Alternation and Redundancy Analysis of the Intersection Problem

JÉRÉMY BARBAY

University of Chile

AND

CLAIRE KENYON

Brown University

Abstract. The *intersection of sorted arrays* problem has applications in search engines such as Google. Previous work has proposed and compared deterministic algorithms for this problem, in an adaptive analysis based on the encoding size of a certificate of the result (cost analysis). We define the *alternation analysis*, based on the nondeterministic complexity of an instance. In this analysis we prove that there is a deterministic algorithm asymptotically performing as well as any randomized algorithm in the comparison model. We define the *redundancy analysis*, based on a measure of the internal redundancy of the instance. In this analysis we prove that any algorithm optimal in the redundancy analysis is optimal in the alternation analysis, but that there is a randomized algorithm which performs strictly better than any deterministic algorithm in the comparison model. Finally, we describe how these results can be extended beyond the comparison model.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Sorting and searching*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Adaptive analysis, alternation analysis, intersection, redundancy analysis, randomized algorithm, intersection of sorted arrays

Authors' addresses: J. Barbay, Departamento de Ciencias de la Computacin Universidad de Chile, Bianco Encalada 2120, Santiago, Chile; email: jbarbay@dcc.uchile.cl; C. Kenyon, Computer Science Department, Brown University, Providence, RI 02912.

1. *Introduction*

We consider search engines where *queries* are composed of several keywords, each associated with a sorted array of references to entries in some database [Witten et al. 1994, p. 136]. 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.

There is an extensive literature on the merging [Hwang and Lin 1972, 1971; Christen 1978; Manacher 1979; de la Vega et al. 1998, 1993] or intersection [Baeza-Yates 2004] of two sorted arrays. The two problems are similar, as both require the algorithm to place each element in the context of the other elements. In relational databases, the intersection of more than two arrays is computed by intersecting the arrays two by two. The only optimization available in this context consists in choosing the *order* in which these sets are intersected, and the literature explores how to use statistics precomputed on the content of the database to choose the best order; see Chaudhuri [1998] and the references therein.

Demaine et al. [2001] showed that a holistic algorithm, which considers the query as a whole rather than as a decomposition of it in smaller two-by-two intersection queries, is more efficient, both in theory and in practice.

In this article we present another theoretical analysis, called the *alternation* analysis [Barbay and Kenyon 2002], based on the nondeterministic complexity of the instance, and prove tight bounds on the randomized computational complexity of the intersection. One intriguing fact of this analysis is that the lower bound applies to randomized algorithms, whereas a deterministic algorithm is optimal. Does this mean that no randomized algorithm can perform better than a deterministic one on the intersection problem? To answer this question, we extend the alternation analysis to the *redundancy* analysis [Barbay 2003], based on a measure of the internal redundancy of the instance. This analysis permits to prove that for the intersection problem, randomized algorithms perform better than deterministic ones in terms 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 parameterized by the measure of difficulty.

We define formally the intersection problem in Section 2, and sketch the alternation analysis and its results in Section 3. We define the redundancy analysis and study it in Section 4: We give and analyze a randomized algorithm in Section 4.1, and prove that this algorithm is optimal in Section 4.2.

We answer the question of the usefulness of randomized algorithms for the intersection problem in Section 5: No deterministic algorithm can be optimal in a redundancy analysis, hence the superiority of randomized algorithms. We list in Section 6 several perspectives of this work.

2. *Definitions*

We consider queries composed of several keywords, each associated to a sorted array of references. The references can be, for instance, addresses of webpages,

$A =$	**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 in a better way the structure of the instance, where the x-coordinate of each element is equal to its value.

the only requirement being a *total order* on them, that is, that all unequal pairs of references can be ordered. To study the intersection problem, we consider any set of two or more arrays, of elements from 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 2.1. We consider 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 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 2.2. 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 2.3. 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 2.4. 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.

Any algorithm (even a nondeterministic one) computing the intersection must prove the correctness of the output: First, it must certify that all 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 certificate that there can be no other elements in the intersection than those output. We define the partition certificate as such a proof.

Definition 2.5. A *partition certificate* is a partition $(I_j)_{j \leq \delta}$ of 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.

3. *Alternation Analysis*

Imagine a function which indicates for each element $x \in$ 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 U in increasing order, is just one less than the minimal size of a partition certificate of the instance, which is called the *alternation* of the instance.

Definition 3.1. The *alternation* δ of an instance $(A_1, \ldots, A_k)$ is the minimal number of intervals forming a partition certificate of this instance.

Example 3.2. 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 of an instance I is also the complexity of the best nondeterministic algorithm on I (plus 1), namely the *nondeterministic complexity*. This nondeterministic complexity forms a weak lower bound on the complexity of any randomized or deterministic algorithm solving I, and hence a natural measure of the difficulty of the instance.

Indeed, among instances of same signature and alternation, it is possible to prove a tight bound on the randomized complexity of the intersection problem: By providing a difficult distribution of instances and using the minimax principle, we prove a lower bound on the complexity of any randomized algorithm solving the problem [Barbay and Kenyon 2002].

THEOREM 3.3 (ALTERNATION LOWER BOUND [BARBAY AND KENYON 2002]). *For any $k \geq 2$, $0 < n_1 \leq \cdots \leq n_k$, and $\delta \in \{4, \ldots, 4n_1\}$, and for any randomized algorithm A_R for the intersection problem, there is an instance of signature at most $(k, n_1, \ldots, n_k)$ and alternation at most δ, such that A_R performs $\Omega(\delta \sum_{i=1}^{k} \log(n_i/\delta))$ comparisons on average on it.*

PROOF. This is a simple application of Lemma 4.9 (stated and proved in Section 4.2) and of the Yao-von Neumann principle [Neumann and Morgenstern 1944; Sion 1958; Yao 1977], described in the following.

—Lemma 4.9 gives a distribution for $\delta \in \{4, \ldots, 4n_1\}$ on instances of alternation *at most* δ; and

—then the Yao-von Neumann principle permits to deduce from this distribution a lower bound on the worst-case complexity of randomized algorithms. □

On the other hand, a simple deterministic algorithm reaches this lower bound. As the class of deterministic algorithms is contained in the class of randomized algorithms, this proves that the bound is tight for randomized algorithms.

THEOREM 3.4 (ALTERNATION UPPER BOUND [BARBAY AND KENYON 2002]). *There is a deterministic algorithm which performs $O(\delta \sum_{i=1}^{k} \log(n_i/\delta))$ comparisons on any instance of signature $(k, n_1, \ldots, n_k)$ and alternation δ.*

PROOF. The deterministic version of algorithm Rand Intersection (see Section 4.1), where the choice of a random array is replaced by the choice of the next array in a fixed order, performs $O(\delta \sum_{i=1}^{k} \log(n_i/\delta))$ comparisons on an instance of signature $(k, n_1, \ldots, n_k)$ and of alternation δ. Its analysis is very similar to the one of the randomized version given in the proof of Theorem 4.7.

Note that this algorithm is distinct from the algorithm presented previously [Barbay and Kenyon 2002], where the algorithm was performing unbounded searches in parallel in the arrays. Here the algorithm performs one unbounded search at a time, which saves some comparisons in many cases, for any arbitrary signature $(k, n_1, \ldots, n_k)$ (but not in the worst case). □

The lower bound applies to any randomized algorithm, when a mere deterministic algorithm is optimal. Does this mean that no randomized algorithms can do better than a deterministic one on the intersection problem? We refine the analysis to answer this question.

4. *Redundancy Analysis*

By definition of the partition certificate, the following is noted.

—For each singleton $\{x\}$ of the partition, any algorithm must find the position of x in all arrays A_i, which takes k searches; and

—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 these 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 fewer 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 4.1. Consider for instance, the interval $I_j = [10, 11)$ in the instance of Figure 1: We see that $r_j = 4$ arrays have an empty intersection with it. A randomized algorithm, choosing an array uniformly at random, has probability r_j/k to find an array which does not intersect I_j, and will do so after at most $\lceil k/r_j \rceil$ trials on average, even if it tries several times in the same array because it doesn't memorize which array it tried before. As the number of arrays k is fixed, the value $1/r_j$ measures the difficulty of proving that no element of I_j is in the intersection of the instance.

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

Definition 4.2. Let $A_1, \ldots, A_k$ be k sorted arrays, and let $(I_j)_{j\le\delta}$ 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* $\rho((I_j)_{j\le\delta})$ *of a partition certificate* $(I_j)_{j\le\delta}$ is the sum $\sum_j \rho(I_j)$ of the redundancies of the intervals composing it.

—The *redundancy* $\rho\left((A_i)_{i\le k}\right)$ *of an instance* of the intersection problem is the minimal redundancy $\min\{\rho\left((I_j)_{j\le\delta}\right),\ \forall (I_j)_{j\le\delta}\}$ of a partition certificate of the instance.

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, hence $\#\{i,\ A_i \cap I = \emptyset\} > 0$.

Example 4.3. The partition certificate $\{(-\infty, 9), [9, 10), [10, 11), [11, +\infty)\}$ has redundancy at most $1/2+1/3+1/4+1/2 = 7/6$ for the instance given Figure 1, and no other partition certificate has a smaller redundancy, hence the instance has redundancy $7/6$.

$A =$	**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	**9**
$G =$	8	**9**	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		9											
$G:$								8	9										19	20

FIG. 2. A much more difficult variant of the instance of Figure 1: Only two elements changed, namely $F[4]$ and $G[2]$ which were equal to 10 and are now equal to 9, but the redundancy is now $\rho = 1/2 + 1 + 1/6 + 1/2 = 2 + 1/6$.

The main idea is that the redundancy analysis permits to measure the difficulty of the instance in a finer way than the alternation analysis: For fixed $k, n_1, \ldots, n_k$ and δ, several instances of signature $(k, n_1, \ldots, n_k)$ and alternation δ may present various levels of difficulty, and the redundancy helps to distinguish between these.

Example 4.4. In the instance from Figure 1, the only way to prove the emptiness of the intersection is to compute the intersection of one of the arrays chosen from $\{A, B, C, D\}$ with one of the arrays chosen from $\{E, F, G\}$, because $9 \in A \cap B \cap C \cap D$ and $10 \in E \cap F \cap G$. For simplicity and without loss of generality, suppose that the algorithm searches to intersect A with another array in $\{B, C, D, E, F, G\}$, and consider the number of unbounded searches performed, instead of the number of comparisons. The randomized algorithm looking for the element of A in a random array from $\{B, C, D, E, F, G\}$ performs on average only 2 searches, as the probability to find an array whose intersection is empty with A is then $1/2$.

On the other hand, consider the instance of Figure 2, a variant of the instance of Figure 1, where element 9 is present in all the arrays but E. 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 before now performs an average of $k/2$ searches, as opposed to 2 searches on the original instance. This difference in difficulty between these very similar instances is not expressed by a difference of alternation, but by a difference of redundancy: The new instance has a redundancy of $1/2 + 1 + 1/6 + 1/2 = 2 + 1/6$, which is larger by one than the redundancy $7/6$ of the original instance. This difference of one corresponds to k more doubling searches for this simple instance. This difference is used in Section 5 to create instances where a deterministic algorithm performs $O(k)$ times more searches and comparisons than a randomized algorithm.

4.1. RANDOMIZED ALGORITHM. For simplicity, we assume that all arrays contain the element $-\infty$ at position 0 and the element $+\infty$ at position n_i+1. Given this convention, the intersection algorithm can ignore the sizes of the sets. This is the case in particular for pipelined computations, where the sets are not completely computed when the intersection starts, for instance, in parallel applications.

An unbounded search looks for an element x in a sorted array A of unknown size, starting at position $init$. It returns a value p such that $A[p-1] < x \leq A[p]$, called the *insertion rank* of x in A. It can be performed combining the doubling search and binary search algorithms [Barbay and Kenyon 2002; Demaine et al. 2001, 2000], and is then of complexity $2\lceil \log_2(p - init) \rceil$, or in a more complicated way [Bentley and Yao 1976] to improve the complexity by a constant factor of less than 2.

```
Algorithm. Rand Intersection (A_1, ..., A_k)
for all i do p_i ← 1 end for
Result ← ∅; s ← 1
repeat
    m ← A_s[p_s]
    #NO ← 0; #YES ← 1;
    while YES < k and #NO = 0
        Let A_s be a random array such that A_s[p_s] ≠ m.
        p_s ← Unbounded Search(m, A_s, p_s)
        if A_s[p_s] ≠ m then #NO ← 1 else #YES ← YES + 1 end if
    endwhile
    if #YES = k then Result ← Result ∪ {m} end if
    for all i such that A_i[p_i] = m do p_i ← p_i + 1 end for
until m = +∞
return Result
```

FIG. 3. The algorithm Rand Intersection: Given k nonempty sorted sets $A_1, \ldots, A_k$ of sizes $n_1, \ldots, n_k$, the algorithm computes in variable *result* the intersection $A_1 \cap \ldots \cap A_k$. Note that the only random instruction is the choice of array in inner loop.

Using unbounded rather than binary search is crucial to the complexity of the intersection algorithm. Consider the task of searching d elements $x_1 \leq x_2 \leq \cdots \leq x_d$ in a sorted array of size n. It requires $d \log n_i$ comparisons using binary search, but less than $2d \log(n_i/d)$ comparisons using unbounded search. To see this, define p_j such that $p_0 = 0$ and $A[p_j] = x_j \,\forall j \in \{1, \ldots, d\}$: The jth doubling search performs no more than $2\log(p_j - p_{j-1})$ comparisons. By concavity of the log, the sums $\sum_{j\leq d} 2\log(p_j - p_{j-1})$ are no larger than $2d\log(\sum_{j\leq d}(p_j - p_{j-1})/d)$. The sum $\sum_{j\leq d}(p_j - p_{j-1})$ is equal to $p_d - p_0$, which is smaller than the size n of the array. Hence, the d doubling searches perform less than $2d\log(n_i/d)$ comparisons.

THEOREM 4.5. *Algorithm Rand Intersection (see Figure* 3*) computes the intersection of the arrays given as input.*

PROOF. Given k nonempty sorted arrays $A_1, \ldots, A_k$ of sizes $n_1, \ldots, n_k$, the Rand Intersection algorithm (Figure 3) computes the intersection $A_1 \cap \ldots \cap A_k$. 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, whether 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, the 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 subarrays $A_i[p_i, \ldots, n_i]$ for all values of i. Any first element $A_i[p_i]$ of a subarray 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 values of the current array have been considered, and m has taken the last value $+\infty$. $\square$

We now analyze the complexity of algorithm Rand Intersection (Figure 3) as a function of the redundancy ρ of the instance. To understand the intuition behind the analysis, consider the following example.

Example 4.6. For a fixed interval I_j, suppose that the algorithm receives six arrays such that A_1, A_2, A_3, and A_4 contain many elements from I_j but have none in common, and such that A_5 and A_6 contain no elements from I_j. Ignore all steps of the algorithm where m takes values out of the interval I_j: The interval defines a *phase* of the algorithm. Suppose that m takes a value in I_j at some point, for instance, from A_1. At each iteration of the external loop, the algorithm ignores the array from which the current value of m was taken, chooses one between the four remaining arrays, searches in the chosen one, and updates the value of m accordingly.

—With probability $3/5$ the algorithm chooses the set A_1, A_2, A_3, or A_4 (depending on which set the current value of m comes from) and potentially fails to terminate the phase.

—With probability $2/5$ the algorithm chooses A_5 or A_6, performs a search in it (there might be elements left from intervals $I_1 \cup \ldots \cup I_{j-1}$), and updates m to a value from I_{j+1}, which terminates the current phase.

We are interested in the number C_i^j of searches performed in each array A_i during this phase. As m takes a value outside of I_j after a search in A_5 or A_6, both C_5^j and C_6^j are random Boolean variables which depend only on the last choice of the algorithm before changing phase: The expectation of C_5^j (respectively, C_6^j) is exactly the probability that A_5 (respectively, A_6) is picked, knowing that one of those is picked, namely $1/2$.

The algorithm can perform many searches in A_1, A_2, A_3, and A_4, so the variables C_1^j, C_2^j, C_3^j, and C_4^j are random integer variables which depend on all the choices of the algorithm but the last. The probability that A_1 is chosen is null if m comes from A_1. Otherwise, it is less than the probability that A_1 is chosen, knowing that m doesn't come from A_1: $\Pr[A_1 \text{ is chosen }] = \Pr[A_1 \text{ is chosen and } m \text{ does not come from } A_1] \leq \Pr[A_1 \text{ is chosen } | m \text{ does not come from } A_1]$. Hence the probability that A_1 is chosen is less than $1/4$.

C_1^j is increased each time A_1 is chosen (probability $a \leq 1/5$), is finalized as soon as A_5 or A_6 is chosen (probability $b = 2/5$), and stays the same each time another array is chosen (probability $c \geq 2/5$). Ignore all the steps where C_2^j, C_3^j, or C_4^j are increased: Knowing that C_2^j, C_3^j, or C_4^j are not increased, the probability that C_1^j is increased is $a/(a+b) \leq 1/3$, and the probability that it is finalized is $b/(a+b) \geq 2/3$. Such a system will iterate at most $3/2$ times on average, and increment C_1^j each time but the last, that is, $3/2 - 1 = 1/2$ times on average. The same reasoning holds for A_2, A_3, and A_4. Hence, in this example $E(C_i^j) = 1/2$ for each set A_i, where 2 is the number of arrays which contain no elements from I_j.

The proof of Theorem 4.7 argues similarly in the more general case.

THEOREM 4.7 (REDUNDANCY UPPER BOUND [BARBAY 2003]). *Algorithm Rand Intersection (Figure 3) performs on average $O(\rho \sum_{i=1}^{k} \log(n_i/\rho))$ comparisons on any instance of signature $(k, n_1, \ldots, n_k)$ and of redundancy ρ.*

PROOF. Let $(I_j)_{j\leq\delta}$ be a partition certificate of minimal redundancy ρ. 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 searches performed by the algorithm during phase j in array A_i, let $C_i = \sum_j C_i^j$ be the number of 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_j is a singleton, and to $\#\{i,\ A_i \cap I_j = \emptyset\}$ otherwise.

Let us consider a fixed phase $j \in \{1, \ldots, \delta\}$, and compute the average number of searches $E(C_i^j)$ performed in each array A_i during phase j. At each iteration of the internal loop, the algorithm chooses an array in which m is not known to be. As m always comes from one array, there are at most $k-1$ of these arrays, hence each array is chosen with probability at least $1/(k-1)$. If the element m currently considered is in the intersection, then each array A_i will be searched and C_i^j is equal to 1. In this case $1/r_j$ is also equal to 1, so that $C_i^j{=}1/r_j{=}E(C_i^j)$.

Suppose that m is *not* in the intersection, and that $A_i \cap I_j$ is empty. Either A_i is never chosen and $C_i^j = 0$, or A_i is chosen and $C_i^j = 1$, because the algorithm will terminate the phase after searching in A_i. The probability that A_i is chosen is at most the probability that it is chosen knowing that this is the last search of the phase.

$$\begin{aligned}\Pr[A_i \text{ is chosen}] &= \Pr[A_i \text{ is chosen and last search}]\\ &\leq \Pr[A_i \text{ is chosen}|\text{ last search}]\end{aligned}$$

As the arrays are chosen uniformly, this probability is $\Pr\{C_i^j = 1\} \leq 1/r_j$, and the average number of searches is at most $E(C_i^j) = 1 * \Pr\{C_i^j = 1\} \leq 1/r_j$.

The interesting case is when m is not in the intersection but $A_i \cap I_j \neq \emptyset$. At each new search, one of three possibilities holds.

(1) C_i^j is incremented by one because the search occurred in A_i, which occurs with probability less than $1/(k-1)$; or

(2) C_i^j is fixed in a final way because an array was found whose intersection with I_j is empty, which occurs with probability $r_j/(k-1)$; or

(3) C_i^j is neither incremented nor fixed if another array was searched but its intersection with I_j is not empty.

The combined probability of the first and second cases is $1/(k-1) + r_j/(k-1)$. Ignoring the third case where C_i^j never changes, the conditional probability of the first case is $\frac{1}{k-1}/(\frac{1}{k-1} + \frac{r_j}{k-1})$. Hence, this system is equivalent to a system where C_i^j is incremented by one with probability at least $1/(1+r_j)$, and fixed with the remaining probability, at most $r_j/(1+r_j)$. Such a system iterates at most $(1{+}r_j)/r_j$ times on average, and increments C_i^j at each iteration but the last: The final value of C_i^j is at most $(1+r_j)/r_j - 1 = 1/r_j$.

Hence, the average number of searches performed in each array A_i during phase j is $E(C_i^j) \leq 1/r_j$. Summing over all phases, this implies that the algorithm performs on average $E(C_i) \leq \sum_j 1/r_j = \rho$ searches in each array A_i.

Let $g_{i,j}^{\ell}$ be the increment of p_i due to the ℓth unbounded search in array A_i during phase j. Notice that $\sum_{j,\ell} g_{i,j}^{\ell} \leq n_i$. The algorithm performs at most $2\log(g_{i,j}^{\ell}+1)$

comparisons during the ℓth search of phase j in array A_i. So it performs at most $2\sum_{j,\ell}\log(g_{i,j}^{\ell}+1)$ 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 $2C_i\log(\sum_{j,\ell} g_{i,j}^{\ell}/C_i+1)$, and because of the preceding remark $(\sum_{j,\ell} g_{i,j}^{\ell}\leq n_i)$, this is smaller than $2C_i\log(n_i/C_i+1)$.

The functions $f_i(x)=2x\log(n_i/x+1)$ 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 $2\rho\log(n_i/\rho+1)$ comparisons between m and an element in array A_i. Summing over i we get the final result, which is $O(\rho\sum_i\log n_i/\rho)$. □

4.2. Randomized Complexity Lower Bound. We now prove that no randomized algorithm can do asymptotically better in $(k, n_1, \ldots, n_k)$. The proof is quite similar to the lower bound of the alternation analysis [Barbay and Kenyon 2002], and differs mostly in Lemma 4.8, which must be adapted to the redundancy.

Lemmas 4.8 and 4.9 are used to prove the alternation lower bound in Theorem 3.3 and to prove the redundancy lower bound in Theorem 4.10.

In Lemma 4.8 we prove an average lower bound on a distribution of instances of alternation and redundancy of at most $\rho=4$ and of intersection size at most 1. We use this result in Lemma 4.9 to define a distribution on instances of alternation and redundancy of at most $\rho\in\{4,4n_1\}$ by combining $p=\theta(\rho)$ subinstances. Applying the Yao-von Neumann principle [Neumann and Morgenstern 1944; Sion 1958; Yao 1977] in Theorem 4.10 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.

Finally, in Lemma 4.11, we prove that any instance of signature $(k, n_1, \ldots, n_k)$ has redundancy ρ of at most $2n_1+1$, so that the redundancy analysis of Theorem 4.10 totally covers all instances for a given signature $(k, n_1, \ldots, n_k)$.

Lemma 4.8. *For any $k\geq 2$, $0<n_1\leq\ldots\leq n_k$, there is a distribution on instances of the intersection problem with signature at most $(k, n_1, \ldots, n_k)$, and with alternation and redundancy at most 4, such that any deterministic algorithm performs at least $(1/4)\sum_{i=2}^{k}\log(2n_i+1)+\sum_{i=2}^{k}1/(2n_i+1)-k+2$ comparisons on average.*

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

Let us draw an index $w\in\{2,\ldots,k\}$ equal to i with probability 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 that $b<d<a<c<e$ (see Figure 4); 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 $A_1[1]$.

Let $x=A_1[1]$ be the first element of the first array. Define x-comparisons to be 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 array does not yield more information than the two comparisons between x and b and between x and d: The positions of elements b and d relative to x permit to deduce their order. Hence, any algorithm performing C comparisons between arbitrary elements can be expressed as an algorithm performing no more than $2C$

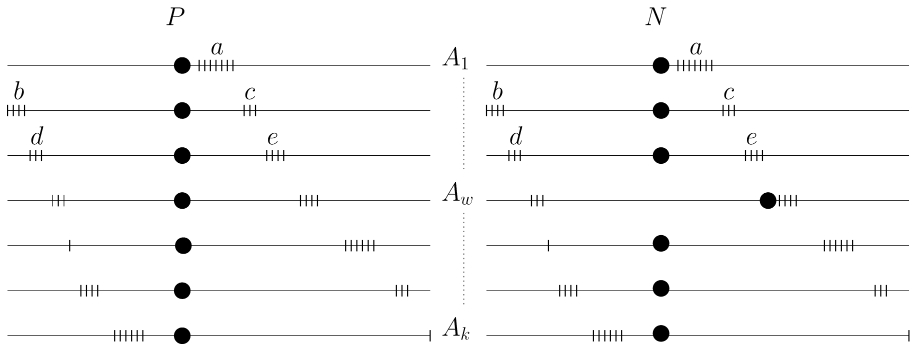

FIG. 4. Distribution on (P, N): Each element of value v is represented by a dot of x-coordinate v, and large dots correspond to the element at position p_i in each array A_i.

x-comparisons, and any lower bound L on the complexity of algorithms using only x-comparisons is an $L/2$ lower bound on the complexity of algorithms using comparisons between arbitrary elements.

The alternation of such instances is at most 4, and the redundancy of such instances is no more than $3+1/(k-1)$, which is less than 4 by the following reasoning.

—The interval $(-\infty, A_1[1])$ is sufficient to certify that no element smaller than x is in the intersection, and stands for a redundancy of at most 1.

—The interval $(A_1[n_1], +\infty,)$ is sufficient to certify 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 stand for a redundancy of at most $1+1/(k-1)$.

The only difference between instances P and N is the relative position of the element $A_w[p_w]$ to the other elements composing the instance, as described in Figure 4. Any algorithm computing the intersection of P has to find the $(k-1)$ positions $\{p_2, \dots, p_k\}$. Any algorithm computing the intersection of N has to find w and the associated position p_w. Any algorithm distinguishing between P and N has to find p_w: We will prove that it needs an average of almost $F/2 = (1/2)\sum_{i=2}^{k} \log_2(2n_i+1)$ x-comparisons to do so on a distribution corresponding to the uniform choice between an instance N and an instance P.

Consider a deterministic algorithm using only x-comparisons to compute the intersection. As the algorithm has not distinguished between P and N until finding w, let X_i denote the number of x-comparisons performed in array A_i for both P or N. Let Y_i denote the number of x-comparisons performed in array A_i for N; and let ξ_i be the indicator variable, which equals 1 exactly if p_i has been determined on instance P. The number of comparisons performed 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 us 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 $E(Y_i\xi_i) \geq \sum_{h=1}^{F_i} \Pr\{Y_i\xi_i \geq h\}$.

These terms can be decomposed using the property that $\Pr\{a \vee b\} \leq \Pr\{a\} + \Pr\{b\}$.

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

The probability $\Pr\{Y_i < h\}$ is bounded by the usual decision tree lower bound: If we consider the binary x-comparisons performed in set A_i, there are at most 2^h leaves at depth less than h. Since the insertion rank of x in A_i is uniformly chosen, these leaves have the same probability and have total probability of at most $\Pr\{Y_i < h\} \leq 2^h/(2n_i + 1) = 2^{h-F_i}$. 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 us fix $p = (p_2, \ldots, p_k)$. There are only $k - 1$ possible choices for w. The algorithm can only differentiate between P and N when it finds w. Let σ denote the order in which these instances are dealt with 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 Eq. (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 that

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

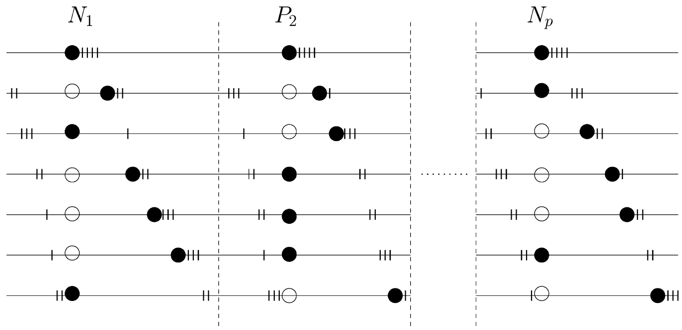

FIG. 5. p elementary instances unified to form a single large instance.

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

LEMMA 4.9. *For any $k \geq 2$, $0 < n_1 \leq \cdots \leq n_k$ 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 of alternation and redundancy at most ρ, 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 subinstances of signature $(k, \lfloor n_1/p \rfloor, \ldots, \lfloor n_k/p \rfloor)$ from the distribution of Lemma 4.8. As $\rho \leq 4n_1$, $p \leq n_1$, and $\lfloor n_1/p \rfloor > 0$, the sizes of all arrays are positive. Let's choose uniformly at random each subinstance I_j between the subinstance P_j whose intersection is a singleton and the subinstance N_j whose intersection is empty. Further, lets form a larger instance I by unifying arrays of same index from each subinstance, such that the elements from two different subinstances never interleave, as in Figure 5.

This defines a distribution on instances of alternation and redundancy of at most ρ (as $4p = 4\lfloor \rho/4 \rfloor \leq \rho$), and of signature at most $(k, n_1, \ldots, n_k)$. Solving this instance implies solving all the p subinstances. Lemma 4.8 gives a lower bound of $(1/4)\sum_{i=2}^{k}\log(2n_i/p+1)+\sum_{i=2}^{k} 1/(2n_i+1)-k+2$ comparisons, on average, for each of the p subproblems, hence a lower bound of

$$(p/4)\sum_{i=2}^{k}\log(2n_i/p+1) + p\left(\sum_{i=2}^{k} 1/(2n_i/p+1) - k+2\right),$$

which is $\Omega(\rho \sum_{i=1}^{k}\log(n_i/\rho))$. $\square$

THEOREM 4.10 (REDUNDANCY LOWER BOUND [BARBAY 2003]). *For any $k \geq 2$, $0 < n_1 \leq \ldots \leq n_k$, and $\rho \in \{4, \ldots, 4n_1\}$, and for any randomized algorithm A_R for the intersection problem, there is an instance of signature at most $(k, n_1, \ldots, n_k)$, and of redundancy at most ρ, such that A_R performs $\Omega(\rho \sum_{i=1}^{k}\log(n_i/\rho))$ comparisons on it, on average.*

PROOF. The proof is identical to that of Theorem 3.3, as the instances generated by the proof are of alternation equal to their redundancy. This is a simple application of Lemma 4.9 and of the Yao-von Neumann principle [Neumann and Morgenstern 1944; Sion 1958; Yao 1977], which we show next.

—Lemma 4.9 gives a distribution for $\rho \in \{4, \ldots, 4n_1\}$ on instances of redundancy of at most ρ; and

—then the Yao-von Neumann principle permits to deduce from this distribution a lower bound on the worst-case complexity of randomized algorithms. □

This analysis is more precise than the lower bound previously presented [Barbay and Kenyon 2002], where the additive term in $-k$ was ignored, although it makes the lower bound trivially negative for large values of the difficulty ρ. Here the additive term is suppressed for $\min_i n_i \geq 128$, and the multiplicative factor between the lower bound and 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 [Bentley and Yao 1976].

One could wonder how the lower bound evolves for redundancy values larger than $4n_1$. The following result shows that no instance with such redundancy can exist.

LEMMA 4.11. *For any $k \geq 2$ and $0<n_1\leq \cdots \leq n_k$, any instance of signature $(k, n_1, \ldots, n_k)$ has redundancy ρ of 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. □

Note that this does not contradict the result from Lemma 4.9, which defines a distribution of instances of redundancy of *at most* $4n_1$.

5. *Comparisons Between the Analyses*

The redundancy analysis is strictly finer than the alternation analysis: Some algorithms optimal for the alternation analysis are not optimal anymore in the redundancy analysis (Theorem 5.1), and any algorithm optimal in the redundancy analysis is optimal in the alternation analysis (Theorem 5.2). So the Rand Intersection algorithm is theoretically better than its deterministic variant in the comparison model, and the redundancy analysis permits a better analysis than the alternation analysis.

THEOREM 5.1. *For any $k \geq 2$, $0<n_1\leq \cdots \leq n_k$, and $\rho \in \{4, \ldots, 4n_1\}$, and for any deterministic algorithm for the intersection problem, there is an instance of signature at most $(k, n_1, \ldots, n_k)$, and of redundancy at most ρ, such that this algorithm performs $\Omega(k\rho \sum_i \log(n_i/k\rho))$ comparisons on it.*

PROOF. The proof uses the same decomposition as that of Theorem 4.10, but uses an adversary argument to obtain a deterministic lower bound. Build $\delta = k\rho/3$ subinstances of signature $(k, \lfloor n_1/\delta \rfloor, \ldots, \lfloor n_k/\delta \rfloor)$ and of redundancy at most 3, such that $x = A_1[1]$ is present in roughly half of the other arrays, as in Figure 6.

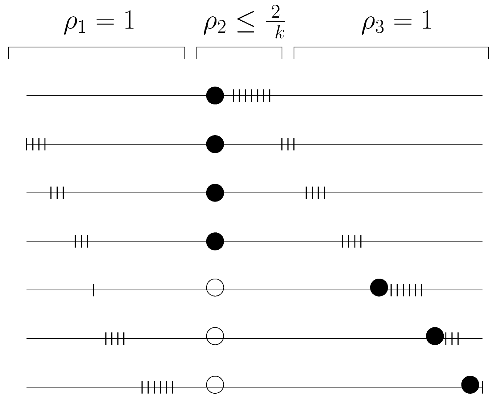

FIG. 6. Element x is present in half of the arrays of the subinstance.

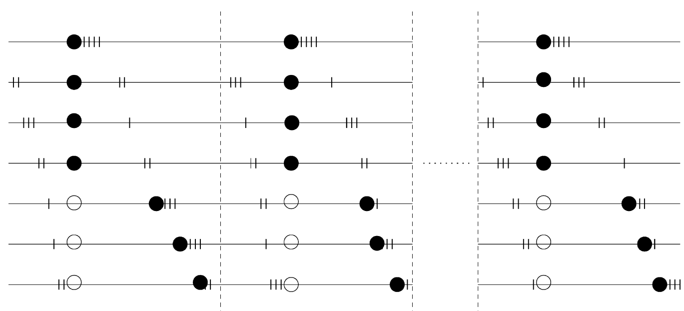

FIG. 7. The adversary performs several strategies in parallel, one for each subinstance.

On each subinstance, an adversary can force any deterministic algorithm to perform a search in each of the arrays containing x, as well as in a single array which does not contain x. Then the deterministic algorithm performs $(1/2)\sum_{i=2}^{k} \log(n_i/\delta)$ comparisons for each subinstance see Figure 7. In total over all subinstances, the adversary can force any deterministic algorithm to perform $(\delta/2)\sum_{i=2}^{k} \log(n_i/\delta)$ comparisons, namely $(k\rho/4)\sum_{i=2}^{k} \log(n_i/k\rho)$, which is $\Omega(k\rho \sum_{i=2}^{k} \log(n_i/k\rho))$. $\square$

As $x \log(n/x)$ is a function increasing with x, $k\rho \sum_i \log(n_i/k\rho)$ is several times larger than the lower bound $\rho \sum_i \log(n_i/\rho)$, hence no deterministic algorithm can be optimal in the redundancy analysis.

THEOREM 5.2. *Any algorithm optimal in the redundancy analysis is optimal in the alternation analysis.*

PROOF. By definition of the redundancy ρ and of the alternation δ of an instance, $\rho \leq \delta$. So if an algorithm performs $O(\rho \sum \log n_i/\rho)$ comparisons, it also performs $O(\delta \sum \log n_i/\delta)$ comparisons. Hence the result, as this is the lower bound in the alternation analysis. $\square$

This proves also that the measure of difficulty of Demaine et al. [2000] is not comparable with the measure of redundancy, as it is not comparable with the measure of alternation [Barbay and Kenyon 2003, Section 2.3]. This means that the two measures are complementary without being redundant in any way, as was so for the alternation. All these measures describe the difficulty of the instance, which can be seen from the following.

—The alternation [Barbay and Kenyon 2003, Section 2.3] describes the number of key blocks of consecutive elements in the instance;
—the *gap cost* [Demaine et al. 2000] describes the repartition of the size of these blocks; and
—the redundancy [Barbay 2003] describes the difficulty of finding each block.

However, only the gap cost and the redundancy matter because the alternation analysis is reduced to the redundancy analysis.

6. *Perspectives*

The t-threshold set and opt-threshold set problems [Barbay and Kenyon 2003] 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 a proper measure is harder to define in this context. As similar techniques are applied to solve queries on semistructured documents [Barbay 2004], the redundancy could be useful in this domain as well, but the definition of the proper measure of difficulty is even more evasive in this context.

Demaine et al. [2001] performed experimental measurements of the performance of various deterministic algorithms for the intersection on their own data, using some queries provided by Google. We performed similar measurements for the deterministic and randomized version of our algorithm, using the same queries and a larger set of data, also provided by Google. The results are quite disappointing, as the randomized version of the algorithm does not perform better than the deterministic one in terms of the number of comparisons or searches, and is much worse in terms of runtime. The fact that the number of comparisons and of searches are roughly the same indicates that most instances of this dataset either have redundancy close to the alternation, because the elements searched are in many of the arrays, or are so easy that both algorithms perform equally well on it. The fact that the runtime is worse is probably linked to the performance of prediction heuristics in the hardware: A deterministic algorithm is easier to predict than a randomized one. It would be interesting to see if these negative results still hold for queries with more keywords and on some datasets, such as those from relational databases, which can exhibit more correlation between keywords.

While we restricted our definition of the intersection problem to sets of arrays and analyzed it in the comparison model, it makes sense to consider other structures for sorted sets, especially in the context of cached or swapped memory, or of succinct encodings of dictionaries. *Hierarchical memory* [Frigo et al. 1999] seems promising for this kind of application, and Bender et al. [2002] proposed a data structure and a *cache-oblivious* algorithm to perform unbounded searches (implemented as finger searches). Our algorithm can easily be adapted to this model, to perform $O(\rho \sum(\log_B(n_i/\rho) + \log^*(n_i/\rho)))$ I/O transfers at the level of cache-size B.

In most of the intersection algorithms, the interactions with each set are limited to accessing an element, given its rank (*select* operator), and searching for the insertion rank of an element in it (*rank* operator): These algorithms can be used with any set implementation which provides those operators. For instance, using sorted arrays such as in this article, the *select* operator takes constant time while the *rank* operator takes time logarithmic in the size of set. While the results of this article are optimal in the comparison model, they are not necessarily optimal in more general models: The computational complexity of the search operators constitute a tradeoff with the size of encoding of the set. For instance, consider a set of n elements from a universe of size m: Raman et al. [2002] proposed a succinct encoding of fully indexable dictionaries using $\log \binom{m}{n} + o(m)$ bits to provide *select* and *rank* operators in constant time. On the other side of the time/space tradeoff, Beame and Fich [2002] proposed a more compact encoding, using $O(n)$ words of $\log m$ bits to provide *select* and *rank* operators in time $O(\sqrt{\log n / \log\log n})$. Encoding the sets using any of these schemas would tremendously improve the computational complexity of the intersection at a small cost in space, which could result in much faster search engines.

ACKNOWLEDGMENTS. This article covers work performed in many places, including the University of Paris-Sud, Orsay, France; the University of British Columbia, Vancouver, Canada; and the University of Waterloo, Waterloo, Canada. The authors wish to thank all the institutions involved, as well as Joel Friedman and the reviewers, who provided many useful comments.

REFERENCES

BAEZA-YATES, R. A. 2004. A fast set intersection algorithm for sorted sequences. In *Proceedings of the Annual Symposium on Combinatorial Pattern Matching (CPM)*, S. C. Sahinalp et al., Eds. Lecture Notes in Computer Science, vol. 3109. Springer, 400–408.

BARBAY, J. 2004. Index-Trees for descendant tree queries in the comparison model. Tech. Rep. TR-2004-11, University of British Columbia. July.

BARBAY, J. 2003. Optimality of randomized algorithms for the intersection problem. In *Proceedings of the 2nd International Symposium on Stochastic Algorithms, Foundations and Applications (SAGA)*, A. Albrecht, Ed. Lecture Notes in Computer Science, vol. 2827. Springer, Heidelberg, 26–38. 3-540-20103-3.

BARBAY, J., AND KENYON, C. 2003. Deterministic algorithm for the t-threshold set problem. In *Proceedings of the 14th Annual International Symposium on Algorithms and Computation (ISAAC)*, H. O. Toshihide Ibaraki, Noki Katoh, Eds. Lecture Notes in Computer Science. Springer, 575–584.

BARBAY, J., AND KENYON, C. 2002. Adaptive intersection and t-threshold problems. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*. ACM, 390–399.

BEAME, P., AND FICH, F. E. 2002. Optimal bounds for the predecessor problem and related problems. *J. Comput. Syst. Sci. 65*, 1, 38–72.

BENDER, M. A., COLE, R., AND RAMAN, R. 2002. Exponential structures for efficient cache-oblivious algorithms. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming (ICALP)*. Springer, 195–207.

BENTLEY, J. L., AND YAO, A. C.-C. 1976. An almost optimal algorithm for unbounded searching. *Inf. Proc. Lett. 5*, 3, 82–87.

CHAUDHURI, S. 1998. An overview of query optimization in relational systems. In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 34–43.

CHRISTEN, C. 1978. Improving the bound on optimal merging. In *Proceedings of 19th Annual Symposium on Foundations of Computer Science (FOCS)*. 259–266.

DE LA VEGA, W. F., FRIEZE, A. M., AND SANTHA, M. 1998. Average-Case analysis of the merging algorithm of Hwang and Lin. *Algorithmica 22*, 4, 483–489.

DE LA VEGA, W. F., KANNAN, S., AND SANTHA, M. 1993. Two probabilistic results on merging. *SIAM J. Comput. 22*, 2, 261–271.

DEMAINE, E. D., LÓPEZ-ORTIZ, A., AND MUNRO, J. I. 2001. 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. Springer, 5–6.

DEMAINE, E. D., LÓPEZ-ORTIZ, A., AND MUNRO, J. I. 2000. Adaptive set intersections, unions, and differences. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 743–752.

FRIGO, M., LEISERSON, C. E., PROKOP, H., AND RAMACHANDRAN, S. 1999. Cache-Oblivious algorithms. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 285.

HWANG, F. K., AND LIN, S. 1972. A simple algorithm for merging two disjoint linearly ordered sets. *SIAM J. Comput. 1*, 1, 31–39.

HWANG, F. K., AND LIN, S. 1971. Optimal merging of 2 elements with n elements. *Acta Inf.* 145–158.

MANACHER, G. K. 1979. Significant improvements to the Hwang-Ling merging algorithm. *J. ACM 26*, 3, 434–440.

NEUMANN, J. V., AND MORGENSTERN, O. 1944. *Theory of Games and Economic Behavior*, 1st ed. Princeton University Press.

RAMAN, R., RAMAN, V., AND RAO, S. S. 2002. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 233–242.

SION, M. 1958. On general minimax theorems. *Pacific J. Math.* 171–176.

WITTEN, I. H., MOFFAT, A., AND BELL, T. C. 1994. *Managing Gigabytes*. VanNostrand Reinhold, New York.

YAO, A. C. 1977. Probabilistic computations: Toward a unified measure of complexity. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS)*, 222–227.