

# Fast Algorithms for Selecting Specific siRNA in Complete mRNA Data

Jaime Davila, Sudha Balla, Sanguthevar Rajasekaran

CSE Department University of Connecticut  
{jdavila,ballasudha,rajasek}@engr.uconn.edu

**Abstract.** The Specific Selection Problem arises from the need to design short interfering RNA (siRNA) that aims at gene silencing. These short sequences target specific messenger RNA (mRNA) and cause the degradation of such mRNA, inhibiting the synthesis of the protein generated by it. In [11] this problem was solved in a reasonable amount of time when restricted to the design of siRNA for a particular mRNA, but their approach becomes too time consuming when trying to design siRNA for each mRNA in a given organism. We devise simple algorithms based on sorting and hashing techniques that allow us to solve this problem for the entire mRNA of the Human in less than 4 hours, obtaining a speedup of almost two orders of magnitude over previous approaches.

## 1 Introduction

The Specific Selection Problem arises from the need to design short interfering RNA (siRNA) that aims at gene silencing [4]. These short sequences target specific messenger RNA (mRNA) and cause the degradation of such mRNA, inhibiting the synthesis of the protein generated by it. These sequences are usually of small length, usually consisting of between 20 and 25 nucleotides. However a length of 21 is used in practice and usually two of the nucleotides are predetermined, so the problem becomes one of designing sequences of length 19.

An important criterion in the design of the siRNA is that the sequence should minimize the risk of off-target gene silencing caused by hybridization with the wrong mRNA. This hybridization may occur because the number of mismatches between the sequence and an unrelated sequence maybe too small or because they share a long enough subsequence.

In [11] this problem was considered in the context of designing an siRNA that would target a particular mRNA sequence. However their approach becomes computationally very demanding in the case of selecting such siRNA for every possible mRNA in a given organism.

In this paper we design simple algorithms that solve this problem by making use of sorting techniques. The algorithm is shown to be practical when processing the complete mRNA of Human and Drosophila, running in less than 4 hours and outperforming previous approaches [11, 9, 12].

In this paper we tackle the problem that arises from constraints that consider mismatches with unintended sequences. Some other constraints have also been

considered in the literature [8, 10]. These other other constraints can be taken into account in pre or post processing stages.

## 2 Specific Selection Problem

We are interested in identifying small  $l$ -mers which will target a particular mRNA sequence, trying to minimize hybridizations with other sequences. Hybridizations could occur, for example, if the number of mismatches between the designed  $l$ -mer and an  $l$ -mer of another sequence is low –say, less than 3–. In this section, we define the problem under formal terms as the  $(l, d)$  *Specific Selection Problem* and we consider practical and efficient algorithms that solve it.

### 2.1 Problem Definition

We denote by  $d_H(x, y)$  the Hamming distance between two strings  $x$  and  $y$ , i.e. the number of mismatches between  $x$  and  $y$ .

**Definition 1.** Let  $x$  and  $s$  be strings over  $\Sigma$  with  $|x| = l$ ,  $|s| = n$  and  $l < n$ .

1. We say  $x$  is an  $l$ -mer of  $s$  if  $x$  is a subsequence of  $s$  and we denote it by  $x \triangleleft_l s$ .
2. We denote by  $d_H(x, s) = \min_{y \triangleleft_l s} d_H(x, y)$

**Definition 2.** Let  $\mathcal{S} = \{s_1, \dots, s_n\}$  and  $x \triangleleft_l s_i$ . We denote by

$$\bar{d}_H(x, \mathcal{S}) = \max_{1 \leq j \neq i \leq n} d_H(x, s_j) \quad (1)$$

A similar concept to  $\bar{d}_H(\cdot, \mathcal{S})$  was introduced in [11] under the name of *mismatch tolerance*.

**Definition 3.** Let  $\mathcal{S} = \{s_1, \dots, s_n\}$  be a set of sequences over  $\Sigma$ . Given  $l$  and  $d$  the  $(l, d)$  *Specific Selection Problem* consists of finding a set of  $l$ -mers  $\mathcal{X} = \{x_1, \dots, x_n\}$  such that

$$\forall (1 \leq i \leq n) : x_i \triangleleft_l s_i \text{ and } \bar{d}_H(x_i, \mathcal{S}) > d \quad (2)$$

That is  $x_i$  appears in  $s_i$  and does not appear in any other  $s_j$  ( $j \neq i$ ) with up to  $d$  errors. In case for a particular  $i$  there is no  $x_i$  that satisfies (2) we set  $x_i = \emptyset$ .

It is clear that this problem can be solved in  $O(N^2)$  time where  $N := \sum_{i=1}^n |s_i|$ . However such an approach becomes impractical when we are dealing with complete mRNA data where  $N$  could be of the order of  $10^8$ .

In [12] this problem was studied under the name of *unique oligo* problem and in [9] a more general problem is considered under the name of *probe design*

problem, imposing more conditions on the designed  $l$ -mers which include homogeneity –which is measured by the melting temperature of the probe and the CG content– and sensitivity –which is calculated using the free energy of the probe–.

Their solution strategy is based on determining whether  $\bar{d}_H(\cdot, \mathcal{S}) \leq d$  for each candidate  $l$ -mer by making use of a precalculated index for small  $q$ -mers or seeds, and then extending contiguous hits of  $q$ -mers with few mismatches. The running time of these approaches depends critically on the values of  $q$  and the number of mismatches which are used, which in turn depends heavily on the combination of values of  $l$  and  $d$ .

In [11] it is pointed out that in cases such as the ones that arise from designing siRNA where  $N \sim 10^8$ ,  $19 \leq l \leq 23$  and  $d = 3, 4$  the previous strategy is computationally very intensive, hence the value of  $\bar{d}_H(\cdot, \mathcal{S})$  is calculated by making use of overlapping –instead of contiguous–  $q$ -mers or seeds allowing a few mismatches, and it is shown that this approach outperforms the previous methods by orders of magnitude. In particular it is claimed that for  $l = 19$ ,  $d = 3$  and  $N = 5 \times 10^7$ ,  $\bar{d}(\cdot, \mathcal{S})$  can be calculated in nearly  $10^{-2}$  seconds on a Xeon CPU with a clock rate of 3.2 GHz and 2GB of main memory. This would imply that if we want to solve the  $(l, d)$  off-target selection problem in this case we would take close to 6 days of calculation. Our method would take close to 3 hours to be solved on a similar machine.

## 2.2 SOS: A Solution Based on Radix Sorting

Let  $x \triangleleft_l s_i$  and assume that  $\bar{d}_H(x, \mathcal{S}) \leq d$ . This means that there is  $y \triangleleft s_j$  ( $j \neq i$ ) with  $d_H(x, y) \leq d$ . If we eliminate the  $\leq d$  characters where  $x$  and  $y$  differ the resulting strings will be identical and easily identifiable if we sort them. Since we don't know which set of positions will work, we need to try all the  $\binom{l}{d}$  combinations. However if  $l$  and  $d$  are small –as it is in our case– the number of possibilities is not that big. Notice that in this case we are using a strategy which is similar to [7, 1] but in a different context.

In other words we are exploiting the fact that if  $l$  and  $d$  are small enough, the number of cases where two  $l$ -mers will differ in less than  $d$  positions is not that big.

This following definition will be used in the description of the algorithm that captures this idea.

**Definition 4.** Let  $x$  be a string over  $\Sigma$  and let  $1 \leq i_1 \leq \dots \leq i_h \leq |x|$ . We call  $x \downarrow_{i_1, \dots, i_h}$  the  $l$  –  $h$ -mer that omits the characters  $x[i_1], \dots, x[i_h]$ .

## Algorithm SOS

1. Given  $\mathcal{S} = \{s_1, \dots, s_n\}$ , generate  $X = \bigcup_{i=1}^n \{(x, i) : x \triangleleft_l s_i\}$ . Let  $C := X$ .
2. For all  $(j_1 \dots j_d)$  with  $1 \leq j_1 < \dots < j_d \leq l$ 
  - (a) Sort the collection of  $X = \{(x, i)\}$  according to the values of  $x \downarrow_{j_1, \dots, j_d}$  using radix-sort.
  - (b) Scan the sorted collection. If  $(y, i)$  and  $(y', j)$  appear consecutively, where  $y \downarrow_{j_1, \dots, j_d} = y' \downarrow_{j_1, \dots, j_d}$  and  $i \neq j$  mark the elements  $(x, i)$  such that  $x \downarrow_{j_1, \dots, j_d} = y \downarrow_{j_1, \dots, j_d}$ .
3. Output the unmarked elements.

**Theorem 1.** *Algorithm SOS can be implemented in  $O(N \frac{l}{w} \binom{l}{d})$  time and  $O(N \log |\Sigma| \frac{l}{w})$  memory, where  $w$  is the word size of the computer.*

*Proof.* We just have to notice that we can represent each letter of the alphabet using  $\log |\Sigma|$  bits, hence we can store each  $l$ -mer using  $\frac{\log \Sigma^l}{w}$  words. The time complexity results follows from noticing that we execute step 2  $\binom{l}{d}$  times.

One big advantage of Algorithm SOS is the fact that for a fixed value of  $l$  and  $d$  the algorithm is linear in  $N$ , making it practical for high values of  $N$ . However it is sensitive on the parameter  $l$  and particularly sensitive on parameter  $d$ , making it practical for values of  $d \leq 5$ .

Notice that we can decrease the memory used by the algorithm SOS to  $O(N)$  by storing the  $l$ -mers in collection  $X$  by their position numbers.

At the core of our SOS algorithm is the use of radix sorting, so is a natural step to try to use a combination of MSD and LSD radix sorting. This type of approach has been tried before in a different problem in [4] but for the sake of completeness we will describe it in full in the following paragraph.

Let  $(j_1, \dots, j_d)$  be a set of positions as in step 2 and let us write  $x' := x \downarrow_{j_1, \dots, j_d}$  where  $(x, i) \in X$ . It is clear that the problem of finding “repeated”  $l - d$ -mers can be solved if we partition the input into buckets  $B_1, \dots, B_{|\Sigma|^k}$  according to the first  $k$  MSD digits of  $x'$ . Then independently on each bucket we do an LSD radix-sort on the remaining  $l - d - k$  digits and eliminate “duplicates” as in step 2b.

An advantage of this approach is the fact that processing every bucket  $B_p$   $p = 1, \dots, |\Sigma|^k$  can be done independently of the others and if we assume that strings  $s_i$  are generated by a random source of characters over  $\Sigma$  with equal probability then the expected size of each  $B_p$  will be  $\frac{N}{|\Sigma|^k}$ .

This implies the following result

**Theorem 2.** *For a fixed  $k > 0$ , algorithm SOS can be implemented in parallel in an  $O(|\Sigma|^{-k} N \frac{l}{w} \binom{l}{d})$  expected time and  $|\Sigma|^k$  processors. The expected memory usage for each processor is  $O(|\Sigma|^{-k} N \log |\Sigma| \frac{l}{w})$ , where  $w$  is the word size of the computer.*

### 2.3 SOS-Hash: A Solution Based on Hashing

The use of hashing for pattern matching related problems was pioneered in [6] and has been used extensively in the pattern matching literature.

Notice that if we fix a given set of positions  $i_1, \dots, i_d$  it is clear that we can find all the  $l$ -mers which differ in those positions by using a hash table as in [3]. In doing so we make use of the representation of  $l$ -mers over  $\Sigma$  as numbers in base  $|\Sigma|$ .

In the following algorithm we will use two hash functions,  $g : \Sigma^l \rightarrow \{0, 1\}$  and  $h : \Sigma^l \rightarrow \{0, \dots, n\}$ .  $g$  will be used to tell whether a particular  $l$ -mer is in the solution set and  $h$  will store the index of the last sequence where a particular  $l$ -mer was found.

#### Algorithm SOS-Hash

1. Initialize  $g$  with 0 values.
2. For all  $(j_1 \dots j_d)$  with  $1 \leq j_1 < \dots < j_d \leq l$ 
  - (a) Initialize  $h$  with 0 values.
  - (b) For all  $i = 1, \dots, n$  and  $x \triangleleft_l s_i$ 
    - i. Let  $x' = x \downarrow_{j_1, \dots, j_d}$ .
    - ii. If  $g(x) = 0$  and  $h(x') = 0$  set  $h(x') = i$ .
    - iii. If  $g(x) = 0$  and  $1 \leq h(x') \neq i$  set  $g(x) = 1$ .
3. For all  $i = 1, \dots, n$  and  $x \triangleleft_l s_i$  output  $x$  if  $g(x) = 0$ .

We use as hash functions  $g(x) = h(x) = \tilde{x} \bmod p$ , where  $\tilde{x}$  is the representation of  $x$  as a number in base  $\Sigma$  and  $p$  is an integer –usually a prime.

**Theorem 3.** *Algorithm SOS-Hash takes  $O(Nd \binom{l}{d} + p)$  time and  $O(p)$  memory.*

*Proof.* We should only point out that given two consecutive  $l$ -mers  $x$  and  $y$  in a given  $s_i$  it is known how to calculate  $g(y)$  given  $g(x)$  in constant time. Furthermore we can calculate the value of  $h(y')$  given the value of  $g(y)$  and then subtracting an appropriate amount that would depend only on the values of  $y$  at the positions  $j_1, \dots, j_d$ .

In case  $l \leq \log_{|\Sigma|} N$  we can use  $p = |\Sigma|^d$  and the algorithm will be exact and its run time will be  $O(Nd \binom{l}{d})$ , using  $O(N)$  memory. For larger values of  $l$  the memory consumption can become impractical so we might settle for a Monte Carlo version of this algorithm. It is a well known result from [6] that if we choose  $p$  to be a prime less than  $N^{1+\epsilon} l \log(N^{1+\epsilon} l)$  the probability that for every  $(j_1, \dots, j_d)$  chosen in step 2 we get a false positive is  $O(\frac{1}{N^\epsilon})$ .

### 3 Specific Selection with Longest Common Factor Constraint

In [5] it was shown that a designed siRNA may bind with an off-target mRNA sequence if they shared a specific number of consecutive matches. This type of constraint was considered in the work of [11] under the name of *longest common factor* and will be made formal in the following definitions.

**Definition 5.** Let  $s$  and  $x$  be strings over  $\Sigma$  with  $|x| = l < |s|$ . We denote by  $lcf(x, s)$  the longest common factor, i.e. the longest common contiguous string shared between  $x$  and  $s$  and  $y$ . In formal terms, we say  $y := lcf(x, s)$ ,  $|y| = t$  iff

$$(y \triangleleft_t x \wedge y \triangleleft_t s) \text{ and } (\forall h > t : z \triangleleft_h x \Rightarrow \neg(z \triangleleft_h s)) \quad (3)$$

**Definition 6.** Let  $\mathcal{S} = \{s_1, \dots, s_n\}$  be a set of sequences over  $\Sigma$ . Given  $l, d, h$  the  $(l, d, h)$  Specific Selection Problem with Longest Common Factor Constraint consists of finding a set of  $l$ -mers  $\mathcal{X} = \{x_1, \dots, x_n\}$  such that they satisfy (4) and (3)

$$\forall(1 \leq i \leq n) : \forall(j \neq i) : |lcf(x_i, s_j)| > h \quad (4)$$

In case for a particular  $i$  there is no  $x_i$  that satisfies (4) we set  $x_i = \emptyset$ .

It is clear that we can solve this problem in two phases. In the first phase we mark the  $l$ -mers which do not satisfy (3). This could accomplish just by sorting all the  $l - h$ -mers and recording the duplicates coming from different  $s_i$ .

In the second phase we can run either SOS or SOS-Hash. Notice that in this case it is not necessary to try all the possible  $(i_1, \dots, i_d)$  with  $1 \leq i_1 \leq \dots \leq i_d \leq n$  but a subset of these, namely the ones that satisfy the condition (5)

$$i_1 - 1 < h \text{ and } i_2 - i_1 < h \text{ and } \dots \text{ and } i_{d-1} - i_d < h \text{ and } n - i_d < h, \quad (5)$$

since by condition (4) the set of  $l$ -mers at the end of Phase (I) will not share a subsequence of length  $h$  or bigger.

It should also be pointed out that in case in the first phase we discard a big fraction of the  $l$ -mers in the input, it would be faster and makes more sense to calculate  $\bar{d}(\cdot, \mathcal{S})$  as in [11] for the unmarked elements.

### 4 Experiments

We implemented algorithm SOS as a C program and tested on the complete mRNA data for Human <sup>1</sup> which  $N = 9.3 \times 10^7$  and  $n = 3.8 \times 10^4$  – and *Drosophila Melanogaster* <sup>2</sup> for which  $N = 4 \times 10^7$  and  $n = 1.9 \times 10^4$ .

The programs were run on a Power Edge Linux Server with 4GB of RAM and dual Xeon 2.66 Ghz CPU's –only one which was used. In processing the Human

<sup>1</sup> From [ftp://ftp.ncbi.nih.gov/refseq/H\\_sapiens/mRNA\\_Prot/human.rna.gz](ftp://ftp.ncbi.nih.gov/refseq/H_sapiens/mRNA_Prot/human.rna.gz)

<sup>2</sup> From <ftp://hgdownload.cse.ucsc.edu/goldenPath/dm1/bigZips/mrna.fa.gz>

mRNA data we used close to 1.5Gb of RAM and in the case of the Drosophila we used close to 700Mb of RAM, due to the fact that we store the  $l$ -mers as 64 bit numbers.

In the particular case of the Human mRNA with  $l = 19$  and  $d = 3$  our algorithm took 3 hours and 22 minutes, outperforming the results in [11] by almost two orders of magnitude.

In table 1 we show the run time, memory usage and number of  $l$ -mers which satisfy the Specific Selection Problem for values of  $l = 19, 20, 21$  and  $d = 3$ . Of particular interest is the fact that as we consider larger values of  $l$  the number of possible  $l$ -mers grows exponentially. In those case we can use conditions such as (4) or the ones defined on [8, 10] in order to prune the space of possible results.

Species	Size(bp)	$l$	Time	Memory Used	Size of Solution
Human	$9.3 \times 10^7$	19	202 m	1.5 Gb	$2.1 \times 10^5$
		20	251 m	1.5 Gb	$2.7 \times 10^6$
		21	344 m	1.5 Gb	$1.1 \times 10^7$
Drosophila	$4 \times 10^7$	19	98 m	0.7 Gb	$8.5 \times 10^5$
		20	124 m	0.7 Gb	$4.5 \times 10^6$
		21	167 m	0.7 Gb	$9.4 \times 10^6$

**Table 1.** SOS Performance for  $d = 3$

In table 2 we consider the Human mRNA dataset and we fix  $l = 19$  and  $d = 3$  and consider the variant of SOS for the Specific Selection Problem with Longest Common Factor Constraint, for values of  $h = 12, 13, 14$ . We include the number of  $l$ -mers at the end of Phase I and II. Notice that in case we use a small value of  $h$  –in our case 12– the number of  $l$ -mers which needs to be pruned in Phase I reduces drastically and it may be more efficient to use the approach in [11]. However as the values of  $l$  increases it is more practical to use an algorithm like SOS.

$h$	Time	Size Phase (I)	Size Phase (II)
12	179 m	$3.3 \times 10^4$	$6.0 \times 10^3$
13	192 m	$1.1 \times 10^6$	$7.0 \times 10^4$
14	200 m	$9.4 \times 10^6$	$1.7 \times 10^5$

**Table 2.** SOS Performance for Human mRNA,  $l = 19$ ,  $d = 3$

## References

1. S. Balla, S. Rajasekaran, Sorting and FFT Based Techniques in the Discovery of Biopatterns, *Bioinformatics Algorithms: Techniques and Applications*. Wiley Book

- Series on Bioinformatics (Series Editors: Yi Pan and Albert Y. Zomaya). to appear.
2. S. Balla, S. Rajasekaran, Space and Time Efficient Algorithms to Discover Endogenous RNAi Patterns In Complete Genome Data, *International Symposium on Bioinformatics Research and Applications (ISBRA 2007)*, May 2007.
  3. F Y.L. Chin and H C.M. Leung, Voting Algorithms for Discovering Long Motifs, *Proceedings of the Third Asia-Pacific Bioinformatics Conference (APBC2005), Singapore*, 261-271, January 2005.
  4. S. Elbashir, J. Harboth, W. Lendeckel, A. Yalcin, K. Weber, T. Tuschli. Duplexes of 21-nucleotide RNAs mediate RNA interference in cultured mammalian cells, *Nature*, 411, 494-498. 2001.
  5. A. Jackson, S. Bartz, J. Schelter, S.Kobayashi, J. Burchard, M. Mao, B. Li, G. Cavet P. Linsley, Expression profiling reveals off-target gene regulation by RNAi , *Nature Biotechnology* , 21, 635-637, 2003.
  6. R. Karp and M. Rabin, Efficient randomized pattern matching algorithms, *IBM Journal of Research and Development*, 31:249-260, March 1987.
  7. S. Rajasekaran, S. Balla, C.-H. Huang, V. Thapar, M. Gryk, M. Maciejewski, M. Schiller, Exact Algorithms for Motif Search ,*Proc. of the Third Asia-Pacific Bioinformatics Conference*, 239-248, 2005.
  8. A. Reynolds, D. Leake, Q. Boese, S. Scaringe, W.S. Marchall, A. Khvorova, Rational siRNA design for RNA interference *Nature Biotechnology*, 22, 326-330, 2004.
  9. W. K. Sung, W. H. Lee, Fast and Accurate Probe Selection Algorithm for Large Genomes, *CSB 2003*, 2003, pp. 65-74
  10. K. Ui-Tei, Y. Naito, F. Takahashi, T. Haraguchi, H. Ohki-Hamazaki, A. Juni, R. Ueda, K. Saigo , Guidelines for the Selection of Highly Effective siRNA Sequences for Mammalian and Chick RNA Interference *Nucleic Acid Research*, 22, 326-330, 2004.
  11. T. Yamada, S, Morishita, Accelerated off-target search algorithm for siRNA. *Bioinformatics* 21(8), 2005, pp. 1316-1324.
  12. J. Zheng, T. J. Close, T. Jiang, S, Lonardi. Efficient selection of unique and popular oligos for large EST databases. *Bioinformatics* 20(13), 2004, pp. 2101-2112.