + \#P$ over all certificates $(\mathcal{I}, P)$ for the instance.*

This is similar to the notion of disorder in [4], and similar to but distinct from the notion of difficulty in [2]. Our measure is essentially the non-deterministic complexity of the instance, i.e. proving that the intersection is I requires $\#P + (k-1)\#\mathcal{I}$ comparisons: $(k-1)\#\mathcal{I}$ equalities to establish that the corresponding elements belong to all the sets, and $\#P$ inequalities to establish that there are no other elements in the intersection.

One question immediately comes to mind: why did we not pick simply the non-deterministic complexity, $\#P + (k-1)\#I$, as our measure of difficulty? In fact, the two parts of the certificate play a very different role, and the core of the theorem is when the intersection is empty – one lower bound, $\Omega(\#I \sum_i \log(n_i/\#I))$, is relatively easy, the other one, $\Omega(\#P \sum_i \log(n_i/\#P))$, which is relevant when the intersection is small or empty, is more difficult. Our definition of difficulty simplifies notations to combine both lower bounds into one.

We will prove a structural result, Theorem 3.1 below, establishing a correspondence between intersection certificates and certain partitions of the real line into intervals. For each certificate we can partition $(-\infty, +\infty)$ into intervals of elements not in the intersection, and singletons consisting of elements of the intersection. That theorem inspired us in the design of the instances used in the proof of the lower bound, and is also used in the analysis of the upper bound.

THEOREM 3.1. *Consider an instance $(A_1, A_2, \ldots, A_k)$ of the Intersection Problem with intersection I. There exists a certificate $(\mathcal{I}, P)$ with $\#\mathcal{I} + \#P \leq d$, if and only if there exists a set of at most $d+1$ intervals $(I_j)_{0 \leq j \leq d}$ such that:*

- *$\bigcup_j I_j \cup I$ is a partition of $(-\infty, +\infty)$, and*
- *for every I_j, there exists a set A_{i_j} such that $A_{i_j} \cap I_j = \emptyset$.*

Proof. We first prove the "only if" direction. The proof is by induction on d. If one of the sets, say A_{i_0}, is empty, then the lemma holds with the partition $I_0 = (-\infty, +\infty)$. This settles the base case $d = 0$. Let $d > 0$ and assume that none of the sets A_i are empty. Let $a = \max\{A_1[1], A_2[1], \ldots, A_k[1]\}$.

Consider the case where $a = A_s[1] \in I$. A new proof $\tilde{P}$ for our instance can be defined by removing any comparison between an element $\leq a$ and an element $\geq a$. We define an interval $I_d = (-\infty, a)$, with an associated set $A_{i_d} = A_s$; we now create a new instance $(A'_1, A'_2, \ldots, A'_k)$, such that for every i, $A'_i = \{x \in A_i | x > a\}$.

Let $\mathcal{I}'$ be the "transcription" of $\mathcal{I} \setminus \{a\}$ on this new instance $(A'_1, A'_2, \ldots, A'_k)$, with the element ranks updated appropriately. Let P' be the "transcription" of $\tilde{P}$ on the instance $(A'_1, A'_2, \ldots, A'_k)$, where the element ranks are updated appropriately, and comparisons involving elements $\leq a$ are removed. It is easy to check that $(\mathcal{I}', P')$ is a certificate for $(A'_1, A'_2, \ldots, A'_k)$, of size at most $d-1$ since $\#\mathcal{I}' = \#\mathcal{I} - 1$. Apply the induction hypothesis to $(A'_1, \ldots, A'_k)$ to define $(I'_j)_{0 \leq j \leq d-1}$; let $I_j = I'_j \setminus (-\infty, a]$ for $j \leq d-1$. Adjoining I_d gives a partition of $(-\infty, +\infty)$ which satisfies the theorem for $(A_1, \ldots, A_k)$.

Now, consider the case where $a = A_s[1] \notin I$. Consider one particular instance of the problem. Of all the inequalities in P, take the inequality $A_u[p] < A_v[q]$ such that the element $A_v[q]$ has smallest value. A new proof $\tilde{P}$ for our instance can be defined by replacing all inequalities involving $A_u[x]$, for every $x \leq p$, by "$A_u[x] < A_s[1]$" (note that $A_v(q) \leq A_s[1]$, otherwise P would not rule out the possibility that $A_1[1] = A_2[1] = \ldots = A_k[1]$ belongs to I). We define an interval $I_d = (-\infty, a)$, and an associated set $A_{i_d} = A_s$; we now create a new instance: $A'_i = A_i$ for every $i \neq u$ and $A'_u = A_u[p+1, \ldots n_u]$.

Let $\mathcal{I}'$ be the "transcription" of $\mathcal{I}$ on this new instance $(A'_1, A'_2, \ldots, A'_k)$, with the element ranks updated appropriately. Let P' be the "transcription" of $\tilde{P}$ on the instance $(A'_1, A'_2, \ldots, A'_k)$, where the inequalities involving an element which has been removed disappear, and the element ranks are updated appropriately. It is easy to check that $(\mathcal{I}', P')$ is a certificate for $(A'_1, A'_2, \ldots, A'_k)$, of size at most $d-1$ since $\#P' \leq \#P - 1$. Apply the induction hypothesis to $(A'_1, \ldots, A'_k)$ to define $(I'_j)_{0 \leq j \leq d-1}$; let $I_j = I'_j \setminus (-\infty, a)$ for $j \leq d-1$. Adjoining I_d gives a partition of $(-\infty, +\infty)$ which satisfies the theorem for $(A_1, \ldots, A_k)$.

We now prove the other direction. Consider a set of intervals $(I_j)_{0 \leq j \leq d}$ satisfying the conditions of the theorem. Let $a = \sup I_j = \inf I_{j+1}$. If $a \in I$, just add the corresponding k-tuple to $\mathcal{I}$. If $a \notin I$, let $A_{i_j}[r]$ be the smallest element of A_{i_j} which is $\geq a$, and $A_{i_{j+1}}[l]$ be the largest element of $A_{i_{j+1}}$ which is $\leq a$ (note that these elements cannot both be equal to a). Add the comparison "$A_{i_{j+1}}[l] < A_{i_j}[l]$" to P. This defines $(\mathcal{I}, P)$, which has size d, and one can easily convince oneself that it is a certificate. $\square$

3.2 A lower bound for Intersection Problem

THEOREM 3.2. *For any positive integers $k, n_1, \ldots, n_k$, and for any integers i_0 and p_0 such that $i_0 + p_0 = \delta$, there exists a distribution D over instances of difficulty $\leq \delta$ on k sets of sizes $n_1, \ldots, n_k$ respectively, with an intersection of size i_0, such that any deterministic algorithm for the Intersection Problem performs on average $\Omega(\delta \sum_i \log(n_i/\delta))$ comparisons.*

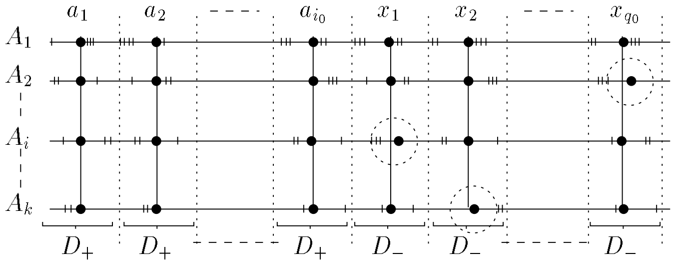

Figure 2: An instance $(A_1, \ldots, A_k)$ of the intersection problem, obtained by drawing from i_0 distributions of type D_+ and q_0 distributions of type D_-.

Proof. Let $q_0 = \lfloor p_0/2 \rfloor$. Let $A_0 = \{a_1, a_2, \ldots, a_{i_0}, x_1, \ldots, x_{q_0}\}$ and choose (z_i) and (u_j) such that $a_1 < z_1 < a_2 < \ldots z_{i_0-1} < a_{i_0} < z_{i_0}$, and $z_{i_0} < x_1 < u_1 < \ldots < x_{q_0-1} < u_{q_0-1} < x_{q_0}$. We define m_i for all $i = 1, \ldots, k$ as equal to $\lfloor n_i/(i_0 + q_0) \rfloor$.

Construct i_0 instances of the Intersection Element Problem for k sets of sizes $m_1, \ldots, m_k$, using the distribution $D_+(l, x, r)$ defined in the proof of Theorem 2.1: more precisely, use distribution $D_+(-\infty, a_1, z_1)$ to create sets $A_1^{(1)}, \ldots, A_k^{(1)}$ which all contain a_1; distribution $D_+(z_1, a_2, z_2)$ to create sets $A_1^{(2)}, \ldots, A_k^{(2)}$, and so on till distribution $D_+(z_{i_0-1}, a_{i_0}, z_{i_0})$ to create sets $A_1^{(i_0)}, \ldots, A_k^{(i_0)}$.

Construct q_0 instances of the Intersection Element Problem for $k-1$ sets of size $m_1, \ldots, m_k$, using the distribution $D_-(l, x, r)$ defined in the proof of Theorem 2.2: more precisely, use distribution $D_-(z_{i_0}, x_1, u_1)$ to create sets $A_1^{(i_0+1)}, \ldots, A_k^{(i_0+1)}$ which almost all contain x_1, distribution $D_-(u_1, x_2, u_2)$ to create sets $A_1^{(i_0+2)}, \ldots, A_k^{(i_0+2)}$, and so on till distribution $D_-(z_{q_0-1}, x_{q_0}, \infty)$ to create sets $A_1^{(i_0+q_0)}, \ldots, A_k^{(i_0+q_0)}$.

Let $A_j = A_j^{(1)} \cup \ldots \cup A_j^{(i_0+q_0)}$. Complete each set if necessary so that the size of each set A_i equals n_i. This creates a random instance $A_1, A_2, \ldots, A_k$ whose intersection is $I = \{a_1, a_2, \ldots, a_{i_0}\}$.

LEMMA 3.1. *The instance $(A_1, \ldots, A_k)$ thus constructed has difficulty at most δ.*

Proof. We use Theorem 3.1. The intervals $(-\infty, a_1)$,(a_i, a_{i+1}), (a_{i_0}, x_1) (x_j, x_{j+1}) and (x_{q_0}, ∞) all have an empty intersection with A_1. For each x_j, there is a set A_{i_j} which does not contain x_j, hence a small open interval around x_j whose intersection with A_{i_j} is empty. This defines $i_0 + q_0 + 1 + q_0 \le \delta + 1$ intervals which satisfy the two conditions of Lemma 3.1, hence the instance has difficulty at most δ. □

Now, let A be an algorithm for the Intersection Problem. Even if a "little birdy" tells A what the distribution is, and that $I \subset A_0$, A still has to solve i_0 positive instances and q_0 negative instances of the Intersection Element Problem to verify that the intersection is exactly $\{a_1, \ldots, a_{i_0}\}$. So by Theorems 2.1 and 2.2, A must perform on average at least $\Omega((i_0 + q_0)\sum_i \log m_i)$ comparisons, hence the proof. □

Applying the Yao-von Neumann principle [7, 6], we finally obtain the following corollary.

COROLLARY 3.1. *Consider instances of difficulty $\le \delta$; the complexity of any randomized algorithm for the Intersection Problem is $\Omega(\delta \sum_i \log(n_i/\delta))$.*

Remark that the proof constructs instances whose difficulty is either $\delta - 1$ or δ, and can in fact easily be modified to construct instances of difficulty exactly δ, so that the Corollary in fact also holds for instances of difficulty $= \delta$.

3.3 An upper bound for Intersection Problem

We use the deterministic algorithm from [2], stated in a slightly simplified form in Algorithm 2. This algorithm relies on doubling search [1], which searches for an element in a set left-to-right, doubling the size of the interval covered at each step (see Algorithm 1). Algorithm 2 finds the elements of the intersection in increasing order, that is, at any time, the element m currently under consideration is a candidate element of the intersection, and at that time the algorithm has already found all the elements of the intersection which are less than m. The value of m is updated when either m is found to be in all the sets (hence belongs to the intersection), or m is found not to be in some set A_i (hence is not in the intersection). The sets which are marked are all the sets in which m has been found so far.

Algorithm 1 Doubling search for x in set A_i

```
Let step_i ← 1
Let position_i ← 1
while position_i ≤ #A_i and A_i[position_i] < x do
  step_i ← 2 × step_i
  position_i ← position_i + step_i
end while
output: "x has insertion rank in (position_i − step_i, position_i]"
```

THEOREM 3.3. *Algorithm 2 computes the intersection of any instance of difficulty δ in $O(k\delta \log(n/\delta))$ comparisons.*

Algorithm 2 Algorithm to compute the intersection $I = A_1 \cap A_2 \cap \ldots \cap A_k$

```
Let M ← min{A_1[n_1], A_2[n_2], ..., A_k[n_k]}
Let m ← max{A_1[1], A_2[1], ..., A_k[1]}
Mark the set to which m belongs
Let ∀i step_i ← 1
while m ≤ M do
  Let A_i be the next non marked set in round robin order
  Perform one step of doubling search for m in A_i, comparing m to some a ∈ A_i.
  if a ≥ m then
    Perform a binary search for m in the relevant interval of A_i.
    Reinitialize step_i ← 1
    if m ∈ A_i then
      Mark A_i
      if all sets are marked then
        I ← I ∪ {m}
      end if
    end if
    if m ∈ I or m ∉ A_i then
      Remove all marks
      Let m ← first element of A_i larger than m
      Mark A_i
    end if
  end if
end while
```

The algorithm was first given by [2], and Theorem 3.3 can be obtained from Corollary 5.1 in [2] (by considering n as an upper bound for each $g_s[i]$). Here we give a direct proof.

This upper bound matches the lower bound of Theorem 3.2 when $\max_i \log n_i = O(\min_i \log n_i)$ and $k = o(n)$, so the algorithm is optimal in this case. This result is then the best possible in the sense that the upper bound is reached by a deterministic algorithm, whereas the lower bound holds even for randomized algorithms.

The upper bound could be greater than the lower bound by a factor of $O(k)$, for instance if $n_1 = \Omega(2^{n_2})$ where $n_1 > n_2 \geq \ldots \geq n_k$. These are extremal conditions, and we think the lower bound is not tight for those instances.

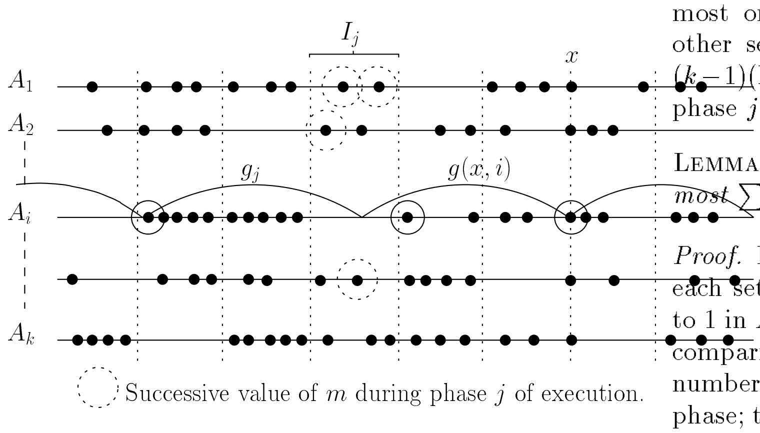

Figure 3: Analysis of the doubling comparisons of Algorithm 2.

Proof. Let us call the comparisons performed by the algorithm *doubling comparisons* or *binary comparisons*, depending on whether they are performed during a doubling search or during a binary search. We focus on doubling comparisons. Consider the intervals $(I_j)_{j \leq \delta}$ obtained by applying Theorem 3.1 to an optimal proof. Each set A_i can be partitioned by using as separators:

- the intersection elements (if any), and
- the intervals I_j associated with A_i (if any, see Theorem 3.1; for each j there is an A_i such that $A_i \cap I_j = \emptyset$).

These separators divide A_i into parts, with g_j the size of the part ending in I_j, and $g(i,x)$ the size of the part ending at $x \in I$ (see Figure 3).

During an execution of Algorithm 2, the successive values of m form a non-decreasing sequence. Say that a comparison is performed in phase j if it is done while m has value in I_j, and in phase x if it is done while m has value $x \in I$.

Lemma 3.2. *During phase j let A_i be the set associated to I_j ($A_i = A_{i_j}$ in the notations of Theorem 3.1). The algorithm performs at most $\log_2 g_j$ doubling comparisons in set A_i.*

Proof. During phase j, m has values in I_j, so we never have $m \in A_i$, hence A_i is never marked. As the doubling search step won't be reset to 1 in A_i, the number of doubling comparisons in set A_i is less than the logarithm of the total number of elements of A_i which are scanned during that period; that number is less than g_j, hence the lemma. □

Doubling search in the $k-1$ sets is performed in round robin order, so for each doubling comparison in A_j at most one doubling comparison is performed in each other set A_l, and so the algorithm performs at most $(k-1)(\log_2 g_j + 1)$ doubling comparisons in total during phase j.

Lemma 3.3. *During phase x, the algorithm performs at most $\sum_i \log_2 g(i,x)$ doubling comparisons in all sets.*

Proof. During phase x, m is fixed and equal to x. For each set A_i, as the doubling search step won't be reset to 1 in A_i until we find $m \in A_i$, the number of doubling comparisons in A_i is less than the logarithm of the total number of elements of A_i which are scanned during that phase; that number is less than $g(i,x)$, hence the lemma. □

Using concavity of logarithms in the sum over all phases j, we find that the number of doubling comparisons in those phases is at most $\delta(k-1)(\log_2(\sum_j g_j/\delta)+1)$. Similarly, using concavity of logarithms in the sum over all phases x and sets A_i, we find that the number of doubling comparisons in those phases is at most $k\#I \log_2(\sum_{x,i} g(i,x)/k\#I)$. Both these numbers are in $O(k\delta \log_2(n/\delta))$, so the total number of doubling comparisons performed by the algorithm is in $O(k\delta \log_2(n/\delta))$.

From this upper bound on the total number of doubling comparisons we can deduce an upper bound on the total number of binary comparisons: in any set, a binary search is preceded by a doubling search in the interval defined by the doubling search. As the binary search uses at most as many comparisons as the doubling search which preceded it, binary comparisons can be charged to doubling comparisons, hence the theorem. □

4 The t-Threshold Problem

Definition 4.1. *Given k ordered sets $A_1, A_2, \ldots, A_k$ of sizes $n_1, \ldots, n_k$ summing to n, the $t-$*Threshold Problem *consists in computing the set of elements which are present in at least t of the k sets:*

$$T_t = \{x : \#\{i \leq k, x \in A_i\} \geq t\}.$$

The k-Threshold Problem is the Intersection Problem. Thus this section can be seen as an extension of the work done on the Intersection Problem, and for $t = k$ Theorem 4.2 yields the bound of Theorem 3.2.

Observe that for the t-Threshold Problem when $t \geq 2$, a certificate for the result must access each element of the output set at least once, hence the size of the output is an obvious lower bound to the complexity of any algorithm. On the other hand, the 1-Threshold Problem is the Union Problem (compute the union of all the sets), which has one significant difference from the other t-Threshold Problems: one might be able to compute the union implicitly without accessing all of its elements: for example, if $k = 2$ and if the algorithms checks that $A_1[\#A_1] < A_2[1]$, then it knows that all the elements are distinct and has an implicit knowledge of the union (in that sense, the Union Problem can be reduced to the 2-Threshold Problem). This was exploited in [2], where an optimal adaptive algorithm was designed for the Union Problem. Henceforth, we will assume that $t \geq 2$.

DEFINITION 4.2. *A certificate $(\mathcal{T}_t, P)$ for the t-Threshold Problem consists of*

1. *a set $\mathcal{T}_t$ of t-tuples, where the jth entry of the t-tuple, for $1 \leq j \leq t$, is a pair specifying a set i_j and a rank in that set, and*
2. *a set P of inequalities of the form "$A_i[p] < A_{i'}[p']$" such that if all are satisfied then the tuples given in $\mathcal{T}_t$ all correspond to distinct elements, and the t-threshold set is restricted to the elements given in $\mathcal{T}_t$.*

Thus, the t-tuple $((i_1, p_1), (i_2, p_2), \ldots, (i_t, p_t))$ means that $A_{i_1}[p_1] = A_{i_2}[p_2] = \ldots = A_{i_t}[p_t]$ and so, that element belongs to T_t. And the inequalities $A_1[1] > A_3[5]$ and $A_2[1] > A_3[8]$, if $t = k - 2$, imply that no element of A_3 of rank less than 6 is in the t-threshold set.

DEFINITION 4.3. *The* difficulty δ *of an instance of the $t-$Threshold Problem is the minimum value of $\#\mathcal{T}_t + \#P$ over all certificates $(\mathcal{T}_t, P)$ for the instance.*

We now present a structural result similar to Theorem 3.1.

THEOREM 4.1. *Consider an instance $(A_1, A_2, \ldots, A_k)$ of the t-Threshold Problem with result T_t. For each certificate $(\mathcal{T}_t, P)$ with $\#\mathcal{T}_t + \#P \leq d$ there exist a set of at most $d+1$ intervals $(I_j)_{0 \leq j \leq d}$ such that:*

- *$\bigcup_j (I_j) \cup T_t$ is a partition of $(-\infty, +\infty)$, and*
- *For every I_j, there exist $k - t + 1$ sets $A_{i_j^q}$, with $1 \leq q \leq k - t + 1$, such that $A_{i_j^q} \cap I_j = \emptyset$.*

Proof. The proof is by induction on d. If $k-t+1$ of the sets are empty, then the Lemma holds with the partition $I_0 = (-\infty, +\infty)$. This settles the base case $d = 0$. Let $d > 0$ and assume that $e \leq k - t$ of the sets are empty. Among all comparisons, take the one whose right hand side is smallest. Let x be that right hand side. Let y be the smallest element of T_t.

- If $x < y$ then $k-t+1$ sets A_s must have $x < A_s[1]$ (to rule out the possibility that $A_{i_1}[1] = \ldots = A_{i_t}[1] \in T_t$); define $I_d = (-\infty, x)$ and use induction on $A'_i = A_i \setminus I_d$.
- If $y < x$ then $k-t+1$ sets A_s must have $y \leq A_s[1]$ (to rule out the possibility that $A_{i_1}[1] = \ldots = A_{i_t}[1] \in T_t$); define $I_d = (-\infty, y)$ and use induction on $A'_i = A_i \setminus I_d$. □

4.1 A lower bound for t-Threshold Problem

THEOREM 4.2. *For any positive integers $k, n_1, \ldots, n_k$, and for any integers i_0 and p_0 such that $i_0 + p_0 = \delta$, there exists a distribution D over instances of difficulty $\leq \delta$ on k sets and $n_1 \geq n_2 \geq \ldots \geq n_k$ elements with a t-threshold set of size i_0, such that any deterministic algorithm for the t-Threshold Problem performs on average $\Omega(\delta \sum_{i=1}^{t} \log(n_i/\delta))$ comparisons.*

Proof. We can limit ourselves to instances were $k = t$ by giving to the algorithm the indices of the $k - t$ smallest sets with the promise that they do not contain any element of the t-threshold set. Then the lower bound of Theorem 3.2 applies. □

Applying the Yao-von Neumann principle again, we finally obtain the following corollary.

COROLLARY 4.1. *Consider instances of difficulty $\leq \delta$; the complexity of any randomized algorithm for the t-Threshold Problem is $\Omega(\delta \sum_{i=1}^{t} \log(n_i/\delta))$.*

4.2 An upper bound for t-Threshold Problem

Algorithm 3 finds the elements of T_t in increasing order. At any time, the element m currently under consideration is a candidate element of T_t; at that time the algorithm has already found all the elements of T_t which are less than m.

The sets which are colored green are the ones in which m has been found so far; the sets which are colored in red are the ones to which m is known not to belong (By convention, if A_i is empty then A_i is colored red and we can assume that $+\infty$ is the corresponding element in H). An element $A_i[p]$ is in H if and only if A_i is red, and $A_i[p]$ is the smallest element of A_i which is strictly larger than m (it is the smallest element of A_i which could conceivably end up in T_t). H is implemented by a heap.

The value of m is updated when m is either found to be in t sets (hence belongs to T_t), or found not to be in $k-t+1$ sets (hence does not belong to T_t). At that time, we must choose a new value for m. The algorithm chooses this new value in such a way that at any time, the number of white sets is at most $t-1$. This guarantees that no element of T_t will be overlooked, hence the correctness of the algorithm.

Specialized to the case $t = k$, the green sets correspond to the marked sets in the Intersection algorithm; all the other sets are white, except when the algorithm finds a set A_i which does not contain m, in which case all green sets are recolored white and m is immediately updated and replaced by the next larger element of A_i. For $t = k$, Algorithm 3 is thus exactly Algorithm 2.

THEOREM 4.3. *Algorithm 3 computes the t-threshold set of any instance of difficulty δ in $O(t\delta \log k \log n)$ comparisons.*

Proof. Let us call the comparisons performed by the algorithm

- *doubling comparisons* if they are performed during a doubling search,
- *binary comparisons* if they are performed during a binary search, and
- *heap comparisons* if they are performed during the manipulation of H.

Consider the intervals $(I_j)_{j\leq\delta}$ obtained by applying Theorem 4.1 to an optimal proof. During an execution of Algorithm 3, the successive values of m form a non-decreasing sequence. Say that a comparison is performed in phase j it it is done while m has value in I_j, and in phase x if it is done while m has value $x \in T_t$

LEMMA 4.1. *During phase j let A_i be one of the set associated to I_j ($A_i = A_{i_j^q}$ in the notations of Theorem 4.1). The algorithm performs at most $\log_2 n$ doubling comparisons in set A_i.*

Proof. During phase j, the variable m has values which belong to I_j. But $I_j \cap A_i = \emptyset$, so during that time we never have $m \in A_i$ and A_i is never colored green; if m is ever less than the element of A_i to which it is compared, that initiates a binary search after which A_i will be colored red and never examined again during phase j, and so the number of doubling comparisons in set A_i is less than $\log_2 n$. □

Let A_i be the last set to become red during phase j, among the $k-t+1$ sets associated to I_j. When A_i becomes red, phase j immediately terminates. While A_i is white, since doubling search is performed in round robin order among the white sets, and as there are at most $(t-1)$ white sets at any time, the algorithm performs at most $(t-1)$ doubling comparisons in other sets for each doubling comparison in A_i. So, in total, the algorithm performs at most $t \log_2 n$ doubling comparisons during phase j.

In any set, a doubling search is followed by a binary search in the interval defined by the doubling search. The binary search uses at most as many comparisons as the doubling search which preceded it, so binary comparisons can be charged to doubling comparisons: the algorithm performs at most $2(t-1)\log_2 n$ comparisons in total during phase j. Since there are exactly δ such phases, the overall number of comparisons during phases j is at most $2\delta(t-1)\log_2 n$. The analysis for phases x is similar.

It remains to analyze the heap comparisons, between elements of H: first the heap H is constructed with cost at most $k \log k$ comparisons. Then in the while loop there are at most as many elements inserted as there are iterations, so there are at most $\delta t \log_2 n$ insertions in total. As there are never more than $k-t+1$ red sets, H never contains more elements, and each insertion cost at most $\log_2(k-t+1)$ comparisons. As there cannot be more removal than insertions in the heap, the number of heap comparisons is no more than $2t\delta \log(k-t+1) \log n$, and adding this to the other comparisons gives a bound of at most $2t\delta(\log(k-t+1)+1)\log n$. □

5 Perspectives

Some obvious open problems include closing the gaps between our lower and upper bounds. Many other problems are also worth considering in this framework.

In the context of database queries, a natural variant of the t-Threshold Problem is the following: Given a set of k ordered sets, find the maximal value t_{max} for t such that the t-Threshold Set is non empty, and return this Threshold Set.

So far, in the applications each set A_i was considered to be the set of database elements which contain the i^{th} word in the k-word query. However, more sophisticated algorithms, given a query word w, design a short list of all words which are variants of w, then find a set $A_{\tilde{w}}$ for each variant $\tilde{w}$. Instead of finding the intersection of all the sets, one is then faced with a new problem, finding the intersection of unions of sets. Of course if one has preprocessed, for each word, the union of the sets corresponding to its variants, then queries can be answered just by performing an intersection. But precomputation has a cost in memory, and in some applications, precomputing the unions cost too much to be practical.

More generally, given k sorted sets and an element x, one can construct a k-bit word such that $x_i = 1$ if and only if $x \in A_i$. Take a boolean function $f : \{0,1\}^k \to 1$, and consider the problem of finding all the elements (or deciding whether there exist any element, in the decision version of the problem) such that $f(x) = 1$. When $f(x_1, \ldots, x_k) = x_1 \vee \ldots \vee x_k$, we get the Union Problem. When $f(x_1, \ldots, x_k) = x_1 \wedge \ldots \wedge x_k$, we get the Intersection Problem. When $f(x_1, \ldots, x_k) = (x_1 + \ldots + x_k \geq t)$, we get the t-Threshold Problem. Can anything be said about the problem for general f?

Acknowledgments: We would like to thank Julien Sebot for introduction to the *Fleming File Sharing System* project, which inspired this work. Also we thank Sophie Laplante for interesting discussions and corrections.

References

[1] John L. Bentley, and Andrew C. Yao, An almost Optimal Algorithm for Unbounded Searching. In Information Processing Letters 5 1976, 82-87.

[2] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro, Adaptive set intersections, unions, and differences. In Proceedings of the *11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2000, 743-752.

[3] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro, Experiments on Adaptive Set Intersections for Text Retrieval Systems. In Proceedings of the *3rd Workshop on Algorithm Engineering and Experiments*, Lecture Notes in Computer Science, 2001, 91-104.

[4] Vladimir Estivill-Castro, and Derick Wood, A Survey of Adaptive Sorting Algorithms. *ACM Computing Surveys*, 1992, 24(4):441–476.

[5] Donald E. Knuth, The Art of Computer Programming, Vol 3, Sorting and Searching, Section 5.3. *Addison-Wesley*, 1973.

[6] John von Neumann, and Oskar Morgenstern, Theory of games and economic behavior, *1st ed. Princeton University Press*, 1944.

[7] Andrew C. Yao, Probabilistic computations: Toward a unified measure of complexity, *In Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS)*, 1977, 222-227.

Algorithm 3 Algorithm for the t-Threshold Problem

Let $M \leftarrow t^{th}$ largest element of the multi-set $\{A_1[n_1], A_2[n_2], \ldots, A_k[n_k]\}$
Let $H = \{A_1[1], A_2[1], \ldots, A_k[1]\}$ (counted with multiplicity).
Let $m \leftarrow t^{th}$ element of H.
Color green all the sets A_i such that $A_i[1] = m$, and remove all copies of m from H.
Color red all the sets A_i such that $A_i[1] > m$.
Color white all the sets A_i such that $A_i[1] < m$, and remove $A_i[1]$ from H.
while $m \leq M$ **do**
  if t sets are green or $k - t + 1$ sets are red **then**
    if t sets are green **then**
      $T_t \leftarrow T_t \cup \{m\}$
    end if
    Take $t - 1 - \#$(white sets) of the green sets and color them white.
    For each remaining green set A_i, insert in H the first element of A_i which is $> m$, and change the color of A_i to red.
    Let $m \leftarrow \min H$; change the color to green for all the sets which have m as a representative in H, and remove m from H.
  end if
  Let A_i be the next white set in round robin order.
  Perform one step of doubling search for m in A_i, comparing m to some $a \in A_i$.
  if $a \geq m$ **then**
    Perform a binary search for m in the relevant interval of A_i.
    Reinitialize doubling search step to 1 in A_i.
    if $m \in A_i$ **then**
      Color A_i in green
    else
      Insert in H the first element of A_i which is strictly larger than m, and color A_i in red
    end if
  end if
end while