

# Synthesizing Arbitrary Genomes

Dennis Shasha\*      Zasha Weinberg†

September 20, 1999

**Corresponding author:** Dennis Shasha (shasha@cs.nyu.edu)  
Courant Institute, New York University, 251 Mercer Street, New York,  
New York, 10012.  
Phone: (212) 998-3086; Fax: (212) 995-4123

**Running head:** Synthesizing Arbitrary Genomes

**Key words:** large-scale DNA synthesis, prosthetic sequences, molecular engineering, nucleotide folding, shearing.

---

\*Department of Computer Science, Courant Institute, New York University, 251 Mercer Street, New York, New York, 10012 (shasha@cs.nyu.edu). Much of the work was done while this author was in INRIA, Rocquencourt, France.

†Department of Computer Science and Engineering, University of Washington Box 352350, Seattle, Washington, 98195 (zasha@cs.washington.edu). Much of the work was done while this author was employed at Nicholson NY, LLC.

## Abstract

Suppose a researcher wants a long arbitrary sequence of nucleotides and asks a lab to synthesize it. Oligonucleotides are generally of length less than 100, so it is necessary to resort to Watson-Crick pairing and ligation for longer strands. Some of the longest strands so far generated have had 15,000 bases, but those strands weren't arbitrary since they were specially designed for the purpose of DNA computing.

Our algorithm produces a recipe for making a length  $n$  strand of arbitrary DNA. The algorithm requires an expected computation time of  $O(n^2)$ , though its worst case time is  $O(n^4)$ . Its expected total laboratory time is  $O(n)$ , assuming that an arbitrary number of oligonucleotides can hybridize in constant time. We illustrate its application on long sequences from Human Chromosome 7 and random sequences of length 10 million. Our algorithm requires the invention of some laboratory techniques, since it requires the use of special enzymes, techniques to prevent shearing, and techniques to prevent folding. Still, this effort will be well recompensed by the medical and scientific benefits of the dream: *build any genome you wish.*

## 1 Introduction

Suppose we are given an arbitrary sequence of letters,  $Sdna$ , drawn from the alphabet A,C,G and T. Our problem is to manufacture the nucleotide sequence  $Sdna$ . A simple technique that almost works is to partition the sequence of letters into consecutive subsequences  $Sdna_1, Sdna_2, \dots, Sdna_n$ , such that each  $Sdna_i$  is of modest length (less than 100 bases). At this length, each  $Sdna_i$  can be individually manufactured using conventional techniques [Gait, 1984]. For each  $Sdna_i$ , we generate a Watson-Crick complement of the end of  $Sdna_i$  and the beginning of  $Sdna_{i+1}$  to link these. We call the  $i^{th}$  linking complement  $Ldna_i$ . Then we mix the  $Sdna_i$  and  $Ldna_i$  together and use ligase and gel electrophoresis in order to produce a strand (with complements at link points) having the proper length. That strand will be  $Sdna$ .

In [Stemmer, 1995], Stemmer used a similar process to create 2-3 kilobase oligonucleotides, with errors in the creation. These errors were intended for mutagenesis studies. Our goal is to create error-free sequences in the spirit of Adleman's classic paper on molecular computation [Adleman, 1994], [Agarwal *et al.*, 1970].

There are several problems with the approach described above: The first is that long strands are susceptible to shear forces, and may break [Paun *et al.*, 1998]. The second is that single-stranded DNA may fold when parts of the strand bind to other parts of the strand. The third is that gel electrophoresis is not precisely accurate at extremely long lengths. The fourth is that some of the linkers may link the wrong strands, e.g.  $Ldna_i$  may link  $Sdna_j$  with  $Sdna_k$  where  $j + 1 \neq k$ .

We ignore the first two problems (shear and folding) for the purpose of this paper, in the hopes that these limitations will disappear with the advance of technology. We address the third problem in that we don't demand complete accuracy of electrophoresis. Our algorithm is mainly directed at avoiding the fourth problem. It gives a way to choose cut points and linkers to avoid false linkages.

Section 2 defines some necessary terms and notation. Section 3 describes the laboratory process we propose for creating long DNA strands. This process consists of a number of experiments carried out in a lab, and a computer program that designs the experiments. Section 4 presents the algorithm for experimental design, including the notion of recursive synthesis. In section 5, we discuss a "prosthetic bridging" strategy for sequences that are difficult to build using the other methods. Section 6 contains the

results of computer simulations of our approach. Section 7 presents conclusions and directions for future work.

## 2 Terms and Notation

**Sdna.** The DNA sequence we wish to synthesize. This denotes both the sequences of bases using the symbols A,C,G and T, as well as the actual physical DNA oligonucleotide itself.

**DNA strand/oligo.** A DNA strand is a sequence of letters from the alphabet A,C,G and T, that “runs” in a particular direction. For consistency, we assume (arbitrarily) that strands run from their 5’ terminus to their 3’ terminus. Strands can be “reversed” to run in the opposite direction simply by flipping them. Strands will not hybridize unless they run in opposite directions from one another. The size of strand  $X$  is denoted  $|X|$ .

**complement of strand.** Always the Watson-Crick complement, wherein A and T are complements, and C and G are complements. The complement of strand  $X$  is written  $\overline{X}$

**reversal of strand.** Changing the direction of the strand, in terms of its 5’ vs. 3’ ends. We write the reversal of strand  $X$  as  $X^r$ .

**concatenation of strands** If strand  $Y$  is concatenated onto the end of  $X$ , this is denoted  $X + Y$ .

**bridging.** We say that oligo  $X^r$  “bridges” oligos  $Y$  and  $Z$  when  $X^r$  will hybridize with  $Y$  and  $Z$  in such a way that  $Y$  and  $Z$  will be adjacent with  $Y$  nearer to the 5’ end of the strand and  $Z$  nearer the 3’ end of the strand. At that point, ligase will join  $Y$  and  $Z$  to create a new single strand,  $Y + Z$ .

## 3 Process

In this section, we describe the basic process for constructing a long strand of DNA. Later we discuss modifications that can arise in exceptional cases.

### 3.1 Steps for DNA Synthesis

Our global strategy for producing  $Sdna$  (see fig.1) consists of two steps:

1. Synthesize  $Sdna$  among other strands that result from side effects.
2. Isolate  $Sdna$  from the other strands.

<b>Fig. 1 goes here</b> (Tables & figures are at end of document, on separate pages.)
---

### 3.1.1 Steps to Synthesize Sdna

1. Run computer program to specify an appropriate set of short oligonucleotides, based on the given *Sdna*. Some of these oligos will be consecutive subsequences from *Sdna*, *Sdna<sub>1..N</sub>*. These will be called strands. The others will be linkers, *Ldna<sub>1..N-1</sub>*, which will link consecutive pairs of *Sdna<sub>1..N</sub>* together *and no other pairs*. (When this last italicized phrase fails to hold, we must build strands or linkers using the recursive synthesis technique described later.)
2. Synthesize oligonucleotides using standard techniques.
3. Amplify oligos. Having more oligonucleotide copies may be necessary to improve the likelihood of constructing *Sdna*.
4. Combine the oligos such that they can hybridize.
5. Apply ligase. This can join together subsequences of *Sdna*, to form *Sdna* itself.

### 3.1.2 Steps to Isolate Sdna

1. Now that we have created *Sdna*, we must isolate it. Our basic idea is to denature the DNA, and use electrophoresis to extract *Sdna*. By design, we will ensure that no other oligo will be of similar length to *Sdna*, if it has the wrong sequence.

Synthesis will produce some proper substrands of *Sdna*. These substrands will always be missing *Sdna<sub>1</sub>*, *Sdna<sub>N</sub>*, or both because of the italicized phrase in step 1 above. If *acc* is the accuracy of electrophoresis (e.g., *acc* = 0.1 means 10% accuracy) for measuring length, then we require  $|Sdna_1|, |Sdna_N| \geq acc \cdot |Sdna|$ . This enables us to distinguish *Sdna* from the proper substrands. If necessary, we can build the longer *Sdna<sub>1</sub>* and *Sdna<sub>N</sub>* using the recursive synthesis procedure presented later in the paper.

We note that there are other methods of isolating *Sdna* (as well as synthesizing it), which avoid this use of electrophoresis.

For example, we could use oligos attached to magnetic beads as Adleman did [Adleman, 1998]. We can design such oligos, *Bdna* and *Edna*, that will bind to *Sdna<sub>1</sub>* and *Sdna<sub>N</sub>*, respectively. We then use them to retrieve *Sdna* only.

## 3.2 Specification of Our Program

We now describe the input/output behavior of the program supporting the synthesis procedure above. The program accepts the following input:

- The sequence *Sdna*, which consists of the symbols A,C,G and T.
- A parameter *maxlen*, defining the maximum length of an oligonucleotide that can be synthesized with standard techniques, currently less than 100.

- An error rate, “allowed difference” (or “*allowed dif*”). This is the edit distance (i.e. operations are insert, deleted change) between two strands  $X_1$  and  $\overline{X_2}$  such that  $X_1$  and  $X_2$  will perform a Watson-Crick pairing. We define this as a constant for any given target strand problem, but must be conservative because the error rate depends on many variables, such as length and C-G content [Hershlag, 1991],[Frutos *et al.*, 1997].

Given this input, the program will produce output in the following form:

- A list of substrands,  $Sdna_i$ , consisting of the symbols A,C,G and T. When joined together, these subsequences will produce  $Sdna$ . No  $Sdna_i$  will be greater in length than  $maxlen$ .
- A list of linkers,  $Ldna_i$ . No  $Ldna_i$  will be greater in length than  $maxlen$ .

The output must be constrained such that we can recover an oligonucleotide with sequence  $Sdna$  in the electrophoresis step. Essentially, this gives us the following requirements:

- $Sdna$  will be produced
- No “bad” oligo will be produced. A “bad” oligo  $X$  is one containing both  $Sdna_1$  and  $Sdna_N$  where  $|X| \simeq |Sdna|$ , but  $X \neq Sdna$ . Our procedure cannot distinguish bad oligos from  $Sdna$ , so it might produce an incorrect output.

In order to eliminate the possibility of producing bad oligos, our algorithm forbids all *erroneous bridgings*. An erroneous bridging occurs when some oligo bridges two other oligos that are not consecutive in the target strand,  $Sdna$ . This can occur when some linker or substrand bridges any of the following:

- $Ldna_i$  and  $Ldna_j$  for any  $i, j$
- $Ldna_i$  and  $Sdna_j$  for any  $i, j$
- $Sdna_i$  and  $Ldna_j$  for any  $i, j$
- $Sdna_i$  and  $Sdna_j$  for  $j \neq i + 1$  (if  $j = i + 1$ , then  $Sdna_i$  and  $Sdna_j$  are consecutive in  $Sdna$ , and so binding them is desirable)

By forbidding erroneous bridgings, we ensure that the only strand containing  $Sdna_1$  and  $Sdna_N$  will be  $Sdna$  itself. In this case there cannot be any bad oligos. (See figure 2 for an example of an erroneous bridging.)

**Fig. 2 goes here** (Tables & figures are at end of document, on separate pages.)

## 4 Construction Algorithm

Having stated the approach and the requirements on the algorithm, we can now describe the algorithm itself. We do this in two stages, beginning with a simplified discussion of the expected case, followed by a discussion of exceptional cases.

## 4.1 Overview of Algorithm

The algorithm takes the input parameters described in section 3.2 and executes as follows:

1. Chop the target strand,  $Sdna$ , into a sequence of consecutive substrands, all of length  $maxlen$ .
2. Create linkers for each pair of consecutive substrands. Each linker is of length  $maxlen$ , and is centered on the point where its substrands meet. That is, the linker between  $Sdna_i$  and  $Sdna_{i+1}$  is formed from the Watson-Crick complement of the last  $maxlen/2$  nucleotides of  $Sdna_i$  and the first  $maxlen/2$  nucleotides of  $Sdna_{i+1}$ .
3. Check to see whether the proposed substrands and linkers build  $Sdna$  without erroneous bridgings. We explain how this check procedure works in the subsection 4.2. If the proposed oligos will work, then the algorithm terminates with success.
4. If there are erroneous bridgings, then we apply the following strategy:
  - (a) Use *recursive synthesis*. Once we know which oligos (linkers or substrands) are involved in the various erroneous bridgings, we coalesce several consecutive substrands into “blocks.” (Formally we partition the original set of substrands making up  $Sdna$  into a set of sets of substrands  $B$  such that each set in  $B$  consists of consecutive substrands.) Each block is built in isolation and in a separate application of our process. The partitioning is done such that, within each block, there are no erroneous bridgings. The algorithm for designing the partition is given in subsection 4.3. Once the blocks are constructed, we continue the algorithm from step 2 where the constructed blocks are viewed as substrands.

Two heuristics can be applied prior to recursive synthesis.

- (a) Shift the center of the linker. For example, if a linker extends 10 bases into each of its substrands, then create a new linker that extends 5 bases into one and 15 bases into the other. This new linker will often escape the erroneous bridgings that caused problems for the initial linker.
- (b) Move the cut points between substrands, e.g. take the last two bases at the end of  $Sdna_i$ , and move them to the beginning of  $Sdna_{i+1}$ .

## 4.2 Algorithm to Test for Erroneous Bridgings

For the purpose of describing this algorithm, we will call both linkers and strands by the single name oligo. Recall that we must avoid erroneous bridgings, which occur whenever an oligo bridges any two oligos other than  $Sdna_i$  and  $Sdna_{i+1}$  for some  $i$ .

In this test, we extend the notion of erroneous bridgings to those that waste DNA. For example, a substrand may bind to a single substrand, without bridging two oligos. In this case, these oligos will not contribute to building  $Sdna$ . This case is rare enough that it has no effect on our expected time, though it is the motivation for the “masking” that we do in the recursive synthesis procedure described below.

The naive algorithm to test for erroneous bridgings is to look at every possible erroneous bridging involving oligos  $A, B, C$ , to see if  $A^r$  bridges  $B + C$ . If it does, then we have found an erroneous bridging. In the worst case there are  $O(N^3)$  tests where  $N$  is the number of substrands (and  $N - 1$  is the number of linkers). Each test may require  $O(maxlen^2)$  time since erroneous bridgings need not be centered on the bridged oligos, so Ukkonen-style techniques [Ukkonen, 1985] cannot be used and the full  $(maxlen \times maxlen)$  dynamic programming matrix must be constructed. Since  $N = |Sdna|/maxlen$ , the total time is  $O((|Sdna|^3)/maxlen)$ . Fortunately, we can speed this up in the expected case.

First, we describe an optimization to avoid testing all triplets. Consider the question: for a given oligo  $A$ , which pairs of oligos  $B, C$  have the property that  $A^r$  will bridge  $B + C$ ? Whenever such a bridging occurs, either the left half of  $A^r$  appears (within *allowedif* edit operations) in  $B$  or the right half of  $A^r$  appears (within *allowedif* edit operations) in  $C$ . If neither of these conditions is true, then  $A^r$  cannot possibly bridge  $B + C$ .

Our algorithm exploits this observation:

```

Let AllOligos be the set of all oligos.
For all  $A \in \textit{AllOligos}$ ,
  Let LeftMatch be the set of oligos  $B$  for which lefthalf( $A^r$ ) appears in  $B$  with up to allowedif edit operations.
  Similarly, RightMatch is the set of oligos  $C$  for which riighthalf( $A^r$ ) appears in  $C$ .
  Test  $A^r$  against the cross products LeftMatch  $\times$  RightMatch and LeftMatch  $\times$  (AllOligos - RightMatch) and (AllOligos - LeftMatch)  $\times$  RightMatch.
  If  $A^r$  matches any of the pairs in these cross products, then
    Oligos  $A, B, C$  form an erroneous bridging.
  End If
End For

```

We expect *LeftMatch* and *RightMatch* to be very small relative to  $N$ , and the calculations of *LeftMatch* and *RightMatch* each check  $O(N)$  oligos. So, in the expected case this optimization will reduce the overall number of searches to  $O(N^2)$ .

This algorithm still requires an  $O(maxlen^2)$  time dynamic programming test to test each candidate oligo (for *LeftMatch* or *RightMatch*), with up to *allowedif* edit operations in the worst case. It also requires an  $O(maxlen^2)$  test to determine if pairs of oligos are bridged by  $A^r$  when *LeftMatch* or *RightMatch* are not empty.

We can optimize these tests in the expected case, by noting that the test will typically be negative when the oligos are sufficiently long and *allowedif* is sufficiently small. We therefore use a technique that reduces the comparison to an expected time  $O(n)$ .

We wish to see if  $X$  (a putative bridge) is a substring of  $Y$  (the concatenation of a pair of oligos), within up to *allowedif* edit operations, for some  $X, Y$ . Suppose, for example, that *allowedif* = 1. Then, we split  $X$  into halves,  $X = X_1 + X_2$ . Suppose further that neither  $X_1$  nor  $X_2$  is an *exact* substring of  $Y$ . Then each of  $X_1, X_2$  must differ by at least 1 edit operation. Therefore  $X = X_1 + X_2$  differs by at least two edit operations, so  $X$  cannot be a substring of  $Y$  within an edit distance of 1.

More generally, we split  $X$  into *allowedif* + 1 pieces, and do an exact substring search on each of them. Exact substring search can be done in  $O(maxlen)$  time (e.g. using the Knuth-Morris-Pratt or Boyer-

Moore algorithms [Cormen *et al.*, 1990]). Therefore, if we can eliminate tests in mismatching strings often enough, we achieve an expected  $O(maxlen)$  time performance.

We note that in some cases, one of the  $allowedif + 1$  pieces will match by chance, causing us to do the expensive  $O(maxlen^2)$  edit distance test. Before doing the expensive test, we may wish to split  $X$  into  $allowedif + E$  pieces, for some  $E \geq 1$ . In this case, we need to do the expensive test only if  $E$  of the pieces match exactly.

Following these two optimizations, the expected running time of the overall algorithm can be computed as follows. Let  $|Sdna|$  be the number of bases in the target sequence. Then the expected running time is  $O((|Sdna|/maxlen)^2 \cdot maxlen) = O((|Sdna|^2)/maxlen)$ , as stated in the abstract.

### 4.3 Details of the Recursive Synthesis Strategy

The recursive synthesis strategy applies whenever there exists at least one erroneous bridging even after applying heuristic techniques (e.g. lengthening linkers or moving them) for resolving this problem. The recursive synthesis strategy builds large substrands in isolation and then uses these to construct still larger substrands in isolation in an iterative fashion until there is one large substrand,  $Sdna$ .

#### 4.3.1 Use of Recursive Synthesis Strategy

Recall that an erroneous bridging involves three oligos,  $A, B$  and  $C$ , where  $A^r$  bridges  $B + C$ , and  $B, C$  are not the consecutive substrands that  $A$  should bridge. At the point at which we decide to use a recursive synthesis, we will have a set of erroneous bridgings that our algorithm has detected.

Recursive synthesis partitions  $Sdna$  into sets of substrands,  $BLKdna_{1..b}$  called “blocks”, such that  $BLKdna_1 + BLKdna_2 + \dots + BLKdna_b = Sdna$  and each  $BLKdna_i$  will contain one or more consecutive substrands from the  $Sdna_i$ 's. Recursive synthesis synthesizes each  $BLKdna_i$  in isolation using the basic algorithm.

Within each  $BLKdna_i$ , no set of three oligos form an erroneous bridging. We achieve this condition as follows. Initially there are no blocks.

```

Set  $b := 1$  and form an empty block  $BLKdna_1$ .
Sort oligos from  $Sdna_1, Ldna_1$  to  $Sdna_N, Ldna_N$ 
For every oligo,  $o$ , in sorted order
  Put  $o$  into  $BLKdna_b$ 
  If this causes an erroneous bridging, then
    Remove  $o$  from  $BLKdna_b$ 
     $b := b + 1$ 
    Form an empty block  $BLKdna_b$ 
    Put  $o$  into  $BLKdna_b$ 
  End If
End For

```

*(The construction guarantees that no block has erroneous bridgings within it. Therefore, each one can be constructed in isolation.)*

If the resulting number of blocks  $b$  is less than the number of oligos  $N$ , then  
 Consider the blocks to be substrands

Create linkers and test for erroneous bridgings using the algorithm of section 4, with the  $BLKdna_i$ 's as the substrands. (*Note: it will sometimes be desirable to "mask" some  $BLKdna_i$ , by binding it with its Watson-Crick complement,  $\overline{BLKdna_i}$ . This prevents a linker or another strand from binding to the block, consuming DNA without contributing to synthesizing the target strand.*)

If there are no erroneous bridgings, then

Declare success

Else

*(We were able to coalesce substrands into blocks. But, we cannot link these blocks. So, we need to do another level of a recursive synthesis)*

Reinvoke this algorithm, with  $BLKdna_{1..b}$  as input.

End If

Else

*(We were not able to make bigger strands than the substrands we already have, so the recursive synthesis did not work.)*

Apply the prosthetic bridging method (see "Prosthetic Bridging" section)

End If

Our test simulations suggest that the recursive synthesis strategy is effective in nearly all tests with human DNA sequences, when it is necessary. However, there are sequences for which recursive synthesis is insufficient. For example, if the *allowedif* parameter is 2, the following strand is impossible to build with the recursive synthesis technique:

$Sdna = AAAAA...AACAAAA...AAGAAAA...$

The problem is that any oligo designed to bind to the sequence of A's with the C, will be only 1 mismatch from the sequence of A's with the G in the middle. The prosthetic bridging method resolves such problems.

## 5 Prosthetic Bridging

The prosthetic bridging process can take any two strands and bridge them. This can be applied recursively to build a full target string. This process requires more laboratory time, and therefore we use it only as a last resort.

### 5.1 The Prosthetic Bridging Process

Given a target  $Sdna_{target}$ , the prosthetic bridging process works as follows (see fig. 3, steps 1-5):

**Fig. 3 goes here** (Tables & figures are at end of document, on separate pages.)

1. Let  $Sdna_1$  and  $Sdna_2$  be such that  $Sdna_{target} = Sdna_1 + Sdna_2$  and  $Sdna_1$  and  $Sdna_2$  are of equal length or differ by one. Let  $Pdna$  be a "prosthetic" sequence (whose construction is explained further below). Let  $SPdna_1 = Pdna + Sdna_1$ ,  $SPdna_2 = Sdna_2 + Pdna$ .  $Pdna$  will be used to restrict the number of erroneous bridgings that can occur. Let  $Ldna$  be a linker that bridges  $Sdna_1 + Sdna_2$ . Synthesize  $Ldna$ ,  $SPdna_1$ , and  $SPdna_2$  using either standard techniques or a recursive synthesis. (If this is impossible directly, use prosthetic bridging to construct those elements.)

2. Combine the oligos such that they can hybridize. Apply ligase. This will form the sequence  $SPdna = SPdna_1 + SPdna_2 = Pdna + Sdna_1 + Sdna_2 + Pdna$ .
3. Isolate  $SPdna$  using electrophoresis.
4. Synthesize strands complementary to  $Pdna$ , and insert them.
5. Apply an exonuclease that will dissolve the now double-stranded  $Pdna$  sequences, leaving  $Sdna_1 + Sdna_2 = Sdna_{target}$ .

One such exonuclease is *Bal 31* [Ausubel *et al.*, 1999]. *Bal 31* degrades double-stranded DNA from both the 5' and 3' ends. However, it also has other behaviors that may be difficult to control.

An alternative is separately to use a 5' to 3' exonuclease such as T7 gene 6 exonuclease [Kerr and Sadowski, 1972] and a 3' to 5' exonuclease such as *exo III* [Rogers and Weiss, 1980]. In this case, we would insert  $\overline{Pdna}$  and then the first exonuclease. Then we would again insert  $\overline{Pdna}$  and then this time the second exonuclease. This will remove the  $Pdna$  sequences at both ends of  $SPdna$ .

We now isolate the target strand by an additional electrophoresis step. Note that we need this extra electrophoresis step, since the exonuclease may not have worked on all occurrences of  $Pdna$ . Also, note that  $Pdna$  will have to be large enough that  $Sdna_1 + Sdna_2$  can be distinguished from  $SPdna_1 + SPdna_2$  using electrophoresis.

## 5.2 Strategy for Constructing Prosthetics

This subsection will show how to construct the oligos,  $Pdna$  and  $Ldna$ , in order to eliminate all potential erroneous bridgings. First, we examine the types of erroneous bridgings, and determine the properties necessary to avoid them. Then we show how to ensure that strands complementary to  $Pdna$  will bind only to  $Pdna$ .

**Avoiding bad targets with  $Ldna$ .** First, we show how to filter out bridgings that would produce bad target oligos containing  $Ldna$ , i.e.  $Ldna + Ldna$ ,  $SPdna_1 + Ldna$ ,  $SPdna_2 + Ldna$ , etc. We do this by choosing the length of  $Ldna$  carefully. We can set  $|Ldna| = \frac{5}{8}|SPdna_1|$  (note  $|SPdna_1| \simeq |SPdna_2|$ ). Then  $|Ldna| + |SPdna_1| = \frac{9}{8}|SPdna|$ . Therefore, this length differs by 12.5% from the desired target,  $SPdna$ . This can easily be distinguished using electrophoresis. Also, there are no other strands that include  $Ldna$  that are any closer in length.

**Avoiding erroneous bridgings involving  $Ldna$ .** It remains to avoid erroneous bridgings that produce strands involving only  $SPdna_1$  and  $SPdna_2$ . There are 4 of these, of the length of  $SPdna$ : the desired target  $SPdna_1 + SPdna_2$ , and the bad targets,  $SPdna_1 + SPdna_1$ ,  $SPdna_2 + SPdna_2$  and  $SPdna_2 + SPdna_1$ .

Now, we show how to make sure that  $Ldna$  does not bridge any of these. To do this, we set  $|Ldna| \geq 2 \cdot |Pdna| + |Sdna_1|$ . By inspection, this will imply that whenever  $Ldna^r$  bridges one of the 3 bad targets,

$Ldna$  will have to overlap  $Pdna$  (e.g., see fig 3, items P1-P3). We will design  $Pdna$  such that  $Ldna$  cannot bind to it. Later, we will describe how to accomplish this.

**Avoiding all other erroneous bridgings.** We have now completely eliminated erroneous bridgings that involve  $Ldna$ , either as part of the target or as the bridger. So, the only remaining potential erroneous bridgings are where  $SPdna_1^r$  or  $SPdna_2^r$  bridge one of the 3 bad targets. For example, consider the case of  $SPdna_1^r$  bridging  $SPdna_1 + SPdna_1$ . Recall that  $SPdna_1 = Pdna + Sdna_1$ . Since we have no control over  $Sdna_1$  (it is given as input), it is difficult to ensure that it does not bridge. However, we have control over  $Pdna$ . We will guarantee that  $SPdna_1^r$  does not bridge a bad target, by ensuring that a part of it – namely  $Pdna^r$  – does not bind to the target.

In this case, we are considering the target  $SPdna_1 + SPdna_1 = Pdna + Sdna_1 + Pdna + Sdna_1$ . Suppose that  $Pdna^r$  binds to the substring  $Sdna_1 + Pdna$ . For this to be true, it must be that either the left half of  $Pdna^r$  binds to a substring of  $Sdna_1$ , or the right half of  $Pdna^r$  binds to a substring of  $Pdna$ , or both. If this were not the case, then some of  $Pdna^r$  is not accounted for.

Therefore, we can guarantee that this erroneous bridging does not occur by constructing  $Pdna$  such that the left half of  $Pdna^r$  does not bind to any substring of  $Sdna_1$  and also the right half of  $Pdna^r$  does not bind to any substring of  $Pdna$ . Expanding on this observation, for all non-consecutive bridgings involving  $SPdna_1$  and/or  $SPdna_2$ , we require that neither half of  $Pdna^r$  binds to any substring of  $Pdna$ ,  $Sdna_1$  or  $Sdna_2$ . We can achieve this goal by imposing two properties on  $Pdna$ .

**Required properties of  $Pdna$ .** The first property of  $Pdna$  is that it is composed exclusively of the bases A or C, instead of the full DNA alphabet. Because neither A nor C bind to either A or C, any substring of  $Pdna^r$  will mismatch any of substring of  $Pdna$ . (Note that the choice of {A,C} is only one alternative; {A,G}, {C,T} and {G,T} would work just as well.). The number of mismatches will be the number of bases in the substring. Therefore, if the length of half of  $Pdna$  is greater than the *allowedif* parameter, the halves will not bind.

The second property of  $Pdna$  is less trivial to achieve. It is that neither of its halves binds within *allowedif* edit operations to  $Sdna_1$ ,  $Sdna_2$  or  $Ldna$ . We will construct a half of  $Pdna$ ,  $HPdna$ , with this property, and set  $Pdna = HPdna + HPdna$ . The algorithm to do this is given in the following subsection. Note that we may need to make  $Pdna$  artificially longer (perhaps by repeating the sequence  $HPdna$ ), in order to make sure that  $SPdna$  can be distinguished from  $Sdna$  using electrophoresis.

Finally, we have to ensure that the strands complementary to  $Pdna$  (i.e.  $\overline{Pdna}^r$  in fig. 3, item 4) bind only to  $Pdna$ . We do this by dividing  $Pdna$  into quarters (going beyond the halves). One quarter,  $Q_{AC}$ , is composed solely of the bases A or C and is designed to ensure that  $Pdna$  does not bind to  $Ldna$ ,  $Sdna_1$  or  $Sdna_2$ . Another quarter,  $Q_{GT}$ , uses only G or T, and also does not bind to  $Ldna$ ,  $Sdna_1$  or  $Sdna_2$ . We set  $Pdna = Q_{AC} + \overline{Q_{GT}}^r + Q_{AC} + \overline{Q_{GT}}^r$ . Since A is complementary to T and C is complementary to G, this guarantees that  $Pdna$  consists only of A or C.  $Pdna$  also contains the unique string  $Q_{AC}$ , and therefore will not bind to  $Ldna$ ,  $Sdna_1$  or  $Sdna_2$ .  $\overline{Pdna}^r$  will contain  $Q_{GT}$ , so will not bind to these strings. This implies that  $\overline{Pdna}^r$  will not bind to anything other than the desired  $Pdna$ .

**Pdna could be smaller in practice.** Heuristics may enable us to use a smaller *Pdna*. First, we may actually be able to build *Pdna* using the full DNA alphabet (i.e. {A,C,G,T}), and find that in fact neither half of *Pdna* binds to a substring of *Pdna* (this is the requirement that made us conservatively use only {A,C}). The second observation is that we need not build  $Q_{AC}$  independently from  $Q_{GT}$ . For example, it may be that the first part of  $\overline{Q_{GT}}$  functions as a viable replacement to the end of  $Q_{AC}$ , allowing for a shorter  $Q_{AC} + \overline{Q_{GT}}$

**Summary.** We have stated the properties of *Ldna* and *Pdna* that ensure that our desired strand will be produced, and that electrophoresis will be able to isolate it (assuming that we can create  $Q_{AC}$  and  $Q_{GT}$ ).

### 5.3 Find Distant Problem

The previous section assumed that we could create strands over two letters of the DNA alphabet that would not bind to any substring of *Ldna*, *Sdna*<sub>1</sub> or *Sdna*<sub>2</sub>, within *allowedif* edit operations. We call this the *Find Distant* Problem. Formally, given a sequence  $S$ , a parameter *allowedif* and an alphabet  $A$ , the Find Distant problem is to find a string  $x$  that whose edit distance from any consecutive subsequence of  $S$  is at least *allowedif* + 1.

In this section, we present two algorithms to solve the Find Distant problem. Our implementation uses a heuristic algorithm, which works well empirically, but which does not guarantee a minimum length solution. We also discuss a brute-force algorithm which guarantees a minimum length result.

The *Find Different* problem is defined formally as follows. (Similar problems to this one, but with Hamming distance instead of edit distance, are discussed in [Amir *et al.*, 1997] and [Gasieniec *et al.*, 1997]).

**Instance:** A set of strings,  $S$  over the alphabet {A,C,G,T}, and an integer  $d$  ( $d$  is *allowedif*).

**Goal:** Find a string  $x$  over {A,C} (or {G,T}), such that there is no substring  $s$  of  $S \in \mathcal{S}$  where  $x$  binds to  $s$  within  $d$  edit operations and  $|x|$  is minimal. Note that we will set  $Q_{AC} = x$  or  $Q_{GT} = x$ .

In the following, we prove that  $|x|$  is logarithmic in  $\max(|S|)$ , for  $S \in \mathcal{S}$ . This shows that, for large  $|S|$ , our *Pdna* strand will be relatively small. Then we discuss the possibility that *maxlen* may need to be greater than desirable, when  $|S|$  is small. We conclude with a sketch of the two algorithms we use to solve this problem.

**Logarithmic Bound is Achievable.** We show that for any input  $\mathcal{S}$ , there must exist a solution  $x$  that is logarithmic in the lengths of the strings in  $\mathcal{S}$ . For simplicity, we concatenate all strings in  $\mathcal{S}$  into one large string,  $S$ . This does not make the problem any computationally easier, but does simplify the analysis.

We consider  $S$  of size  $|S|$ , and hypothesize that we can find a solution  $x$ , with  $|x| = k$ , for a given  $k$ . Now we consider the conditions under which this is true.

Let us define a function,  $Elim(k, d)$ , that gives the number of potential choices for  $x$  that could be eliminated by a well-chosen  $S$ , given  $k$  and  $d$ . (That is, any such  $x$  would be within edit distance  $d$  of some substring of  $S$ .) For a given  $k$ , there are  $2^k$  potential choices of  $x$ . So, if  $2^k > Elim(k, d)$ , then there is at least one valid  $x$  to choose.

Recall that  $S$  is over the alphabet  $\{A,C,G,T\}$ , while  $x$  is over  $\{A,C\}$ . Fortunately, given a string over  $\{A,C,G,T\}$ , we will never increase its edit distance to  $x$  by replacing any A and C with G or T. Therefore, we need only to consider  $S$  over  $\{G,T\}$ .

Next, we define  $ElimExact(k, d)$ , which is the number of eliminations within exactly  $d$  edit operations. So,  $Elim(k, d) = \sum_{i=0}^d ElimExact(k, i)$ . To estimate an upper bound for  $ElimExact$ , we note that each operation is either: insert a G, insert a T, delete anything or change (toggle between G/T). Also, each of the operations could operate at any of the  $k$  positions of the string. Finally, we note that there are less than  $|S|$  strings of size  $k$  in  $S$ . Thus,  $ElimExact(k, d) \leq (4k)^d |S|$ . So,  $Elim(k, d) \leq \sum_{i=0}^d (4k)^i |S|$ . Summing the power series,  $Elim(k, d) \leq \frac{4k^{d+1}-1}{4k-1} |S|$ .

Now, there exists some valid  $x$  if  $2^k \geq \frac{4k^{d+1}-1}{4k-1} |S|$ , that is if some potential choices of  $x$  cannot be eliminated with the given  $k$  and  $d$ . Taking the logarithm of both sides, we see that  $k$  is logarithmic in  $|S|$ , as desired.

**May require the substrands to be long.** We have shown that  $|Q_{AC}|$  and  $|Q_{GT}|$  are logarithmic with respect to the size of  $Sdna_1, Sdna_2$  and  $Ldna$ . However, when they are small, we may require a relatively high substrand length ( $maxlen$ ) to build  $Sdna$ .

We can estimate the required  $maxlen$  for a given  $k$ , using two inequalities. First, we note that  $|Pdna|=4k$ . We assume that electrophoresis is accurate to a factor of  $acc$  (e.g. 0.01 for 1%). The longest strand we will need to create is  $Ldna$ , and  $|Ldna| = (1 + 2 \cdot acc)(|Pdna| + |Sdna_1|)$ , in order for us to be able to distinguish bad targets involving  $Ldna$  from the desired intermediate target  $SPdna$ , using electrophoresis. Then we assume a parameter,  $buildup$ , which is the factor by which the oligos grow, with each step of recursive synthesis.

The first inequality is that  $|Ldna| \cdot buildup \leq |Sdna|$ . So our target is less than our longest oligo by a factor of  $buildup$ . The second inequality is  $|Ldna| \leq maxlen$ . Substituting the above equation for  $|Ldna|$ , we find that  $maxlen \geq \frac{8k(1+2 \cdot acc)}{2 - buildup \cdot (1+2 \cdot acc)}$ .

According to this formula, and our estimate of  $Elim$ , we would need  $maxlen$  of at least 288, if  $allowedif = 3$ ,  $buildup = 1.1$  and  $acc = 0.01$ . 1% accuracy is aggressive, but realistic for small oligos. For larger oligos, the logarithmic bound ensures that the  $maxlen$  parameter will become less restrictive.

In our experimental simulations with random sequences, we have found that the actual values of  $maxlen$  are much lower. For example, we show below that the primary algorithm for the find distant problem typically produces  $k \leq 10$  for  $|S| = 1000$ . This would lead to  $maxlen \leq 92$  for  $acc = 0.01$  and  $buildup = 1.1$ . However, our (pessimistic) estimate puts  $k \simeq 23$ , which would lead to  $maxlen = 214$ .

We note that our estimation of the  $Elim$  function may be very pessimistic, especially since it overlooks the fact that the same string may be “eliminated” many times. Moreover, the sequences that require the prosthetic bridging method will typically be highly repetitive, so most “eliminated” strings will indeed be duplicates.

If our  $Elim$  function is not greatly pessimistic, then this may represent a problem, since  $maxlen > 90$  is difficult to achieve in practice using standard DNA synthesis techniques. Once  $maxlen \simeq 650$ , however, we can build large  $Sdna$ , with  $buildup = 1.5$ .

**Primary Algorithm for the Find Distant Problem.** Our primary algorithm is a simple greedy one, which consists of the following steps:

```

Let  $x$  be the tentative answer.
 $x := \epsilon$ , the empty string.
Create  $IsProblematic_{1..|S|}$ , an array of boolean flags. Set all entries to true. (If  $IsProblematic_i$  is false, then we say the substring of  $S$  beginning at position  $i$  is no longer problematic, i.e. its edit distance with the proposed string  $x$  is already more than  $d$ )
While there are problematic strings
  For all  $y \in \{A, C\}$ 
    Find the edit distances of  $x + y$  and all still-problematic substrings in  $S$ .
    Select the value of  $y$  (either A or C) that yields the greatest average edit distance calculated over all still-problematic substrings. Set  $x := x + y$ .
  For all  $0 \leq i \leq |S|$ 
    If the substring starting at  $i$  is no longer problematic, then
      Set  $IsProblematic_i$  to false.
    End If
  End For
End While

```

In our experimental simulations, this algorithm produces strings that are logarithmic in the length of the input with very small variance. The lengths are typically low:

with  $allowedif = 2$ ,  $|S| = 100$  leads to two letter difference strings of length between 6 and 7 (6.6 on the average)

$|S| = 1000$  leads to length 9-10 (9.1 on the average)

$|S| = 10000$  leads to length 11-12 (11.3 on the average)

$|S| = 100000$  leads to length 13-14 (13.7 on the average)

Empirically, then, the length of the different string is proportional to  $c \cdot \log_2 |S|$ , where  $c \leq 1$ . The results are even better when  $S$  has many repeats. However, we have not proven that it has a logarithmic bound in the worst case.

**Brute-Force Algorithm for the Find Distant Problem.** In the brute force algorithm, suppose we wish to find an answer of length  $k$ .

```

Let  $L$  be a an empty list, which will contain strings over  $\{A,C\}$ .
For all substrings  $s$  of  $S$  of lengths
   $k - allowedif, 1 + k - allowedif, \dots, k + allowedif$ ,
  Generate all substrings  $s'$  that are within  $d$  edit operations of  $s$ .
  Add  $s'$  to  $L$ .
End For
Sort  $L$  lexically.
For all elements of  $L$  (in sorted order),
  If there is a gap between the current and next elements (i.e. a valid string of length  $k$  lexically between them).
    The missing value is an acceptable answer,  $x$ .
  End If
End For
If there was no gap, then no solution exists for length  $L$ , so set  $L$  to  $L + 1$  and repeat.

```

This algorithm runs in  $O(|S| \cdot k^d)$  time and space. This is acceptable for us, since our values for  $d$  are typically low.

## 6 Simulation Results

We have implemented our construction algorithms in the K programming language (<http://www.kx.com/>), and have run test simulations. We have used two types of test data: randomly-generated DNA sequences and human DNA sequences.

The simulations on human DNA may be more significant than the tests on random DNA, because researchers are likely to want to synthesize data similar to human DNA. Our human DNA test sequences are various subsequences of an approximately 2.1 million base sequence of human DNA generated by the University of Washington Genome Center (<http://www.ncbi.nlm.nih.gov/genome/seq/ctg.cgi?CTG=Hs7.965>). All simulations used  $allowedif = 2$ .

Table 1 summarizes our results, showing how often the more time-consuming recursive synthesis and prosthetic bridging procedures are required in practice, assuming that strands of length  $maxlen$  have already been synthesized. Fig. 4 shows how this changes as a function of the size and type of DNA sequence.

<b>Table 1 goes here</b> (Tables & figures are at end of document, on separate pages.)
--

<b>Fig. 4 goes here</b> (Tables & figures are at end of document, on separate pages.)
---

The data shows that we encounter far fewer erroneous bridgings for randomly-generated data. This makes intuitive sense, since erroneous bridgings require duplicate sub-sequences and duplication is less common with random sequences.

Intuitively, recursion is more likely to be required for longer  $Sdna$ . Indeed, the average number of isolated blocks for  $Sdna$  is near 5 when  $|Sdna|$  is 50 kilobytes, so we are essentially building 10 Kb blocks, and then link them together to become 50 Kb blocks. The same holds true for the full 2.1 megabyte human DNA sequence and for multi-megabyte random sequences.

Based on these results, we consider the extent of recursion that would be required to build a 3 Gb sequence (i.e. the length of the human genome with all chromosomes end-to-end). We note that with about 833 substrands (50000/60), random sequences almost always require no recursion. With human sequences, however, having 167 substrands means that we require 1.7 recursive sub-steps on average (where 1 recursive step implies no sub-steps at all). So, on average, each recursive step will grow the largest target strand by a factor of  $167/1.7=97$ .

The number of parallel recursive steps required for synthesis is therefore  $\log_{97} \frac{3 \cdot 10^9}{maxlen}$ . For  $maxlen = 60$ , this is about 4. The total number of applications of our process would be  $\sum_{i=1}^4 \frac{3 \cdot 10^9}{maxlen \cdot 97^i} \simeq 5.05 \cdot 10^5$ . We note that our estimate may be pessimistic. For example, we were able to build the 2.1 Mb human DNA sequence with 1500 ( $\frac{2.1 \cdot 10^6}{1333}$ ) sub-steps, which is much more than 97.

These 500,000 applications of recursive synthesis will require a new microarray technology.

## 7 Conclusions and Future Work

This paper presents an algorithm that constructs a recipe for synthesizing arbitrary strands of DNA. The algorithm essentially constructs a candidate set of small oligos and then adjusts them until only the desired strand will be constructed. Our techniques for adjustment comprise, essentially, recursive synthesis and the prosthetic bridging methods. The algorithm also uses several promising heuristics for rapid checking of candidate solutions.

We divide our future work into laboratory, algorithmic, and engineering tasks.

1. The laboratory task is to realize the assumptions we have made: (i) avoidance of shearing and (ii) avoidance of self-folding. One idea is to build the target sequence using double-stranded DNA with sticky ends [Watson *et al.*, 1997].
2. The algorithmic task is to find an algorithm that is guaranteed to find a solution requiring no recursive synthesis if there is such a solution. This will not enable us to construct new sequences, since we can already use recursive synthesis and prosthetic bridging, but it will make the laboratory work less expensive.
3. The engineering task is to minimize the cost of synthesizing the initial set of oligos. For example, if researchers wish to synthesize a 1 Mb DNA sequence using our method, they must synthesize a total of 1 Mb (in separate oligos) using traditional synthesis methods. One approach to reduce this cost is to build duplicate sub-sequences only once.

Sophisticated ways of efficiently creating large sets of oligonucleotides also exist. For example, methods have been proposed to efficiently synthesize many oligos for the purposes of molecular computation, e.g. [Bach *et al.*, 1996]. This introduces a notion of cost to the output of our algorithm outputs. That is, certain legitimate sets of  $Sdna_i$  and  $Ldna_i$  may be cheaper to synthesize than others.

## 8 Acknowledgements

We would like to thank Steve Rozen for his many helpful comments. Michael Benedik and Mike MacDonell have also been very helpful. We would also like to thank Nat Goodman for underscoring the virtues of tests on human DNA.

## References

- [Adleman, 1994] Adleman, L.M. 1994. Molecular Computation of Solutions to Combinatorial Problems. *Science*, 266, 1021-1024.
- [Adleman, 1998] Adleman, L.M. 1998. Computing with DNA. *Scientific American*, 279, 54-61.

- [Agarwal *et al.*, 1970] Agarwal, K.L., Büchi, H., Caruthers, M.H., Gupta, N., Khorana, H.G., Kleppe, K., Kumar, A., Ohtsuka, E., Rajbhandary, U.L., van de Sande, J.H., Sgaramella, V., Weber, H., Yamada, T. 1970. Total Synthesis of the Gene for an Alanine Transfer Ribonucleic Acid from Yeast. *Nature*, 227, 27-34.
- [Amir *et al.*, 1997] Amir, A., Apostolico, A. and Lewenstein, M. 1997. Inverse Pattern Matching. *Journal of Algorithms*, 24, 325-339.
- [Ausubel *et al.*, 1999] Ausubel, F.M., Brent, R., Kingston, R.E., Moore, D.D., Seidman, J.G., Smith, J.A., Struhl, K., Albright, L.M., Coen, D.M. and Varki, A., eds. 1999. *Current Protocols in Molecular Biology*. John Wiley and Sons, USA.
- [Bach *et al.*, 1996] Bach, E., Condon, A., Glaser, E. and Tanguay, C. 1996. DNA Models and Algorithms for NP-complete Problems, 290-299. In IEEE, *1996 IEEE Conference on Computational Complexity*. IEEE Computer Society Press, Los Alamitos, USA.
- [Cormen *et al.*, 1990] Cormen, Thomas H., Leiserson, Charles E. and Rivest, Ronald L. 1990. *Introduction to Algorithms*. MIT Press, New York, USA.
- [Frutos *et al.*, 1997] Frutos, A.G., Liu, Q., Thiel, A.J., Sanner, A.M.W., Condon, A.E., Smith, L.M. and Corn, R.M. 1997. Demonstration of a word design strategy for DNA computing on surfaces. *Nucleic Acids Res.*, 25, 4748-4757.
- [Gait, 1984] Gait, M.J., ed. 1984. *Oligonucleotide Synthesis: A practical approach*. IRL Press, Washington, D.C., USA.
- [Gasieniec *et al.*, 1997] Gasieniec, L., Indyk, P. and Krysta, P. 1997. External Inverse Pattern Matching, 90-101. In Apostolico, A. and Hein, J., eds., *Combinatorial Pattern Matching, 8th Annual Symposium, CPM 97, Aarhus, Denmark, June 30 - July 2, 1997, Proceedings*. Springer-Verlag, New York, USA.
- [Hershlag, 1991] Hershlag, D. 1991. Implications of ribozyme kinetics for targeting the cleavage of specific RNA molecules. *Proc. Natl. Acad. Sci. USA*, 88, 6921-6925.
- [Kerr and Sadowski, 1972] Kerr, C., and Sadowski, P.D. 1972. Gene 6 exonuclease of bacteriophage T7 I. Purification and properties of the enzyme. *J. Biol. Chem.*, 247, 305-310.
- [Paun *et al.*, 1998] Paun, G., Rozenberg, G. and Salomaa, A. 1998. *DNA Computing*. Springer-Verlag, New York, USA.
- [Rogers and Weiss, 1980] Rogers, S.G. and Weiss, B. 1980. Exonuclease III of *Escherchia coli* K-12, an AP exonuclease. *Meth. Enzymol.*, 65, 201-211.
- [Stemmer, 1995] Stemmer, W.P. 1995. Single-step assembly of a gene and entire plasmid from large numbers of oligodeoxyribonucleotides. *Gene*, 164, 49-54.

- [Ukkonen, 1985] Ukkonen, E. 1985. Finding approximate patterns in strings. *Journal of Algorithms*, 6, 132-137.
- [Watson *et al.*, 1997] Watson, J.D., Gilman, M., Witkowski, J. and Zoller, M. 1997. *Recombinant DNA*. W.H. Freeman and Company, New York, USA.

<b>Sequence source</b>	<b>Sequence length</b>	<b>maxlen</b>	<b># of sequences tested</b>	<b>% requiring recursive synthesis</b>	<b>% requiring prosthetic build</b>
Random	10,000	60	20	0%	0%
Human	10,000	60	20	30%	0%
Random	50,000	60	20	0%	0%
Human	50,000	60	20	85%	10%
Human	2,100,000	10,000	1	0%	0%
Human	2,100,000	1,333	1	0%	0%
Random	10,000,000	50,000	1	0%	0%

Table 1. Recursion needed in simulated experiments. This table shows how often recursive and prosthetic applications are required for different types of DNA sequences, and with different lengths.

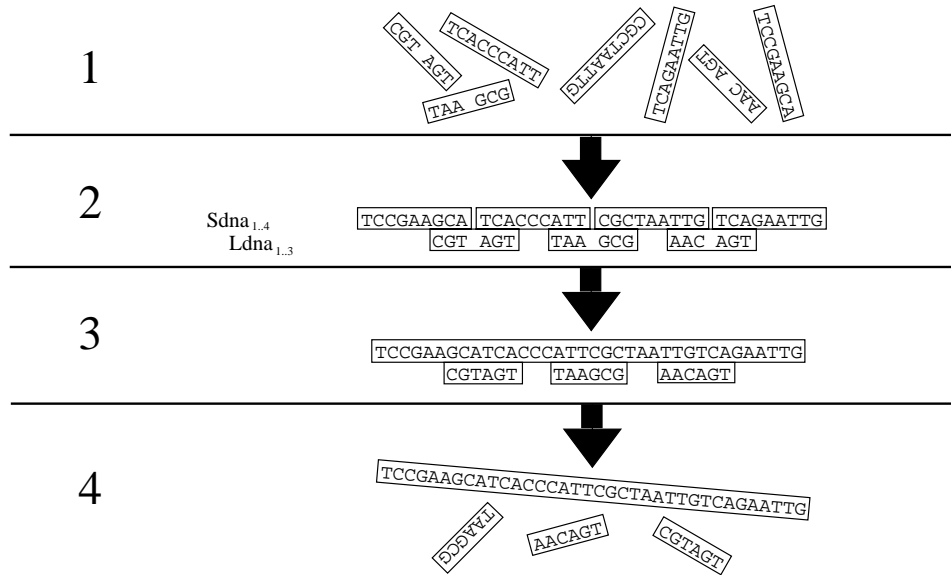


Fig. 1. Making the target strand, based on Adleman's method: (1) Oligos (substrands,  $Sdna_i$ , and linkers,  $Ldna_i$ ) are synthesized and combined in a test tube. (2) Linker  $Ldna_i$  hybridizes to the end of substrand  $Sdna_i$  and the beginning of  $Sdna_{i+1}$ , bringing the substrands together. In this case, we say that  $Ldna_i$  "bridges"  $Sdna_i$  and  $Sdna_{i+1}$ . (There are other, undesirable, ways for the strands to hybridize other than the one shown. Later, we will deal with this problem.). (3) Ligase is applied, joining the substrands together into one long strand - the target strand. (4) The DNA is denatured. Because the target strands are longer than the other strands, we can separate them using electrophoresis. Alternatively, we can use magnetic beads and complementarity.

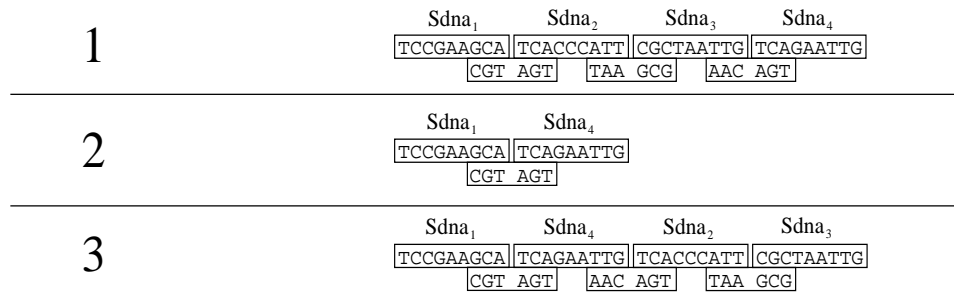


Fig 2. Erroneous bridgings. (1) Strands are supposed to bind to yield the target strand (as in fig. 1). The  $Sdna_i$  are labeled for convenience. (2) An “erroneous bridging”. This bridging is erroneous because  $Sdna_1$  and  $Sdna_4$  are linked, but are not supposed to be. (3) The result of an erroneous bridging. Once ligase is applied, this can result in a strand of the same size as the target strand, but with the wrong sequence.

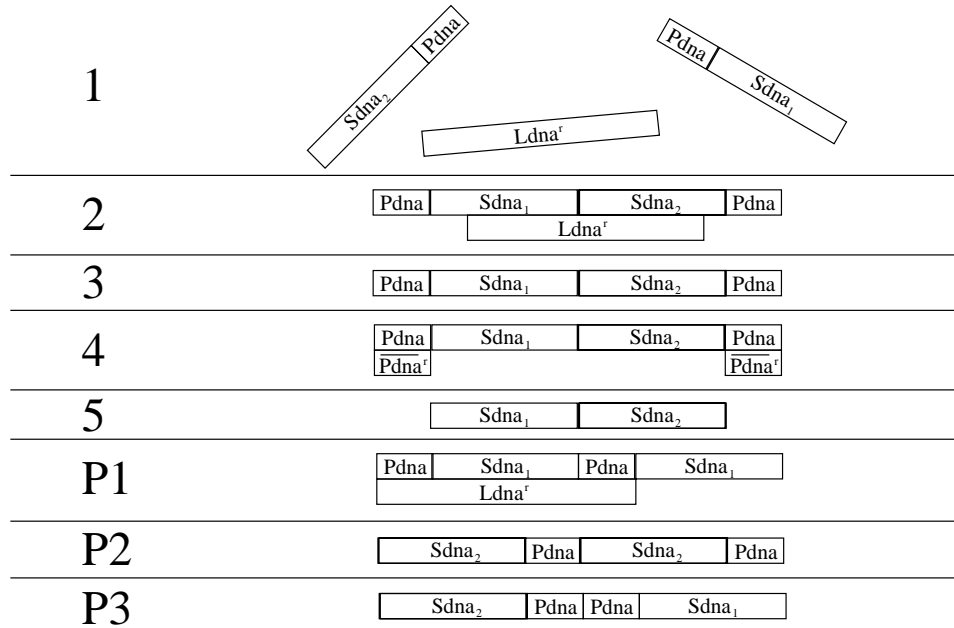


Fig 3. Prosthetic Bridging method diagrams. (1) We synthesize these oligos initially. (2) Oligos hybridize together, and  $SPdna_1$  and  $SPdna_2$  are ligased together. (3)  $SPdna = SPdna_1 + SPdna_2$  is isolated using electrophoresis. (4) We insert  $\overline{Pdna}$ , which binds to  $Pdna$ , creating a double-stranded region of DNA. (5) A double-strand specific exonuclease eliminates the double-stranded  $Pdna$  regions, leaving  $Sdna_{target}$ .

Potential problems: (P1)  $Ldna$  is long enough that an erroneous bridging will imply that it binds to  $Pdna$ . In this case, the bad target is  $SPdna_1 + SPdna_1$ . (P2) The bad target is  $SPdna_2 + SPdna_2$ . (P3) The bad target is  $SPdna_2 + SPdna_1$ .

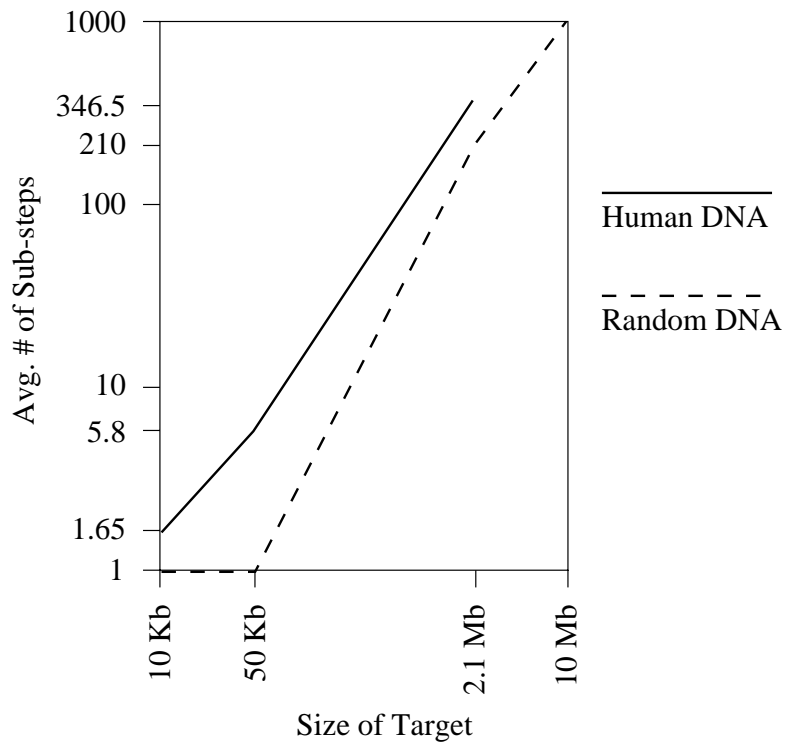


Fig. 4. Recursive synthesis effort. This graph shows the average number of separate blocks ( $BLKdna_{1..b}$ ) required for recursive synthesis for the different types and sizes of sequences. Random sequences require less recursion, since they have fewer (near) duplicate subsequences.