

# Primer Design for MPLA Experiments

Jeroen Laros

January 13, 2005

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Goals</b>	<b>4</b>
<b>3</b>	<b>Finding good primers</b>	<b>5</b>
<b>4</b>	<b>Pairwise alignment</b>	<b>8</b>
<b>5</b>	<b>Generating the pairwise alignment output</b>	<b>10</b>
<b>6</b>	<b>Experiments</b>	<b>11</b>
<b>7</b>	<b>Conclusion</b>	<b>12</b>

# 1 Introduction

The goal of this project is the following. Given a small single strand of DNA (DeoxyriboNucleic Acid), find particular pieces within this strand that are (almost) unique, and adhere to certain demands like bonding energy and temperature. Iterate this process for 40.000 given strands.

A primer is a small piece of DNA which is (almost) unique for a particular DNA sequence. The reason we are interested in primers is because we can use them for all sorts of purposes. For example, we can see if a certain gene is present in the genome or we can use primers to copy or isolate certain pieces of DNA. The latter is what we focus on in this document; it is a technique referred to as PCR (Polymerase Chain Reaction) [2]. With PCR it is convenient when the used primers are unique.

**Properties of DNA** A DNA molecule has a lot of interesting properties, some of which are so important that we need to take them into account. The most important of them is the bonding energy. It takes some energy to separate two pieces of DNA; the same amount of energy is gained when the pieces are merged. We can make a good approximation of the bonding energy needed by counting the occurrences of C and G and by looking how the occurrences are spread over the piece of DNA.

This is because the C-G base pairs are responsible for most of the bonding energy, the A-T bonding is made up of two H-bridges and the C-G bonding is made up of three H-bridges. It would seem that the C-G bond is 1.5 times stronger than the A-T bond, however this is not the case because there are more than one type of H-bridges and the bonding energy of H-bridges depends on the neighboring atoms as well.

There is a formula [1] that gives the bonding energy given a strand of DNA and by using this formula we can also calculate the temperature at which a strand of DNA lets go of another strand of DNA (in practice this is the temperature at which the double helix structure (of a particular strand of DNA) breaks down and forms two single strands); this is called the *melting temperature*. Basically, the formula counts all types of neighbouring basepairs and assigns an energy based upon these numbers and a lookup table. The values in this table were experimentally determined.

Another important property of DNA is that single strands are oriented. Because of this a piece of DNA has two distinguishable ends; the 5" and the 3" end; DNA in nature is always read from 5" to 3". The last thing we have to take into account is that DNA is a double stranded helix which consists of complementary strings. These strings are read in different directions, which means that when we encounter a string, we also encounter the *reverse complement* of this string so on one hand we can only read in one direction, but on the other hand we read the reverse complement simultaneously; from now on we shall refer to the reverse complement simply as the *complement*.

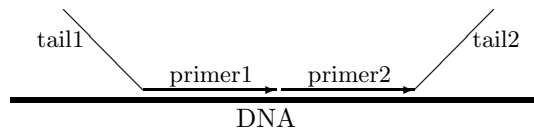


Figure 1: Primers sticking to the DNA at temperature  $T_1$ .

**Properties of primers** If we want to have good primers, we have to make sure that they only stick to one particular piece of the genome. This also means that primers may not stick to each other or themselves. The ability of two pieces of DNA to stick to each one another can be approximated by aligning them and evaluating matches and mismatches, also see section 4.

This project is at the request of Peter H.K.G. Taschner of the Bioinformatics Support Group and in cooperation with Hendrik Jan Hoogeboom and Walter A. Kusters, both of the Leiden Institute of Advanced Computer Science.

## 2 Goals

The primary goal of this project is to find primers in certain isolated areas of the genome, where we already know that these areas are important, and in a way we want to have a fingerprint of each area.

In the case of MPLA (Multiplex Ligation-dependent Probe Amplification) we first want to have two primers that are adjacent; these primers are supplied with tails, one for the left primer and one for the right one. The tails are known in advance. Then the primers are merged together with an enzyme. The result of this is a piece of DNA that can be recognized by its tails and therefore it can be amplified in order to find out whether or not the piece of target DNA (the piece of DNA our strand is designed for) is present or not.

This is globally what happens:

1. At a certain temperature  $T_1$  both primers have merged with the DNA as can be seen in Figure 1.
2. By adding an enzyme the primers are merged as can be seen in Figure 2.
3. By increasing the temperature to  $T_2 > T_1$  the merged primer pair lets go and is ready for multiplication (or amplification) using PCR and the (complement of the) tails as primers. The primer pair can be recognized by its tails (as can be seen in Figure 3).

To reach these goals, we have to do two things:

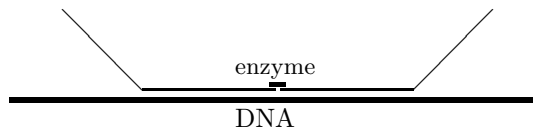


Figure 2: Primers merging

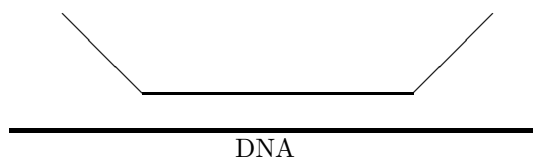


Figure 3: Primer pair letting go at temperature  $T_2$

- We have to alter the behavior of a frequently used program for finding primers. This will be discussed in Section 3.
- We have to test all primers with respect to each other, because the primers are used simultaneously. This will be discussed in Section 4.

### 3 Finding good primers

There is a much used program that can find primers: Primer3 [4]. This program is able to take a lot of parameters into account. For instance we may specify the melting temperature, as well as the desired length of the primer. Primer3 also checks that the primer does not stick to itself or to other primers. However, for our purposes Primer3 needed a couple of alterations.

Because we did not want to alter the overall behavior of Primer3, we only expanded it with a couple of options.

Primer3 is available via a webinterface and as a downloadable program. We have downloaded version 0.9\_test from September the 21-th of 1998 (the last version we could find) and altered the source.

**Primer pairs** Primer3 determines a primer pair on both strands, the second primer is found on the complementary strand, but since need two primers on the same strand an in the same direction we have to complement the second primer.

In order to find two primers on the same DNA strand (and therefore in the same direction), we searched for the complement on the complementary DNA under

the assumption that the bonding energy is invariant under complementation. This new option is called `PRIMER_SAME_STRAND`. For the default value 0, the program behaves as before.

**Head-tail primer pairs** We have to find primers that make contact, with no overlap. This is needed to accommodate for step 2. This new option is named `PRIMER_CONTACT`. Note that Primer3 has the option to specify a gap length between the primers, but when to 0, it is interpreted as ‘default’, i.e., no preference given.

**Post-processing** In order to make the output suitable for post-processing we added the option to give minimal output. The reason for this is that after we have found thousands of primers, we also want to make sure that all these primers do not stick to each other. Rather than pairwise alignment, this test is not done with Primer3 because Primer3 is not designed to test large amounts of primers for alignment. A different program discussed later on is. The new post-processing option for Primer3 is named `PRIMER_FRONTEND`. It will print bare output to standard error in addition to the normal output.

**Clamps** We have implemented a number of clamps to make sure a primer starts or ends with a G or a C. The default clamp (`PRIMER_GC_CLAMP`) was not applicable because it was on the wrong side (the 3’ end) of the primer. The reason for those clamps is that certain enzymes do not work when a strand of DNA does not start with C or G. The new options are named `PRIMER_CG_L_CLAMP` and `PRIMER_CG_R_CLAMP`. Both of them apply to the 5’ end of the primers, and the L and R indicate on which of the two primers the clamps are activated. The clamps are applied after we look whether or not the primers must be found on the same strand, so it will always be on the 5’ end of the primer.

**Overview** An overview of all implemented options and the values we have assigned to them in our experiments, can be found in Table 1. By default all options are set to 0; this is to ensure that the normal behavior of Primer3 is not affected. Options are given to Primer3 via stdin, see the README of Primer3 for more information.

Apart from the options we implemented ourselves, we used the options shown in Table 2 from the regular program to get the desired output.

Furthermore the executable is called with the options shown in Table 3 (also from the regular program).

A sample of the desired output which we can use to add the tails and then investigate further can be seen in Figure 4. The lines starting with a ; are comments and usually display the name of the strand of DNA where the primer pair is found. If no good primer pair is found, there will be three lines of comment (as can be seen in the example).

Name	Value	Description
PRIMER_CONTACT	1	Makes sure the primers make contact.
PRIMER_CG_L_CLAMP	1	Makes sure the 5" end of the left primer starts with C or G.
PRIMER_CG_R_CLAMP	0	Makes sure the 5" end of the right primer starts with C or G.
PRIMER_SAME_STRAND	1	Finds two primers on the same strand.
PRIMER_FRONTEND	1	Prints the primer candidates to standard error.

Table 1: Primer3: Implemented options and their values for our purposes

Name	Value	Description
PRIMER_EXPLAIN_FLAG	1	Enable verbose output and statistics.
PRIMER_PRODUCT_SIZE_RANGE	40-60	Specifies the length of the two primers plus the space in between.
PRIMER_MIN_TM	62	Minimum melting temperature for a primer oligo (in degrees Celsius).
PRIMER_OPT_TM	65	Optimum melting temperature.
PRIMER_MAX_TM	70	Maximum melting temperature.

Table 2: Primer3: Used existing options

Name	Description
-strict_tags	Give an error for unrecognized input options.
-format_output	Print a more user-oriented report for each sequence.

Table 3: Primer3: Command line parameters

```

;A1
CCTGGTATGACAACGAATTTGGCT
ACAGCAACAGGGTGGTGGAC
;A2
CACTTAAATACCTCGTGTATGGTGCAA
TCAGACCACAAAATCAGAAGCTGG
;A3
CCACGCCAGACATTGTGTCA
CGCATCACGCAGTACATCGC
;B4
;
;

```

Figure 4: Primer3; Bare output as prepared using our new option `PRIMER_FRONTEND`

**Tails** The fixed tails are added with a small script that takes input like the example in Figure 4 and adds the left tail to the left end of the odd primers and the right tail to the right end of the even primers. The tails are named `MLPAF` (`GGGTTCCCTAAGGGTTGGA`) and `MPLAR` (`TCTA-GATTGGATCTTGCTGGCGC`) and are added (by a simple shellsript) to the bare output of Primer3 and before the real post-processing begins.

## 4 Pairwise alignment

As mentioned before, the primers that are found still need to be tested against each other to make sure that they do not stick together. A program called `ntdpal` (it is included in the Primer3 distribution and can be used by typing `make ntdpal` in the source directory) does something similar. The primary difference between what `ntdpal` does and what we want is that `ntdpal` calculates the alignment of two primers and we want to know how well the primers stick to each other. So what we need to do is to (reverse) complement the second primer for the program. We name this option `-c`.

We test for two types of alignment: *global* [3] and *local* [5] and in this particular case the medical engineers were interested in global *end-anchored* alignment and normal local alignment. The end-anchor means that the end of the first sequence must take part in the alignment. `Ntdpal` and `bulkal` use the following abbreviations for the different types of alignment: `g`: global, `G`: global end-anchored, `l`: local and `L`: local end-anchored.

The global alignment algorithm calculates a kind of edit-distance between two strings using dynamic programming; with distance we mean a kind of topological or Hamming distance.

There are two things to worry about when testing for an alignment: a mismatch and the possibility of a gap. For example we want to align the strings `ATGTTCA`

```

ATGT-TCA
|  |  |  |
AGGTAACA
12345678

```

Figure 5: Global alignment example

Option	<code>ntdpal</code>	<code>bulkal</code>	Explanation
<code>gap</code>	-100	-200	The “gap opening” penalty.
<code>gapl</code>	-100	-200	The “gap extension” penalty.
<code>max_gap</code>	3	1	The maximum allowable size for a gap.
<code>check_chars</code>	1	0	Check for illegal characters in the input when non-0.
<code>score_only</code>	0	1	Only print the score when non-0.

Table 4: Differences in initialization functions between `ntdpal` and `bulkal`

with `TGTTACCT`, the result of which can be seen in Figure 5.

We see that at position 2, 5 and 6 there is a mismatch and we see that at position 5 a gap is inserted, otherwise position 7 and 8 would not match.

The reason that the gap is inserted is because we give awards for matches, and punishment for mismatches and gaps. We give 1 point for a match, -1 for a mismatch and -2 for a gap. Since the overall score of this alignment does not change when we insert a gap (the gap could also be inserted at position 6) and because we are looking for the maximal alignment, the gap is inserted. The overall global alignment score of this alignment is therefore  $1 - 1 + 1 + 1 - 2 - 1 + 1 + 1 = 1$ . The maximal gap length is set to 1.

Local alignment is very similar to global alignment, except that it looks for substrings, which means that we do not give punishment for deletion of a prefix and/or suffix of one of the sequences, another way of looking at this is that gaps at the beginning or the end receive no penalty.

**Bulkal** In order to analyze the primer pairs in a reasonable amount of time, we chose to load all primers into memory and then test them. We did this by making our own `ntdpal` program which we named `bulkal`.

Then we encountered another difference. It seemed that a lot of standard variables were set differently in the program `ntdpal`, so we also took the initialization function of `Primer3` (`set_dpal_args()`) and incorporated it in `bulkal`. The differences between the two initialization functions can be seen in Table 4.

The program `bulkal` does not require any options. Redirect or pipe the input through the executable, and then human-readable information will be printed on `stdout`, and progress indication output on `stderr`; to omit the information and produce binary output, use the `-i` option.

```

|GGGTTCCCTAAGGGTTGGACCTGGTATGACAACGAATTTGGCT|
|ACAGCAACAGGGTGGTGGACTCTAGATTGGATCTTGCTGGCGC|    l score = 4
|GGGTTCCCTAAGGGTTGGACCTGGTATGACAACGAATTTGGCT|
|ACAGCAACAGGGTGGTGGACTCTAGATTGGATCTTGCTGGCGC|    G score = 1
|GGGTTCCCTAAGGGTTGGACCTGGTATGACAACGAATTTGGCT|
|GGGTTCCCTAAGGGTTGGACCTAAATACCTCGTGTATGGTGCAA|    l score = 7
|GGGTTCCCTAAGGGTTGGACCTGGTATGACAACGAATTTGGCT|
|GGGTTCCCTAAGGGTTGGACCTAAATACCTCGTGTATGGTGCAA|    G score = 4
|GGGTTCCCTAAGGGTTGGACCTGGTATGACAACGAATTTGGCT|
|TCAGACCACAAAATCAGAAGCTGGTCTAGATTGGATCTTGCTGGCGC| l score = 6
|GGGTTCCCTAAGGGTTGGACCTGGTATGACAACGAATTTGGCT|
|TCAGACCACAAAATCAGAAGCTGGTCTAGATTGGATCTTGCTGGCGC| G score = 1

```

Figure 6: Bulkal: ASCII output

**Internals of bulkal** Because `bulkal` gets pre-formatted text (as output by `Primer3` under our new option `PRIMER_FRONTEND`) there are not many options necessary, but since global alignment depends on which primer is the left one and which is the right one, we internally keep track of which primer is which: all odd primers are on the left side, all even primers are on the right side. An input sample can be seen in Figure 4; each primer is tested for local and global alignment with every other primer. The result of this is two scores per primer pair, one for local and one for global alignment.

The output of this program is either of the form seen in Figure 6 or in binary form. The data in this example can be interpreted as follows. The local alignment score between the first left primer and the first right primer is 4, the Global alignment score is 1, the local alignment score between the first left primer and the second left primer is 7, the global alignment score is 4 and so on. At the moment we do not analyze the obtained scores.

## 5 Generating the pairwise alignment output

We can now load the candidate primer pairs from the output file of (the adapted) `Primer3` and test each pair after we have added the tails.

**Storing the information** Let  $n$  be the number of primers. Because there are  $\binom{n}{2}$  combinations of primer pairs, we chose to store only the scores. We store each score in one byte, so the output file is  $\binom{n}{2}$  bytes large. Concrete: for our 40000 primers, there will be 763 Megabytes of output. This can be reduced by a factor of two if we do not allow 256, but 16 possible values for the score (which is not a bad idea, since there are very few primer pairs that have a score higher than 16, in which case we could make a separate list of these pairs (when these pairs would be of any interest)).

**Retrieving the information** We can find the score of the primer pair  $(i, j)$  at position:

$$k = \sum_{\ell=1}^i (n - \ell) + j - i - 1,$$

where  $0 \leq i < n - 1$ ,  $i < j < n$ ,  $0 \leq k < \binom{n}{2}$ . We can rewrite this into an explicit function:

$$k = \frac{i(2n - i - 1)}{2} + j - i - 1$$

We can see this is equivalent by induction over  $i$ :

*Proof.* Hypothesis:

$$\sum_{\ell=1}^i (n - \ell) = \frac{i(2n - i - 1)}{2}$$

Base:

$$i = 1 \rightarrow n - 1 = \frac{2n - 2}{2}$$

Induction step:

$$\begin{aligned} \sum_{\ell=1}^{i+1} (n - \ell) &= \sum_{\ell=1}^i (n - \ell) + (n - i - 1) \\ &= \frac{i(2n - i - 1)}{2} + (n - i - 1) \\ &= \frac{(i + 1)(2n - (i + 1) - 1)}{2} \end{aligned}$$

■

As long as  $k = \binom{n}{2} < 4$  Gigabytes, we can address it with an 32-bits integer, which makes  $n < 92683$ .

## 6 Experiments

All experiments were done on an Intel PIV Mobile at 1.7 GHz with 256MB of memory.

**HUMLIB96** The first experiments were done on HUMLIB96 (2001), the Human OligoLibrary from Compugen Incorporated, which consists of 18861 strands of DNA with a length of 61 nucleotides. It took Primer3 3 minutes and 55 seconds to complete its analysis; the result consisted of 16076 primer pairs. Adding the tails took 21.5 seconds.

The post processing step took 14 hours and 41 minutes to complete and it resulted in an output file of 493 MB in binary form; we did not examine the ASCII output because of disk limitations, but we expect the output to be roughly 54 times larger than the binary output.

**Ultra conserved sequences** We also did experiments on *ultra conserved sequences*; these are sequences found in more than one species, like men and mice. We had 482 stands of DNA with a mean length of 262 nucleotides. It took Primer3 28 seconds to complete and the result consisted of 448 primer pairs. The time it took to add the tails is too short to make it worth mentioning. The post processing step took 48 seconds and resulted in an output file of 392 kB in binary form, in ASCII form it is 85 MB.

**Physical experiments** To our knowledge only the data from the experiment on the ultra conserved sequences is used in practice. At the time of writing the designed primers are synthesized.

## 7 Conclusion

We conclude that determining primers on a large scale is reasonably doable, but the alignment of all possible pairs is the main bottleneck.

Further research can be analyzing the output of `bulkall` and by using the result of this analysis to select the best primers. The primers that are thrown away can be replaced by other primers by doing another iteration of the entire process (starting with Primer3 to `bulkall`). It should be fairly easy to let Primer3 return the second- or third-best primers, this way we can try to find at least one primer pair per given strand.

## References

- [1] Breslauer, K.J., Frank, R., Blöcker, H. and Marky, L.A. (1986) Predicting DNA duplex stability from the base sequence. *Proc. Natl. Acad. Sci. USA* 83, 3746–3750.
- [2] Dieffenbach, C.W and Dveksler, G.S. (1995) PCR primer: A laboratory manual. CSHL press, Cold Spring Harbor, USA.
- [3] Needleman, S. B. and Wunsch, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins (1970). *J. Mol. Biol.* 48, 443–453.
- [4] Rozen, Steve, Skaletsky, Helen J., Primer3 (1996, 1997, 1998), [http://www-genome.wi.mit.edu/genome\\_software/other/primer3.html](http://www-genome.wi.mit.edu/genome_software/other/primer3.html)
- [5] Smith, T. and Waterman, M. (1981). Identification of common molecular subsequences. *J. Mol. Biol.* 147, 195–197.