ADAPTIVE SET INTERSECTIONS

by

Erik D. Demaine
Alejandro Lopez-Ortiz
J. Ian Munro

TR98-120, July 1998

Faculty of Computer Science
University of New Brunswick
Fredericton, N.B. E3B 5A3
Canada

Phone:	(506) 453-4566
Fax:	(506) 453-3566
E-mail:	fcs@unb.ca
www:	http://www.cs.unb.ca

Adaptive Set Intersections

Erik D. Demaine* Alejandro López-Ortiz† J. Ian Munro*

Abstract

We introduce a new data structure, corresponding to a fully persistent SET ADT, motivated by the efficient evaluation of boolean queries in text retrieval systems. We then focus on a key operation required by the ADT, namely the problem of efficiently evaluating an intersection expression over a static collection of integer sets. We present an optimal nondeterministic algorithm for computing the shortest proof of nonintersection for disjoint sets in the comparison model. Then we propose an adaptive algorithm for computing such proofs in deterministic fashion. The number of comparisons performed by this algorithm falls within a small multiplicative factor of the theoretical best. This algorithm is generalized to compute the intersection of nondisjoint collections of sets with the same competitive factor.

1 Introduction and Motivation

In this paper we consider the problem of evaluating a set expression over a given static domain composed of a collection of sets of integers. This problem is motivated by queries to a database such as a search engine. On a text-retrieval system, a standard multiterm query is composed of strings of *terms* connected via the boolean operators `and`, `or` and `not`, for example, `"computer" and "science"`. Each term represents the set of all documents containing that term. The boolean operators represent the corresponding operations over these sets. That is, a sequence of `and` terms is mapped to the intersection of the sets implicitly defined by those terms. Similarly a sequence of `or` operations maps to the union of document sets.

During the last decade the average size of a text collection has grown exponentially. For example in the mid 1980s the Oxford English Dictionary was considered a large collection at 300MB of text. Currently, a large collection, such as the Lexis/Nexis database, holds over one terabyte of text. Another common example is the World Wide Web, estimated to contain around 100 gigabytes of text.[1] For this case, a single query term of a randomly selected word

*Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada, email: {`eddemaine,imunro`}`@uwaterloo.ca`

†Faculty of Computer Science, University of New Brunswick, P. O. Box 4400, Fredericton, N. B. E3B 5A3, Canada, email: `alopez-o@unb.ca`

[1]All figures are estimates, and rapidly increasing. At time of printing SIGIR Forum estimates Lexis/Nexis at over seven terabytes of text, and the web at two terabytes [12]. Other sources report more conservative growth.

from an English dictionary matches anywhere between a few hundred to several millions of documents. In fact we obtained an average of 37,500 different documents matching a single-term query, as described above, on the AltaVista search engine. However, the actual average result size is larger, as users search using a restricted subset of English (as well as other languages). If we restrict ourselves to a dictionary made from the logs of actual queries to a web search engine we see that the average query term appears in the order of 1,000,000 documents.

This means that a **single** query term over a large collection gives result sets of impractical size. Currently a user must type several terms to narrow the result set to a usable size, as query logs confirm. Thus the efficient evaluation of an arbitrary boolean query requires fast algorithms for the evaluation of arbitrary set expressions over sets of millions of elements in size. If the query is well constructed, the matching set is then likely to consist of a small number of documents as compared to the sizes of the sets being intersected.

While some "complex" instances will force one to look at all the documents involved, it is sensible to require "simple" instances to be solved more efficiently. For example, computing the intersection of two ordered sets of size n that interleave perfectly requires $\Omega(n)$ operations (essentially the time required to merge them and determine that no two consecutive values are the same). On the other hand, if all the values in one set are strictly less than all the values in the other set, we should be able to determine that the intersection is empty in constant time.

Indeed the latter case is particularly useful when one of the query terms does not occur as often as the others. For example, consider the query `"traincrash" and "Summerview"`. The first term is likely to result in evenly distributed occurrences over the index as there are train crashes all over the world. In contrast the peaceful town of Summerview appears infrequently on the news. An adaptive algorithm exploits the structure of the sets being intersected, producing faster results in those cases where the answer can be computed more efficiently.

For simplicity of exposition we will first analyze and propose algorithms for the case in which the answer set is empty and then generalize the algorithms for nonempty intersections. A natural measure of the hardness of the input for an intersection operation, under the comparison model, is the information-theoretic lower bound, namely, the size of the optimal encoding of a proof showing that the sets do not intersect.

Usually, operations are performed over a predetermined collection of sets, namely the word index. Often these sets are represented as B-trees or variations thereof. It is thus important that the algorithms proposed support these structures. Set operations are an extension of the dictionary abstract data type (ADT). Moreover, the result of a set operation is sometimes used for further processing (e.g., ranking or finding similar documents). We need then to be able to use the result of a set operation as input to another operation of the same type.[2]

To summarize, the requirements call for a SET ADT over a collection of integer sets. The operations allowed are UNION(A, B), INTERSECTION(A, B), and DIFFERENCE(A, B), where A, B are two integer sets, not necessarily disjoint. The SET ADT is fully persistent in that the

[2]An example of this concept is the UNION-FIND ADT, in which the set resulting from a union can then be unioned again with other preexisting sets in an efficient manner.

resulting set can be used in any future set operation. Operations produce "self-replicating" structures which allow the evaluation of set expressions with parse trees of arbitrary depths.

It is readily apparent that the proposed SET ADT, if efficiently implemented, satisfies the requirements of a multiterm text retrieval system. In this paper we propose efficient adaptive algorithms for the set-intersection problem and postpone the union and difference operations for discussion in a forthcoming paper.

The rest of this paper is outlined as follows. In Section 2, we introduce the problem. In Sections 3 and 4 we consider optimal nondeterministic algorithms for finding a proof of empty intersection. In Section 5 we present matching upper and lower bounds on deterministic algorithms for computing the intersection of a collection of sets. Lastly in Section 6 we do average case analysis for random instances.

2 Problem Statement

Let us give a formal statement of the problem we are concerned with. Suppose we are given a collection of n sets $A^1, \ldots, A^n$, where

$$A^s = \left\{a_1^s, \ldots, a_{n_s}^s\right\}.$$

Each set is in sorted order; that is, $a_i^s < a_j^s$ for all $i < j$. We are interested in computing the intersection of the sets, that is, $A^1 \cap \cdots \cap A^n$, in an optimal factor away from the information-theoretic lower bound on the difficulty of the problem.

We will often consider the equivalent problem of *proving* that the sets are disjoint. The sets are called *disjoint* if $A^1 \cap \cdots \cap A^n = \emptyset$. Note that this is different from the sets being *pairwise disjoint*, that is, $A^s \cap A^t = \emptyset$ for all s, t. In particular, any proof that two of the sets are disjoint is a proof that all of the sets are disjoint.

A *proof* is a set of formal comparisons (all strict inequalities), written in parentheses to denote formality. For example,

$$\left\{(a_2^5 < a_9^3), (a_3^3 < a_1^1)\right\}$$

is a 2-comparison proof of nothing in particular. The *length* of a proof P, denoted $|P|$, is simply the number of comparisons.

A *proof of nonintersection* is a proof such that if the stated comparisons were true, then it must be that the sets are disjoint. In other words, there are no choices of $i_1, \ldots, i_n$ such that $a_{i_1}^1, \ldots, a_{i_n}^n$ are pairwise incomparable, where possible comparisons are defined by those in the proof plus the assumption that every set is in sorted order.

3 General Structure of Proofs of Nonintersection

A natural question at this point is, what do proofs of nonintersection look like? Let us define the concept of elimination in a proof P, which will essentially represent the elements that have been proved to be not in the intersection $A^1 \cap \cdots \cap A^n$. First, call an element a_i^s *minimal* [*maximal*] if for every $j < i$ [$j > i$], a_j^s is eliminated. Now define a_i^s to be *eliminated* if either

1. there is a $j \geq i$ such that $(a^s_j < b) \in P$ where b is minimal, or
2. there is a $j \leq i$ such that $(a^s_j > b) \in P$ where b is maximal.

That is, if $(a^s_j < b) \in P$ and b is minimal, then a^s_i is eliminated for all $i < j$. Similarly, if $(a^s_j > b) \in P$ and b is maximal, then a^s_i is eliminated for all $i > j$. Furthermore, this is a complete definition of elimination.

The motivation for this definition is the following lemma.

Lemma 1 *If a^s_j is eliminated, then it cannot be in $A_1 \cap \cdots \cap A_n$. In particular, if a proof eliminates an entire set, then it is a proof of nonintersection.*

Proof: Consider some sequentialization of the "elimination steps," that is, eliminating a^s_i for all $i < j$ because $(a^s_j < a^t_k) \in P$ and a^t_k is minimal, or the symmetric case. Assume that up to some point all eliminated elements are not in the intersection, and consider the next elimination step. Let a^s_i be one of the eliminated elements; we will consider all possible values it could have. If $a^t_{j-1} < a^s_i < a^t_j$ for some $j \leq k$, then a^s_i is not in A^t, so it cannot be in the intersection. Otherwise, $a^s_i = a^t_j$ for some $j < k$. But by the induction hypothesis, a^t_j (and hence a^s_i) is not in the intersection. Therefore, in all cases a^s_i cannot be in the intersection, as desired. □

Indeed, the converse of the second part of this lemma holds. First, we need to show that elements are eliminated only on either end of a set.

Lemma 2 *If a set has uneliminated elements, they are all consecutive.*

Proof: Suppose $i < j < k$ is such that a^s_i and a^s_k are uneliminated, but a^s_j is eliminated. Assume by symmetry that there is an $l \geq j$ such that $(a^s_l < b) \in P$ for some b that is minimal. Now $l \geq j > i$, so a^s_i is also eliminated, a contradiction. □

Now we can show the importance of the idea of elimination.

Theorem 1 *Any proof of nonintersection eliminates an entire set, that is, at least one of the sets has every element eliminated.*

Proof: Suppose for contradiction that every set has at least one uneliminated element. By Lemma 2, the uneliminated elements in each set are consecutive. Let u_s be the index of the smallest uneliminated element $a^s_{u_s}$ in set A^s. We claim that $a^1_{u_1}, \ldots, a^n_{u_n}$ are pairwise incomparible, and hence P is not a proof of nonintersection.

Suppose P implies that $a^s_{u_s} < a^t_{u_t}$. In other words, there is some path of comparisons

$$(a^s_{j_0} < a^{s_1}_{i_1}),\ (a^{s_1}_{j_1} < a^{s_2}_{i_2}),\ \ldots,\ (a^{s_k}_{j_k} < a^t_{i_{k+1}}),\ \in P,$$

$$\text{where } \ u_s \leq j_0,\ i_1 \leq j_1,\ \ldots,\ i_k \leq j_k,\ i_{k+1} \leq u_t.$$

Initially, $a^t_{u_t}$ and hence $a^t_{i_k}$ is minimal. Because of the comparison $(a^{s_k}_{j_k} < a^t_{i_{k+1}})$, $a^{s_k}_{j_k}$ and hence $a^{s_k}_{i_k}$ is eliminated and minimal. Assume in general that $a^{s_m}_{j_m}$ is eliminated and minimal. Because of the comparison $(a^{s_{m-1}}_{j_{m-1}} < a^{s_m}_{i_m})$, $a^{s_{m-1}}_{j_{m-1}}$ and hence $a^{s_{m-1}}_{i_{m-1}}$ is eliminated and minimal. By induction, $a^s_{j_0}$ and hence $a^s_{u_s}$ is eliminated, a contradiction. □

4 Shortest Proofs and Nondeterministic Algorithms

So far, proofs have had no order and hence the effect of a particular comparison, in terms of elimination, is unclear. In an ordered proof, we say that the kth comparison *eliminates* an element if the subproof with the first k comparisons has this element eliminated; it *newly eliminates* an element if in addition just the first $k-1$ comparisons do not have this element eliminated. A *basic comparison* is a comparison $(a_i^s < a_j^t)$ that newly eliminates elements just in A^s or just in A^t. An *incremental ordering* of a proof P is an ordering such that the comparisons are basic up to but not including the first comparison that eliminates an entire set.

Consider the following nondeterministic algorithm for ordering a proof P. First, choose any comparison left that newly eliminates some elements. Repeat until there are no such comparisons. Second, choose the remaining comparisons in any order.

Lemma 3 *Any execution of this nondeterministic algorithm results in an incremental ordering of P.*

Proof: Suppose we add the comparison $(a_j^s < a_k^t)$ during the first phase. For this to cause any elimination, it must newly eliminate elements in A^s or A^t. For elimination to occur in other sets, it must be because of new elements becoming minimal or maximal. If both a_k^t was minimal and a_j^s was maximal, then this comparison entirely eliminates set A^s and A^t, so we do not have to consider it or future comparisons. Hence, we can assume that either a_k^t was minimal or a_j^s was maximal, but not both.

Without loss of generality, consider the case where a_k^t was minimal, which causes a_i^s to newly become minimal for all $i \leq j+1$. It is also possible that elements in A^s newly become maximal, but this only occurs when a_i^s was eliminated for all $i \geq j$, which means that the comparison entirely eliminates the set A^s, so we do not have to consider this or future comparisons. For the new minimal elements to cause more new eliminations, there must have already been a comparison $(a_l^u < a_m^s)$ for some $m \leq j+1$. This comparison was chosen in the first phase (because it was chosen earlier), and hence either a_m^s was already minimal or a_l^u was already maximal. In the former case, a_i^u was already eliminated for all $i \leq l$, and hence there are no new eliminations. In the latter case, a_i^s was already eliminated for all $i \geq m$ (and hence for all $i \geq j+1$), which means that the set A^s just became entirely eliminated, so we do not have to consider this or future comparisons.

Next consider a comparison chosen in the second phase. Assume by induction that there are no remaining comparisons that cause new elimination. Because this comparison did not newly eliminate any elements and hence did not make any new elements minimal or maximal, it cannot make any future comparisons cause elimination. Therefore, no comparisons in the second phase cause any elimination. □

Let $P = [c_1, \ldots, c_n, \ldots, c_m]$ be an incremental proof, where c_n is the first comparison eliminating an entire set. The comparisons $c_{n+1}, \ldots, c_m$ have no purpose in terms of proving nonintersection, so we call them *useless*. Consider a useful comparison $c_k = (a_i^s < a_j^t)$. We call c_k *low* if a_j^t is minimal, and *high* if a_i^s is maximal, at the time when c_k is added to the proof. Because c_k newly eliminates elements, it must be either low or high. By definition of incremental, c_k is not both low and high for $k < n$, so we call it *low-only* or *high-only.* Also, because c_n entirely eliminates the two sets it compares between, it must be both low and high, or *low-high.*

First let us show that the low and high comparisons are effectively independent.

Lemma 4 *Let P be an incremental proof of nonintersection. Let Q be the reordering of P that puts all the low-only comparisons before the high comparisons, maintaining the relative order between two low comparisons or two high comparisons. Then Q is also an incremental ordering of P.*

Proof: Applying bubble sort, we can perform this permutation by a sequence of swaps of adjacent low-only and high-only comparisons. Consider a low-only comparison $c = (a_i^s < a_j^t)$ and a high-only comparison $d = (a_k^u < a_l^v)$. Then a_j^t is minimal and a_k^u is maximal; c eliminates a_m^s for $m \leq i$ and d eliminates a_m^v for $m \geq l$. Now consider interchanging the order of c and d, that is, putting c before d when c used to be after d. This will only affect c's elimination if a_j^t is no longer minimal, i.e., if d newly eliminated a_m^t for some $m < j$. Choosing m to be the smallest possible, this means that $a_m^t = a_l^v$. But because d makes $a_m^t = a_l^v$ minimal, a_m^t must have already been minimal, which means that d was low, a contradiction. Similarly, d's elimination is only affected if c was also high, a contradiction. $\square$

Now we claim that we can ignore high-only comparisons.

Lemma 5 *If P is an incremental proof of nonintersection, then there exists another incremental ordering of P such that all useful comparisons are low.*

Proof: First reorder P according to Lemma 4 so that the low-only comparisons come before all high comparisons. We shall now reverse the order of the high comparisons. Note that this does not affect the low comparisons. We claim that it turns the high comparisons into low comparisons, and hence by Lemma 3 keeps the ordering incremental. We shall prove this by executing a sequence of permutations, repeatedly putting the last high comparison immediately before the first high comparison.

Consider the last high comparison $(a_i^s < a_j^t)$. It is also low, i.e. a_j^t is minimal, so it eliminates a_k^s for $k \geq i$. We need to show that the comparison would also be low if we moved it to immediately before the first high comparison, i.e., that a_j^t was already minimal before all the high comparisons. If this were not the case, there must be a high-only comparison that newly eliminated a_j^t. Since it is high-only, it must be of the form $(b < a_i^t)$, where b is maximal and a_i^t was already minimal from the low-only comparisons, which completely eliminates A^t. Hence, the first high comparison is also low, the desired result. $\square$

Finally, we argue about the inflexibility in the choice of the comparisons.

Theorem 2 *The following greedy algorithm generates a shortest proof of nonintersection. After choosing $c_1, \ldots, c_{i-1}$, let $c_i = (a_i^s < b)$ where b is the largest minimal element in all the sets, and a_i^s is an element less than b such that a_{i+1}^s is maximized. Here $a_{n_i+1}^s$ is defined to be infinity.*

Proof: Consider a shortest incremental proof P. It cannnot have any useless comparisons; otherwise we could simply remove them for a shorter proof. Hence assume by Lemma 5 that all comparisons are low.

Consider $c = (a_i^s < a_j^t) \in P$. Because c is low, a_j^t is minimal. We will call a_i^s the *eliminatee* and a_j^t the *eliminator.* We claim that the choice of a_j^t is essentially fixed. If a_j^t is not the largest minimal element e, we can replace c by $(a_i^s < e)$; because $e > a_j^t$, it will eliminate the same elements, so all the future comparisons will work as before. Hence, assume that the eliminator of each comparison in P is chosen to be the largest minimal element at the time. The choice of i is also fixed: we can replace i with the largest value such that c is actually true. This will eliminate (in particular) all the elements eliminated before, so all the future comparisons will work as before.

All that remains is the choice of s, the set containing the eliminatee. Call a comparison c *good* if the eliminatee a_i^s is chosen such that a_{i+1}^s is maximum. Suppose that there is a comparison $c = (a_i^s < b) \in P$ that is not good. Let $a_{i'}^{s'}$ be such that $c' = (a_{i'}^{s'} < b)$ is good. Let us consider replacing c by c'. If c eliminated an entire set, then c was already good, a contradiction. Otherwise, the only effect on future comparisons is that a_{i+1}^s can no longer be an eliminator. Consider a comparison d that uses a_{i+1}^s as a eliminator. Then we can modify d to use $a_{i'+1}^{s'}$ as its eliminator; this will cause the same elimination since $a_{i'+1}^{s'} > a_{i+1}^s$.

We can keep perfoming such modifications to c and corresponding d's. Each one causes c's eliminator to strictly increase, and never decreases the value of other eliminators. Since there are only a finite number of possible values for each eliminator, we can only perform these modifications a finite number of times. When we are done, all comparisons will be good. That is, we will have a proof generated by the algorithm, of optimal length $|P|$. □

Note that the optimal proof can be obtained as well by considering the complements of the sets being intersected. Each complement consists of a collection of open intervals. The intersection of the given sets is empty if and only if there exists a collection of intervals covering the entire real line. It is possible to show that such a cover corresponds to a proof of nonintersection, and indeed the smallest-cardinality cover corresponds to a shortest proof of nonintersection. As expected, such collection can be obtained using a greedy algorithm similar to Theorem 2.

5 Optimal Proofs and Deterministic Algorithms

Theorem 2 gives us a simple greedy algorithm characterizing shortest proofs. The attractiveness of the algorithm is that it is nondeterministically optimal, running in $|P|$ time. However, the algorithm is not useful on a deterministic machine, because choosing to eliminate the best set and location within that set will take more than a unit of time. The rest of this paper is about how to actually find proofs of nonintersection in optimal (deterministic) time.

The key to this problem is the definition of "optimal." We need a measure of the difficulty of an instance, relative to which we can then evaluate algorithms and lower bounds. The natural measure of difficulty in the comparison model is the information-theoretic lower bound, that is, the size of the optimal binary encoding of a proof of nonintersection.

How do we best encode a proof? Call an element *compared* if it occurs in one of the proof's comparisons. Because compared elements can be arbitrarily spaced out in each set, it is natural to encode the size of the gaps (i.e., the differences in index) between compared elements, which costs $\lg g$ for each gap of size g, where $\lg g = \lceil \log_2(1+g) \rceil$. There are two details left out of this idea. First, by appropriately switching between specifying gaps from the low and high sides, we can avoid encoding the largest gap in each set. Second, we need to specify the pairing between compared elements that forms the proof.

Here is an encoding that fills in both of these details. Take any incremental ordering of the proof P (which exists by Lemma 3), and let $c = (a_i^s < a_j^t)$ be the first comparison. Assume that c is low; the high case is symmetric. First encode s and t using $\lceil \log_2 n \rceil$ bits each. Encode i by specifying the smallest gap g to an already compared element in A^s (using essentially $\lg g$ bits), along with whether that gap is from the low or high side (using one bit). Encode j by specifying that c is low (using one bit). The cost of encoding c in this way is thus

$$2\lceil \log_2 n \rceil + \lg g + 2.$$

By induction, we obtain a formula for the cost for encoding an entire proof P. We will define the *cost* of P, denoted $c(P)$, to be slightly different and easier to work with; it will underestimate the "true" formula by at most a factor of two. Let $g_0^s, \ldots, g_{p_s}^s$ denote the gaps in A^s for P, including the "end gaps" before the first element and after the last compared element in A^s. Then we define

$$c(P) = |P| \lg n + g(P), \tag{1}$$

where $g(P)$ is the *gap cost* of P, defined by

$$g(P) = \sum_{s=1}^{n} \left(\sum_{i=0}^{p_s} \lg g_i^s - \max_{0 \le i \le p_s} \lg g_i^s \right). \tag{2}$$

Indeed, the described encoding of a proof is optimal in the following sense.

Theorem 3 *Let the number n of sets and the sizes $n_1, \ldots, n_n$ of these sets be fixed. Given any proof P of nonintersection for such an instance, there are*

$$\Omega\left(\max\left\{2^{|P| \lg n},\ 2^{g(P)-|P|}\right\}\right) = 2^{\Omega(c(P))}$$

proofs of nonintersection for other such instances, each of whose cost is at most $c(P)$. In other words, if we fix any language for encoding all proofs of nonintersection, almost every proof P of nonintersection requires $\Omega(c(P))$ bits to be encoded in this language.

Proof: Note that the proofs of nonintersection we are seeking can be for different instances, because we are concerned with encoding a general proof of nonintersection, without knowing the instance. However, n and the n_i's cannot vary, because we presume they are known by the message decoder.

Also note that the factor p in the proof is because of the use of lg in the the definition of cost of a proof, which presumes that any information costs at least one bit, and that partial amounts of information consume a whole bit.

First let us show a lower bound of $\Omega(2^{|P| \lg n})$ proofs of nonintersection. Let p be any positive integer, and let N be the total number of elements. Note that the message decoder can remove redundant comparisons, so we effectively encode a proof with no duplicate comparisons, implying that $p < N$. Pick two sets A^s and A^t with $s < t$, and define the first comparison to be $(a^s_{n_s} < a^t_1)$. For each $2 \le k \le p$, pick two arbitrary elements a^s_i and a^t_j with $s < t$ that were not previously compared, and define the kth comparison to be $(a^s_i < a^t_j)$. This results in a proof of nonintersection for any instance satisfying $a^s_i < a^t_j$ whenever $s < t$. There are at least n^2 choices per comparison, so we have found $n^{2p} = 2^{2p \log_2 n}$ proofs of nonintersection with length p, showing the desired bound.

Now we will show a lower bound of $\Omega(2^{g(P)-|P|})$ proofs of nonintersection. Let P be the given proof in the theorem statement. First assume that $g^s_i > 1$ for all i and s. This means that no two consecutive elements in a set are used in the proof.

From this proof P, we construct several other proofs (for different instances) as follows. We decrease every gap, except the largest gap in each set, to any amount less than or equal to the gap in P. The pairing between elements stays the same; we simply move the compared elements. To compensate for these shrinking gaps, the largest gap in each set grows, and hence remains largest. These modified gap sizes induce moved positions of the compared elements. More precisely:

- For $s = 1, \ldots, n$:
  1. Let
  $$a^s_{i_1} \le \cdots \le a^s_{i_m} \le a^s_{i_{m+1}} \le \cdots \le a^s_{i_{p_s}}$$
  denote the elements in A^s that are used in P, where $a^s_{i_m}$ and $a^s_{i_{m+1}}$ bound the largest gap $i_{m+1} - i_m$ in P.
  2. Let $i_0 = 0$ and $i_{p_s+1} = n_s + 1$, so that the gap g^s_k is simply $i_{k+1} - i_k$, even at the ends.
  3. Define $j_0 = i_0$ and $j_{p_s+1} = i_{p_s+1}$.
  4. For $k = 1, \ldots, m$:
     - Replace $a^s_{i_k}$ by $a^s_{j_k}$ for any chosen j_k such that $1 \le j_k - j_{k-1} \le i_k - i_{k-1}$.
  5. For $k = p_s, \ldots, m+1$:
     - Replace $a^s_{i_k}$ by $a^s_{j_k}$ for any chosen j_k such that $1 \le j_{k+1} - j_k \le i_{k+1} - i_k$.

Because we only decrease gap sizes, except for the largest gap in each set which does not affect $g(P)$, the gap cost of any constructed proof is at most the gap cost of P. Furthermore, the length of the proof and the number of sets do not change, so the $p \lg n$ term in Equation (1) does not change; hence, the total cost of any constructed proof is at most the cost of P.

Now at each execution of Step 4 or 5, we have g^s_i choices for the value of j_k. Therefore, the number of proofs constructed using this technique is

$$\prod_{s=1}^{n} \prod_{\substack{0 \le i \le p_s \\ i \ne m_s}} g^s_i,$$

which is $2^{g(P)}$, i.e., the exponentiation of Equation (2).

The case where there are gap costs of 1 follows using the same technique, by noticing that their contribution to the $\Omega(2^{g-p})$ is zero, because the gap cost cancels out with p. □

It now makes sense to talk about *optimal* proofs of nonintersection, that is, proofs of nonintersection with minimum cost. We call this minimum cost the *difficulty* $\mathcal{D}$ of the instance, because it is an information-theoretic lower bound on the running time. It turns out that another important measure on proofs is the gap cost; we will denote the minimum gap cost by $\mathcal{G}$. Note that it is possible for this gap cost to only be realized by suboptimal proofs, that is, ones with cost higher than $\mathcal{D}$.

The next two sections prove matching lower and upper bounds on the competitive ratio of the time complexity to the difficulty. The bound is $\Theta(n\mathcal{G}/\mathcal{D})$, which is somewhat less than $\Theta(n)$.

5.1 Lower Bound

In this section, we prove the following theorem.

Theorem 4 *Given positive integers n, p, and g ($p \leq g$), and given an algorithm for finding proofs of nonintersection, there is a collection of n sets having a p-comparison proof of nonintersection with cost $O(p \lg n + g)$, such that every proof of nonintersection has gap cost $\Omega(g)$, and the algorithm takes $\Omega(ng)$ time on this input. In other words, the competitive ratio is $\Omega(ng/(p \lg n + g))$ in the worst case.*

The basic idea is to construct a parameterized class of instances, and have an adversary pick a bad instance for the algorithm. Let $\ell_1, \ldots, \ell_p$ be positive integers summing to g, such that each is either $\lfloor g/p \rfloor$ or $\lceil g/p \rceil$. An ℓ_i will represent the lg of a gap in the proof of nonintersection.

First let us describe the parameters for an instance. Pick $p+1$ "magic" values $m_0 < \cdots < m_p$. Pick a sequence $s_0, \ldots, s_p$ of set numbers such that $s_{i-1} \neq s_i$ for all $1 \leq i \leq p$. Furthermore, each $s \in \{1, \ldots, n\}$ must occur at least every $2n$ elements in the sequence. One way to do this is to concatenate several permutations of $\{1, \ldots, n\}$, chosen randomly such that the first value of one permutation is different from the last value of the previous permutation. Finally, pick integers $k_1^s, \ldots, k_p^s$ such that $\lg k_i^s = \ell_i$, except for $k_i^{s_{i-1}}$ which is defined to be zero.

Then we construct an input as follows (see Figure 1). Each magic value m_i occurs in every set except A^{s_i}; we will denote the occurrence of m_i in A^s by m_i^s. In every set A^s, there are precisely k_i^s elements strictly between m_{i-1} and m_i. In particular, $A^{s_{i-1}}$ has no elements strictly between m_{i-1} and m_i; indeed, it also has no elements equal to m_{i-1}.

Next let us describe the proof of nonintersection. Note that there are no elements before $m_1^{s_0}$, and hence it is minimal. Suppose in general that $m_i^{s_{i-1}}$ is minimal. Then we can use it to eliminate all elements less than m_i in A^{s_i}. But there are no elements between m_i (inclusive) and m_{i+1} (exclusive) in A^{s_i}, so this elimination makes $m_{i+1}^{s_i}$ minimal. This continues by induction until we find that "m_{p+1}^p" is minimal, that is, A^p is entirely eliminated.

This gives a p-comparison proof of nonintersection, but what is its cost?

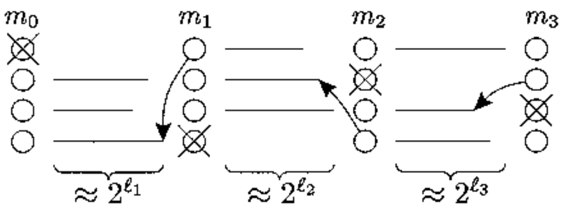

Figure 1: *Illustration of lower-bound construction. Circles show magic elements, and crossed-out circles indicate missing magic values. Arrows indicate comparisons that form a proof of nonintersection.*

Lemma 6 *The described proof has cost $O(p \lg n + g)$.*

Proof: Consider the elimination of all elements less than m_i in A^{s_i}. By construction of the sequence $s_1, \ldots, s_p$, either $i \leq 2n$ or there exists $i - 2n \leq j < i$ such that $s_j = s_i$. In other words, this comparison eliminates at most the last $2n$ gaps. Each gap between magic elements has a lg of $\ell_i \leq g/p+1$, and thus is of size at most $2^{g/p+1} - 1$; if we count the magic element too, the -1 disappears. Hence, the number of eliminated elements is at most

$$2n2^{g/p+1} = n2^{g/p+2}.$$

Now there are two gaps created by this comparison, one of which is of size one, and the other of which is of the size above. Hence, the piece of the gap cost attributed to this comparison is at most

$$1 + \lg n2^{g/p+2} \leq 1 + \lg n + \lg 2^{g/p+2} = \lg n + g/p + 4.$$

Summing over all comparisons, we get a gap cost of at most $p(4 + \lg n) + g$. The total cost of the proof is this plus $p \lg n$, and hence is $O(p \lg n + g)$ as desired. □

It turns out that this is the only proof of nonintersection for this instance, except for two minor details described now. Let P be a proof of nonintersection. A comparison c in P is called *discardable* if $P - c$ still proves nonintersection. P is called *minimal* if none of its comparisons can be discarded. A *contraction* is a set of comparisons in P of the form

$$\{(a_1 < a^s_{i+1}), \ldots, (a_j < a^s_{i+1}), (a^s_i < b)\}$$

such that $a^s_{i+1} < b$. In other words, these comparisons can be replaced with

$$\{(a_1 < b), \ldots, (a_j < b)\}$$

while still proving nonintersection. P is *contraction-free* if it has no contractions.

Lemma 7 *There is only one minimal contraction-free proof of nonintersection for the described instance.*

Proof: Applying Lemma 5, let P be incrementally ordered such that all its comparisons are low, and consider stepping through this ordering. Initially, $m_1^{s_0}$ is minimal. Assume by induction that the proof so far is as desired, and that $m_k^{s_{k-1}}$ is now minimal. Consider the first comparison $c = (a_i^s < b)$ in P after this assumption becomes true.

Now m_{k-1} occurs in every set except $A^{s_{k-1}}$, and $m_k^{s_{k-1}}$ just became minimal, so no element that is at least m_{k-1} could have been eliminated. If $b \leq m_{k-1}$, c tells us nothing and so can be discarded, contradicting minimality of P. Hence, the only possible choice for b is $m_k^{s_{k-1}}$. Suppose that c does not make $m_{k+1}^{s_k}$ minimal, that is, $a_{i+1}^s \neq m_{k+1}^{s_k}$. Then $a_{i+1}^s \leq m_k^{s_{k-1}}$. For c to be undiscardable, there must be another comparison using a_{i+1}^s as an eliminator. But if we take the set of all such comparisons together with c, we get a contraction, because $a_{i+1}^s \leq m_k^{s_{k-1}}$, contradicting that P is contraction-free. Therefore, $c = (m_{k+1}^{s_k} < m_k^{s_{k-1}})$, the desired result. □

This tells us something about all proofs of nonintersection.

Lemma 8 *Every proof of nonintersection for the described instance has gap cost $\Omega(g)$.*

Proof: If we discard some comparisons, and make some contractions, we can only decrease the gap cost. Hence by Lemma 7, this proof is the one described above. Clearly, it eliminates at least one gap of size $k_i^s = \Omega(g/p)$ in each step, so the gap cost is $\Omega(g)$. □

Finally, we can show a lower bound on the running time of the algorithm. The algorithm is allowed to know the magic values $m_1, \ldots, m_p$, as well as $\ell_1, \ldots, \ell_p$, that is, the approximate gap sizes. The algorithm does not know the exact gap sizes (the k_i^s's), nor the numbers s_i of the sets missing the magic values.

Lemma 9 *The algorithm must determine the s_i's and $k_{i-1}^{s_i}$'s, independently of each other.*

Proof: By Lemma 7, every proof of nonintersection compares $m_i^{s_{i-1}}$ for all i, and hence it must figure out the s_i's and $k_{i-1}^{s_i}$'s. To determine that an element is magic, we must determine that it is equal to another element in another set. Equivalently, we can test whether the element is equal to the known value of m_i. Determining either of these does not help us find other magic elements. □

Hence, the algorithm's job reduces to p independent subjobs, each of the following form: given n sorted sets, each of unknown size whose lg is ℓ_i, find the unique set whose last element is not magic. We need the following fundamental theorem, which to our knowledge has not appeared before.

Lemma 10 *Consider the following problem: given n sorted sets, each of size k, and given an element e, find the unique set not containing e. In the comparison model, any algorithm solving this problem takes $\Omega(n \lg k)$ time.*

Proof: Suppose that the element e actually occurred everywhere. Then the algorithm must determine position in every set at which e occurs, using queries with constant-size answers. Encoding this information takes $\lg k$ bits per set, proving a lower bound of $\Omega(n \lg k)$ time. □

Therefore, the algorithm takes $\Omega(ng)$ time, proving Theorem 4.

5.2 Upper Bound

Consider the following algorithm for finding a proof of nonintersection. Essentially, it "gallops" in parallel through all the sets, from both the low and high sides. Galloping consists of doubling the jump in position each iteration, until it "overshoots" the current eliminator (which will always be on the low side). Upon overshooting, the other parallel processes pause while the overshooter does a binary search to find the largest eliminatable element, and chooses the next higher element as the new eliminator.

In more detail, the algorithm works as follows.

- Initialize low-jump(s) and high-jump(s) to 1, and done(s) to 0, for each $s \in \{1, \ldots, n\}$.
- Initialize elim-set to 1 and eliminator to a_1^1.
- For s ranging through $\{1, \ldots, n\}$ cyclicly:
  - Skip this step if $s =$ elim-set.
  - Low step:
    1. Let $p =$ done(s) $+$ low-jump(s).
    2. If $a_p^s \geq$ eliminator (we overshot),
       (a) Binary search in the interval [done(s) $+ 1, p$] to find the smallest p' with $a_{p'}^s \geq$ eliminator.
       (b) If $p' - 1 >$ done(s), add ($a_{p'-1}^s <$ eliminator) to the proof.
       (c) Set done(s) to $p' - 1$, and low-jump(s) to 1.
       (d) Set elim-set to s, and eliminator to $a_{p'}^s$.
    3. Otherwise, double low-jump(s) and set done(s) to p.
  - High step:
    1. Let $p = n_s + 1 -$ high-jump(s).
    2. If $a_p^s <$ eliminator (we overshot),
       (a) Binary search in the interval $[p, n_s]$ to find the largest p' with $a_{p'}^s <$ eliminator.
       (b) If $p' >$ done(s), add ($a_{p'}^s <$ eliminator) to the proof.
       (c) If $p' = n_s$, stop.
       (d) Set done(s) to p', and low-jump(s) to 1.
       (e) Set elim-set to s, and eliminator to $a_{p'+1}^s$.
       (f) Reset high-jump(s) to one.
    3. Otherwise, double high-jump(s).

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

Theorem 5 *The algorithm runs in $O(n\mathcal{G})$ time, and makes at most $8n\mathcal{G}$ comparisons.*

Proof: The initialization steps take $O(n)$ time and no comparisons, and since $\mathcal{G}$ is a positive integer, this is within the bound. The cost of performing the binary searches can be amortized away as follows. Suppose we binary search inclusively between done$(s) + 1$ and $p =$ done$(s) +$ low-jump; the high case is similar. This takes at most $t = \lg$ low-jump comparisons. But low-jump $= 2^i$ where i is the number of iterations we have already executed on this side of A^s since the last overshooting. Hence, $t = i + 1$, so we can charge the binary-search time t to these $i > 0$ iterations, and no other overshooting iterations will charge to the same iterations (because we reset the jump now). This amortization is the source of the first factor of two in the comparison bound.

Let P be a proof with gap cost $\mathcal{G}$, and let it be incrementally ordered with all low comparisons by Lemma 5. Let $c = (a_i^s < a_j^t)$ be the first comparison in P. We want to evaluate the number of iterations the algorithm spends to eliminate a_i^s. The low and high parts of the algorithm run effectively in parallel; this causes the second factor of two in the comparison bound.

There are two main cases. In the first case, the gap below a_i^s (of size g say) is not the largest gap in A^s. Ignoring the binary-search cost as described above, galloping effectively occurs in lock-step parallel over the sets. Local to A^s, the number of comparisons for galloping to a_i^s or beyond is $\lg g$. The other sets have had at most the same number of iterations, thus adding a factor of n.

The second case is when the gap below a_i^s is the largest gap in A^s. If the sum of the other gaps' sizes is at least g, then we can charge the cost ($\lg g$) of running through the largest gap to the other gaps in A^s. These gaps will not be charged to again, because there is only one largest gap in A^s that the gap cost does not count (i.e., for which we must avoid paying directly). If, on the other hand, the other gaps' sizes sum up to some value h less than g, then the high step finds a_i^s from the high side in $\lg h$ iterations. But $\lg h$ is at most the sum of the lgs of the other gaps in A^s, so again we can charge to these gaps. These amortizations add the last factor of two to the comparison bound.

Therefore, eliminating a_i^s takes $\lg g$ amortized comparisons, unless g is the largest gap in its set, in which case it takes zero amortized comparisons. By induction, this holds for all future comparisons in P. □

Let us turn to the case where the intersection is not necessarily empty, and we are asked to compute it. Define a *proof of correctness* to be a proof that demonstrates the intersection elements (by making $n - 1$ equality comparisons each), and forms a proof of nonintersection on the remaining elements. Similar to before, we can define the gaps in such a proof, and hence the optimal gap cost $\mathcal{G}$. Then we have the following result.

Corollary 1 *The intersection of n sorted sets can be computed in $O(n\mathcal{G})$ time and at most $8n\mathcal{G}$ comparisons.*

Proof: This follows from a simple modification to the algorithm above, namely whenever a comparison with the eliminator returns "equal," stop galloping in that set and increase the occurrence count of the eliminator. If the occurrence count reaches n, output the eliminator as part of the intersection, and take its successor as the new eliminator. □

Note that the described algorithms perform just as well on B-trees or related structures. We only need to start with the leftmost and rightmost leaves, and then gallop inwards from each side. This can be easily performed by traversing the parent and child pointers in a B-tree, with only a constant-factor overhead.

6 Expected Case

Suppose we generate k sets, each containing n random values chosen uniformly from the unit interval $[0,1]$. What is the expected length of a shortest proof of nonintersection? Suppose that we knew the expected value v of the maximum of the minimum element in each set. This gives us an initial eliminator of expected value v. We can use it to eliminate any value in another set with value less than v, so we can ignore all elements with values less than v. We have thus reduced to the subproblem of random values chosen from $[v,1]$. The process continues in the same way, reducing to the subproblems $[2v,1]$, $[3v,1]$, etc. in the expected case. Hence, the expected length of a shortest proof is $1/v$. The rest of this section will evaluate v.

First, what is the probability distribution of the minimum element of a set A^i of size n, whose elements are chosen uniformly from $[0,1]$? Let us denote this value by m_i. We will evaluate the probability that m_i is less than some $0 \le x \le 1$. Note that "m_i is less than x" is equivalent to "either the first element is less than x, or it is not and the second element is less than x, or both are not and the third element is less than x, etc." Also note that the probability of a particular element being less than x is x. Hence,

$$\Pr[m_i < x] = \sum_{i=0}^{n-1}(1-x)^k x = 1-(1-x)^n, \quad \text{and so} \quad \Pr[m_i > x] = (1-x)^n. \tag{3}$$

Although it will not be useful for our purposes, let us evaluate the expected value of m_i. To do this, we need to compute the probability density function D of the minimum value. But

$$\begin{aligned} 1-(1-x)^n = \Pr[m_i < x] &= \int_0^x \mathrm{D}[m_i = x]dy, \\ \text{so}\ \ \mathrm{D}[m_i = x] &= \frac{d}{dx}1-(1-x)^n = n(1-x)^{n-1}. \end{aligned}$$

Now we can take the expected value:

$$\mathrm{E}[m_i] = \int_0^1 x \cdot n(1-x)^{n-1}dx = \frac{1}{n+1}.$$

Next we want to compute the expected value of the maximum M of the m_i's. We can do this using an approach similar to the above, using the probability distribution of m_i given in (3). First we want to compute the probability distribution of M:

$$\begin{aligned} \Pr[M > x] &= \sum_{i=0}^{k-1}\left(1-(1-x)^n\right)^i(1-x)^n = 1-\left(1-(1-x)^n\right)^k, \\ \text{so}\quad \Pr[M < x] &= \left(1-(1-x)^n\right)^k, \\ \text{and}\quad \mathrm{D}[M = x] &= \frac{d}{dx}\left(1-(1-x)^n\right)^k = -k\left(1-(1-x)^n\right)^{k-1} n(1-x)^{n-1}. \end{aligned}$$

Now we can find the expected value, by substituting $y = 1 - x$:

$$
\begin{aligned}
\mathrm{E}[M] &= -kn \int_0^1 x \cdot (1-x)^{n-1} \left(1 - (1-x)^n\right)^{k-1} dx \\
&= kn \int_0^1 (1-y) y^{n-1} (1-y^n)^{k-1} dy \\
&= kn \left(\underbrace{\int_0^1 y^{n-1} (1-y^n)^{k-1}}_{=\frac{1}{kn}} - \underbrace{\int_0^1 y^n (1-y^n)^{k-1}}_{=\frac{\left(\frac{1}{n}\right)!(k-1)!}{n\left(k+\frac{1}{n}\right)!}} \right) \\
&= 1 - k \frac{(\frac{1}{n})!(k-1)!}{(k+\frac{1}{n})!} \\
&= 1 - \frac{(\frac{1}{n})!k!}{(k+\frac{1}{n})!} \\
&= \frac{\gamma + \psi(k+1)}{n} + O\left(\frac{1}{n^2}\right) \\
&= \frac{\ln k}{n} + \frac{\gamma}{n} + \frac{1}{2kn} + O\left(\frac{1}{k^2 n} + \frac{1}{n^2}\right).
\end{aligned}
$$

where

$$
\psi(x+1) = x! \frac{dx!}{dx} = \ln x + \frac{1}{2x} + O\left(\frac{1}{x^2}\right).
$$

Therefore, the expected length of a shortest proof is $1/\mathrm{E}[M]$, which is asymptotically $n/\ln k$.

7 Conclusion

In this paper we have considered adaptive algorithms for computing the intersection of sorted sets. To do this we have made a detailed study of sets of comparisons proving that a given intersection is empty. Our main results are matching upper and lower bounds on the worst-case ratio of the running time to the information-theoretic lower bound on the difficulty of the problem. The bound is the number of sets times a ratio $\mathcal{G}/\mathcal{D}$ which is slightly less than one, depending on the instance. In other words, there is an algorithm that performs slightly better than a factor of the number of sets away from the information-theoretic lower bound, and this is the best possible in the worst case.

Acknowledgments.

We thank Ming Li for helpful discussions. This work was supported by NSERC.

References

[1] Aho, A., Hopcroft, J., Ullman, J. *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1974.

[2] Baeza-Yates, R. *Efficient Text Searching*, Ph.D. Thesis, Dept. of Computer Science, U. of Waterloo, 1989.

[3] Boyer, R., Moore, J. A Fast String Searching Algorithm. *CACM*, 20:762-772, 1977.

[4] Bloom, B.H. Some Techniques and Trade-offs Affecting Large Data Base Retrieval Times. *ACM NC*, pp. 83–95, 1969.

[5] Feldman, J.A. and Rovner, P.D. An Algol-Based Associative Language. *CACM*, 12(8):439–449, 1969.

[6] Frakes, W., Baeza-Yates, R. *Information Retrieval*, Prentice Hall, 1992.

[7] Heaps, H.S. and Thiel, L.H. Optimum Procedures for Economic Information Retrieval. *Inform. Stor. and Retrieval*, 6(2):137–153, 1970.

[8] Hwang, F.K. and Lin, S. A Simple Algorithm for Merging Two Disjoint Linearly-Ordered Sets. *SIAM J. Comput.*, 1(1):31–39, 1972.

[9] Hwang, F.K. Optimal Merging of 3 Elements with n Elements. *SIAM J. Comput.*, 9(2):298–320, 1980.

[10] Knuth, D. *The Art of Computer Programming*, Vol. 3 Sorting and Searching. Addison-Wesley, 1973.

[11] Knuth, D., Morris, J., Pratt V. Fast Pattern Matching in Strings, *SIAM J. Comput.*, 6:323–350, 1977.

[12] Lesk, Michael. “Real World” Searching Panel at SIGIR 1997. *SIGIR Forum*, Spring 1998.

[13] Manber, U., Myers, G. Suffix Arrays: A New Method for On-line String Searches, in *1st Symposium on Discrete Algorithms*, pp. 319-327, 1990.