# Optimality of Randomized Algorithms for the Intersection Problem

Jérémy Barbay

Department of Computer Science, University of British Columbia,
201-2366 Main Mall, Vancouver, B.C. V6T 1Z4 CANADA
`jeremy@cs.ubc.ca`

**Abstract.** The "Intersection of sorted arrays" problem has applications in indexed search engines such as Google. Previous works propose and compare deterministic algorithms for this problem, and offer lower bounds on the randomized complexity in different models (cost model, alternation model).
We refine the alternation model into the *redundancy model* to prove that randomized algorithms perform better than deterministic ones on the intersection problem. We present a randomized and simplified version of a previous algorithm, optimal in this model.
**Keywords:** randomized algorithm, intersection of sorted arrays.

## 1 Introduction

We consider search engines where *queries* are composed of several keywords, each one being associated with a sorted array of references to entries in a database. The answer to a *conjunctive query* is the intersection of the sorted arrays corresponding to each keyword. Most search engines implement these queries. The algorithms are in the *comparison model*, where comparisons are the only operations permitted on references.

The intersection problem has been studied before [**?**,**?**,**?**], but the lower bounds apply to randomized algorithms, when some deterministic algorithms are optimal. Does it mean that no randomized algorithms can do better than a deterministic one on the intersection problem ?

In this paper we present a new analysis of the intersection problem, called the *redundancy* analysis, more precise and which permits to prove that for the intersection problem, randomized algorithms perform better than deterministic algorithms in term of the number of comparisons. The redundancy analysis also makes more natural assumptions on the instances: the worst case in the alternation analysis is such that an element considered by the algorithm is matched by almost all of the keywords, while in the redundancy analysis the maximum number of keywords matching such an element is parametrized by the measure of difficulty.

We define formally the intersection problem and the *redundancy* model in Section 2. We give in Section 3 a randomized algorithm inspired by the `small adaptive` algorithm, and give its complexity in the redundancy model, for which we prove it is optimal in Section 4. We answer the question of the utility of randomized algorithm for the intersection problem in Section 5: no deterministic algorithm is optimal in the redundancy model. We list in Section 6 several points on which we will try to extend this work.

## 2 Definitions

In the search engines we consider, queries are composed of several keywords, and each keyword is associated to a sorted array of references. The references can be, for instance, addresses of web pages, but the only important thing is that there is a *total order* on them, i.e. all unequal pair of references can be ordered. To study the problem of intersection, we hence consider any set of arrays on a totally ordered space to form an instance [**?**]. To perform any complexity analysis on such instances, we need to define a measure representing the size of the instance. We define for this the *signature* of an instance.

**Definition 1 (Instance and Signature).** *We consider $\mathcal{U}$ to be a totally ordered space. An* instance *is composed of $k$ sorted arrays $A_1, \ldots, A_k$ of positive sizes $n_1, \ldots, n_k$ and composed of elements from $\mathcal{U}$.*

*The* signature *of such an instance is $(k, n_1, \ldots, n_k)$. An instance is "of signature* at most*" $(k, n_1, \ldots, n_k)$ if it can be completed by adding arrays and elements to form an instance of signature exactly $(k, n_1, \ldots, n_k)$.*

*Example 1.* Consider the instance of Figure 1, where the ordered space is the set of positive integers: it has signature $(7, 1, 4, 4, 4, 4, 4, 4)$

**Definition 2 (Intersection).** *The* Intersection *of an instance is the set $A_1 \cap \ldots \cap A_k$ composed of the elements that are present in $k$ distinct arrays.*

*Example 2.* The intersection $A \cap B \cap \ldots \cap G$ of the instance of Figure 1 is empty as no element is present in more than 4 arrays.

| | | | | |
|---|---|---|---|---|
| $A =$ | **9** | | | |
| $B =$ | 1 | 2 | **9** | 11 |
| $C =$ | 3 | **9** | 12 | 13 |
| $D =$ | **9** | 14 | 15 | 16 |
| $E =$ | 4 | **10** | 17 | 18 |
| $F =$ | 5 | 6 | 7 | **10** |
| $G =$ | 8 | **10** | 19 | 20 |

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $A :$ | | | | | | | | | 9 | | | | | | | | | | | |
| $B :$ | 1 | 2 | | | | | | | 9 | | 11 | | | | | | | | | |
| $C :$ | | | 3 | | | | | | 9 | | | 12 | 13 | | | | | | | |
| $D :$ | | | | | | | | | 9 | | | | | 14 | 15 | 16 | | | | |
| $E :$ | | | | 4 | | | | | | 10 | | | | | | | 17 | 18 | | |
| $F :$ | | | | | 5 | 6 | 7 | | | 10 | | | | | | | | | | |
| $G :$ | | | | | | | | 8 | | 10 | | | | | | | | | 19 | 20 |

**Fig. 1.** An instance of the intersection problem: on the left is the array representation of the instance, on the right is a representation which expresses better the structure of the instance, where the abscissa of elements are equal to their value.

Any algorithm (even non-deterministic) computing the intersection must certify the correctness of the output: first, it must certify that all the elements of the output are indeed elements of the $k$ arrays; second, it must certify that no element of the intersection has been omitted by exhibiting some proof that there can be no other elements in the intersection than those output. We define the partition-certificate as a proof of the intersection.

**Definition 3 (Partition-Certificate).** *A* partition-certificate *is a partition $(I_j)_{j\leq\delta}$ of $\mathcal{U}$ into intervals such that any singleton $\{x\}$ corresponds to an element $x$ of $\cap_i A_i$, and each other interval $I$ has an empty intersection $I \cap A_i$ with at least one array $A_i$.*

Imagine a function which indicates for each element $x \in \mathcal{U}$ the name of an array not containing $x$ if $x$ is not in the intersection, and "all" if $x$ is in the intersection. The minimal number of times such a function *alternates* names, for $x$ scanning $\mathcal{U}$ in increasing order, is also the minimal size of a partition-certificate of the instance (minus one), which is called *alternation.*

**Definition 4 (Alternation).** *The* alternation $\delta(A_1, \ldots, A_k)$ *of an instance $(A_1, \ldots, A_k)$ is the minimal number of intervals forming a partition-certificate of this instance.*

*Example 3.* The alternation of the instance in Figure 1 is $\delta = 3$, as we can see on the right representation that the partition $(-\infty, 9), [9, 10), [10, +\infty)$ is a partition-certificate of size 3, and that none can be smaller.

The alternation measure was used as a measure of the difficulty of the instance [**?**], as it is the non-deterministic complexity of the instance, and as there is a lower bound increasing with $\delta$ on the complexity of any randomized algorithm. By definition of the partition-certificate:

- for each singleton $\{x\}$ of the partition, any algorithm must find the position of $x$ in all arrays $A_i$, which takes $k$ searches;
- for each interval $I_j$ of the partition, any algorithm must find an array, or a set of arrays, such that the intersection of $I_j$ with this array, or with the intersection of those arrays, is empty.

The cost for finding such a set of arrays can vary and depends on the choices performed by the algorithm. In general it requires less searches if there are many possible answers. To take this into account, for each interval $I_j$ of the partition-certificate we will count the number $r_j$ of arrays whose intersection with $I_j$ is empty. The smaller is $r_j$, the harder is the instance: $1/r_j$ measures the contribution of this interval to the difficulty of the instance.

*Example 4.* Consider for instance the interval $[10, 11)$ in the instance of Figure 1: $r_j = 4$. A random algorithm choosing an array uniformly has probability $r_j/k$ to find an array which do not intersect $[10, 11)$, and will do so on average before $k/r_j$ trials, even if it doesn't memorize which array it tried before. $k$ being fixed, $1/r_j$ measures the difficulty of proving that no element of $[10, 11)$ is in the intersection of the instance.

We name the sum of those contributions the *redundancy* of the instance, and it forms our new measure of difficulty:

**Definition 5 (Redundancy).** *Let $A_1, \ldots, A_k$ be $k$ sorted arrays, and let $(I_j)_{j\leq\delta}$ be a partition-certificate for this instance.*

- *The* redundancy $\rho(I)$ of an interval or singleton $I$ *is defined as equal to* 1 *if $I$ is a singleton, and equal to* $1/\#\{i,\ A_i \cap\ I = \emptyset\}$ *otherwise.*
- *The* redundancy $\rho((I_j)_{j\leq\delta})$ of a partition-certificate $(I_j)_{j\leq\delta}$ *is the sum* $\sum_j \rho(I_j)$ *of the redundancies of the intervals composing it.*
- *The* redundancy $\rho\left((A_i)_{i\leq\ k}\right)$ of an instance *of the intersection problem is the minimal redundancy of a partition-certificate of the instance,* $\min\{\rho\left((I_j)_{j\leq\delta}\right),\ \forall (I_j)_{j\leq\delta}\}$.

Note that the redundancy is always well defined and finite: if $I$ is not a singleton then by definition there is at least one array $A_i$ whose intersection with $I$ is empty, and $\#\{i,\ A_i \cap\ I = \emptyset\} > 0$.

*Example 5.* The partition-certificate $(-\infty, 9), [9, 10), [10, 11), [11, +\infty)$ has redundancy at most $\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{2}$, and no other partition-certificate has a smaller redundancy, hence our instance has redundancy $\frac{7}{6}$.

The redundancy analysis permits to measure the difficulty of the instance in a finer way than with the alternation: for fixed $k, n_1, \ldots, n_k, \delta$, several instances of signature $(k, n_1, \ldots, n_k)$ and alternation $\delta$ may present different difficulties for any algorithm, and different redundancies.

*Example 6.* In the instance from Figure 1 the only way to prove that the intersection of those arrays is empty, is to compute the intersection of one of the arrays from $\{A, B, C, D\}$ with one of the arrays from $\{E, F, G\}$. For simplicity, and without loss of generality, we suppose the algorithm searches to intersect $A$ with another array in $\{B, C, D, E, F, G\}$, and we focus for this example on the number of unbounded searches performed, instead of the number of comparisons: the randomized algorithm looking for the element of $A$ in an array from $\{B, C, D, E, F, G\}$ chosen at random performs on average only 2 searches in the first instance, as the probability to find an array whose intersection is empty with $A$ is then $\frac{1}{2}$.

On the other hand, consider the instance of a subtle variant of the instance of Figure 1, where the element 9 would be present in all the arrays but one, for instance $E$ (only two elements needs to change, $F[4]$ and $G[2]$ which were equal to 10 and are now equal to 9). As the two instances have the same signature and alternation, the alternation analysis yields the same lower bound for both instances. But the randomized algorithm described above performs now on average $k/2$ searches, as opposed to 2 searches on the original instance. This difference of performance is not expressed by a difference of alternation, but is expressed by a difference of redundancy: the new instance has a redundancy of $\frac{1}{2}+1+\frac{1}{2} = 2$ *larger* than the redundancy $\frac{7}{6}$ of the original instance[1].

## 3 Randomized algorithm

The algorithm we propose here is a randomized and simplified version of the `small adaptive` algorithm [?]. It uses the `unbounded search` algorithm, which looks for an element $x$ in a sorted array $A$ of unknown size, starting at position *init*, with complexity $2\lceil\log_2(p - init)\rceil$, where $p$ is the insertion position of the element in the array. It returns a value $p$ such that $A[p-1] < x \leq A[p]$. This algorithm has already been studied before, it can be implemented using the doubling search and binary search algorithms [?,?,?,?], or directly to improve the complexity by a constant factor [?].

Given $t \in \{1, \ldots, k\}$, and $k$ non-empty sorted sets $A_1, \ldots, A_k$ of sizes $n_1, \ldots, n_k$, the `rand intersection` algorithm (algorithm 1) computes

[1] This is just a particular case given as an example, see Section 5 for the general proof.

the intersection $I{=}A_1{\cap}\ldots{\cap}A_k$. For simplicity, we assume that all arrays contain the element $-\infty$ at position 0 and the element $+\infty$ at position $n_i + 1$.

The algorithm is composed of two nested loops. The outer loop iterates through potential elements of the intersection in variable $m$ and in increasing order, and the inner loop checks for each value of $m$ if it is in the intersection.

In each pass of the inner loop, the algorithm searches for $m$ in one array $A_s$ which potentially contains it. The invariant of the inner loop is that, at the start of each pass and for each array $A_i$, $p_i$ denotes the first potential position for $m$ in $A_i$: $A_i[p_i - 1] < m$. The variables #`YES` and #`NO` count how many arrays are known to contain $m$, and are updated depending on the result of each search.

A new value for $m$ is chosen every time we enter the outer loop, at which time the current subproblem is to compute the intersection on the sub-arrays $A_i[p_i, \ldots, n_i]$ for all values of $i$. Any first element $A_i[p_i]$ of a sub-array could be a candidate, but a better candidate is one which is larger than the last value of $m$: the algorithm chooses $A_s[p_s]$, which is by definition larger than $m$. Then only one array $A_s$ is known to contain $m$, hence #`YES` $\leftarrow 1$, and no array is known *not to* contain it, hence #`NO` $\leftarrow 0$. The algorithm terminates when all the values of the current array have been considered, and $m$ has taken the last value $+\infty$.

**Algorithm 1** `Rand Intersection` $(A_1, \ldots, A_k)$

Given $k$ non-empty sorted sets $A_1, \ldots, A_k$ of sizes $n_1, \ldots, n_k$, the algorithm computes in variable $I$ the Intersection $A_1 \cap \ldots \cap A_k$. Note that the only random instruction is the choice of the array in the inner loop.

```
for all $i$ do $p_i \leftarrow 1$ end for
$I \leftarrow \emptyset$; $s \leftarrow 1$
repeat
  $m \leftarrow A_s[p_s]$
  #NO $\leftarrow 0$; #YES $\leftarrow 1$;
  while #YES $< k$ and #NO $= 0$ do
    Let $A_s$ be a random array s.t. $A_s[p_s] \neq m$.
    $p_s \leftarrow$ Unbounded Search$(m, A_s, p_s)$
    if $A_i[p_i] \neq m$ then #NO $\leftarrow 1$ else #YES $\leftarrow$ YES $+ 1$ end if
  end while
  if #YES $= k$ then $I \leftarrow I \cup \{m\}$ end if
  for all $i$ such that $A_i[p_i] = m$ do $p_i \leftarrow p_i + 1$ end for
until $m = +\infty$
return $T$
```

**Theorem 1.** *Algorithm* `rand intersection` *(algorithm 1) performs on average $O(\rho \sum \log(n_i/\rho))$ comparisons on an instance of signature $(k, n_1, \ldots, n_k)$ and of redundancy $\rho$.*

*Proof.* Let $(I_j)_{j\leq\delta}$ be a partition-certificate of minimal redundancy $\rho$. Each comparison performed by the algorithm is said to be performed in *phase* $j$ if $m \in I_j$ for some interval $I_j$ of the partition. Let $C_i^j$ be the number of binary searches performed by the algorithm during phase $j$ in array $A_i$, let $C_i = \sum_j C_i^j$ be the number of binary searches performed by the algorithm in array $A_i$ over the whole execution, and let $(r_j)_{j\leq\delta}$ be such that $r_j$ is equal to 1 if $I$ is a singleton, and to $\#\{i,\, A_i \cap\ I_j = \emptyset\}$ otherwise.

Let's consider a fixed phase $j \in \{1, \ldots, \delta\}$: if the phase is positive (if $m \in I$) then $C_i^j = 1\,\forall i = 1, \ldots, k$. Remark that in this case $1/r_j$ is also equal to 1, so that $C_i^j = 1/r_j$. If the phase is negative (if $m \notin I$), $C_i^j$ is a random variable.

- If $A_i \cap I_j = \emptyset$ then $C_i^j \in \{0, 1\}$ as the algorithm will terminate the phase whenever searching in $A_i$. The probability to do such a search is $\Pr\{C_i^j = 1 | A_i \cap\ I_j = \emptyset\} = \frac{1}{r_j}$, so the average number of searches is $E(C_i^j | A_i \cap\ I_j = \emptyset) = 1 * \Pr\{C_i^j = 1 | A_i \cap\ I_j = \emptyset\} = \frac{1}{r_j}$.
- If $A_i \cap\ I_j \neq \emptyset$ then at each new search, either $C_i^j$ is incremented with probability $\frac{1}{k-1}$, because the search occurred in $A_i$; or $C_i^j$ is fixed in a final way with probability $\frac{r_j}{k-1}$, because an array of empty intersection with $I$ was searched; or neither incremented nor fixed, with probability $1-\frac{1+r_j}{k-1}$, in the other cases. This system is equivalent to a system where $C_i^j$ is incremented with probability $\frac{1}{1+r_j}$ and fixed with probability $\frac{r_j}{1+r_j}$. From this we can deduce that $C_i^j$ is incremented on average $\frac{1+r_j}{r_j} - 1 = \frac{1}{r_j}$ times before it is fixed.

So the algorithm performs on average $E(C_i) = \sum_j \frac{1}{r_j} = \rho$ binary searches in array $A_i$.

Let $g_{i,j}^l$ be the increment of $p_i$ due to the $l$th unbounded search in array $A_i$ during phase $j$. Notice that $\sum_{j,l} g_{i,j}^l \leq n_i$. The algorithm performs $2\log(g_{i,j}^l + 1)$ comparisons during the $l$th search of phase $j$ in array $A_i$. So it performs $2\sum_{j,l} \log(g_{i,l}^l + 1)$ comparisons between $m$ and an element of array $A_i$ during the whole execution. Because of the concavity of the function $\log(x + 1)$, this is smaller than $2C_i \log(\sum_{j,l} \frac{g_{i,j}^l}{C_i} + 1)$, and because of the preceding remark $\left(\sum_{j,l} g^l_{i,j} \leq n_i\right)$, this is still smaller than $2C_i \log(\frac{n_i}{C_i}+1)$.

The functions $f_i(x){=}2x\log(\frac{n_i}{x}{+}1)$ are concave for $x{\leq}n_i$, so $E(f_i(C_i)){\leq}f_i(E(C_i))$. As the average complexity of the algorithm in array $A_i$ is $E(f(C_i))$, and as $E(C_i)=\rho$, on average the algorithm performs less than $2\rho\log(\frac{n_i}{\rho}+1)$ comparisons between $m$ and an element in array $A_i$. Summing over $i$ we get the final result, which is $O(\rho\sum_i \log\frac{n_i}{\rho})$. □

## 4 Randomized Complexity Lower Bound

We prove now that no randomized algorithm can do asymptotically better. The proof is quite similar to the lower bound of the alternation model [**?**], and differs mostly in lemma 1, which must be adapted to the redundancy and whose lower bound is improved by a constant multiplicative factor.

In Lemma 1 we prove a lower bound on average on a distribution of instances of redundancy at most $\rho=4$ and of output size at most 1. We use this result in Lemma 2 to define a distribution on instances of redundancy at most $\rho\in\{4,4n_1\}$ by combining $p=o(\rho)$ sub-instances. In Lemma 3 we prove that any instance of signature $(k,n_1,\ldots,n_k)$ has redundancy $\rho$ at most $2n_1+1$, so that the result of lemma 2 holds for any $\rho\geq$ 4. Finally applying the Yao-von Neumann principle [**?**,**?**,**?**] in Theorem 2 this gives us a lower bound of $\Omega(\rho\sum_{i=2}^{k}\log(n_i/\rho))$ on the complexity of any randomized algorithm for the Intersection problem.

**Lemma 1.** *For any $k\geq 2$, and $0<n_1\leq\ldots\leq n_k$, there is a distribution on instances of the Intersection problem with signature at most $(k,n_1,\ldots,n_k)$, and redundancy at most 4, such that any deterministic algorithm performs at least $\frac{1}{4}\sum_{i=2}^{k}\log n_i+\sum_{i=2}^{k}\frac{1}{2n_i+1}-k+2$ comparisons on average.*

*Proof.* Let $C$ be the total number of comparisons performed by the algorithm, and for each array $A_i$ note $F_i=\log_2(2n_i{+}1)$, and $F=\sum_{i=2}^{k}F_i$.

Let's draw an index $w\in\{2,\ldots,k\}$ equal to $i$ with probability $\frac{F_i}{F}$, and $(k-1)$ positions $(p_i)_{i\in\{2,\ldots,k\}}$ such that $\forall\ i$ each $p_i$ is chosen uniformly at random in $\{1,\ldots,n_i\}$. Let $P$ and $N$ be two instances such that in both $P$ and $N$, for any $1{<}i{<}j{\leq}k$, $a{\in}A_1$, $b,c{\in}A_i$ and $d,e{\in}A_j$ then $b{<}A_i[p_i]{<}c$ and $d{<}A_j[p_j]{<}e$ imply $b{<}d{<}a{<}c{<}e$ (see Figure 2); in $P$, $A_w[p_w]{=}A_1[1]$; in $N$ $A_w[p_w]{>}A_1[1]$; and such that the elements at position $p_i$ in all other arrays than $A_w$ are equal to $A_1[1]$.

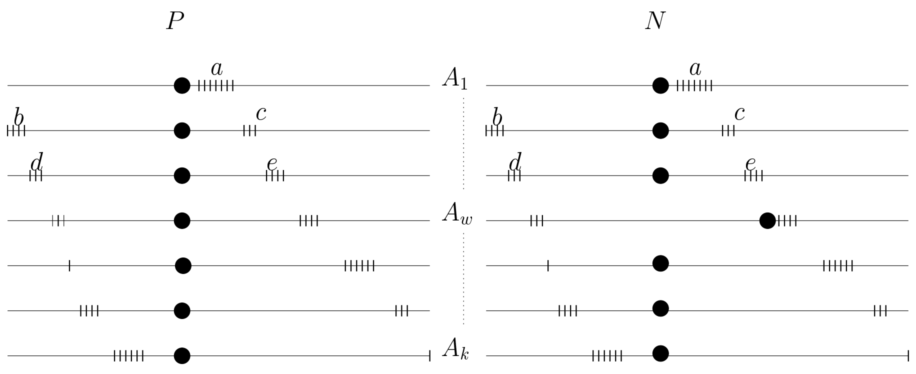

**Fig. 2.** Distribution on $(P, N)$: elements are represented by a dot of abscissa their value, full large dots correspond to the element at position $p_i$ in each array $A_i$.

Let $x = A_1[1]$ be the first element of the first array. Note $x$-comparisons the comparisons between any element and $x$. Because of the special relative positions of the elements, a comparison between two elements $b$ and $d$ in any arrays doesn't yield more information than two comparisons between $x$ and $b$ and between $x$ and $d$: the relative positions to $x$ of elements $b$ and $d$ permit to deduce their order. Hence any algorithm performing $C$ comparisons between arbitrary elements can be expressed as an algorithm performing no more than $2C$ $x$-comparisons, and any lower bound $L$ on the complexity of algorithms using only $x$-comparisons is a $L/2$ lower bound on the complexity of algorithm using comparisons between arbitrary elements.

The redundancy of such instances is no more than 4: the interval $(-\infty, A_1[1])$ is sufficient to certificate that no element smaller than $x$ is in the intersection, and stand for a redundancy of at most 1; the interval $(A_1[n_1], +\infty, )$ is sufficient to certificate that no element larger than $A_1[n_1]$ is in the intersection, and stands for a redundancy of at most 1; the interval $[A_1[1], A_1[n_1]]$ is sufficient in $N$ to complete the partition-certificate, and stands for a redundancy of at most 1; the singleton $\{x\}$ and the interval $(A_1[1], A_1[n_1]]$ are sufficient in $P$ to complete the partition-certificate, and stands for a redundancy of at most $1+\frac{1}{k-1}$.

The only difference between instances $P$ and $N$ is the relative position of element $A_w[p_w]$ to the other elements composing the instance, as described in Figure 2. Any algorithm computing the intersection of $P$ has to find the $(k-1)$ positions $\{p_2, \ldots, p_k\}$. Any algorithm computing the intersection of $N$ has to find $w$ and the afferent position $p_w$. Any algorithm distinguishing between $P$ and $N$ has to find $p_w$: we will prove that it needs on average almost $\frac{F}{2} = \frac{1}{2}\sum_{i=2}^{k} \log_2(2n_i+1)$ comparisons to do so.

Let $A$ be a deterministic algorithm using only $x$-comparisons to compute the intersection. As the algorithm $A$ has not distinguished between $P$ and $N$ till it found $w$, let $X_i$ denote the number of $x$-comparisons performed by $A$ in array $A_i$ for both $P$ or $N$. Let $Y_i$ denote the number of $x$-comparisons performed by $A$ in array $A_i$ for $N$; and let $\xi_i$ be the indicator variable which equals 1 exactly if $p_i$ has been determined by $A$ on instance $P$. The number of comparisons performed by $A$ is $C = \sum_{i=2}^{k} X_i$. Restricting ourselves to arrays in which the position $p_i$ has been determined, we can write $C \geq \sum_{i=2}^{k} X_i\xi_i = \sum_{i=2}^{k} Y_i\xi_i$.

Let's consider $E(Y_i\xi_i)$: the expectancy can be decomposed as a sum of probabilities $E(Y_i\xi_i) = \sum_h \Pr\{Y_i\xi_i \geq h\}$, and in particular $E(Y_i\xi_i) \geq \sum_{h=1}^{F_i} \Pr\{Y_i\xi_i \geq h\}$. Those terms can be decomposed using the property $\Pr\{a \vee b\} \leq \Pr\{a\} + \Pr\{b\}$:

$$\begin{aligned}
\Pr\{Y_i\xi_i \geq h\} &= \Pr\{Y_i \geq h \wedge \xi_i = 1\} \\
&= 1 - \Pr\{Y_i < h \vee \xi_i = 0\} \\
&\geq 1 - \Pr\{Y_i < h\} - \Pr\{\xi_i = 0\} \\
&= \Pr\{\xi_i = 1\} - \Pr\{Y_i < h\} \qquad (1)
\end{aligned}$$

The probability $\Pr\{Y_i < h\}$ is bounded by the usual decision tree lower bound: if we consider the binary $x$-comparisons performed by algorithm $A$ in set $A_i$, there are at most $2^h$ leaves at depth less than $h$. Since the insertion position of $x$ in $A_i$ is uniformly chosen, these leaves are equiprobable and have total probability at most $\Pr\{Y_i < h\} \leq \frac{2^h}{2n_i+1} = 2^{h-F_i}$. Those terms for $h \in \{1, \ldots, F_i\}$ form a geometric sequence whose sum is equal to $2(1 - 2^{-F_i})$, so $E(Y_i\xi_i) \geq F_i \Pr\{\xi_i = 1\} - 2(1 - 2^{-F_i})$. Then

$$\begin{aligned}
E(C) \geq \sum_{i=2}^{k} E(Y_i\xi_i) &\geq \sum_{i=2}^{k} F_i \Pr\{\xi_i = 1\} - \sum_{i=2}^{k} 2(1 - 2^{-F_i}) \\
&\geq \sum_{i=2}^{k} F_i \Pr\{\xi_i = 1\} + 2\sum_{i=2}^{k} 2^{-F_i} - 2(k-2). \qquad (2)
\end{aligned}$$

Let's fix $p = (p_2, \ldots, p_k)$. There are only $k-1$ possible choices for $w$. Algorithm $A$ can only differentiate between $P$ and $N$ when it finds $w$. Let $\sigma$ denote the order in which these instances are dealt with by $A$ for $p$ fixed. Then $\xi_i = 1$ if and only if $\sigma_i \leq \sigma_w$, and so $\Pr\{\xi_i = 1|p\} = \sum_{j:\sigma_j \geq \sigma_i} F_j/F$.

Summing over $p$, and then over $i$, we get an expression of the first term in Equation (2):

$$\Pr\{\xi_i = 1\} = \sum_p \Pr\{\xi_i = 1|p\} \Pr\{p\} = \sum_p \sum_{j:\sigma_j \geq \sigma_i} \frac{F_j}{F} \Pr\{p\}$$

$$\sum_{i=2}^{k} F_i \Pr\{\xi_i = 1\} = \sum_p \sum_{i=2}^{k} \sum_{j:\sigma_j \geq \sigma_i} \frac{F_i F_j}{F} \Pr\{p\} = \sum_p \Pr\{p\} \sum_{i=2}^{k} \sum_{j:\sigma_j \geq \sigma_i} \frac{F_i F_j}{F}.$$

In the sum, each term "$F_i F_j$" appears exactly once, and

$$\left(\sum_i F_i\right)^2 = 2 \sum_i \sum_{i \leq j} F_i F_j - \sum_i {F_i}^2,$$

hence

$$\sum_{i=2}^{k} \sum_{j:\sigma_j \geq \sigma_i} F_i F_j = \frac{1}{2} \left( \left( \sum_{i=2}^{k} F_i \right)^2 + \sum_{i=2}^{k} {F_i}^2 \right),$$

which is independent of $p$. Then we can conclude:

$$\sum_{i=2}^{k} F_i \Pr\{\xi_i = 1\} = \frac{1}{2} \frac{1}{F} \left( \left( \sum_{i=2}^{k} F_i \right)^2 + \sum_{i=2}^{k} {F_i}^2 \right) \sum_p \Pr\{p\} = \frac{1}{2} \sum_{i=2}^{k} F_i.$$

Plugging this into Equation (2), we obtain a lower bound of $\frac{1}{2}\sum_{i=2}^{k} F_i + 2\sum_{i=2}^{k} 2^{-F_i} - 2(k-2)$, which is $\frac{1}{2}\sum_{i=2}^{k} \log_2(2n_i+1) + 2\sum_{i=2}^{k} \frac{1}{2n_i+1} - 2(k-2)$ on the average number of $x$-comparisons $E(C)$ performed by any deterministic algorithm restricted to $x$-comparisons. This in turn implies a lower bound of $\frac{1}{4}\sum_{i=2}^{k} \log_2(2n_i+1) + \sum_{i=2}^{k} \frac{1}{2n_i+1} - (k-2)$ on the average number of comparisons performed by *any* deterministic algorithm, hence the result. □

**Lemma 2.** *For any $k \geq 2$, $0 < n_1 \leq \ldots \leq n_k$ and $\rho \in \{4, \ldots, 4n_1\}$, there is a distribution on instances of the Intersection problem of signature at most $(k, n_1, \ldots, n_k)$, and redundancy at most $\rho$, such that any deterministic algorithm performs on average $\Omega(\rho \sum_{i=1}^{k} \log(n_i/\rho))$ comparisons.*

*Proof.* Let's draw $p = \lfloor \rho/4 \rfloor$ pairs $(P_j, N_j)_{j \in \{1,\ldots,p\}}$ of sub-instances of signature $k, \lfloor n_1/p \rfloor, \ldots, \lfloor n_k/p \rfloor)$ from the distribution of lemma 1. As $\rho \leq 4n_1$, $p \leq n_1$ and $\lfloor n_1/p \rfloor > 0$ hence the sizes of all the arrays are positive. Let's choose uniformly at random each sub-instance $I_j$ between the positive sub-instance $P_j$ and the negative sub-instance $N_j$, and form a larger instance $I$ by unifying the arrays of same index from each sub-instance, such that the elements from two different sub-instances never interleave, as in Figure 3

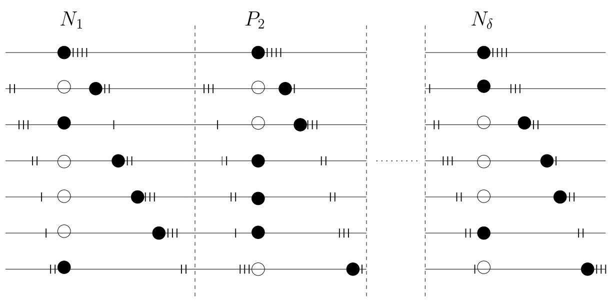

**Fig. 3.** $p$ elementary instances unified to form a single large instance.

This defines a distribution on instances of redundancy at most $4p$ so less than $\rho$, and of signature at most $(k, n_1, \dots, n_k)$. Solving this instance implies to solve all the $p$ sub-instances. Lemma 1 gives a lower bound of $\frac{1}{4}\ \sum_{i=2}^{k}\ \log(2n_i/p+1) + \sum_{i=2}^{k} \frac{1}{2n_i+1} - k{+}2$ comparisons on average for each of the $p$ sub problems, hence a lower bound of $\frac{p}{4}\ \sum_{i=2}^{k}\ \log(2n_i/p + 1) + p(\sum_{i=2}^{k} \frac{1}{2n_i/p+1} - k{+}2)$, which is $\Omega(\rho \sum_{i=1}^{k} \log(n_i/\rho))$. □

**Lemma 3.** *For any $k \geq 2$, $0 < n_1 \leq \dots \leq\ n_k$, any instance of signature $(k, n_1, \dots, n_k)$ has redundancy $\rho$ at most $2n_1 + 1$.*

*Proof.* First observe that there is always a partition-certificate of size $2n_1 + 1$. Then that the redundancy of any partition-certificate is by definition smaller than the size of the partition. Hence the result. □

**Theorem 2.** *For any $k \geq 2$, $0 < n_1 \leq \dots \leq\ n_k$ and $\rho \in \{4, \dots, 4n_1\}$, the complexity of any randomized algorithm for the Intersection problem on instances of signature at most $(k, n_1, \dots, n_k)$, and redundancy at most $\rho$ is $\Omega(\rho \sum_{i=1}^{k} \log(n_i/\rho))$.*

*Proof.* This a simple application of lemma 2, lemma 3 and of the Yao-von Neumann principle [**?**,**?**,**?**]:

– lemma 2 gives a distribution for $\rho \in \{4, \dots, 4n_1\}$ on instances of redundancy *at most* $\rho$,

– and lemma 3 proves that there are no instances of redundancy more than $2n_1+1$, hence the result of lemma 2 holds for any $\rho \geq 4$.
– Then the Yao-von Neumann principle permits to deduce from this distribution a lower bound on the complexity of randomized algorithms. □

This analysis is much finer than the previous lower bound presented in [**?**], where the additive term in $-k$ was ignored, although it makes the lower bound trivially negative for large values of the difficulty $\delta$. Here the additive term is suppressed for $\min_i n_i \geq 128$, and the multiplicative factor between the lower bound and the upper bound is reduced to 16 instead of 64. This technique can be applied to the alternation analysis of the intersection with the same result. Note also that a multiplicative factor of 2 in the gap comes from the unbounded searches in the algorithm and can be reduced using a more complicated algorithm for the unbounded search [**?**].

## 5 Comparisons with the alternation model

The redundancy model is strictly finer than the alternation model: some algorithms, optimal for the alternation model, are not optimal anymore in the redundancy model (theorem 3), and any algorithm optimal in the redundancy model is optimal in the alternation model (theorem 4). So the `rand intersection` algorithm is theoretically better than its deterministic variant, and the redundancy model permits a better analysis than the alternation model.

**Theorem 3.** *Any deterministic algorithm performs $\Omega(k\rho\sum_i \log(n_i/k\rho))$ comparisons in the worst case over all instances of signature at most $(k, n_1, \dots, n_k)$ and redundancy at most $\rho$.*

*Proof.* The proof uses the same decomposition than the proof of theorem 2, but uses an adversary argument to obtain a deterministic lower bound. Build $\delta = \frac{k\rho}{3}$ sub-instances of signature $(k, \lfloor\frac{n_1}{\delta}\rfloor, \dots, \lfloor\frac{n_k}{\delta}\rfloor)$, redundancy at most 3, such that $x = A_1[1]$ is present in half of the other arrays, as in Figure 4.

On each sub-instance an adversary can force any deterministic algorithm to perform a search in each of the arrays containing $x$, and in a single array which does not contain $x$. Then the deterministic algorithm performs $\frac{1}{2}\sum_{i=2}^{k}\log\frac{n_i}{\delta}$ comparisons. In total over all sub-instances, the adversary can force any deterministic algorithm to perform $\frac{\delta}{2}\sum_{i=2}^{k}\log\frac{n_i}{\delta}$ comparisons, i.e. $\frac{k\rho}{4}\sum_{i=2}^{k}\log\frac{n_i}{k\rho}$, which is $\Omega(k\rho\sum_{i=2}^{k}\log\frac{n_i}{k\rho})$. □

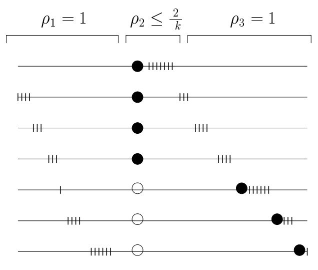

**Fig. 4.** Element $x$ is present in half of the arrays of the sub-instance.

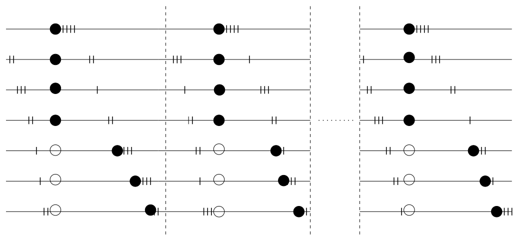

**Fig. 5.** The adversary performs several strategies in parallel, one for each sub-instance.

As $x\log(n/x)$ is an increasing function of $x$, $k\rho\sum_i \log(n_i/k\rho) > \rho\sum_i \log(n_i/\rho)$ and no deterministic algorithm is optimal in the redundancy model.

**Theorem 4.** *Any algorithm optimal in the redundancy model is optimal in the alternation model.*

*Proof.* By definition of the redundancy $\rho$ and of the alternation $\delta$ of an instance, $\frac{\delta}{k} \leq \rho \leq \delta$. So if an algorithm performs $O(\rho \sum \log \frac{n_i}{\rho})$ comparisons, it also performs $O(\delta \sum \log \frac{n_i}{\delta})$ comparisons. Hence the result as this is the lower bound in the alternation model. $\square$

This proves also that the measure of difficulty of Demaine, López-Ortiz and Munro [?] is not comparable with the measure of redundancy, as it is not comparable with the measure of alternation [?, Section 2.3]. This means that the two measures are complementary, without being redundant in any way, as it was for the alternation. All those measures describe the difficulty of the instance:

– the *alternation* describes the number of key blocs of consecutive elements in the instance;
– the *cost* describes the repartition of the size of those blocs;
– the *redundancy* describes the difficulty to find each bloc.

But only the *cost* and the *alternation* matter, because the alternation analysis is reduced to the redundancy analysis.

## 6 Perspectives

The $t$-threshold set and opt-threshold set problems [?] are natural generalizations of the intersection problem, which could be useful in indexed search engines. The redundancy seems to be important in the complexity of these problems as well, but we failed to define the proper measure there. Once the proper definition of a certificate and of the difficulty measure is found, the results of this paper should be generalized to the $t$-threshold set and opt-threshold set problems.

Deterministic algorithms for the intersection have been studied on practical data [**?**]. The performance of the randomized versions of those algorithms, in terms of the number of comparisons *and* in terms of running time, will be studied. We expect for instance the average number of comparisons to decrease, as the randomized algorithm is more independent from the structure of the instance than the deterministic one.

*Acknowledgments:* This work was done at UBC, Vancouver, Canada, during a post-doc internship founded by the "Institut National de Recherche en Informatique et Automatisme" (INRIA) of France, under the mentor-ship of Joël Friedman. The redundancy analysis is a development of joint work with Claire Kenyon on the alternation analysis. The author wishes to thank all these people.