The Bit Vector Intersection Problem

(Preliminary Version)

Richard M. Karp* Orli Waarts† Geoffrey Zweig‡
University of California at Berkeley and ICSI

Abstract

This paper introduces the bit vector intersection problem: given a large collection of sparse bit vectors, find all the pairs with at least t ones in common for a given input parameter t. The assumption is that the number of ones common to any two vectors is significantly less than t, except for an unknown set of $O(n)$ pairs. This problem has important applications in DNA physical mapping, clustering, and searching for approximate dictionary matches.

We present two randomized algorithms that solve this problem with high probability and in subquadratic expected time. One of these algorithms is based on a recursive tree-searching procedure, and the other on hashing. We analyze the tree scheme in terms of branching processes, while our analysis of the hashing scheme is based on Markov chains. Since both algorithms have similar asymptotic performance, we also examine experimentally their relative merits in practical situations. We conclude by showing that a fundamental problem arising in the Human Genome Project is captured by the bit vector intersection problem described above and hence can be solved by our algorithms.

1 Introduction

In the *bit vector intersection* problem, one is given a collection of n sparse k-dimensional bit vectors, and the goal is to find all the pairs with at least t ones in common. The model we study is a generalization of the following simple situation.

- Each position in a vector is 1 with probability $\leq P$;
- Except for an unknown set of Cn pairs of vectors, the number of ones common to any two vectors is at most kP^2;
- $t = kP^s$, where $1 \leq s < 2$; thus t is significantly greater than the expected number of common ones in a typical pair of vectors.

The exceptional pairs in the second property are called *overlapping* pairs.

This problem has important applications in DNA physical mapping, clustering, and searching for approximate dictionary matches.

The original motivation for our problem came from computational biology. In the process of DNA physical mapping, a chromosome is decomposed into many overlapping pieces called *clones*. Each clone is *fingerprinted* experimentally, and it is desired to reconstruct the relationship of the clones on the intact chromosome using these fingerprints. A key step in this reconstruction process is to identify pairs of fragments with a high degree of overlap, as evidenced by the similarity of their fingerprints. Current mapping projects with roughly $10,000$ clones spend an inordinate amount of time finding overlapping pairs of clones [10]. Larger projects [11] will involve on the order of 10^5 or 10^6 clones. Hence, it is important to improve upon the obvious quadratic algorithm in which all pairs of clones are compared. In Section 5 we present abstractions of the two variants of this problem which occur most frequently in biological situations, and show that both conform to the assumptions of our model and hence can be solved by our algorithms with the stated time complexity.

The problem of finding highly-intersecting pairs of bit vectors is related to the nearest-neighbor, range-searching, and partial-match problems that have been intensively studied in the past (cf. [2, 16, 6, 15, 4, 7, 1, 14, 3, 5]). Of the above problems, the closest to ours is the problem of finding the nearest Hamming-distance neighbors to a given vector (cf. [13, 8, 5]).

*E-Mail: karp@cs.berkeley.edu
†Work supported in part by NSF postdoctoral fellowship.
E-Mail: waarts@cs.berkeley.edu
‡E-Mail: zweig@cs.berkeley.edu

Interestingly, a simple encoding reduces the problem of finding the nearest Hamming-distance neighbors to the problem of finding highly intersecting bit vectors. The opposite does not hold.

The two most common approaches to the problem of finding the closest Hamming-distance neighbors are based on tree-like data structures such as tries, and hashing schemes based on error correcting codes [13, 5]. None of these methods can be directly applied to the problem of finding highly intersecting pairs.

We present two randomized algorithms for solving the bit vector intersection problem, one based on a recursive tree-searching procedure, and the other on hashing. Both procedures are shown to have a running time of $O(n^{s+\iota})$, where ι is a small positive value. In the tree-based procedure, under commonly occurring conditions, ι approaches zero asymptotically (see Section 3.1), while in the hashing procedure ι is bounded from below by a small constant.

Our algorithms are substantially different from current methods for solving the above-mentioned related problems in that they generate multiple representations for each item; current tree-based methods partition the input space so that each item lies in exactly one leaf, and current hashing methods place each item in exactly one bin.[1] Our algorithms rely on a measured duplication of item representations. The analyses we present, based on branching processes for the tree approach, and based on Markov chain analysis for the hashing procedure, are also novel.

The basic operation of our algorithms can be stated very simply. At each non-leaf node, the tree algorithm randomly selects a fixed number of coordinates and then, for each selected coordinate in turn, it recursively analyses all the bit vectors that test positive for that coordinate. The branching factor is thus the number of coordinates selected, and multiple representation results when a vector has more than one of the selected coordinates, and is thus processed along more than one of the branches. The algorithm terminates when a fixed depth is reached. We show that each time this algorithm is run, each pair of overlapping vectors will occur together at a leaf node with a probability that is independent of n, while the probability that a nonoverlapping pair occurs together tends to 0. The above two facts admit a subquadratic randomized algorithm that with multiple repetitions identifies all highly overlapping pairs with high probability.

The hashing-based method is very different, but is also based on multiple representations of the bit vectors. Loosely stated, an arbitrary ordering is imposed on the coordinates and hash keys are generated for each vector from all l-tuples of consecutive positive features. We show that each time the algorithm is run, each pair of highly overlapping vectors will tend to be mapped into identical buckets, while the probability that a nonoverlapping pair is found together is significantly smaller. These facts result again in a subquadratic randomized algorithm that with multiple repetitions identifies all overlapping pairs with high probability.

Since both algorithms have similar asymptotic performance, in Section 6, we examine experimentally their relative merits in practical situations.

Due to lack of space, proofs are omitted or only sketched.

[1] [5] raises the question of multiple hashing schemes, based on error correcting codes, in the context of Hamming-distance nearest neighbors.

2 The Idealized Bit Vector Intersection Problem

In the *bit vector intersection* problem, we are given a set of n binary vectors, $v_1, \ldots, v_n$, of size k each. Let n_i denote the number of ones in vector v_i, and let $n_{i,j}$ denote the number of ones that vectors v_i and v_j have in common, *i.e.* $\sum_a v_{i,a} v_{j,a} = n_{i,j}$. The goal is to find all pairs of vectors v_i, v_j for which $n_{i,j} \geq t$, for a given parameter t.

The problem is called the *idealized* bit vector intersection problem if the following requirements are satisfied. For a given parameter $0 \leq \mu \leq 1$ and given constant parameters P, s, C, d, where d is sufficiently large, we have:

1. $k \geq d \log_{1/P} n$;
2. $\forall i,\ n_i \leq (1+\mu)kP$;
3. there are at most Cn pairs i, j such that $n_{ij} > (1+\mu)Pn_i$;
4. $\frac{t}{k} = P^s$, where $1 \leq s < 2 - \log_{1/P}(2(1+\mu)^2)$.

The exceptional pairs in the third property are called *overlapping* pairs.

The first property is not very restrictive since we can always assume that all vectors are distinct (since duplicates can be eliminated very efficiently); and for distinct vectors, k is at least $\log n$. The intuition behind the second property is that the given vectors are sparse. The intuition behind the third property is that for most pairs of vectors, values of corresponding entries are independent. Moreover, the second, third and fourth properties together make the pairs of vectors one is looking for, *i.e.* pairs with at least t ones in common, distinguishable from the other pairs.

2.1 A Relaxed Variant

The Data tree algorithm requires even weaker assumptions than those of the idealized nearest pairs problem. Specifically, it only requires that for a given $0 \leq \mu \leq 1$ and given constant parameters P, B, s, we have:

1. $\frac{t}{k} = P^s$, where $1 \leq s < 2 - \log_{1/P}(1+\mu)^2$;
2. $\sum_{i,j}(\frac{n_{i,j}}{k})^{\lceil \log_{1/P} n \rceil} \leq Bn^{\log_{1/P}(1+\mu)^2}$.

As shown in the full paper, these properties are always satisfied by the idealized nearest pairs problem (with the same μ, P and s).

2.2 The Asymptotic Bit Vector Intersection Problem

For the application to biology considered in Section 5, it is natural to consider a family of instances in which P and s are fixed, n tends to infinity, and $\mu = O(1/\log n)$. We call this setting the *Asymptotic* Bit Vector Intersection Problem. In this setting, the fourth property of the idealized problem requiring that $1 \leq s < 2 - \log_{1/P} 2(1+\mu)^2)$ can be replaced by the requirement that $1 \leq s < 2 - \log_{1/P} 2$. Analogously, the first property of the relaxed variant can be replaced by the requirement that $1 \leq s < 2$; and the second property is replaced by $\sum_{i,j}(\frac{n_{i,j}}{k})^{\lceil \log_{1/P} n \rceil} \leq B$.

3 Data Tree Algorithm

The data tree algorithm solves the idealized bit vector intersection problem by iterating the following *one-shot* data tree algorithm.

First Step The algorithm constructs a rooted tree of depth $\lceil \log_{1/P} n \rceil$, in which each internal node has $\lceil (1 + \frac{1}{\log n})k/t \rceil$ children. Each edge independently is labeled with an integer drawn from the uniform distribution over $\{1, 2, \ldots, k\}$, representing one of the coordinates of the k-dimensional vectors $v_1, v_2, \ldots, v_n$. A vector v_i from the set $\{v_1, v_2, \ldots, v_n\}$ is said to occur at a given node if v_i has a 1 in every coordinate which occurs as a label on an edge of the path from the root to that node. In particular, every vector v_i occurs at the root.

Second Step For each of the leaves, the algorithm compares all pairs of vectors occurring at the leaf. It finds among them all pairs v_i, v_j with $n_{i,j} \geq t$, and outputs them.

The data tree algorithm The data tree algorithm iterates the above one-shot algorithm, and outputs the set of all pairs that were output by the one-shot algorithm in any of the iterations. The number of iterations is determined by the desired upper bound on the probability of error, *i.e.* of not detecting some pairs v_i, v_j for which $n_{i,j} \geq t$. The probability of error will decrease exponentially with the number of iterations.

3.1 Analysis

This section analyses the correctness of the data tree algorithm (*i.e.* the probability of detecting all pairs v_i, v_j for which $n_{i,j} \geq t$), and the amount of work it performs (*i.e.* the expected number of comparisons and tests). Lemma 3.1 shows that for each pair of vectors v_i, v_j for which $n_{i,j} \geq t$, the probability that the pair will be detected by the one-shot algorithm is $\Omega(1/\log n)$. To analyze the amount of work we distinguish between a *comparison* of a pair of vectors, as done in the second step of the one-shot algorithm, and a *primitive test* that finds whether $v_{i,j} = 1$, as done in the first step of the one-shot algorithm. Lemma 3.2 shows that the expected number of comparisons performed by the one-shot algorithm is $O(n^{s+\log_{1/P}(1+\mu)^2})$ and is $O(n^s)$ in the asymptotic problem (since by definition of the asymptotic problem $\mu = O(1/\log n)$). Notice that by definition of s, the exponent in $n^{s+\log_{1/P}(1+\mu)^2}$ is less than two. Lemma 3.3 shows that the expected number of primitive tests performed by the one-shot algorithm is $O(n^{s+\log_{1/P}(1+\mu)^2} \log n)$, which is $O(n^s \log n)$ in the asymptotic problem. The performance of the bit vector intersection algorithm is summarized by Theorem 3.4.

Lemma 3.1 *For each pair of vectors q, r for which $n_{q,r} \geq t$, the probability that pair (q, r) is output by the one-shot data tree algorithm is $\Omega(1/\log n)$.*

Sketch of Proof: Within the tree generated by the one-shot algorithm, consider the subtree consisting of all those nodes at which both q and r occur. This subtree may be viewed as being generated by a branching process in which the number of offspring of a parent is the number of heads in $\lceil (1 + \frac{1}{\log n})k/t \rceil$ tosses of a coin with probability of heads t/k. Since the expected number of heads is $\geq 1 + \frac{1}{\log n}$, the branching process is supercritical, and hence there is some positive probability of nonextinction in such a process.

Let X denote this positive probability of survival. Then X is the solution of the equation: $X =$

$\sum_{a=0}^{\infty} p_a X^a$, where p_a is the probability that the number of children of a parent is a. Let $\lambda = 1 + \frac{1}{\log n}$. Then $p_a \approx e^{-\lambda}\frac{\lambda^a}{a!}$, and hence X must approximate the solution of the equation: $X = \sum_{a=0}^{\infty} e^{-\lambda}\frac{\lambda^a}{a!}X^a$. Straightforward algebraic manipulations yield that $X = \Omega(1/\log n)$. ■

Lemma 3.2 *The expected number of pairs compared in the second stage of the one-shot data tree algorithm is at most $O(n^{s+\log_{1/P}(1+\mu)^2})$, which is $O(n^s)$ in the asymptotic bit vector intersection problem.*

Lemma 3.3 *The expected number of primitive tests performed in the first stage of the one-shot data tree algorithm is $O(n^{s+\log_{1/P}(1+\mu)^2}\log n)$, which is $O(n^s \log n)$ in the asymptotic bit vector intersection problem.*

Lemmas 3.1, 3.2, and 3.3 immediately imply:

Theorem 3.4 *For every $\tau > 0$, iterating the one-shot data tree algorithm $2.5\ln(n^2/\tau)\log n$ times and outputting the set of all pairs output in these iterations, gives a bit vector intersection algorithm whose probability of error is $\leq \tau$, and which performs $O(n^{s+\log_{1/P}(1+\mu)^2}\log^2 n \log(1/\tau))$ comparisons and $O(n^{s+\log_{1/P}(1+\mu)^2}\log^3 n \log(1/\tau))$ primitive tests.*

In the asymptotic problem, the resulting algorithm performs $O(n^s \log^2 n \log(1/\tau))$ comparisons and $O(n^s \log^3 n \log(1/\tau))$ primitive tests.

4 Hashing-Based Algorithm

Denote $\lceil \log_{1/P} n \rceil$ by l. Define a *lexicographic l-tuple* as a strictly increasing sequence of l integers from the set $\{1, 2, \cdots, k\}$. The lexicographic l-tuple $i_1, i_2, \cdots, i_l$ is said to be a *consecutive l-tuple* of the bit vector v if v contains ones in coordinates $i_1, i_2, \cdots, i_l$, and zeros in all the other coordinates in the interval $[i_1..i_l]$. A lexicographic l-tuple that is consecutive for both v_i and v_j is called a *matching consecutive l-tuple* for v_i and v_j.

Choose an integer b such that $bn^{-s+1} \leq 1$. A consecutive l-tuple whose first coordinate is not greater than b is defined as *eligible* consecutive l-tuple.

The hashing-based algorithm solves the idealized bit vector intersection problem by iterating a *one-shot* hashing-based algorithm. The one-shot algorithm uses a hash table and proceeds in three steps as follows.

First step The one-shot algorithm first erases all current values appearing in the hash table. Then it computes a random permutation σ on $1, \ldots, k$, and, for each vector v_i, reorders v_i according to σ (*i.e.* it computes a new v_i, say v_i', such that for each a, $v'_{i,a} := v_{i,\sigma(a)}$). In the following we refer to the resulting vector (v_i' of above) by v_i.

Second step For each vector v_i, and for each eligible consecutive l-tuple belonging to v_i, the index i is placed in a hash address determined by the l-tuple. In the following, whenever we say that integers i, j appear in the same cell of the hash table, we refer to those v_i, v_j that are placed in that cell because of a matching eligible consecutive l-tuple for the two vectors. We assume that the hash table is sufficiently large so that the number of collisions resulting from distinct eligible consecutive l-tuples hashing to the same cell is dominated by the number of collisions resulting from matching eligible consecutive l-tuples. This means that the size of the hash table is proportional to the number of hashes done by the one-shot algorithm.

Third step The one-shot algorithm compares each pair of vectors v_i, v_j whose indices appear in the same entry of the hash table (*i.e.* all pairs of vectors which have some matching eligible consecutive l-tuple), and outputs from among those pairs all those for which $n_{i,j} \geq t$. The one-shot algorithm keeps track of compared pairs so that each pair is compared by it at most one time. (A pair may be re-compared in other iterations of the one-shot algorithm.)

The hashing based algorithm The hashing-based algorithm iterates the one-shot algorithm, and outputs the set of all pairs that were output by the one-shot algorithm in any of the iterations. The probability of error will decrease exponentially with the number of iterations.

4.1 Analysis

This section analyses the correctness of the hashing-based algorithm, *i.e.* the probability of detecting all pairs v_i, v_j for which $n_{i,j} \geq t$, and the amount of work the algorithm performs, *i.e.* the expected number of hashes and comparisons it does. Recall the definition of nonoverlapping pairs from section 2. We first bound from below the probability that two vectors with $n_{i,j} \geq t$ will have a matching eligible consecutive l-tuple, and bound from above the probability that two nonoverlapping vectors will have such an l-tuple (Section 4.1.1). Values for l and for the number of iterations of the one-shot algorithm are then chosen so that all pairs that have at least t 1's in common will be detected with high probability, and that in addition, the amount of work is minimized (Section 4.1.2).

4.1.1 Probability of a Matching Pair

Let $M_{i,j}$ be the probability that, in the one-shot hashing-based algorithm, v_i and v_j have a matching eligible consecutive l-tuple. Let $\kappa > 0$ be a sufficiently small constant such that $\log_{1/P}((2(1+\mu)-P^{s-1})(1+\kappa)/(1-\kappa)) \leq \log_{1/P}(2(1+\mu))$. (Note that by definition of the idealized configuration such a κ can be found.) The following lemmas are proved:

Lemma 4.1 *There exists a constant $0 < c_1 \leq 1$, such that for all vectors v_i, v_j such that $n_{i,j} \geq t$,*

$$M_{i,j} \geq c_1 b n^{-s+1-\log_{1/P}((2(1+\mu)-P^{s-1})(1+\kappa))} .$$

Notice that the lower bound on M_{ij} in the above lemma is at most one, due to our assumption that $bn^{-s+1} \leq 1$.

Lemma 4.2 *There exists a constant $c_2 > 0$, such that for all nonoverlapping vectors v_i, v_j,*

$$M_{i,j} \leq c_2 b n^{-1+\log_{1/P}((1+\mu)/(1-\kappa))} .$$

Sketch of Proof of Both Lemmas We first observe the following: The number of coordinates in which either v_i or v_j contains a 1 is $n_i + n_j - n_{ij}$, and, among these coordinates, the number in which both v_i and v_j contain a 1 is n_{ij}; thus $M_{i,j}$ is the probability that, if we randomly permute the coordinates of a bit vector which has size $n_i + n_j - n_{ij}$ and contains n_{ij} ones, the resulting vector will contain an eligible consecutive l-tuple.

Observing the above, we proceed in three steps. In the first step we bound the probability of having an eligible consecutive l-tuple in a vector r (of length $|r|$) each of whose entries is 1 with probability Q. This is done by constructing a Markov chain whose probability of reaching a final state is the same as the probability that the above vector r has an eligible consecutive l-tuple.

In the second step we show how the results obtained in the first step can be used, as a black box, in order to bound $M_{i,j}$. In particular, denote by M_Q the probability that the above random vector r has an eligible consecutive l-tuple. We show that for a random vector r of size $n_i + n_j - n_{ij}$, if $Q = Q_0(i,j)$ is slightly larger than $n_{ij}/(n_i + n_j - n_{ij})$, then $M_{ij} \leq (1+\epsilon)M_{Q_0(i,j)}$; and if $Q = Q_1(i,j)$ is slightly smaller than $n_{ij}/(n_i + n_j - n_{ij})$, then $M_{ij} \geq (1-\epsilon)M_{Q_1(i,j)}$, where ϵ is a small positive value.

In the third step we choose ϵ so that if v_i, v_j are overlapping, and v_f, v_g are nonoverlapping, then $(1+\epsilon)M_{Q_0(f,g)} << (1-\epsilon)M_{Q_1(i,j)}$. ■

4.1.2 Correctness and Efficiency

Lemma 4.3 *The number of hashes done by the one-shot hashing based algorithm is $\leq bn$.*

Lemma 4.4 *Let c_1, c_2 be the constants of Lemmas 4.1 and 4.2. The expected number of comparisons done by the one-shot hashing based algorithm is $\leq c_2 b n^{1+\log_{1/P}((1+\mu)/(1-\kappa))} + Cn$, where C is the C from the definition of the idealized configuration.*

Theorem 4.5 *For every $\tau > 0$, iterating the one-shot algorithm $2.5\ln(n^2/\tau) \cdot n^{s-1+\log_{1/P}((2(1+\mu)-P^{s-1})(1+\kappa))}/(c_1 b)$ times and outputting the set of all pairs output in these iterations, gives a bit vector intersection algorithm whose probability of error is $\leq \tau$, and which performs at most $\frac{2.5}{c_1} \cdot n^{s+\log_{1/P}(2(1+\mu))}\ln(n^2/\tau)$ hashes and at most an expected $2.5(\frac{c_2}{c_1} n^{s+\log_{1/P}(2(1+\mu)^2)} + \frac{C}{c_1 b} n^{s+\log_{1/P}(2(1+\mu))})\ln(n^2/\tau)$ comparisons.*

In the asymptotic problem, i.e. $\mu = O(1/\log n)$, *the resulting algorithm performs $O(n^{s+\log_{1/P} 2}\log(n/\tau))$ hashess and expected $O(n^{s+\log_{1/P} 2}\log(n/\tau))$ comparisons.*

The above bound on the amount of work performed by the hashing scheme is $O(n^{s+\alpha})$, where α decreases with μ. Unlike the case of the data tree scheme, α does not vanish when $\mu = 0$, because it is bounded below by a small fixed constant. A closer look shows that the amount of work performed by the algorithm is indeed $\Omega(n^{s+\alpha'})$, where α' depends on P and s. Hence our data tree scheme is asymptotically more efficient than our hashing based scheme. Nevertheless, due to its small constants, the hashing scheme performs quite well in practice, as can be seen in Section 6.

5 An Application to Computational Biology

The following *Physical Mapping Problem* is central to the Human Genome Project and many other efforts in molecular biology: given a *clone library* - i.e., a large set of relatively short pieces (clones) taken from a long DNA molecule, determine the structure of the long molecule from information about the individual clones. A key step in solving this problem is to determine which pairs of clones overlap substantially. In this section we shall show that two versions of this problem satisfy the conditions of our Idealized Problem, and thus can be handled effectively by the methods of this paper. The first version is called the

Poisson Features Problem, and the second version is called the *Restriction Sites Problem*. In both versions we are given a set of n clones that satisfies the following properties:

1. Each clone is an interval of length 1 contained in the interval $[0, N]$;
2. The left-hand end points of the clones are independent random variables, each of which has the uniform distribution over $[0, N-1]$;
3. n/N is bounded above by a constant.

In both problems the goal is to produce a list that (i) contains all pairs of clones whose intersection is an interval of length at least w, for a given fixed parameter w, where $0 < w < 1$; and (ii) does not contain any nonoverlapping clones.

In the *Poisson Features Problem* we assume that, for $i = 1, 2, \ldots, k$, the occurrences of feature i along the interval $[0, N]$ are according to a Poisson process of rate λ, and the Poisson processes corresponding to different features are mutually independent.

In the *Restriction Sites Problem* restriction sites occur along a DNA molecule according to a Poisson process of rate λk. A clone has feature i if there are two consecutive restriction sites on the clone such that the distance between them lies in the interval $(a_{i-1}, a_i]$, where $a_0 = 0$, $a_k = 1$ and $a_0 < a_1 < a_2, \ldots < a_{k-1} < a_k$. Moreover, the a_i are chosen so that the probability that a clone has feature i is the same for all i.

Both the Poisson Features Problem and the Restriction Sites Problem satisfy with high probability the conditions of our Idealized Bit Vector Intersection Problem provided we assume that (i) P is a constant less than $1/2$; (ii) k, the number of features, is greater than $r \log_{1/P} n$, where $r \geq d$ is a sufficiently large constant and d is the d from the definition of the idealized configuration (the role of r will become clear in Section 5.1); and (iii) w is sufficiently large with respect to P.[2] (All three assumptions are easily satisfied in practical situations.)

To establish this we prove that the parameters P, C, t, μ and s can be chosen so that with high probability the following properties hold, where n_i is the number of features possessed by clone i, and n_{ij} is the number of features possessed by both clone i and clone j.

1. for all i, $n_i \leq (1+\mu)kP$;
2. at most Cn pairs of clones overlap;
3. if clones i and j do not overlap then $n_{ij} \leq (1+\mu)Pn_i$;
4. $\frac{t}{k} = P^s$, where $1 \leq s < 2 - \log_{1/P}(2(1+\mu)^2)$;
5. whenever clones i and j overlap along an interval of length at least w, $n_{ij} \geq t$; and whenever i and j do not overlap, $n_{ij} < t$.

The first four properties are required by the definition of the idealized bit vector intersection problem. For the last property, recall that the bit vector intersection problem requires us to identify all pairs with $n_{i,j} \geq t$. Hence, the last property ensures that identifying those pairs will, with high probability, detect all those clones that overlap for length at least w and will not detect any nonoverlapping pair.

In the following, we will slightly abuse the definitions and refer to the above five properties as the idealized properties.

Section 5.1 shows that with high probability, the Poisson Features problem satisfies the idealized properties defined above. The proof that the Restriction Sites problem also satisfies the idealized properties is deferred to the full paper, where we also deal with generalizations of both problems in which the features may occur with unequal probabilities.

In fact, as seen below, μ can be decreased with k and can be taken as $O(1/\log n)$ when $k = \Omega(\log^2 n)$. In practical situations k is usually $\Omega(\log^2 n)$. Thus, both the Poisson Features Problem and the Restriction Sites Problem satisfy the conditions of the asymptotic bit vector intersection problem (see Section 2.2).

5.1 Proof that the Poisson Features Problem Satisfies the Idealized Properties with High Probability

Let OVERLAP denote the number of overlapping clones.

Lemma 5.1 *For each $c > 0$, there exists a constant D, such that* $\Pr\left(\text{OVERLAP} \geq D\frac{n}{N}n\right) \leq O(n^{-c})$.

Lemma 5.1 establishes the second idealized property. Next we show that the remaining idealized properties are satisfied.

First we choose μ. The probability that in an interval of length q there will be some occurrences of feature f is $1 - e^{-\lambda q}$. Thus, the probability that feature f will occur in a given clone is $1 - e^{-\lambda} \stackrel{\text{def}}{=} P$. Similarly, the probability that feature f will occur in two given clones that overlap for length q is $1 + e^{-\lambda(2-q)} -$

[2] For satisfying the requirements of the relaxed variant of the problem (see Section 2.1) it is not necessary that w be sufficiently large but w only need be a constant. Thus, the Data tree algorithm will solve the above problems also for a negligible small constant w.

$2e^{-\lambda} \stackrel{\text{def}}{=} m(q)$. Since w is fixed, there exists some constant $0 < \mu$ such that $(1-\mu)m(w) > (1+\mu)P^2$. We choose this μ.

The expected number of different features in a clone is $k(1-e^{\lambda}) = kP$. The Chernoff inequality will thus yield:

Fact 5.2 *For each $c > 0$, there is a constant r, so that if $k \geq r \log_{1/P} n$, then for each i,*

$$(1) \quad \Pr\left(n_i \geq (1+\mu)\cdot k \cdot P\right) \leq n^{-c} ;$$

$$(2) \quad \Pr\left(n_i \leq (1-\frac{\mu}{3})\cdot k \cdot P\right) \leq n^{-c} .$$

Clearly Part (1) of Fact 5.2 implies the first idealized property.

Denote by $I(i,j,q)$ the event in which clones i,j overlap for length q. The following fact follows similarly to Fact 5.2.

Fact 5.3 *For each i,j,*

$$(1) \ \Pr(n_{i,j} \leq (1-\mu)\cdot km(q) \mid I(i,j,q)) \leq e^{-\mu^2 km(q)/2} ;$$

$$(2) \ \Pr(n_{i,j} \geq (1+\frac{\mu}{3})\cdot km(q) \mid I(i,j,q)) \leq e^{-\mu^2 km(q)/27} .$$

Note that $m(0) = kP^2$. Thus Part (2) of Fact 5.3 immediately implies:

Fact 5.4 *For each $c > 0$, there is a constant r, so that if $k \geq r \log_{1/P} n$, then for each i,j,*

$$\Pr\left(n_{i,j} \geq (1+\frac{\mu}{3})\cdot kP^2 \mid I(i,j,0)\right) \leq n^{-c} .$$

Clearly Fact 5.4, Part (2) of Fact 5.2, and the fact that by definition $0 \leq \mu \leq 1$, imply the third idealized property.

Define $t = (1-\mu)m(w)$. Define $D(i,j)$ as the event in which clones i,j overlap for length $\geq w$. The following lemma follows from Fact 5.3 and the choice of μ.

Lemma 5.5 *For each $c > 0$, there is an r so that if $k \geq r \log_{1/P} n$, then for each i,j,*

$$(1) \quad \Pr\left((n_{i,j} \leq t \mid D(i,j)\right) \leq n^{-c} ;$$

$$(2) \quad \Pr\left((n_{i,j} \geq t \mid I(i,j,0)\right) \leq n^{-c} .$$

Clearly Lemma 5.5 establishes the fifth idealized property.

Finally, since $m(w) = 1 + e^{-\lambda(2-w)} - 2e^{-\lambda}$, we immediately get:

Fact 5.6 *If w is sufficiently large, and P is a constant less than $1/2$, then $\frac{t}{k} = P^s$, where $1 \leq s < 2 - \log_{1/P}(2(1+\mu)^2)$.*

Fact 5.6 establishes the fourth idealized property. Note that if $k = \Omega(\log^2 n)$, then the statements in the above lemmas hold also for $\mu = O(1/\log n)$.

6 Computational Results

This section describes preliminary computational results for our implementation of the data tree and hashing algorithms, where occurrences of features are as defined by the Poisson Features problem (see Section 5). In the full paper we describe computational results also for the case that the features are as defined by the Restriction Sites problem.

Implementation Define l to be the integer closest to $\log_{1/p} n$. The data tree algorithm is implemented with minor modifications to the description given in Section 3. In addition to comparing all the pairs of vectors found together in nodes at depth l, the algorithm compares all the pairs of vectors in nodes closer to the root whenever the number of vectors in a node falls below a small constant, currently 20. The procedure also avoids testing the same feature multiple times along a branch, or multiple times at a single node. Both the one-shot data tree and the one-shot hashing scheme avoid all duplicate comparisons.

Both algorithms require an estimate of t, the number of features which two highly overlapping clones will share. In practical situations it may be difficult to determine this value, and therefore we have used a simple sampling technique to aid in this determination. Recall that in the biological problem n unit-length clones are positioned in the interval $[0, N]$. The expected number of clones covering any particular point, n/N, is determined by the underlying geometry of the problem and is independent of the number of features present or the accuracy with which they are measured. We take advantage of this fact to determine an appropriate value for t as follows.

1. State the number, C, of clones which are expected to have high overlap with any given clone.
2. Compute the intersection of $h * n$ randomly chosen pairs where h is a small constant.
3. Choose t so that $h * C$ of the sample pairs have an intersection of t or more.

The estimate of the number of overlapping clones, C, is the main input parameter to these routines. The number of one-shot repetitions must also be specified. The number of sampling comparisons is $20 * n$ in all the experiments discussed below.

Problem Parameters Problem instances were generated with Poisson features as discussed in Section 5, with n/N equal to 10. 500 features were used, and λ was 0.04, thus giving each clone approximately

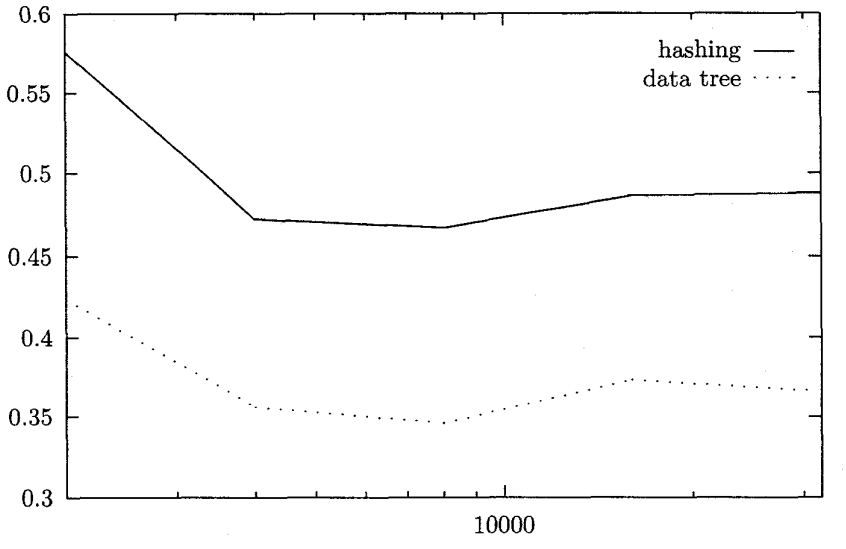

Figure 1: Fraction of physically overlapping pairs of clones found by the two one-shot algorithms. The log-scale horizontal axis indicates the number of clones. Data points are plotted for 2,000, 4,000, 8,000, 16,000, and 32,000 clones.

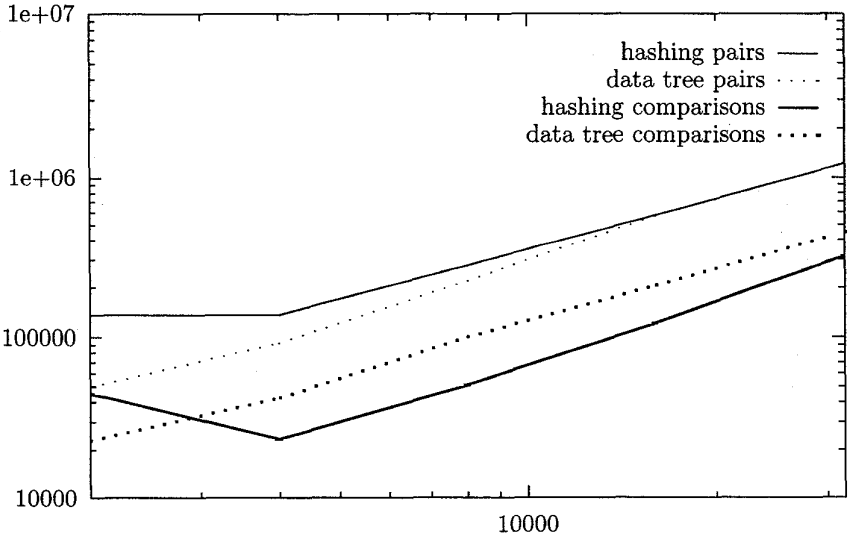

Figure 2: Measures of work. The number of pairs found together by the two algorithms, and the number of comparisons actually made when avoiding duplications. Both axes are log-scale. The number of pairs found together by the hashing algorithm is consistent with a growth rate of $n^{1.1}$. For the data tree this figure is approximately $n^{1.2}$.

20 positive features. Unless otherwise noted, the results presented are averaged over three problem instances, and the desired number of highly overlapping clones used as an input parameter C was 10.

Algorithm Performance Figure 1 shows the success rate of the one-shot algorithms.[3] This is defined as the number of physically overlapping pairs of clones found with an intersection of t or more, expressed as a fraction of the total number of physically overlapping pairs in the problem instance. Hence, for the data tree this is the number of physically overlapping pairs found together at a leaf with an intersection of t or more, and for the hashing algorithm this is the number of physically overlapping pairs with a matching l-tuple and an intersection of t or more. The number of non-physically overlapping pairs found by the algorithms is negligible and omitted. The relative behavior illustrated in Figure 1 was confirmed in a preliminary manner for several k and t values.

The abrupt drop in both algorithms' success rate in going from 2,000 to 4,000 clones is due to a jump in l from 2 to 3. The larger l-tuple size decreases the probability of linking overlapping clones for the one-shot hashing procedure, and the increase in the data tree's depth likewise decreases its one-shot success rate.

We expect that as n increases asymptotically the probability with which the one-shot hashing algorithm finds overlapping pairs will decrease below that of the one-shot data tree in accord with the fact that the probability of success of the one-shot hashing algorithm (Lemma 4.1) decreases more rapidly with n than the probability of success of the one-shot data tree (Lemma 3.1).

We also expect that after repeated iterations both schemes will find the same number of overlapping pairs, and this is indeed what we observe.

Figure 2 shows the amount of work done by the two one-shot algorithms: the number of pairs of clones found together either in leaf nodes or in hash buckets, and the number of non-duplicate pairs whose intersection is actually calculated. The growth rate is nearly linear. The lower two curves indicate that a considerable amount of work is avoided by skipping duplicate comparisons. The value of s implied by the sampling procedure - slightly over 1.2 - is in good accord with the observed growth rates. The observed running times are also consistent. In terms of absolute magnitude, the hashing scheme required approximately 50 seconds for 16,000 clones, and the data tree required roughly 60 seconds.

[3]Note that the actual success probability of the two one-shot algoithms is about double than that indicated by Figure 1, since the goal of the algorithms is not to find all physically overlapping pairs, but only those which overlap for a sufficiently large interval so that they have at least t features in common. The number of such pairs is about half the number of the physically overlapping pairs.

References

[1] A.V. Aho and J.D. Ullman. Optimal Partial-Match Retrieval When Fields are Independently Specified. *ACM Transactions on Database Systems.* 4:168-179. 1979.

[2] J.L. Bentley. Multidimensional Divide and Conquer. *Communications of the ACM.* 23:214-229. 1980.

[3] W.A. Burkhard. Hashing and Trie Algorithms for Partial Match Retrieval. *ACM Transactions on Database Systems.* 1:175-187. 1976.

[4] K. Clarkson. Fast algorithms for the All-Nearest-Neighbors Problem. *Proceedings of the 24th Annual Symposium on the Foundations of Computer Science.* 226-232. 1983.

[5] D. Dolev, Y. Harari, N. Linial, N. Nisan, and M. Parnas. Neighborhood Preserving Hashing and Approximate Queries. *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms.* 251-259. 1994.

[6] J.H. Friedman, F. Baskett, and L.J. Shustek. An Algorithm for Finding Nearest Neighbors. *IEEE Transactions on Computers.* 1000-1007. October, 1975.

[7] J.H. Friedman, J.L. Bentley, and R.A. Finkel. An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software.* 3:209-226. 1977.

[8] P. Klier and R.J. Fateman. On Finding the Closest Bitwise Matches in a Fixed Set. *ACM Transactions on Mathematical Software.* 17:88-97. 1991.

[9] Y. Kohara, K. Akiyama, and K. Isono. The Physical Map of the Whole E. coli Chromosome: Application of a new Strategy for Rapid Analysis and Sorting of a Large Genomic Library. *Cell* 50:495-508. 1987.

[10] D.O. Nelson and T.P. Speed. Statistical Issues in Constructing High Resolution Physical Maps. *Statistical Science.* 9:334-354. 1994.

[11] M.V. Olson, University of Washington. Private Communication.

[12] M.V. Olson, J.E. Dutchik, M.Y. Graham, G.M. Brodeur, C. Helms, M. Frank, M. MacCollin, R. Scheinman, and T. Frank. Random-Clone Strategy for Genomic Restriction Mapping in Yeast. *Proceedings of the National Academy of Science USA* 83:7826-7830. 1986.

[13] R.L. Rivest. On the Optimality of Elias's Algorithm for Performing Best-Match Searches. *Information Processing.* 74:678-681. 1974.

[14] R.L. Rivest. Partial-Match Retrieval Algorithms. *SIAM Journal of Computing.* 5:19-49. 1976.

[15] P.M. Vaidya. An $O(n \log n)$ Algorithm for the All-Nearest-Neighbors Problem. *Discrete and Computational Geometry.* 4:101-115. 1989.

[16] P.N. Yianilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms.* 311-321. 1993.