BIOINFORMATICS

Vol. 20 no. 13 2004, pages 2101–2112
doi:10.1093/bioinformatics/bth210

Efficient selection of unique and popular oligos for large EST databases†

Jie Zheng[1], Timothy J. Close[2], Tao Jiang[1] and Stefano Lonardi[1,*]

[1]Department of Computer Science and Engineering and [2]Department of Botany and Plant Sciences, University of California, Riverside, CA 92521, USA

Received on September 29, 2003; revised on January 2, 2004; accepted on February 10, 2004
Advance Access publication April 1, 2004

ABSTRACT

Motivation: Expressed sequence tag (EST) databases have grown exponentially in recent years and now represent the largest collection of genetic sequences. An important application of these databases is that they contain information useful for the design of gene-specific oligonucleotides (or simply, oligos) that can be used in PCR primer design, microarray experiments and genomic library screening.

Results: In this paper, we study two complementary problems concerning the selection of short oligos, e.g. 20–50 bases, from a large database of tens of thousands of ESTs: (i) selection of oligos each of which appears (exactly) in one unigene but does not appear (exactly or approximately) in any other unigene and (ii) selection of oligos that appear (exactly or approximately) in many unigenes. The first problem is called the unique oligo problem and has applications in PCR primer and microarray probe designs, and library screening for gene-rich clones. The second is called the popular oligo problem and is also useful in screening genomic libraries. We present an efficient algorithm to identify all unique oligos in the unigenes and an efficient heuristic algorithm to enumerate the most popular oligos. By taking into account the distribution of the frequencies of the words in the unigene database, the algorithms have been engineered carefully to achieve remarkable running times on regular PCs. Each of the algorithms takes only a couple of hours (on a 1.2 GHz CPU, 1 GB RAM machine) to run on a dataset 28 Mb of barley unigenes from the HARVEST database. We present simulation results on the synthetic data and a preliminary analysis of the barley unigene database.

Availability: Available on request from the authors.

Contact: stelo@cs.ucr.edu

1 INTRODUCTION

Expressed sequence tags (ESTs) are partial sequences of expressed genes, usually 200–700 bases long, which are generated by sequencing from one or both ends of cDNAs. The information in an EST allows researchers to infer functions of the gene based on similarity to genes of known functions, source of tissue and timing of expression, and genetic map position. EST sequences have been widely accepted as a cost-effective method to gather information about the majority of expressed genes in a number of systems. They can be used to accelerate various research activities, including map-based cloning of genes that control traits, comparative genome analysis, protein identification and numerous methods that rely on gene-specific oligonucleotides (or oligos, for short) such as the DNA microarray technology.

Owing to their utility, speed with which they may be obtained and the low cost associated with this technology, many individual scientists and large genome sequencing centers have been generating hundreds of thousands of ESTs for public use. EST databases have been growing exponentially since the first few hundreds of sequences obtained in the early 1990s by Adams *et al*. (1991), and now they represent one of the largest collection of genetic sequences. As of April 2004, the number of sequences deposited in NCBI's dbEST (Boguski *et al*., 1993) has reached 20 million sequences.

With the advent of whole genome sequencing, it may appear that ESTs have lost some of their appeal. However, the genomes of many organisms that are important to society, including the majority of crop plants, have not been fully sequenced yet, and the prospects for large-scale funding to support the sequencing of any but a few in the immediate future is slim to none. In addition, several of our most important crop plants have genomes that are of daunting sizes and present special computational challenges because they are comprised mostly of highly repetitive DNA. For example, the *Triticeae* (wheat, barley and rye) genomes, each with a size of about 5×10^9 bp per haploid genome (this is about twice the size of maize, 12 times the size of rice and 35 times the size of the *Arabidopsis* genomes), are too large to seriously consider whole genome sequencing at the present time.

Of the many EST databases, we have been especially interested in the dataset of barley (*Hordeum vulgare*). Barley is a premiere model for *Triticeae* plants due to its diploid

*To whom correspondence should be addressed.

†A preliminary version of this work was presented at the *Symposium on Combinatorial Pattern Matching*, Morelia, Mexico, and included in its *Proceedings*, pp. 273–283, LNCS 2676, Springer (2003).

genome and a rich legacy of mutant collections, germplasm diversity, mapping populations (see http://www.css.orst.edu/barley/nabgmp/nabgmp.htm) and the recent accumulation of other genomics resources such as bacterial artificial chromosome (BAC) (Yu *et al*., 2000) and cDNA libraries (Close *et al*., 2001; Michalek *et al*., 2002). Nearly 400 000 publicly available ESTs derived from barley cDNA libraries are present in dbEST. About 350 000 of these sequences have been quality-trimmed, cleaned of vector and other contaminating sequences, preclustered using the software TGICL (http://www.tigr.org/tdb/tgi/software/) and clustered into final assemblies of 'contigs' (i.e. overlapping EST sequences) and 'singletons' (i.e. non-overlapping EST sequences) using CAP3 (Huang and Madan, 1999). The collection of the singletons and consensus sequences of the contigs, called unigenes, form our main dataset. As of April 2004, the collection had 53 804 EST unigenes of a total of 35 689 750 bases. This dataset can be obtained from http://harvest.ucr.edu/ using the HARVEST viewer. Hereafter, we will use the term EST to denote a unigene obtained from the assembly of one or more ESTs.

In this paper, we study two computational problems arising in the selection of short oligos (e.g. 20–50 bases) from a large EST database. One is to identify oligos that are unique to each EST in the database. The other is to identify oligos that are popular among the ESTs. More precisely, the unique oligo problem asks for the set of all oligos each of which appears (exactly) in one EST sequence but does not appear (exactly or approximately) in any other EST sequence, whereas the popular oligo problem asks for a list of oligos that appear (exactly or approximately) in the largest number of ESTs[1].

A unique oligo can be thought of as a 'signature' that distinguishes an EST from all the others. Unique oligos are particularly valuable as locus-specific PCR primers for placement of ESTs at single positions on a genetic linkage map, on microarrays for studies of the expression of specific genes without signal interference from other genes, and to probe genomic libraries (Han *et al*., 2000) in search of specific genes.

Popular oligos can be used to screen efficiently large genomic libraries. They could allow one to simultaneously identify a large number of genomic clones that carry expressed genes using a relatively small number of (popular) probes and thus save considerable amounts of money. In particular for the database under analysis, it has been shown previously by a number of independent methods that the expressed genes in *Triticeae* are concentrated in a small fraction of the total genome. In barley, this portion of the genome, often referred to as the gene-space, has been estimated to be only 12% of the total genome (Barakat *et al*., 1997). If this is indeed true, then 12% of the clones in a typical BAC library would carry expressed genes, and therefore also the vast majority of barley genes could be sequenced by focusing only on this 12% of the genome. An efficient method to reveal the small portion of BAC clones derived from the gene-space has the potential for tremendous cost savings in the context of obtaining the sequences of the vast majority of barley genes. The most commonly used barley BAC library has a 6.3-fold genome coverage, 17-filter set with a total of 313 344 clones (Yu *et al*., 2000). This number of filters is inconvenient and costly to handle, and the total number of BAC clones is intractable for whole genome physical mapping or sequencing. However, a reduction in this library to a gene-space of only 12% of the total would make it fit into two filters that would comprise only ∼600 Mb. This is about the same size as the rice genome, which has been recently sequenced. A solution for the popular oligo problem should make it possible to develop an effective greedy approach to BAC library screening, enabling a very inexpensive method of identifying a large portion of the BAC clones from the gene-space. This would also likely accelerate progress in many crop plant and other systems that are not being considered for whole genome sequencing.

1.1 Our contribution

In this paper, we present an efficient algorithm to identify all unique oligos in the ESTs and an efficient heuristic algorithm to enumerate the most popular oligos. Although the unique and popular oligos problems are complementary in some sense, the two algorithms are very different because unique oligos are required to appear in the ESTs while the popular oligos are not. In particular, the heuristic algorithm for popular oligos is much more involved than that for unique oligos, although their (average) running times are similar. The algorithms combine well-established algorithmic and data structuring techniques such as hashing, approximate string matching and clustering, and take advantage of the facts that (i) the number of mismatches allowed in these problems is usually small and (ii) we usually require a pair of approximately matched strings to share a long common substring [called a common factor in Rahmann (2002)]. These algorithms have been carefully engineered to achieve satisfactory speeds on PCs, by taking into account the distribution of the frequencies of the words in the input EST dataset. For example, running each of the algorithms for the barley EST dataset from HARVEST takes only a couple of hours (on a 1.2 GHz AMD machine). This is a great improvement over other brute-force methods, like the ones based on BLAST[2]. Simulations results show that the number of missed positives by the heuristic algorithm for popular oligos is very limited and can be controlled very effectively by adjusting the parameters.

[1]Note that, a popular oligo does not necessarily have to appear exactly in any EST.

[2]For example, one can identify unique oligos by repeatedly running BLAST for each EST sequence against the entire dataset. This was the strategy previously employed by the HARVEST researchers.

1.2 Previous related work

The problem of finding infrequent and frequent patterns in sequences is a common task in pattern discovery. A quite large family of pattern discovery algorithms has been proposed in the literature and implemented in software tools. Without pretending to be exhaustive, we mention MEME (Bailey and Elkan, 1995), PRATT (Jonassen *et al*., 1995; Jonassen, 1997), TEIRESIAS (Rigoutsos and Floratos, 1998), CONSENSUS (Hertz and Stormo, 1999), GIBBS SAMPLER (Lawrence *et al*., 1993; Neuwald *et al*., 1995), WINNOWER (Pevzner and Sze, 2000; Keich and Pevzner, 2002), PROJECTION (Tompa and Buhler, 2001; Buhler and Tompa, 2002), VERBUMCULUS (Apostolico *et al*., 2003, 2004), MITRA (Eskin and Pevzner, 2002), among others. Although these tools have been demonstrated to perform very well on small and medium size datasets, they cannot handle large datasets such as the barley EST dataset that we are interested in. In particular, some of these tools were designed to attack the 'challenge' posed by Pevzner and Sze (2000), which is in the order of a few kb. Among the more general and efficient tools, we tried to run TEIRESIAS on the 28 Mb barley EST dataset on an 1.2 GHz Athlon CPU with 1 GB of RAM, without being able to obtain any result (probably due to lack of memory).

The unique oligo problem has been studied in the context of probe design (Li and Stormo, 2001; Rahmann, 2002; Rouillard *et al*., 2002). The algorithms in (Li and Stormo, 2001; Rouillard *et al*., 2002) consider physical and structural properties of oligos and are very time-consuming. [The algorithm in Rouillard *et al*. (2002) also uses BLAST]. A very recent algorithm by Rahmann (2002) is, on the other hand, purely combinatorial. It uses suffix arrays instead of hash tables, and requires ∼50 h for a dataset of 40 Mb on a high-performance Compaq Alpha machine with 16 GB of RAM. However, his definition of unique oligos is slightly different from ours (to be given in the next section).

The rest of the paper is organized as follows. Section 2 defines the unique and popular oligo problems formally. The algorithms are presented in Section 3. Experimental results on the barley EST dataset and simulation results can be found in Section 4. In Section 5, we draw some concluding remarks.

2 PRELIMINARIES

We denote the input dataset as $X = \{x_1, x_2, \ldots, x_k\}$, where the generic string x_i is an EST sequence over the alphabet $\Sigma = \{\text{A,C,G,T}\}$ and k is the cardinality of the set. Let n_i denote the length of the i-th sequence, $1 \leq i \leq k$. We set $n = \sum_{i=1}^{k} n_i$, which represents the total size of the input. A string (or oligo) from Σ is called an l-mer if its length is l.

Given a string x, we write $x_{[i]}$, $1 \leq i \leq |x|$, to indicate the i-th symbol in x. We use $x_{[i,j]}$ as a shorthand for the substring $x_{[i]}, x_{[i+1]}, \ldots, x_{[j]}$ where $1 \leq i \leq j \leq n$, with the convention that $x_{[i,i]} = x_{[i]}$. Substrings in the form $x_{[1,j]}$ correspond to the prefixes of x, and substrings in the form $x_{[i,n]}$ to the suffixes of x. A string y occurs at position i of another string x if $y_{[1]} = x_{[i]}, \ldots, y_{[m]} = x_{[i+m-1]}$, where $m = |y|$. For any substring y of x, we denote by $f_x(y)$ the number of occurrences of y in x. $f_X(y)$ denotes the total number of occurrences of y in $x_1, \ldots, x_k$.

The color-set of y in the set $X = \{x_1, x_2, \ldots, x_k\}$ is a subset $\text{col}(y) = \{i_1, i_2, \ldots, i_l\}$ of $\{1, 2, \ldots, k\}$ such that if $i_j \in \text{col}(y)$ then y occurs at least once in x_{i_j}. We also say that y has colors $i_1, i_2, \ldots, i_l$. The number of colors, l, of y is denoted as $c_X(y)$. Clearly $f_X(y) \geq c_X(y)$.

Given two strings x and y of the same length, we denote by $H(x, y)$ the Hamming distance between x and y, i.e. the number of mismatches between x and y. If $H(x, y) \leq d$, we say that x d-matches y and x is a d-mutant of y. The set of all the strings that d-match x is called the d-neighborhood of x. The notion of occurrences and colors can be extended to d-occurrence and d-colors by allowing up to d mismatches. If a string y has d-mutants at j distinct positions in a string x, we say that y has j d-occurrences in x. If a string y has at least one d-occurrence in each of j sequences in X, we say that y has j d-colors in X.

In the context of DNA hybridization, most papers define the specificity of an l-mer in terms of its mismatches to the length-l substrings of target sequences, although some also consider its physical and structural characteristics such as melting temperature, free-energy, GC-content, and secondary structure (Li and Stormo, 2001; Rouillard *et al*., 2002). In 2002, Rahman took a more optimistic approach and used the length of the longest common substring [called the longest common factor (LCF)] as a measure of unspecificity. Given the nature of our target applications, we will take a conservative approach in the definitions of unique and popular oligos.

DEFINITION 2.1. *Given the set $X = \{x_1, x_2, \ldots, x_k\}$ of ESTs and integers l and d, a* unique oligo *is an l-mer y such that y occurs in one EST and the number of d-colors of y in X is exactly one. In other words, y appears exactly in some EST but does not appear approximately in any other EST.*

Suppose that strings x and y have the same length. For any given constants c, d, we say that string x (c, d)-matches string y if x and y can be partitioned into substrings as $x = x_1x_2x_3$ and $y = y_1y_2y_3$ with $|x_i| = |y_i|$ such that (i) $|x_2| = c$, (ii) $x_2 = y_2$ and (iii) the string x_1x_3 d-matches the string y_1y_3. In the above partition, we call x_2 a core in the (c, d)-match between x and y. [Note that a (c, d)-match may have many cores]. The notion of d-occurrences and d-colors can be easily extended to (c, d)-occurrences and (c, d)-colors.

DEFINITION 2.2. *Given the set $X = \{x_1, x_2, \ldots, x_k\}$ of ESTs and integers l, d, c and T, a* popular oligo *is an l-mer y such that the number of (c, d)-colors of y in X is greater than or equal to T. In other words, the l-mer y appears approximately in at least T ESTs.*

The use of pooled oligo probes for BAC library screening (Han *et al.*, 2000) generally have lengths from 24 to 40 bases. Given this range and based on discussion with researchers from the Triticeae community, we consider $l = 33$ and $d = 5$ in the unique oligo problem. In the popular oligo problem, we consider $d = 1, 2, 3$ and $c = 20$ and $l = 36$.

3 METHODS

Our goal is to determine unique and popular oligos for a given set X of EST sequences. Although the objectives of the two problems seem complementary, our algorithms are quite different. The algorithm for the popular oligo problem turns out to be much more involved because popular oligos are not necessarily contained in the ESTs. Nevertheless, both algorithms share some common strategies such as the idea of separating dissimilar strings as early as possible to reduce the search space. To achieve this in the popular oligo problem, we first find cores (i.e. c-mers) that appear exactly in at least two ESTs. Then we cluster all length-l substrings of the ESTs by their cores of length c using a hash table, and then, within each cluster, we cluster again the l-mers based on Hamming distance between regions flanking the cores. Candidate popular oligos are then enumerated from the small groups resulted from the two clusterings, and their number of colors are counted. For the unique oligo problem, the notion of cores, however, does not exist. On the other hand, due to the small number of mismatches allowed (relative to l), two l-mers that d-match each other must contain a pair of substrings that 1-match each other. Such substrings are called seeds. We can thus cluster the ESTs by their seeds using a dictionary. For each cluster, we compare the ESTs in the cluster by counting mismatches in the regions flanking the seeds. The details are given below.

3.1 Unique oligos

Recall that the unique oligo problem is to identify length-l substrings of the ESTs that have exactly one d-color in the dataset X, for a given value of d. Our strategy is first to eliminate all these l-mers that cannot be unique oligos. The algorithm is based on the following observation. Assume that x and y are two l-mers such that $H(x, y) \leq d$. Divide both x and y into $t = \lfloor d/2 \rfloor + 1$ substrings. That is $x = x_1 x_2 \cdots x_t$ and $y = y_1 y_2 \cdots y_t$, where the length of each substring is $q = \lceil l/t \rceil$, except possibly for the last one. In practice, one can always choose l and d so that l is a multiple of t and hence x and y can be decomposed into t substrings of length q, which we call seeds. It is easy to see that since $H(x, y) \leq d$, at least one of the seeds of x has at most one mismatch with the corresponding seed of y.

Using this idea, we design an efficient two-phase algorithm. In the first phase, we cluster all the possible seeds from the ESTs into groups such that within each group, a seed has no more than one mismatch with the other seeds. In the second phase, we check whether extending the flanking regions of a seed would result in a d-match with the corresponding extension of any other seed in the same group. If so, the l-mer given by this extension is not a unique oligo.

Phase 1. (Hit) We file all q-mers (seeds) from the input ESTs into a dictionary with 4^q entries. (If 4^q cannot fit in the main memory, one could use a hash table of an appropriate size.) Each entry of the table points to a list of locations where the q-mer occurs in the EST sequences. Using the table we can immediately locate identical seeds in the ESTs.

Phase 2. (Extension) We compare the corresponding flanking regions of each pair of matching seeds to determine whether they can be extended to a pair of l-mers that d-match each other. Here, we also collect seeds that have exactly one mismatch with each other as follows. For each table entry corresponding to a seed y, we record a list of other seeds that have exactly one mismatch with y, by looking up table entries that correspond to all the 1-mutants of y. This list is called a mutant list of y. We examine all the seeds in the mutant list, and compare the flanking regions of the q-mers and that of y in the same way as we did for identical seeds, except that now the cutoff for the number of mismatches in the flanking regions is $d - 1$.

The algorithm is summarized in Figure 1.

Time complexity. Suppose that the total number of bases in X is n. The time complexity of phase one is simply $\Theta(qn+4^q)$, where the second term reflects the time needed to initialize the hash table. The time complexity of phase two depends on the distribution of the number of seeds filed into each table entry. Simply speaking, if the distribution is more or less uniform (which is the case in our experiment) and each table entry contains $r \approx n/4^q$ identical seeds, the number of comparisons within the table entry is $O(r^2)$. The number for comparisons for each mutant lists of size $3q$ is $O(qr^2)$. Each comparison requires extension of the seeds and takes $2(l - q)$ time. Since there are 4^q entries in the table, the overall time complexity is $O[(l-q)qr^2 4^q]$. Given the exponential dependency on q, one needs to make sure that q is not too large before using the algorithm. (Again, in our experiment on the barley dataset, $l = 33, d = 5$ and $q = 11$.)

In practicing EST data analysis, we also need to consider the reverse complementary strand of each EST, which implies more stringency in the choice of unique oligos. The above algorithm can be easily modified to take into account reverse complementary EST strands without a significant increase in complexity.

3.2 Popular oligos

Recall that the objective is to find all l-mers that have sufficiently large number of (c, d)-colors in X. Since popular oligos are not required to appear exactly in the EST sequences,

```
UNIQUE-OLIGO-SELECTION(X, l, m)
Input:  EST sequences X = {x_1, x_2, ..., x_k}
        l: length of the oligos to be reported
        d: maximum number of mismatches for non-unique oligos
Output: Mark each unique l-mers in X
1  t, q ← ⌊d/2⌋ + 1, ⌈l/t⌉
2  table ← HIT(X, q)
3  EXTENSION(X, t, q, table, d)
```

```
COMPARE(table[i][j], table[i][k], d)
1  q_1 ← the q-mer located at table[i][j]
2  q_2 ← the q-mer located at table[i][k]
3  for each pair of l-mers that contain q_1 and q_2 as seeds respectively do
4        c ← H(q_1, q_2)
5        if c ≤ d then
6           mark the two l-mers as "non-unique"
```

```
HIT(X, q)
1  for i ← 1 to 4^q do
2     initialize table[i]
3     index[i] ← 0
4  for i ← 1 to k do
5     for j ← 1 to n_i − q + 1 do
6        key ← MAP(x_{i,[j,...,j+q−1]})
7        table[key][index[key]] ← ⟨i, j⟩
8        index[key] ← index[key] + 1
9  return table
```

```
EXTENSION(X, t, q, table, m)
1  for i ← 1 to 4^q do
2     List mut ← mutant list of table[i]
3     len ← # of records in table[i]
4     for j ← 1 to len do
5        for k ← j + 1 to len do
6           COMPARE(table[i][j], table[i][k], m)
7        for h ← 1 to # of records in mut do
8           mutlen ← # of records in table[mut[h]]
9           for k ← 1 to mutlen do
10             COMPARE(table[i][j], table[mut[h]][k], m)
```

```
MAP(string S)
1  map S into an integer X with function f : {A,C,G,T} → {0, 1, 2, 3}
2  return X
```

Fig. 1. The algorithm for identifying unique oligos.

the algorithm based on exhaustive enumeration is computationally very expensive. In fact, one can easily show that the problem is NP-hard in general.

The 'exhaustive' algorithm considers all l-mers occuring in the ESTs and for each l-mer, enumerates all its (c,d)-mutants and count their number of (c,d)-colors. However, the number of (c,d)-mutants of an l-mer over the DNA alphabet is more than $\binom{l-c}{d}3^d$. As a consequence, the exhaustive method becomes computionally impractical due to its memory requirement as soon as the input size reaches the order of hundreds of thousands of bases (like the barley dataset)[3].

[3]When $d = 3$ and $c = 20$, $\binom{l-c}{d}3^d = \binom{13}{3}3^3 = 7722$ for the barley dataset. Hence, the straightforward algorithm would have to count the number of colors for about $7722 \cdot 28 \times 10^6 = 217 \times 10^9$ l-mers.

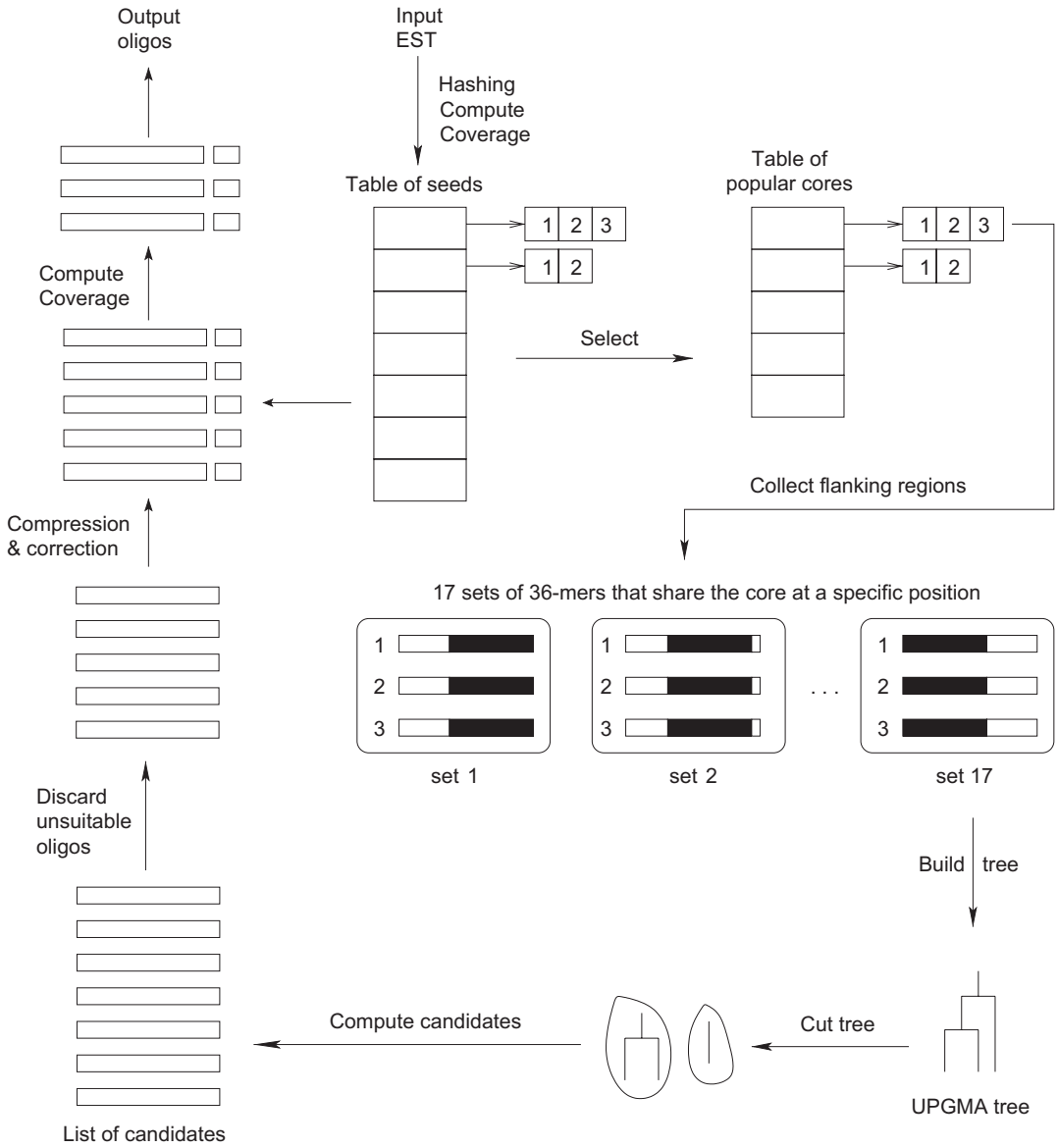

Fig. 2. An overview of the algorithm for selecting popular oligos. For convenience of illustration, the length of the oligos is assumed to be $l = 36$, and the length of the cores is assumed to be $c = 20$.

We can reduce the search space using the same idea as in the algorithm for unique oligos, except here that the role of seeds is played by cores. Observe that, if a (popular) oligo has (c, d)-occurrences in many ESTs, many of these ESTs must contain length-l substrings that share common cores. Based on this observation, we propose a heuristic strategy that first clusters the l-mers in the EST sequences into groups by their cores, and then enumerates candidate l-mers by comparing the members of each cluster in a hierarchical way.

An outline of the algorithm is illustrated in Figure 2. Here, we determine the popularity of the cores (i.e. length-c substrings) from the ESTs in the first step. For each popular core, we consider extension of the cores into l-mers by including flanking regions and cluster them using a well-known hierarchical clustering method, called unweighted pair group method with arithmetic mean (UPGMA) (Swofford, 2002). We recall that UPGMA builds the tree bottom-up in a greedy fashion by merging groups (or subtrees) of data points that have the smallest average distance. Based on the clustering tree, we compute the common oligos shared by the l-mers by performing set intersection. These common oligos shared by many l-mers become candidate popular oligos. Finally, we count the number of colors of these candidates, and output the oligos with at least T colors. A more detailed description

is given below. A complete example of the algorithm on a toy dataset is also given in the section 3.2.1.

Phase 1. We compute the number of colors for all c-mers in the ESTs to determine whether they could be candidate cores for popular l-mers, using a hash table. According to our definition, a popular oligo should have a popular core. We therefore set a threshold T_c on the minimum number of colors of each popular core, depending on T, c, l and X. All cores that have a number of colors below T_c are filtered out, and considered 'unpopular'. However, since an l-mer can (c,d)-match another l-mer with any of its $l-d+1$ cores, it is possible that we might miss some popular oligos that critically depend on unpopular cores. The parameter T_c represents a tradeoff between precision and efficiency. We will show in Section 4 the effect of changing T_c on the output. We will see that in practice we might miss only a negligible number of popular oligos.

Phase 2. Here we collect the substrings flanking the popular cores. For each popular core, we construct $l-c+1$ sets of substrings, one for each possible extension of the core into an l-mer. Each set contains substrings of length $l-c$ constructed by concatenating the left and right flanking regions.

Phase 3. For each set of flanking substrings, we would like to identify all $(l-c)$-mers that have d-occurrences in many of these substrings. In order to achieve this efficiently, we first cluster the substrings according to their mutual Hamming distance using the well-known hierarchical clustering method UPGMA. In the process of building the clustering tree, whenever the Hamming distance between a pair of leaves in the tree is zero we compress the distance matrix by combining the identical strings into one entry. This significantly reduces the running time not only because the tree becomes smaller, but also because the number of common d-mutants of two different l-mers is much less than that of two identical ones. As we can see later, a significant proportion of the running time is spent on the intersection of the sets of d-mutants. Compressing the distance matrices avoids intersecting identical sets of d-mutants, which is expensive and also useless. We then create a set of d-mutants for each substring represented at the leaves and traverse the tree bottom-up. At each internal node u, we compute the intersection of the two sets attached to the children, using a hash table based on the hash function described in (Wesselink, 2002). This intersection represents all the $(l-c)$-mers that have d-occurrences in all the leaves (substrings) under the node u. As soon as the intersetion of some internal node, say u, becomes empty, we cut the tree at u. Each subtree represents a cluster, and the set of $(l-c)$-mers attached to the root are the elements of the cluster. The size of the cluster is therefore equal to the number of leaves in the tree. Because small clusters are unlikely to contain popular oligos, we discard all trees whose size is smaller than $C_{\min}$. At the end of this process, we obtain a collection of sets of $(l-c)$-mers, each of which, together with the popular core, represents a candidate popular oligo.

Phase 4. Given the candidate popular oligos, we need to count their number of (c,d)-colors. Before counting, we radix-sort the candidates and remove duplicates. More precisely, due to the possibly very large number of candidates and duplicates (as in the barley case), we sort the candidates in several stages as follows. For each core, we radix-sort all the candidates derived from the core and remove duplicates. Then, we merge the sorted list into the sorted list containing all the candidates from those cores that have already been processed.

Time complexity. Phase 1 costs $O(cn)$ time. In phase 2, if the number of popular cores selected in the first step is p and the average number of occurrences of the cores is r, this phase costs $O[nr(l-c)]$. For phase 3, the time for building a UPGMA tree, including the computation of the distance matrix, is $O[(l-c)r^2]$, where r stands for the number of strings to be clustered. Since a (binary) UPGMA tree with r leaves has $2r-1$ nodes, the time for traversing (and pruning) the tree is $O\left[r\binom{l-c}{d}3^d\right]$, where $\binom{l-c}{d}3^d$ is the number of d-mutants at each leaf. Finally for phase 4, if the total number of candidates is m, counting the colors for the candidates, excluding the time for radix-sort, costs time $O[rm(l-c)]$.

3.2.1 An example of the popular oligo algorithm We show a small example to illustrate each step of the algorithm. Again, let l denote the length of oligos, c the length of cores, d the maximum number of mismatches between two oligos and T_c the threshold on the minimum number of colors of popular cores. In this toy example, $l=8, c=5, d=1, T_c=3$. The input comprises four artificial EST sequences as shown in Figure 3. `EST1` and `EST3` are highly similar, which represents the similarity between some of the EST sequences in real data.

Phase 1. A total of 7 cores out of 148 possible cores are selected as popular cores. Each entry of the hash table points to a list of positions of the occurrences of the core in the input ESTs.

Phase 2. We collect the flanking regions for the seven cores. Figure 4 shows the four sets of flanking regions for `AAGGC`. Each string in the set is obtained by concatenating the left and the right flanking regions. Note that the fourth set has one fewer element than the other three sets. The reason is that the core `AAGGC` occurs at the right boundary of `EST0`, and therefore has a shorter flanking region.

Phase 3. We cluster the flanking regions using UPGMA (Fig. 5). In Figure 6, we show the clusters for set 2 of the core `AAGGC`. Observe that the Hamming distance between entries 2 and 4 is zero and therefore distance matrix is compressed by combining the identical strings into one entry. We then need to enumerate all the 1-mutants of the strings denoted by the leaves of the trees, i.e. `AAA`, `TGG` and `GGC`. Because leaf 1 and leaf 3 share the same parent, we apply intersection on their

>EST0
TGGAGTCCTCGGACACGATCACATCGACAATGTGAA
GGCGA
>EST1
GTGAAGGAGGTAGATCAAATAGAGCCTGCCCTAAAA
AGGCAGCTTATAATCTCCACTGCT
>EST2
TCCGACTACTGCACCCCGAGCGGATCACACAATGGAA
GGCCCGTGCGC
>EST3
GTGAAGGAGGTAGATACTCGTATACGATCACTGCCTA
AAAAGGCAGCTTATAATCTCCATATCGCTG

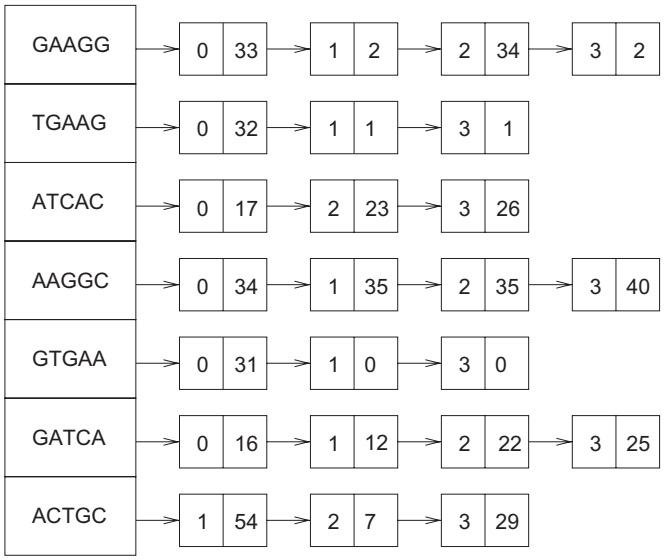

Fig. 3. The example dataset ($l = 8, c = 5, d = 1, T_c = 3$). The table of popular cores.

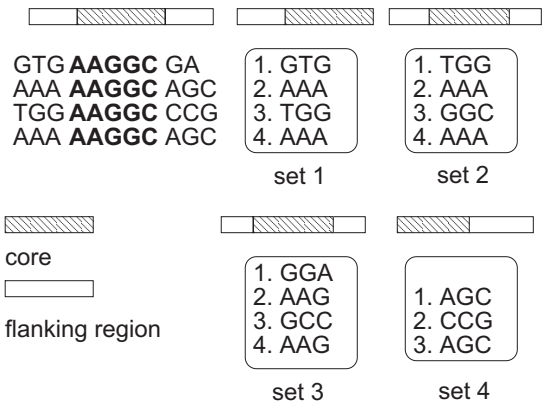

Fig. 4. Collecting flanking regions for the core. There are four sets of flanking regions for AAGGC.

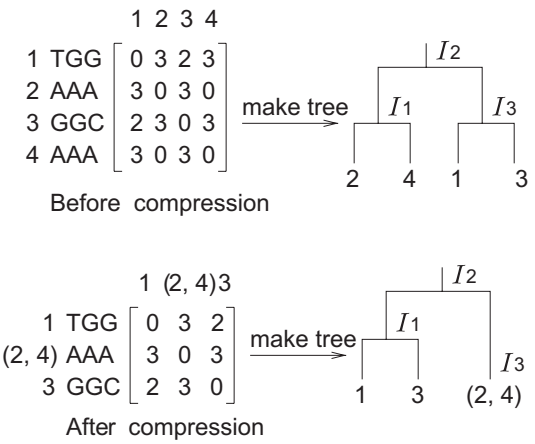

Fig. 5. UPGMA tree construction for set 2 of the core AAGGC.

sets of 1-mutants and get the set $I_1 = \{\texttt{GGG, TGC}\}$. Then, we apply intersection between I_1 and I_3, where I_3 represents the set of the 1-mutants of leaf AAA. Since the resulting intersection I_2 is empty, we prune the tree at I_2 and separately output the strings in I_1 and I_3 as flanking regions of candidate popular oligos. Note that, we output the 1-mutants of I_3 even if it is represented only by one node, because it is actually the intersection of two occurrences of AAA, and therefore, all elements of I_3 have at least two d-matches in the EST sequences.

3.2.2 Post processing of popular oligos As it turns out, the set of oligo candidates generated by the algorithm described above cannot be used directly because in practice it produces too many candidates, most of which are very similar to one another. To give an example, when the threshold T_c on the color of the cores is five, the number of candidates generated from the Barley EST database is about 527 million (see Fig. 8). Post processing is therefore necessary to reduce the number of oligos. We call this final cleaning phase, oligo compression.

The first step in the cleaning phase is to remove oligos that may be unsuitable for hybridization. If the GC-content of an oligo is outside a given range, say 45–55%, the hybridization probably would not take place. For the same reason, oligos containing repeats such as poly-(A)s, poly-(T)s, etc., should not be considered suitable. Oligos with unsuitable GC-content or containing simple repeats are, in fact, eliminated before compression.

In order to explain the oligo compression phase, let us introduce a couple of useful concepts. If an oligo (c, d)-occurs at least once in an EST sequence, we say that oligo covers the EST. In general, each oligo covers a set of ESTs, and each

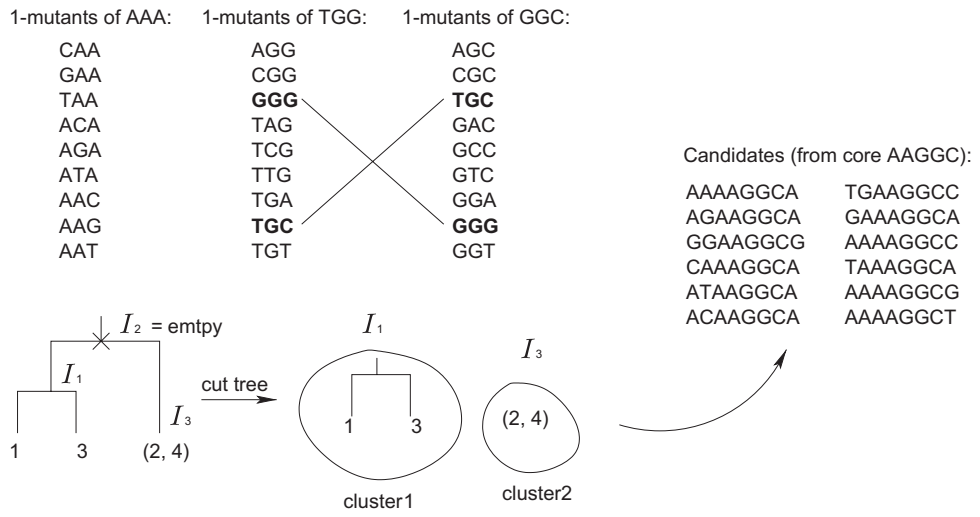

Fig. 6. Clustering set 2 of the core AAGGC.

EST is covered by a set of oligos. Recall that the number of ESTs covered by an oligo corresponds to the (c, d)-colors of that oligo. For a set S of oligos, the coverage ratio is the ratio between the number of ESTs covered by the oligos in S and the size of the set S. Intuitively, the coverage ratio is the average number of (c, d)-colors per oligo.

The objective of oligo compression is to maximize the coverage ratio, i.e. to select a set of popular oligos S from the large pool of candidates, such that the number of covered ESTs is not changed, but the number of selected popular oligos is minimized. More specifically, while we were trying to reduce the size of S, we made sure that the number of covered ESTs would not decrease. It turns out that the general problem of oligo compression is a variant of the Set Covering problem, which is known to be NP-complete (see, Garey and Johnson, 1979).

Since the general problem is NP-complete, it is unlikely that there exists a poly-time algorithm that finds the optimal solution. As a workaround, we use a greedy strategy that, in general, will find a suboptimal solution. The algorithm is articulated in two steps.

In the first step, for each covered EST we select a set of oligos with high colors, as follows. When a candidate oligo w is generated, we obtain the set of ESTs covered by w. For each covered ESTs, we decide whether oligo w should be discarded or kept as the top candidate. At the end of this step, we get a pool of ESTs each of which is covered by several oligos. In the second step, we use the greedy approximation algorithm for the Set Covering problem (see, e.g. Cormen *et al*., 1990). First, we select an oligo with the highest number of colors, and remove all ESTs covered by this oligo from the EST pool. Then, we update the colors of all other oligos. We repeat the latter step until the EST pool becomes empty. Note that updating the colors iteratively is critical because the intersection of the sets of covered ESTs for the initial pool of candidates is likely to be non-empty. The greedy algorithm has a ratio bound as $H(C_{\max})$, where $C_{\max}$ is the maximum number of colors, and $H(d)$ denotes the d-th harmonic number (see Cormen *et al*., 1990). The method is simple and space efficient since it can compress oligos on-line and avoid storing hundreds of million of candidates in main memory.

4 RESULTS

We have implemented both algorithms in C++ and tested the programs on a desktop PC with a 1.2 GHz AMD Athlon CPU and 1 GB RAM, under Linux. The current version of the program is available from the authors upon request. The code can be compiled using the GNU g++ compiler under several platforms. The compiler g++ can be freely obtained at http://gcc.gnu.org/. We are currently designing a web server to make the program more user-friendly.

4.1 Simulations

To evaluate the performance of our heuristics for selecting popular oligos, we first ran a few simulations as follows. We generated a set of artificial ESTs by creating first a set of k random sequences and then injecting a controlled number of approximate occurrences of a given set of oligos. The initial set of oligos, denoted by $I_1, \ldots, I_s$, was also generated randomly over Σ. Each oligo I_i was assigned a predetermined number of colors C_i. We decided that the distribution of the C_i should be Gaussian, i.e. we defined $C_i = c_{\max} e^{-i^2/2}/\sqrt{2\pi}$, where $c_{\max}$ is a fixed constant that determines the maximum number of colors. As discussed above, the positions in between the oligos were filled with random symbols over the DNA alphabet.

We then ran our program for popular oligos on the artificial ESTs dataset and output a set of candidate oligos $O_1, \ldots, O_t$ with their respective colors $C'_1, \ldots, C'_t$. The output oligos were

# of occurrences	# of seeds
0	242399
1-9	3063288
10-19	708745
20-29	120698
30-39	31637
40-49	11908
50-5049	15629

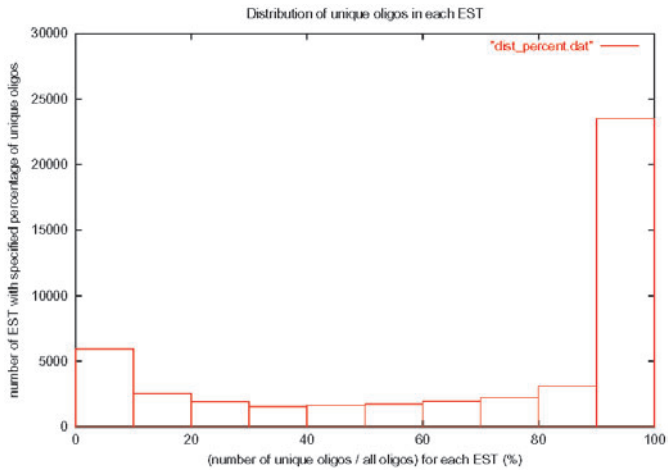

Fig. 7. Left: distribution of frequencies of seeds in barley ESTs. Right: Distribution of unique oligos. The horizontal axis stands for the percentage of unique oligos over all 33mers in an EST, and the vertical axis stands for the number of ESTs whose unique oligos are at a certain percentage of all its 33mers.

Table 1. The average relative errors between the number of colors in the input and the number of colors in output for a simulated experiment ($n = 1\,440\,000$, $k = 2000$, $c = 20$, $c_{\max} = 100$, $s = 100$, $l = 36$)

	$d = 2$	$d = 3$
$T_c = 10$	0.0155	0.0500
$T_c = 15$	0.0003	0.0033
$T_c = 20$	0.0048	0.0005
$T_c = 25$	0.0008	0.0023
$T_c = 30$	0.0005	0.0028

sorted by colors, i.e. $C'_i \geq C'_j$, if $i < j$. Since the output contained redundant candidates that came from the mutations of the original popular oligos, we removed those candidates that were a d-mutant of another oligo with an higher number of colors. More precisely, if O_i was a d-mutant of O_j, and $1 \leq i < j \leq t$, then O_j was discarded.

Finally, we compared the pair (I, C) with (O, C'). The more similar (O, C') is to (I, C), the better is the heuristic of our algorithm. Recall that I and O were sorted by decreasing number of colors. We compared the entries in (I, C) with the ones in (O, C'), position by position. For each $1 \leq i \leq u$, where $u = \min(s, t)$, we computed the average difference between C and C' as $E = (1/u) \sum_{i=1}^{u} (|C_i - C'_i|)/(C'_i)$. If we assume that I and O contain the same set of oligos, the smaller is E, the more similar is (I, C) to (O, C'). To validate this assumption, we also searched the list of oligos I in O, to determine whether we missed completely some oligos.

Table 1 shows the value of E for four runs of the program on a dataset of $n = 1\,440\,000$ bases comprising by $k = 2000$ sequences each of size 720. We generated a set of $s = 100$ oligos with a maximum number of colors $c_{\max} = 100$. In this analysis, we fixed the length of the core to be $c = 20$, whereas the maximum number of mismatches d outside the core and the threshold T_c were varied. The results show that the average relative error is below 2%. We also compared the list of input oligos with the list of output oligos and we found that sometimes the program misses one or two oligos out of 100. However, the number of colors of these missed oligos is always near the threshold T_c. We never miss an oligo whose number of color is above $T_c + 10$.

4.2 Experimental results

The main dataset is a collection barley ESTs from HARVEST containing $k = 46\,145$ EST sequences with a total of $n = 28\,475\,017$ bases. Before performing the searches, we first cleaned the dataset by removing Poly(T) and Poly(A) repeats.

As mentioned above, our first task was to search for unique oligos of length $l = 33$ with a minimum number of mismatches $d = 5$. Based on these parameters, each oligo was divided into three seeds of length $q = 11$. Hence, our dictionary table had $4^{11} \approx 4$ million entries. The efficiency of our algorithm critically depends on the statistical distribution of the seeds in the dictionary. The statistics of the seeds in our experiment (before the extension phase) is shown in table in the left-hand side of Figure 7. Clearly, most seeds occur less than 20 times in the ESTs and this is the main reason why our algorithm was able to solve the dataset efficiently. The final distribution of unique oligos is shown in the right-hand side of Figure 7. Note that, there are many ESTs (slightly more than half of the entire dataset) whose length-33 substrings are almost unique oligos. In particular, there are 13 430 ESTs whose length-33 substrings are all unique oligos and there are 2159 ESTs that contain no unique oligos. The whole computation took 2 h and 26 min and used about 200 MB of memory.

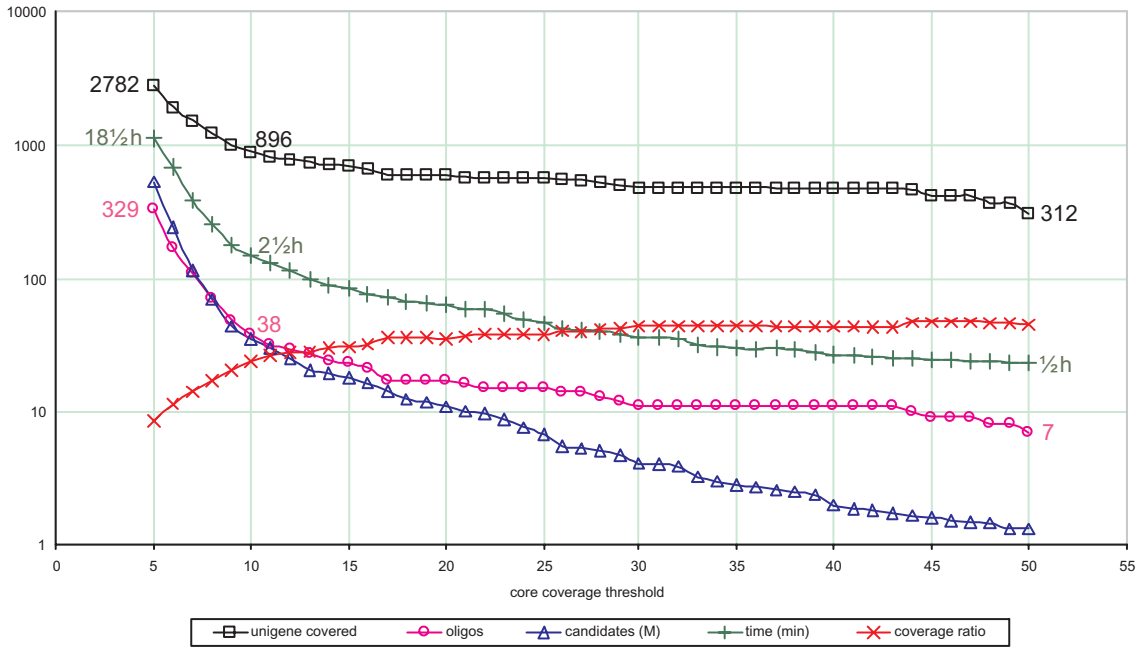

Fig. 8. Results of running the algorithm on the Barley dataset. Shown are the number of candidates generated by the algorithm (in millions), the number of ESTs covered, the final number of popular oligos, the coverage ratio and the time taken by the algorithm (for different choices of T_c).

Table 2. Distribution of the number of colors of the cores

Colors	Number of cores
1	22 523 412
2–10	21 28 677
11–20	5148
21–30	1131
31–40	492
41–50	346
51–60	242
61–70	77
71–80	34
81–90	29
91–100	43
101–176	19

The left column is the range of the number of colors. The right column is the number of cores with a certain number of color.

Our second task was to search for popular oligos with length $l = 36$ and core length $c = 20$. We considered different choices for the maximum number of mismatches d outside the core and the threshold T_c on the minimum number of colors for the popular cores. The threshold $C_{\min}$ on the size of the clusters was set equal to the value of threshold T_c.

The distribution of the number of colors of the cores is shown in Table 2. From the table, we can see that the number of cores decreases almost exponentially as the number of colors increases. On the other hand, cores with low colors are unlikely to contribute to popular oligos. Therefore, it is important to filter them out to increase the efficiency.

The running time of this program varies with the parameters d and T_c, as shown in Figure 8. The memory used in the program was mainly for storing the candidate popular oligos. Figure 8 also shows the number of candidates generated by the algorithm (in millions), the number of ESTs covered, the final number of popular oligos and the coverage ratio, for different choices of the threshold T_c. The post-processing, especially the oligo compression phase, reduces the number of candidates from hundreds of millions to less than a thousand, while the number of covered ESTs was kept unchanged. By selecting the appropriate threshold T_c, one can obtain a set of oligos with good coverage ratio in a few hours using a standard desktop PC.

5 CONCLUSIONS

We have proposed two algorithms to find unique and popular oligos in large EST databases. The size of our dataset, in the order of tens of millions of bases, was the real challenge due to the limitation in the size of main memory in common 32-bits architectures. Our algorithms were able to produce a solution in a reasonable amount of time on a regular PC with a modest amount of memory. As far as we know, no other existing tools are capable of handling such a dataset with limited resources. Simulations show that the number of missed oligos by the

heuristic algorithm for popular oligos is negligible and can be controlled very effectively by adjusting the parameters. Although the algorithms were initially designed to address the challenges from the barley EST dataset, the methods can be easily adapted to solve similar problems concerning infrequent and frequent oligos on other large datasets. The software will be released in the public domain in the near future.

ACKNOWLEDGEMENTS

The authors would like to thank Steve Wanamaker for producing the sequence assemblies and unigene datasets that were used in this work. This research was supported in part by NSF grants ITR-0085910, DBI-0321756, National Key Project for Basic Research (973) grant 2002CB512801, USDA/CSREES grant 2002-35300-12548 and USDA/CSREES/IFAFS grant 2001-52100-11346.

REFERENCES

Adams,M.D., Kelley,J.M., Gocayne,J.D., Dubnick,M., Polymeropoulos,M.H., Xiao,H., Merril,C.R., Wu,A., Olde,B., Moreno,R.F. *et al*. (1991) Complementary DNA sequencing: expressed sequence tags and human genome project. *Science*, **252**, 1651–1656.

Apostolico,A., Bock,M.E. and Lonardi,S. (2003) Monotony of surprise and large-scale quest for unusual words (extended abstract). In Myers,G., Hannenhalli,S., Istrail,S., Pevzner,P. and Waterman,M. (eds), *Proceedings of Research in Computational Molecular Biology (RECOMB)*, Washington, DC, April. *J. Comput. Biol.*, **10**, 3–4, 283–311.

Apostolico,A., Gong,F. and Lonardi,S. (2004) Verbumculus and the discovery of unusual words. *J. Comput. Sci. Technol.*, **19**, 22–41.

Bailey,T.L. and Elkan,C. (1995) Unsupervised learning of multiple motifs in biopolymers using expectation maximization. *Machine Learn.*, **21**, 51–80.

Barakat,A., Carels,N. and Bernardi,G. (1997) The distribution of genes in the genomes of gramineae. *Proc. Natl Acad. Sci., USA*, **94**, 6857–6861.

Boguski,M., Lowe,T. and Tolstoshev,C. (1993) dbEST—database for "expressed sequence tags". *Nat. Genet.*, **4**, 332–333.

Buhler,J. and Tompa,M. (2002) Finding motifs using random projections. *J. Comput. Biol.*, **9**, 225–242.

Close,T., Wing,R., Kleinhofs,A. and Wise,R. (2001) Genetically and physically anchored EST resources for barley genomics. *Barley Genet. Newslett.*, **31**, 29–30.

Cormen,T.H., Leiserson,C.E. and Rivest,R.L. (1990) *Introduction to Algorithms*. MIT Press, Cambridge, MA.

Eskin,E. and Pevzner,P.A. (2002) Finding composite regulatory patterns in DNA sequences. *Proceedings of the International Conference on Intelligent Systems for Molecular Biology*. AAAI press, Menlo Park, CA, pp. S181–S188.

Garey,M.R. and Johnson,D.S. (1979) *Computers and Intractability: a Guide to the Theory of NP-completeness*. Freeman, New York.

Han,C., Sutherland,R., Jewett,P., Campbell,M., Meincke,L., Tesmer,J., Mundt,M., Fawcett,J., Kim,U., Deaven,L. and Doggett,N. (2000) Construction of a BAC contig map of chromosome 16q by two-dimensional overgo hybridization. *Genome Res.*, **104**, 714–721.

Hertz,G.Z. and Stormo,G.D. (1999) Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*, **15**, 563–577.

Huang,X. and Madan,A. (1999) CAP3: a DNA sequence assembly program. *Genome Res.*, **9**, 868–877.

Jonassen,I. (1997) Efficient discovery of conserved patterns using a pattern graph. *Comput. Appl. Biosci.*, **13**, 509–522.

Jonassen,I., Collins,J.F. and Higgins,D.G. (1995) Finding flexible patterns in unaligned protein sequences. *Protein Sci.*, **4**, 1587–1595.

Keich,U. and Pevzner,P.A. (2002) Finding motifs in the twilight zone. *Annual International Conference on Computational Molecular Biology*, Washington, DC, April, pp. 195–204.

Lawrence,C.E., Altschul,S.F., Boguski,M.S., Liu,J.S., Neuwald,A.F. and Wootton,J.C. (1993) Detecting subtle sequence signals: a Gibbs sampling strategy for multiple alignment. *Science*, **262**, 208–214.

Li,F. and Stormo,G.D. (2001) Selection of optimal DNA oligos for gene expression arrays. *Bioinformatics*, **17**, 1067–1076.

Michalek,W., Weschke,W., Pleissner,K. and Graner,A. (2002) EST analysis in barley defines a unigene set comprising 4,000 genes. *Theor. Appl. Genet.*, **104**, 97–103.

Neuwald,A., Liu,J. and Lawrence,C. (1995) Gibbs motif sampling: detecting bacterial outer membrane protein repeats. *Protein Sci.*, **4**, 1618–1632.

Pevzner,P.A. and Sze,S.-H. (2000) Combinatorial approaches to finding subtle signals in DNA sequences. *Proceedings of the International Conference on Intelligent Systems for Molecular Biology*. AAAI press, Menlo Park, CA, pp. 269–278.

Rahmann,S. (2002) Rapid large-scale oligonucleotide selection for microarrays. *Proceedings of the First IEEE Computer Society Bioinformatics Conference (CSB'02)*. IEEE Press, Los Alamitos, pp. 54–63.

Rigoutsos,I. and Floratos,A. (1998) Combinatorial pattern discovery in biological sequences: the TEIRESIAS algorithm. *Bioinformatics*, **14**, 55–67.

Rouillard,J.-M., Herbert,C.J. and Zuker,M. (2002) Oligoarray: genome-scale oligonucleotide design for microarrays. *Bioinformatics*, **18**, 486–487.

Swofford,D. (2002) *PAUP: Phylogenetic Analysis Using Parsimony version 4.0 beta 10*. Sinauer Associates, Sunderland, Massachusetts.

Tompa,M. and Buhler,J. (2001) Finding motifs using random projections. *Annual International Conference on Computational Molecular Biology*, Montreal, Canada, April, ACM Press, NY, pp. 67–74.

Wesselink,J.-J., de la Iglesia,B., James,S.A., Dicks,J.L., Roberts,I.N. and Rayward-Smith,V.J. (2002) Determining a unique defining DNA sequence for yeast species using hashing techniques. *Bioinformatics*, **18**, 1004–1010.

Yu,Y., Tomkins,J.P., Waugh,R., Frisch,D.A., Kudrna,D., Kleinhofs,A., Brueggeman,R.S., Muehlbauer,G.J., Wise,R.P. and Wing,R.A. (2000) A bacterial artificial chromosome library for barley (*Hordeum vulgare* L.) and the identification of clones containing putative resistance genes. *Theor. Appl. Genet.*, **101**, 1093–1099.