Hindawi Publishing Corporation
Advances in Bioinformatics
Volume 2013, Article ID 793130, 6 pages
http://dx.doi.org/10.1155/2013/793130

Research Article

Efficient Serial and Parallel Algorithms for Selection of Unique Oligos in EST Databases

Manrique Mata-Montero,[1] Nabil Shalaby,[2] and Bradley Sheppard[1,2]

[1] *Department of Computer Science, Memorial University, Canada*
[2] *Department of Mathematics and Statistics, Memorial University, Canada*

Correspondence should be addressed to Nabil Shalaby; nshalaby@mun.ca

Received 15 October 2012; Accepted 14 February 2013

Academic Editor: Alexander Zelikovsky

Obtaining unique oligos from an EST database is a problem of great importance in bioinformatics, particularly in the discovery of new genes and the mapping of the human genome. Many algorithms have been developed to find unique oligos, many of which are much less time consuming than the traditional brute force approach. An algorithm was presented by Zheng et al. (2004) which finds the solution of the unique oligos search problem efficiently. We implement this algorithm as well as several new algorithms based on some theorems included in this paper. We demonstrate how, with these new algorithms, we can obtain unique oligos much faster than with previous ones. We parallelize these new algorithms to further improve the time of finding unique oligos. All algorithms are run on ESTs obtained from a Barley EST database.

1. Introduction

Expressed Sequence Tags (or ESTs) are fragments of DNA that are about 200–800 bases long generated from the sequencing of complementary DNA. ESTs have many applications. They were used in the Human Genome Project in the discovery of new genes and are often used in the mapping of genomic libraries. They can be used to infer functions of newly discovered genes based on comparison to known genes [1].

An oligonucleotide (or oligo) is a subsequence of an EST. Oligos are short, since they are typically no longer than 50 nucleotide bases. Oligos are often referred to in the context of their length by adding the suffix "mer". For example, an oligo of length 9 would be referred to as a 9-mer. The importance of oligos in relation to EST databases is quite significant. An oligo that is unique in an EST database serves as a representative of its EST sequence. The oligonucleotides (or simply oligos) contained in these EST databases have applications in many areas such as PCR primer design, microarrays, and probing genomic libraries [2–4].

In this paper we will improve on the algorithms presented in [2] to solve the *unique oligos search* problem. This problem requires us to determine all oligos that appear in one EST sequence but not in any of the others. In addition, we will consider two oligos to be virtually identical if they fall within a certain number of mismatches from each other. In the appendix we include all the algorithms used and developed in this paper.

2. The Unique Oligos Search Problem

In this paper we use the notation $\mathrm{HD}(x, y)$ to denote the Hamming Distance between the strings x and y. Given an EST database $D = \{x_1, x_2, \ldots, x_k\}$, where x_i is a string over the alphabet {A, C, G, G}, integers d and l, and l-mer y, we say that y occurs approximately in D if there exists a substring z of some EST x_i such that $\mathrm{HD}(y, z) \leq d$. We also say that an m-mutant list of a string s is a list of all possible strings, s^*, of length $|s|$ over the alphabet {A, C, G, T} such that $\mathrm{HD}(s, s^*) \leq m$. Such a string s^* is referred to as an m-mutant of s. A unique oligo of D is defined as an l-mer u such that u occurs exactly in one EST and does not occur approximately in any other EST. The unique oligos search problem is the problem of finding all unique oligos in an EST database.

Require: EST database $D = \{x_1, x_2, \ldots, x_k\}$, integer l (length of unique oligos) and integer d (maximum number of mismatches between non-unique oligos)
Ensure: All unique l-mers in D
(1) $q \leftarrow l/(\lfloor d/2 \rfloor + 1)$
(2) $posi \leftarrow$ findqmers(q) (hashtable of positions of all qmers in D)
(3) **for** $i \leftarrow 1$ to 4^q {split loop iterations among processors} **do**
(4) $x \leftarrow i$ as a base 4 integer of length q
(5) $mismatchlist \leftarrow$ list of base 4 integers of length q mismatching x by 1 digit
(6) $modifiedmismatchlist \leftarrow$ the numbers in $mismatchlist$ in base 10
(7) $mut \leftarrow$ list of each $hashtable[i]$ for all $i \in modifiedmismatchlist$
(8) goo2($q, l, d, posi[i], mut$)
(9) **end for**

Algorithm 1: Algorithm for the unique oligos problem.

Many algorithms have been presented to solve this problem [5, 6]. The algorithm presented in [2] relies on an observation that if two l-mers agree within a specific Hamming Distance, then they must share a certain substring. These observations are presented in this paper as theorems.

Theorem 1. *Suppose one has two l-mers l_1 and l_2 such that $HD(l_1, l_2) \le d$. If one divides them both into $\lfloor d/2 \rfloor + 1$ substrings, $l_1^1 l_1^2 \cdots l_1^{\lfloor d/2 \rfloor + 1}$ and $l_2^1 l_2^2 \cdots l_2^{\lfloor d/2 \rfloor + 1}$, and each l_j^i, except possibly $l_j^{\lfloor d/2 \rfloor + 1}$, has length $\lceil l/(\lfloor d/2 \rfloor + 1) \rceil$, then there exists at least one $i_0 \in \{1, 2, \ldots, \lfloor d/2 \rfloor + 1\}$, such that $HD(l_1^{i_0}, l_2^{i_0}) \le 1$.*

Proof. Suppose by contradiction that for any $i \in \{1, 2, \ldots, \lfloor d/2 \rfloor + 1\}$, l_1^i and l_2^i have at least 2 mismatches. Then $\mathrm{HD}(l_1, l_2) \ge d + 2$ which is a contradiction to the fact that $\mathrm{HD}(l_1, l_2) \le d$. □

Using this observation, an algorithm was presented in [2] which solves the unique oligos search problem in time $O((l - q)qr^2 4^q)$. The algorithm can be thought of as a two-phase method. In the first phase we record the position of each q-mer in the database into a hash table of size 4^q. We do so in such a way that for each q-mer x over the alphabet $\{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$ we have that $hashtable[hashfunction[x]] = \{\{s_1, p_1\}, \{s_2, p_2\}, \ldots, \{s_n, p_n\}\}$ whereby s_i is an EST sequence, p_i is the position of x within that sequence, and n is the number of occurrences of x in the database. In the second phase, we extend every pair of identical q-mers into l-mers and compare these l-mers for nonuniqueness. We also do the same for pairs that have a Hamming Distance of 1. If they are nonunique, we mark them accordingly. Theorem 1 guarantees that if an l-mer is nonunique, then it must share a q-mer substring that differs by at most one character with another q-mer substring from another l-mer. Hence, if an l-mer is nonunique, it will be marked during phase two.

Assuming there are n symbols in our EST database, the filing of the q-mers into the hash table takes time $\Theta(qn)$. In phase two, we assume that the distribution of q-mers in the database is uniform; in other words, that each table contains $r \approx n/4^q$ entries. Thus we have $O(r^2)$ comparisons within each table entry. Each q-mer also has a 1-mutant list of size $3q$, so, we have $O(qr^2)$ comparisons for each entry in the table. Also, the time required to extend each pair of q-mers to l-mers is $2(l - q + 1)$. Given that we have 4^q entries in the hash table, we have a total time complexity of

$$O\left((l-q)\,qr^2 4^q\right) = O\left((l-q)\,q\left(\frac{n}{4^q}\right)^2 4^q\right) = O\left(\frac{(l-q)\,qn^2}{4^q}\right), \tag{1}$$

where

$$q = \frac{l}{\lfloor d/2 \rfloor + 1}. \tag{2}$$

In [7], several variations of Theorem 1 are presented. We can use these theorems to generate similar algorithms with slightly different time complexities.

Theorem 2. *Suppose one has two l-mers l_1 and l_2 such that $HD(l_1, l_2) \le d$. If one divides them both into $d + 1$ substrings, $l_1^1 l_1^2 \cdots l_1^{d+1}$ and $l_2^1 l_2^2 \cdots l_2^{d+1}$, and each l_j^i, except possibly l_j^{d+1}, has length $\lceil l/(d+1) \rceil$, then there exists at least one $i_0 \in \{1, 2, \ldots, d+1\}$, such that $l_1^{i_0} = l_2^{i_0}$.*

Proof. Suppose by contradiction that we cannot find any $i_0 \in \{1, 2, \ldots, d + 1\}$ such that $l_1^{i_0} = l_2^{i_0}$. Then there exists at least one mismatch between l_1^i and l_2^i for each $i \in \{1, 2, \ldots, d + 1\}$, and thus we have at least $d + 1$ mismatches which contradicts the fact that $\mathrm{HD}(l_1, l_2) \le d$. □

Based on Theorem 2 we can design a second algorithm that works in a similar way to Algorithm 1. The major difference between these algorithms is that in Algorithm 2 we are not required to do comparisons with each hash table entries mutant list. This means we have $O(r^2)$ comparisons within each table entry which yields a total time complexity of

$$O\left((l-q)\,r^2 4^q\right) = O\left((l-q)\left(\frac{n}{4^q}\right)^2 4^q\right) = O\left(\frac{(l-q)\,n^2}{4^q}\right), \tag{3}$$

Require: EST database $D = \{x_1, x_2, \ldots, x_k\}$, integer l (length of unique oligos) and integer d (maximum number of mismatches between non-unique oligos)
Ensure: All unique l-mers in D
(1) $q \leftarrow l/(d+1)$
(2) $posi \leftarrow$ findqmers(q) (hashtable of positions of all qmers in D)
(3) **for** $i \leftarrow 1$ to 4^q {split loop iterations among processors} **do**
(4) goo($q, l, d, posi[i]$)
(5) **end for**

Algorithm 2: Algorithm for the unique oligos problem.

Require: EST database $D = \{x_1, x_2, \ldots, x_k\}$, integer l (length of unique oligos) and integer d (maximum number of mismatches between non-unique oligos)
Ensure: All unique l-mers in D
(1) $q \leftarrow l/(\lfloor d/3 \rfloor + 1)$
(2) $posi \leftarrow$ findqmers(q) (hashtable of positions of all qmers in D)
(3) **for** $i \leftarrow 1$ to 4^q {split loop iterations among processors} **do**
(4) $x \leftarrow i$ as a base 4 integer of length q
(5) $mismatchlist \leftarrow$ list of base 4 integers of length q mismatching x by at most 2 digits
(6) $modifiedmismatchlist \leftarrow$ the numbers in $mismatchlist$ in base 10
(7) $mut \leftarrow$ list of each $hashtable[i]$ for all $i \in modifiedmismatchlist$
(8) goo2($q, l, d, posi[i], mut$)
(9) **end for**

Algorithm 3: Algorithm for the unique oligos problem.

Require: EST database $D = \{x_1, x_2, \ldots, x_k\}$, integer q
Ensure: A hashtable of all $qmer$ positions.
(1) $hashtable \leftarrow$ a hashtable of all $qmer$ positions in D
(2) **for** $i \leftarrow 1$ to k **do**
(3) **for** $j \leftarrow 1$ to length($D[i]$) $- q + 1$ **do**
(4) $hashedqmer \leftarrow$ map($D[i], j, j + q - 1$)
(5) $hashtable[hashedqmer] \leftarrow$ Append(hashtable[$hashedqmer$], $\{i, j\}$)
(6) **end for**
(7) **end for**

Algorithm 4: Findqmers (q).

(1) $r \leftarrow$ substring(s, i, j)
(2) $t \leftarrow r$ under the transformation {A, C, G, T} $\rightarrow$ {0, 1, 2, 3}
(3) **return** t

Algorithm 5: Map (string s, i, j).

(1) $r \leftarrow$ substring of s from character i to character j
(2) **return** r

Algorithm 6: Substring (string s, i, j).

```
(1) posi ← a list of positions of a specified qmer in D
    (posi = {{x1, y1}, {x2, y2}, ...} where {x, y} corresponds to position y of sequence x)
(2) mut ← a list of positions of qmers in D that mismatch this qmer by either 1 or 2 characters
    (depending on the filtration algorithm using this function)
(3) for i ← 1 to length(posi) do
(4)     for j ← i + 1 to length(posi) do
(5)         if posi[i][1] ≠ posi[j][1] then
(6)             lq1 ← list of l-mers generated from the extension of the qmer in position posi[i]
(7)             lq2 ← list of l-mers generated from the extension of the qmer in position posi[j]
(8)             for x ← 1 to length(lq1) do
(9)                 for y ← 1 to length(lq2) do
(10)                    if HD(lq1[x], lq2[y]) ≤ d then
(11)                        mark the lmers as non-unique
(12)                    end if
(13)                end for
(14)            end for
(15)        end if
(16)    end for
(17)    for k ← 1 to length(mut) do
(18)        if posi[i][1] ≠ mut[k][1] then
(19)            lq1 ← list of l-mers generated from the extension of the qmer in position posi[i]
(20)            lq2 ← list of l-mers generated from the extension of the qmer in position mut[k]
(21)            for x ← 1 to length(lq1) do
(22)                for y ← 1 to length(lq2) do
(23)                    if HD(lq1[x], lq2[y]) ≤ d then
(24)                        mark the lmers as non-unique
(25)                    end if
(26)                end for
(27)            end for
(28)        end if
(29)    end for
(30) end for
```

Algorithm 7: goo2($q, l, d, posi, mut$).

```
(1) posi ← a list of positions of qmer in D
    (posi = {{x1, y1}, {x2, y2}, ...} where {x, y} corresponds to position y of sequence x)
(2) for i ← 1 to length(posi) do
(3)     for j ← i + 1 to length(posi) do
(4)         if posi[i][1] ≠ posi[j][1] then
(5)             lq1 ← list of l-mers generated from the extension of the qmer in position posi[i]
(6)             lq2 ← list of l-mers generated from the extension of the qmer in position posi[j]
(7)             for x ← 1 to length(lq1) do
(8)                 for y ← 1 to length(lq2) do
(9)                     if HD(lq1[x], lq2[y]) ≤ d then
(10)                        mark the lmers as non-unique
(11)                    end if
(12)                end for
(13)            end for
(14)        end if
(15)    end for
(16) end for
```

Algorithm 8: goo($q, l, d, posi$).

Table 1: Results of serial algorithms.

Algorithm	l	d	q	Dataset	Time taken (secs)	Non-unique oligos
Algorithm 2	28	6	4	1 (78 ESTs)	163	46,469
Algorithm 1	28	6	7	1 (78 ESTs)	131	46,469
Algorithm 3	27	6	9	1 (78 ESTs)	231	46,564
Algorithm 2	28	6	4	2 (2838 ESTs)	197, 500	1,611,241
Algorithm 1	28	6	7	2 (2838 ESTs)	117, 714	1,611,241
Algorithm 3	27	6	9	2 (2838 ESTs)	94, 317	1,614,235

Table 2: Results of parallel algorithms on 12 processors.

Algorithm	l	d	q	Dataset	Time taken (secs)	Non-unique oligos
Algorithm 2	28	6	4	1 (78 ESTs)	33	46,469
Algorithm 1	28	6	7	1 (78 ESTs)	29	46,469
Algorithm 3	27	6	9	1 (78 ESTs)	66	46,564
Algorithm 2	28	6	4	2 (2838 ESTs)	40, 420	1,611,241
Algorithm 1	28	6	7	2 (2838 ESTs)	22, 848	1,611,241
Algorithm 1	27	6	9	2 (2838 ESTs)	18, 375	1,614,235

where

$$q = \frac{l}{d+1}. \tag{4}$$

A third theorem was also briefly mentioned [7]; however, it was not implemented in an algorithm. We use this theorem to create a third algorithm to solve the unique oligos search problem.

Theorem 3. *Suppose one has two l-mers l_1 and l_2 such that $HD(l_1, l_2) \leq d$. If one divides them both into $\lfloor d/3 \rfloor + 1$ substrings, $l_1^1 l_1^2 \cdots l_1^{\lfloor d/3 \rfloor + 1}$ and $l_2^1 l_2^2 \cdots l_2^{\lfloor d/3 \rfloor + 1}$, and each l_j^i, except possibly $l_j^{\lfloor d/3 \rfloor + 1}$, has length $\lceil l/(\lfloor d/3 \rfloor + 1) \rceil$, then there exists at least one $i_0 \in \{1, 2, \ldots, \lfloor d/3 \rfloor + 1\}$, such that $HD(l_1^{i_0}, l_2^{i_0}) \leq 2$.*

Proof. Suppose by contradiction that for any $i \in \{1, 2, \ldots, \lfloor d/3 \rfloor + 1\}$, l_1^i and l_2^i have at least 3 mismatches. Then $\mathrm{HD}(l_1, l_2) \geq d + 3$ which is a contradiction to the fact that $\mathrm{HD}(l_1, l_2) \leq 2$. □

The algorithm is somewhat similar to Algorithm 1. The main difference is that we compare every q-mer to q-mers in its corresponding 2-mutant list, rather than its 1-mutant list. Each q-mer has $9\binom{q}{2} + 3q = 9q(q-1)/2 + 3q$ 2-mutants, so we have $O(q^2 r^2)$ comparisons for each entry in the hash table yielding a total time complexity of

$$\begin{aligned} O\left((l-q)\, q^2 r^2 4^q\right) &= O\left((l-q)\, q^2 \left(\frac{n}{4^q}\right)^2 4^q\right) \\ &= O\left(\frac{(l-q)\, q^2 n^2}{4^q}\right), \end{aligned} \tag{5}$$

where

$$q = \frac{l}{\lfloor d/3 \rfloor + 1}. \tag{6}$$

It is important to note the 4^q term in the denominator of our time complexity expressions. Since this term is exponential, it will have the largest impact on the time taken to run our algorithms. Based on this observation, we expect Algorithm 3 to run the fastest, followed by Algorithm 1 and then Algorithm 2.

3. Implementation

We implement these algorithms using C on a machine with 12 Intel Core i7 CPU 80 @ 3.33 GHz processors and 12 GB of memory. The datasets we use in this implementation are Barley ESTs taken from the genetic software HarvEST by Steve Wanamaker and Timothy Close of the University of California, Riverside (http://harvest.ucr.edu/). We use two different EST databases, one with 78 ESTs and another with 2838. In our experiments we search for oligos of lengths 27 and 28 since they are common lengths for oligonucleotides. As we increase the size of the database, we see that Algorithm 3 is the most efficient as anticipated (data shown in Tables 1 and 2).

One important thing to note about all of these algorithms is the fact that the main portion of them is a for loop which iterates through each index of the hash table. It is also obvious that loop iterations are independent of each other. These two factors make the algorithms perfect candidates for parallelism. Rather than process the hash table one index at a time, our parallel algorithms process groups of indices simultaneously. Ignoring the communication between processors, our algorithms optimally parallelize our three serial algorithms.

There are many APIs in different programming languages that aid in the task of parallel programming. Some examples of this in the C programming language are OpenMP and POSIX Pthreads. OpenMP allows one to easily parallelize

a C program amongst multiple cores of a multicore machine [8]. OpenMP also has an extension called Cluster OpenMP which allows one to parallelize across multiple machines in a computing cluster.

A new trend in parallel programming is in the use of GPUs. GPUs are the processing units inside computers graphics card. C has several APIs which allow one to carry out GPU programming. The two such APIs are OpenCL and CUDA [9, 10].

In the second implementation of our algorithms we use OpenMP to parallelize our algorithms throughout the 12 cores of our machine. We can easily see that we achieve near optimal parallelization with our parallel algorithms; that is, the time taken by the parallel algorithms is approximately that of the serial algorithms divided by the number of processors.

4. Conclusion

In this paper we used three algorithms to solve the unique oligos search problem which are extensions of the algorithm presented in [2]. We observed that we can achieve a significant performance improvement by parallelizing our algorithms. We can also see that Algorithm 3 yields the best results for larger databases. For smaller databases, however, the time difference between each pair of algorithms is negligible, but results in Algorithm 3 being the slowest, and this is due to the time required to compute the mismatches of each q-mer. Other algorithms can be obtained by setting q to different values. See Algorithms 1, 2, 3, 4, 5, 6, 7, and 8.

References

[1] M. D. Adams, J. M. Kelley, J. D. Gocayne et al., "Complementary DNA sequencing: expressed sequence tags and human genome project," *Science*, vol. 252, no. 5013, pp. 1651–1656, 1991.

[2] J. Zheng, T. J. Close, T. Jiang, and S. Lonardi, "Efficient selection of unique and popular oligos for large EST databases," *Bioinformatics*, vol. 20, no. 13, pp. 2101–2112, 2004.

[3] S. H. Nagaraj, R. B. Gasser, and S. Ranganathan, "A hitchhiker's guide to expressed sequence tag (EST) analysis," *Briefings in Bioinformatics*, vol. 8, no. 1, pp. 6–21, 2007.

[4] W. Klug, M. Cummings, and C. Spencer, *Concepts of Genetics*, Prentice-Hall, Upper Saddle River, NJ, USA, 8th edition, 2006.

[5] F. Li and G. D. Stormo, "Selection of optimal DNA oligos for gene expression arrays," *Bioinformatics*, vol. 17, no. 11, pp. 1067–1076, 2001.

[6] S. Rahmann, "Rapid large-scale oligonucleotide selection for microarrays," in *Proceedings of the 1st IEEE Computer Society Bioinformatics Conference (CSB '02)*, pp. 54–63, IEEE Press, Stanford, Calif, USA, 2002.

[7] S. Go, *Combinatorics and its applications in DNA analysis [M.S. thesis]*, Department of Mathematics and Statistics, Memorial University of Newfoundland, 2009.

[8] OpenMP.org, 2012, http://openmp.org/wp/.

[9] Khronos Group, "OpenCL—The open standard for parallel programming of heterogeneous systems," 2012, http://www.khronos.org/opencl/.

[10] Nvidia, "Parallel Programming and Computing Platform—Cuda—Nvidia," 2012, http://www.nvidia.com/object/cuda_home_new.html.