DNA Sequences Useful for Computation

Eric B. Baum
NEC Research Institute
4 Independence Way
Princeton, NJ 08540

Abstract

Recent proposals for DNA based computing [1], [2], [3] encode Boolean vector component values with sequences of DNA. It has previously been assumed that sufficient length random subsequences could be used to encode component values. However use of such subsequences will inadvertently result in long complementary subsequences. Complementary subsequences of sufficient length would stick to each other and cause mistakes or delays in computation. We suggest some constraints on DNA subsequences to be used in encodings, and describe maximal sets of subsequences satisfying these constraints.

A seminal paper of Adleman[1] recently sparked growing interest in the use of DNA and the methods of molecular biology to do computation. Adleman's approach required the encoding of computer science problems into DNA sequences, and relied heavily on "extraction" of sequences containing a particular subsequence by use of a complementary subsequence. Subsequently Lipton[2] proposed an approach for using DNA to solve Satisfiability and other problems in the computational class NP. Lipton proposed a particular encoding of Boolean vectors and relied on similar extraction operations using complementary subsequences. Baum proposed to use a similar encoding, and some variants, for Content Addressable memories[3]. Several other proposed computer algorithms rely on similar encodings and methods[4],[5].

For each of these proposals, a set of DNA subsequences must be chosen. Practical considerations will force these subsequences to satisfy certain requirements. Lipton and Adleman suggested using random subsequences. However, practical requirements impose constraints that can not be met by random sequences, and it is not *a priori* obvious that they can be satisfied at all. This paper describes such constraints and proposes sets of subsequences which satisfy them.

For concreteness we briefly review the encoding of Lipton[2]. One wishes to encode the Boolean vectors $\{0,1\}^n$, where n is about 60. For each $i = 1, ..., n$, one chooses two subsequences of DNA X_i and Y_i, corresponding respectively to a 0 or a 1 in the ith component. A vector in $\{0,1\}^n$ is then encoded by the concatenation of the appropriate subsequences, perhaps spaced by a fixed subsequence, or a subsequence corresponding to the number of the component[1]. This evidently requires

[1]Given a directed graph, Adleman[1] described a procedure for constructing an associated set of DNA molecules by mixing in subsequences corresponding to all nodes in his graph, and subsequences corresponding to all edges, which served to splint the node sequences together. Lipton[2] views the encoding of Boolean vectors as a version of Adleman's directed graph construction for a certain fixed graph. In this view, each spacer subsequence must be distinct, or Lipton's graph would collapse, with all those nodes of degree four assigned the identical sequence being identified. We remark that, if instead of mixing in all DNA molecules simultaneously, as Adleman did, those corresponding to the distinct nodes in Lipton's graph are poured in sequentially, so that molecules are formed coding first for 2 bit strings, then for

at least 120 suitable subsequences. In some of the algorithms proposed[4],[5], the initial vectors are extended by appending additional subsequences in a similar fashion (say corresponding to a tag that the vector encoded satisfies some Boolean circuit) so that the number of subsequences needed will grow substantially, and may reach tens of thousands or more. If enough suitable subsequences are not available, this will constrain algorithmic possibilities.

Let Z be a sequence of DNA. Then let $\bar{Z}$ denote the sequence of DNA which is Watson-Crick complementary to Z. The Watson Crick complement of the sequence Z is the sequence gotten by replacing A by T and C by G and vice versa, and then taking the sequence in the reverse order. For example if $Z = AGTCC$, then $\bar{Z} == GGACT$.

Now given an encoding as above, a key operation in many of the algorithms proposed is an "extract". Here one produces a subsequence $\bar{X}_i$ or $\bar{Y}_i$, and affixes it to a magnetic bead. Placing these magnetic beads into one's test-tube computer, the subsequences introduced bind[2] to any molecules already present containing the complementary subsequence. These bound molecules can now be extracted magnetically. This process allows one to search the testtube for vectors having particular component values.

Now we must impose some constraints on our molecules to avoid two difficulties. First, when we come to extract X_i, say, we want $\bar{X}_i$ to stick only to molecules containing X_i. If there is a sufficiently long subsequence in common to X_i and some molecule not containing X_i, we may have erroneous extracts. To avoid this we will require there be no long subsequence common to X_i and either any one of the other X_i or Y_i, or to the overlap region between some X_i or Y_i and whatever sequence may follow it on a molecule. Second, we don't want our molecule to stick to itself, or one of the other molecules in our soup. If this happened, it would gum up our computer and slow it down or make it unreliable. Thus we will require that there is no long subsequence such that both it and its complement may appear.

We will assume in this paper that a subsequence of length k is long enough to present a possibility of sticking, and seek not to have accidental collisions as long as k. It is not clear how large a k may be permissable in practice. It is possible that conditions may enforce k as low as 5, and seems likely that our computer will work more accurately and more rapidly the lower a k we can work with. Achieving low k values is a difficult constraint. If suitable subsequences can not be found, the entire enterprise of DNA based computing along the lines discussed to date is potentially in jeopardy.

Previous authors (e.g. [1], [2]) had noted that by using sufficiently long random sequences of bases as component subsequences X_i, the base itself would not appear. However if complementary subsequences of a finite length $k+1$ may stick, it is no help that the whole sequence is not present.

Thus we are motivated to ask the following question.

Question 1: Find a spacer sequence S_0 and a set of sequences $\{Z_j\}$ as large as possible with the property that there is no subsequence x of length k, which (a) occurs in any two sequences SZ_jS or (b) x occurs in some $S_0Z_jS_0$ and $\bar{x}$ occurs in some $S_0Z_iS_0$. We will also require that the Z_j are themselves at least of length k, so that they can be extracted on.

As DNA computing evolves, related questions will be asked. For example, we might also ask for a set of spacer sequences, which we will not extract on, and a set of Z_j which we will. We hope

3 bit words, etc., then molecules can potentially be built up coding for all n bit strings but using a single, identical spacer.

[2]This is an idealization. In practice, of course, some molecules may bind incorrectly or fail to bind where they should.

analysis similar to that presented here can be applied to such generalizations.

We assert first that without loss of generality the sequences Z_j can all be taken to be of length k. If they are longer, this merely induces more constraints. Using longer sequences will thus require a smaller set, but may have practical advantages in the extract operation. We discuss this briefly at the end.

Next we observe that the set of sequences of length k are divided into equivalence classes by the operation Complement. There are 4^k strings of length k. If $k = 2m+1$ (for m an integer) then no sequence is equal to its complement, so there are $4^k/2$ equivalence classes each of size 2. If $k = 2m$, then there are 4^m sequences equal their complements (since each of the first m components can be chosen in any of 4 ways, thus determining a sequence if we demand it is equal its complement). So for even k we have $(4^k - 4^m)/2$ equivalence classes of size 2 and 4^m of size 1.

For future reference, we note that if we restrict to sequences neither beginning nor ending with A nor T, then if $k = 2m+1$ we find analagously that there are $4^{k-1}/2$ equivalence classes each of size 2, and if $k = 2m$ there are $2 \times 4^{m-1}$ classes of size 1 and $2 \times 4^{k-2} - 4^{m-1}$ classes of size 2.

Say we choose as spacer sequence k X's, for X some base. Then it is apparent that we can form a large set satisfying our constraints by choosing one element from each equivalence class except those which either begin or end with X or $\bar{X}$.

However we can build a larger set. For concreteness, let the spacer be k A's. So that the above rule would not have allowed sequences beginning or ending with T's or A's. Assume for the moment $k = 2m+1$ so there are no equivalence classes of size 1. For each string S of length $k-1$, not beginning nor ending with A or T, and chosen such that $S \neq \bar{S}$, consider its complement $\bar{S}$. We may choose ST and TS, provided we do not choose $\bar{S}T$ nor $T\bar{S}$. Having accumulated each pair ST and TS we must then strike out of consideration, from every equivalence class in the set of length k strings, any that either begin with or end with $\bar{S}$. What we hope is that this will not remove from consideration *both* elements of any equivalence classes. When does this happen? For both elements of an equivalence class to be removed, we must remove both a string SX and the string $\bar{X}\bar{S}$. This would happen when we have added two strings S_1T and TS_2 such that $\overline{\bar{S}_1X} \equiv \bar{X}S_1 = Y\bar{S}_2$. But this implies $\bar{S}_1 = S_2$, which we have forbidden.

Note that, for those $k-1$ strings which are self complementary, we can not both add ST, and TS, and any representative of the class XS, for $X = \{C, G\}$. So we exclude ST and TS of this form.

Thus for each size 2 equivalence class of $k-1$ length strings not beginning nor ending with A nor T, we may choose a representative S, and add both TS and ST. We additionally may add a representative from every length k class which does not end or begin with A or T, selecting a member still allowed after the necessary cross offs. Following the count of number of equivalence classes from above, this simple algorithm allows us to select a set of size $2 \times (2 \times 4^{k-3} - 4^{m-1}) + 2 \times 4^{k-2} = 3 \times 4^{k-2} - 2 \times 4^{m-1}$.

Can we build an even larger set? If you add ST, then you can't add $\bar{S}A$, nor $A\bar{S}$. If you add SA, then you would not be able to add XS, and you would also be unable to add $\bar{S}\bar{X}$. Likewise AS is ruled out. So there is no possible gain in terms of adding strings beginning or ending with A.

However it is possible to create a larger set, as described below, by using length $k-1$ sequences which are permitted to end in A and then appending a T.

We denote by R_i^j a string of length i whose first element is neither A nor T. The superscript here serves solely to denote different such strings and will be omitted where no confusion is possible. Let

Y^j denote a single base either C or G. Again the superscript is only to distinguish different elements. So Y is simply an alternative name for R_1.

If $S = R_{k-2}A$, then $\bar{S}$ begins with T. We already know that if we add ST, we can not add $\bar{S}T$. So we can either choose to consider adding strings ending in AT, or ending and beginning in T, but not both. We consider the set of strings of form $R_{k-2}AT$. First note that we can safely add all strings of form $R_{k-3}AAT$ or $R_{k-3}TAT$, precisely because[3] we have excluded strings which both end and begin with T.

Next we seek to add strings of form $R_{k-3}YAT$, for $Y \in \{C, G\}$. It will be clearer if we denote these Q^iAT, where the i indexes different strings, the Q are of length $k-2$, and they neither begin nor end with A nor T. When we add a string of form QAT, then we can not add a string of form $T\bar{Q}X$. So we will seek instead to include $T\bar{X}Q$. This will be ok unless

$$\bar{X}Q^i = \bar{Q}^jY \tag{1}$$

for any Y, where we have included both Q^i and Q^j. To insure this does not happen, each time we add a string Q^iAT to our set, we must cross off conflicting $Q's$. That is, each time we add a Q, we must construct two others by $CQ = \bar{Q}'Y$, and $GQ = \bar{Q}''Y$ and cross Q' and Q'' off from consideration. However, each time we add $Q = R_{k-3}C$, we may add also $R_{k-3}G$, with no further cross offs. So for each two Q's we add, we must cross off two. Now note, equation 1 will never cross off any potential Q which has an A or a T in its second to last position. That is: because Q^i in equation 1 never begins with A nor T, by definition, any Q^j satisfying this equation on the right hand side will not have an A nor a T in its second to last position.

In summary, this procedure gets the following strings. All strings of form $R_{k-3}AAT$ and $R_{k-3}TAT$. All strings of form $R_{k-4}ACAT$,$R_{k-4}TCAT$,$R_{k-4}AGAT$,$R_{k-4}TGAT$. Half of the strings of form $R_{k-4}CCAT$,$R_{k-4}GCAT$, $R_{k-4}CGAT$, $R_{k-4}GGAT$. Half of the strings of form ST for $S \neq \bar{S}$ and $S = R_{k-2}C$ or $R_{k-2}G$, and for each of these the corresponding strings TS. A representative of each class neither ending nor beginning with A nor T. Assuming $k = 2m+1 \geq 5$, this works out to be $3 \times 4^{k-2} + 5 \times 4^{k-4} - 2 \times 4^{m-1}$.

If $k = 2m$, virtually identical analysis yields a set consisting of:(1)all strings of form $R_{k-3}AAT$ and $R_{k-3}TAT$, (2)all strings of form $R_{k-4}ACAT$,$R_{k-4}TCAT$,$R_{k-4}AGAT$,$R_{k-4}TGAT$; (3) half of the strings of form $R_{k-4}CCAT$,$R_{k-4}GCAT$, $R_{k-4}CGAT$, $R_{k-4}GGAT$; (4) half of the strings of form $R_{k-2}CT$ and $R_{k-2}GT$ and for each the corresponding string of form $TR_{k-2}C$ and $TR_{k-2}G$, and a representative of each length k class of size 2 neither beginning nor ending with A nor T. This works out to be also $3 \times 4^{k-2} + 5 \times 4^{k-4} - 2 \times 4^{m-1}$.

Is there any possibility of a still larger set? We have representatives of each size 2 class not beginning nor ending with A nor T. As discussed, considering adding classes beginning or ending with A will exclude more additional strings than in this set. We have a representative of every possible class of form ST or TS where S neither begins nor ends with A nor T. As discussed, considering strings both beginning with and ending with T will do worse than the construction given. So the only additional freedom is to consider strings of form ST where S ends in T or A. The reader can convince himself that using strings ending in T will cost more than it gains. We have analyzed the set of strings ending in AT in order to get this maximal set. Thus we see that if any string ommitted here is included, at least an equal number of strings included here would have to be ommitted.

[3]That is, if we call $R_{k-3}AA$ string S, adding ST would conflict with $\bar{S}T$ but we are not adding this since it both begins and ends with T, so there is no conflict.

The reader should, with thought, be able to satisfy himself that using a different sequence as spacer will not expand the set. By using k A's, we have been able to rule out all translations into the spacer with only a requirement that neither the first nor last letter of any sequence be A. If we used a more complex spacer sequence, we acquire additional constraints. If we used a spacer sequence beginning with k A's, and ending with k copies of some other letter, we could generalize our construction, but would not achieve a larger one.

We have:

Theorem: The maximum size set of strings satisfying question 1, for $k = 2m+1$ or $k = 2m$, $k \geq 5$, is $3 \times 4^{k-2} + 5 \times 4^{k-4} - 2 \times 4^{m-1}$.

Note, one could imagine modifying question 1 by using different spacers S^i to come after different sequences. We do not believe a larger set could be produced in this way, but have not investigated the question in exhaustive detail.

The constraints are quite restrictive for small k. The number of sequences we can have in our set is given below for various values of k.

k= 4	5	6	7	8	9	
N(k)= 50	208	816	3232	13440	54144	

Clearly k must be at least five for any hope of interesting applications. k as low as five may however be consistent with many DNA computing applications, potentially speeding up and avoiding errors in retrieval. If it is practically possible to work with large k, say as large as 9 for example, we might consider sacrificing the number of distinct sequences achievable in order to work with longer sequences. If we work with length L sequences, for $L > k$, then in an extract step we would have a stronger fit between the retrieval sequence and its mate. The number of such sequences achievable is certainly no larger than $N(k)/(L-k)$. We have not yet investigated how large a set may actually be constructed.

In conclusion we have investigated how DNA sequences may be constructed having properties useful to prevent extraction errors, and to prevent our DNA computer from inadvertently becoming gummed up.

I thank Peter Kaplan and Dan Boneh for conversations.

References

[1] Adleman, L. M., "Molecular computation of solutions to combinatorial problems", Science Vol 266 pp 1021-1024 (1994).

[2] Lipton, R. J.,"Speeding up computations via molecular biology", Science Vol 268, pp542-544 (1995).

[3] Baum, E. B., "Building an associative memory vastly larger than the brain", Science Vol 268,pp583-585 (1995).

[4] Boneh, D., C. Dunworth, R. Lipton, J. Sgall, "On the computational power of DNA", preprint, available at http://www.CS.Princeton.EDU/ dabo/biocomp.html.

[5] Boneh, D., C. Dunworth, R. Lipton, "Breaking DES using a molecular computer" preprint, available at http://www.CS.Princeton.EDU/ dabo/biocomp.html.