Beginnings of Molecular Computing
&
Biological Mathematics

Christian Jacob

CPSC 601.73

23 January 2003

Adleman's Experiments

- Leonard Adleman was able to use encoded DNA to solve the Hamiltonian Path for a single-solution 7-node graph.
- The drawbacks to using DNA as a viable computational device mainly deal with the amount of time required to actually analyze and determine the solution from a test tube of DNA.
- For Adleman's experiment, oligonucleotides of length 20 are required to encode the vertices and edges of the graph.
- Due to the nature of DNA's 4-base language, this allowed for 4^{20} different combinations.
- Even longer oligonucleotides would be required for larger graphs.

Defining a Rule Set

- Given the nature of DNA, we can easily determine a set of rules to operate on DNA.
- Defining a Rule Set allows for "programming" the DNA, much like programming on a conventional computer.

- The rule set assumes the following:
  - DNA exists in a test tube.
  - DNA is in single stranded form.

Operation: Merge

- Merge merges two test tubes of DNA to form a single test tube.
- Given test tubes N_1 and N_2 we can merge the two to form a single test tube N such that N consists of all the elements in $N_1 \cup N_2$.
- Formal Definition:
  - $N = merge(N_1, N_2)$

Operation: Amplify

- Amplify takes a test tube of DNA and duplicates it.
- Given test tube N_1 we duplicate it to form test tube N, which is identical to N_1.
- Formal Definition:
  - $N = duplicate(N_1)$

Operation: Detect

- Detect looks at a test tube of DNA and returns *true* if it has at least a single strand of DNA in it, *false* otherwise.
- Given test tube N, *detect* returns *ture* if it contains at least a single strand of DNA, else return *false*.
- Formal Definition:
  - *detect*(N)

Operation: Separate & Extract

- Separate separates the contents of a test tube of DNA based on some subsequence of bases.

- Given a test tube N and a word *w* over the alphabet {A, C, G, T}, produce two tubes +(N, *w*) and –(N, *w*), where +(N, *w*) contains all strands in N that contain the word *w,* and –(N, *w*) contains all strands in N that do not contain the word *w*.

- Formal Definition:

  – N □ +(N, *w*)
  – N □ -(N, *w*)

Operation: Length-Separate

- Length-Separate takes a test tube and separates it based on the length of the sequences.
- Given a test tube N and an integer *n* we produce a test tube that contains all DNA strands with length less than or equal to *n*.
- Formal Definition:
  - $N \rightarrow (N, \rightarrow n)$

Operation: Position-Separate

- Position-Separate takes a test tube and separates the contents of a test tube of DNA based on some beginning or ending sequence.

- Given a test tube N_1 and a word *w*, produce the tube N consisting of all strands in N_1 that begin/end with the word *w*.

- Formal Definition:

  - $N \leftarrow B(N_1, w)$
  - $N \leftarrow E(N_1, w)$

A Simple DNA Computation Example

- From the given rules, we can now manipulate our strands of DNA to get a desired result.
- Here is an example DNA program that looks for DNA strands that contain the subsequence AG and the subsequence CT:

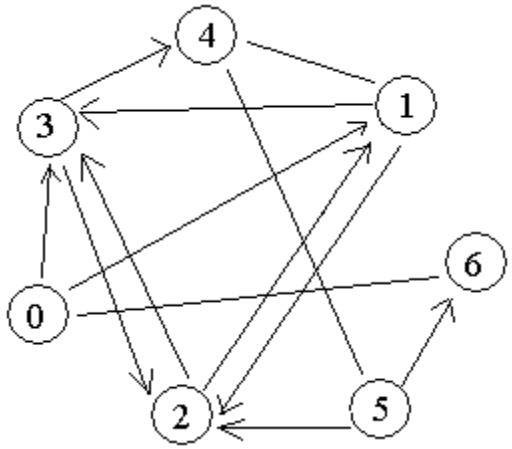
1. input(N)
2. N ← +(N, AG)
3. N ← +(N, CT)
4. *detect*(N)

An Explanation …

1. input(N)
   - ***Input a test tube N containing single stranded sequences of DNA***
2. N □ +(N, AG)
   - ***Extract all strands that contain the AG subsequence.***
3. N □ +(N, CT)
   - ***Extract all strands that contain the CT subsequence.***
   - ***Note that this is done to the test tube that has all AG subsequence strands extracted, so the final result is a test tube which contains all strands with both the subsequence AG and CT.***
4. *detect*(N)
   - ***Returns TRUE if the test tube has at least one strand of DNA in it, else returns FALSE.***

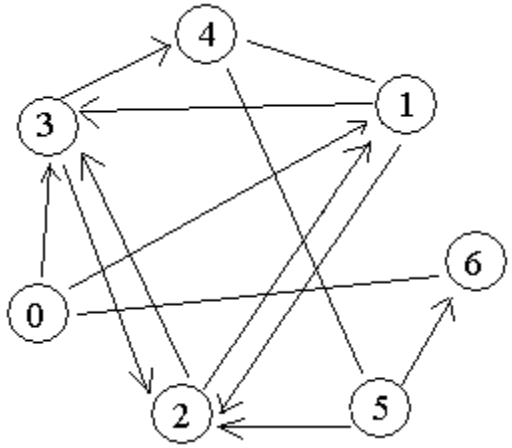
Back to Adelman's Experiment…

- Now that we have some simple rules at our disposal, we can easily create a simple program to solve the Hamiltonian Path problem for a simple 7-node graph as outlined by Adelman.



The Program: 7-Node Hamilton-Path

1. input(N)
2. N ← B(N, s_0)
3. N ← E(N, s_6)
4. N ← +(N, ← 140)
5. for i = 1 to 5 do begin N ← +(N, s_i) end
6. detect(N)



Explanation(I)

1. Input(N)
   - Input a test tube N that contains all of the valid vertices and edges encoded in the graph.

2. N ← B(N, s_0)
   - Separate all sequences that begin with the starting node.

3. N ← E(N, s_6)
   - Further separate all sequences that end with the ending node.

Explanation(II)

5. N □ (N, □ 140)
   - Further isolate all strands that have a length of 140 nucleotides or less (as there are 7 nodes and a 20 oligonucleotide encoding).

6. for i = 1 to 5 do begin N □ +(N, s_i) end
   - Now separate all sequences that have the required nodes, thus giving us our solution(s), if any.

7. detect(N)
   - See if we actually have a solution within our test tube.

Adding Memory – The Sticker Model

- In most computational models, we define a memory, which allows us to store information for quick retrieval.
- DNA can be encoded to serve as memory through the use of its complementarity properties.
- We can directly correlate DNA memory to conventional bit memory in computers through the use of the so called "Sticker Model."

The Sticker Model

- We can define a single strand of DNA as being a memory strand.
- This memory strand serves as the template from which we can encode bits into.
- We then use complementary stickers to attach to this template memory strand and encode our bits.

How It Works (I)

- Consider the following strand of DNA:

CCCC	GGGG	AAAA	TTTT

- This strand is divided into 4 distinct sub-strands.
- Each of these sub-strands has exactly one complementary sub-strand as follows:

GGGG	CCCC	TTTT	AAAA

How It Works (II)

- As a double Helix, the DNA forms the following complex:

CCCC	GGGG	AAAA	TTTT
GGGG	CCCC	TTTT	AAAA

- If we were to take each sub-strand as a bit position, we could then encode binary bits into our memory strand.

How it Works (III)

- Each time a sub-sequence sticker has attached to a sub-sequence on the memory template, we say that the bit slot is *on*.

- If there is no sub-sequence sticker attached to a sub-sequence on the memory template, then we say that the bit slot is *off*.

Some Memory Examples

- For example, if we wanted to encode the bit sequence 1001, we would have:

CCCC	GGGG	AAAA	TTTT
GGGG			AAAA

- This is a direct coding of 1001 into the memory template.

Disadvantages

- This is a rather good encoding, however, as we increase the size of our memory, we have to ensure that our sub-strands have distinct complements in order to be able to "set" and "clear" specific bits in our memory.

- We have to ensure that the bounds between sub-sequences are also distinct to prevent complementary stickers from annealing across borders.

- The biological implications of this are rather difficult, as annealing long strands of sub-sequences to a DNA template is very error-prone.

Advantages

- The clear advantage is that we have a distinct memory block that encodes bits.
- The differentiation between subsequences denoting individual bits allows a natural border between encoding sub-strands.
- Using one template strand as a memory block also allows us to use its complement as another memory block, thus effectively doubling our capacity to store information.

So now what?

- Now that we have a memory structure, we can being to migrate our rules to work on our memory strands.
- We can add new rules that allow us to program more into our system.

Operation: Separate

- Separate now deals with memory strands. It simply takes a test tube of DNA memory strands and separates it based on what is turned on or off.

- Given a test tube, N, and an integer *i,* we separate the tubes into +(N, *i*) which consists of all memory strands for which the *i-th* sub-strand is turned on (e.g. a sticker is attached to the *i-th* position on the memory strand).
  The –(N, *i*) tube contains all memory strands for which the *ith* sub-strand is turned off.

- Formal Definition:
  - N ⇒ +(N, *i*)
  - N ⇒ –(N, *i*)

Operation: Set

- Set simply sets a position on a memory position (i.e.. turns it *on* if it is *off*) on a strand of DNA.
- Given a test tube, N, and an integer i, where $1 \leq i \leq k$ (k is the length of the DNA memory strand), we set the *i-th* position to *on*.
- Formal Definition:
  - *set*(N, i)

Operation: Clear

- Clear simply clears a position on a memory position (i.e.. turns it *off* if it is *on*) on a strand of DNA.
- Given a test tube, N, and an integer *i,* where $1 \leq i \leq k$ (*k* is the length of the DNA memory strand), we clear the *i-th* position to *off*.
- Formal Definition:
  - *clear*(N, i)

Operation: Read

- Read simply reads a test tube, which has an isolated memory strand and determines what the encoding of that strand is.
- Read also reports when there is no memory strand in the test tube.
- Formal Definition:
  - *read*(N)

Defining a Library

- To effectively use the Sticker Model, we define a *library* for input purposes.
- The library consists of a set of strands of DNA.
- Each strand of DNA in this library is divided into two sections:
  - an initial data input section, and
  - a storage/output section.

Library Setup

- The formal notation for a library is as follows:
  - (k, l) *library* (where k and l are integers, $l \leq k$)
- k refers to the size of the memory strand.
- l refers to the length of the positions allowed for input data.
- The initial setup of the memory strand is such that the first l positions are set with input data, and the last $k - l$ positions are clear.

A simple Example

- Consider the following encoding for a library:

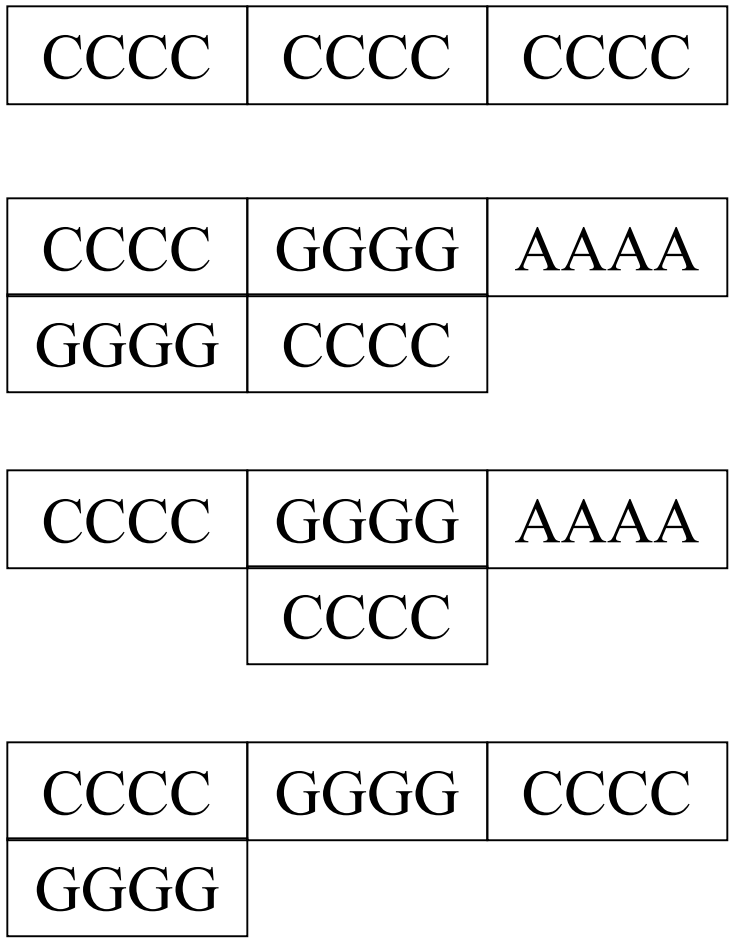
  (3, 2) *library*.

- This means we have a memory strand that is of size 3, and has 2 positions allowed for input data.

- Thus the first 2 positions are used for input data, and the final position is used for storage/input.

A Quick Visualization

- Here is a visualization of this library:



Memory Considerations

- From this visualization we see that we can achieve an encoding of 2^l different kinds of memory complexes.

- We can formally define a memory complex as follows:

$$w0^{k-l},$$

where w is the arbitrary binary sequence of length l, and 0 represents the off state of the following k-l sequences on the DNA memory strand.

An Interesting Example: Minimal Set Cover

- Consider the following NP-complete problem:
  - Minimal Set Cover
    - Given a finite set S = {1, 2, …, p} and a finite collection of subsets {C_1, …, C_q} of S, we wish to find the smallest subset I ⊆ {1, 2, …, q} such that all of the points in S are covered by this subset:

$$\bigcup_{i \in I} C_i = S.$$

- We can solve this problem by using the brute force method of going through every single combination of the subsets {C_1, …, C_q}.
- We will use our rules to implement the same strategy using our DNA system.

Minimal Set Cover: Using DNA (I)

- The initial test tube N_0 will be a

$$(p+q,\ q)\ \textit{library}.$$

- This basically means that our memory stick has $p+q$ positions to model the p points we want to cover and the q subsets that we have in the problem.
- The q will be our data input positions, which are the q subsets that we have in the problem.
- What we have is the first q positions are the data input section, and the last p positions are our storage area.

Minimal Set Cover: Using DNA (II)

- We encode all of the subsets that we have in our problem into the first q positions of our DNA strand.
- The memory complexes in N_0 represent all possible subsets I of the set $\{1, 2, \ldots, q\}$.
- This set contains at least one potential solution to our problem.

- Each position in our q positions represents a single subset that is in our problem.
- A position that is turned *on* represents inclusion of that set in the solution.
- We simply go through each of the possibilities for the q subsets in our problem.

Minimal Set Cover: Using DNA (III)

- The *p* positions represent the points that we have to cover, one position for each point.
- The algorithm takes each set in *q* and checks which points in *p* it covers.
- Then it sets that particular point position in *p* to *on*.
- Once all of the positions in *p* are turned *on*, we know that we have a sequence of subset covers that covers all points.
- Then all we have to do is look at all solutions and determine which one contains the smallest amount of subset covers.

Minimal Set Cover: But How is it Done?

- So far we have mapped each subset cover to a position and each point to a position.
- However, each subset cover has a set of points, which it covers.
- How do we encode this into our algorithm?
- We do this by introducing a program specific rule, known as *cardinality*.

Definition: Cardinality of a Set

- The cardinality of a set, X, returns the number of elements in a set.
- Formally, we define cardinality as:
  - *card*(X).

- From this we can determine what elements are in a particular subset cover in terms of its position relative to the points in *p*.

- Therefore, the elements in a subset C_i, where $1 \leq i \leq q$, are denoted by C_i^j, where $1 \leq j \leq card(C_i)$.

Minimal Set Cover: Checking Each Point

- Now that we can easily determine the elements within each subset cover, we can proceed with the algorithm.
- We check each position in *q* and if it is turned *on*, we simply see what points this subset covers.
- For each point that it covers, we set the corresponding position in *p* to *on*.
- Once all positions in *p* have been turned on, then we have a solution to the problem.

Minimal Set Cover: The Program …

N_0 is a $(p+q, q)$ library.

for $i = 1$ to q

  Separate $+(N_o, i)$ and $-(N_o, i)$

  for $j = 1$ to $card(C_i)$

    $Set(+(N_o, i), q + C_i^j)$

  $N_o \leftarrow merge((N_o, i), -(N_o, i))$

for $i = q + 1$ to $q + p$

  $N_o \leftarrow +(N_o, i)$

Minimal Set Cover: Unraveling it All (I)

//Loop through all of the positions from 1 to q

for $i = 1$ to q

//Separate all of the on and off positions.

Separate $+(N_o, i)$ and $-(N_o, i)$

//loop through all of the elements that the subset covers.

for $j = 1$ to $card(C_i)$

//Set the appropriate position that this element covers in p.

$Set(+(N_o, i), q + C_i^j)$

//Now, merge both of the solutions back together.

$N_o \leftarrow merge(+(N_o, i), -(N_o, i))$

Minimal Set Cover: Unraveling it All (II)

//Finally, we loop through all of the positions in p …

for $i = q + 1$ to $q + p$

//… and separate all strands that have position i on.

N_o □ +(N_o, i)

- The final N_0 contains only memory complexes, where each of the p last substrands is *on*.
- That is, all these solutions cover the set S.

Minimal Set Cover: Notes on DNA Computing

- In a sequential computation the amount of work is enormous: for $q = 100$, we have to apply the procedure for each of the 2^{100} memory complexes.

- Things are different with DNA and the Sticker model:

  – All memory complexes in N_0, where the first substrand is *on* (that is C_1 is one of the sets in the proposed cover of S), are processed *simultaneously*.

  – The result is brought over to the next step.

  – Here the memory complexes having the second substrand *on* are processed *simultaneously*.

  – Hence, only q steps are required, rather than 2^q steps.

Minimal Set Cover: Output of the Solution

- Now we have all of the potential solutions in one test tube, we still have to determine the final solution.
- Note that the Minimal Set Cover problem finds the smallest number of subsets that covers the entire set.
- In our test tube, we have all of the solutions that cover the set, and one of these will have the smallest amount of subsets.
- We therefore have to write a program to determine this.

Minimal Set Cover: Finding the Solution …

for $i = 0$ to $q - 1$

  for $j = i$ down to 0

    separate $+(N_j, i + 1)$ and $-(N_j, i + 1)$

    $N_{j+1} \leftarrow merge(+(N_j, i + 1), N_{j+1})$

    $N_j \leftarrow -(N_j, i + 1)$

read(N_1);

else if it was empty, *read*(N_2);

else if it was empty, *read*(N_3);

…

Minimal Set Cover: Finding the Solution (2)

- The program takes each test tube and separates them based on number of positions in q turned on.
- Thus for example, …
  - all memory strands with 1 position in q turned on are separated into one test tube N_1,
  - all memory strands with 2 positions in q turned on are separated into one test tube N_2,
  - etc.
- Once this is done, we simply read each tube starting with the smallest number of subsets turned on to find a solution to our problem (of which there may be many).

Example: Extracting the Solution

- Minimal set cover:
  extracting the (minimal set) solution
- $q = 4$
- C_3 and any combination of sets C_1, C_2, C_4 cover S.
- No combination of C_1, C_2, C_4 covers S.

- $N_0 = \{ (1, 3), (2, 3), (3,4),$
  $(1, 2, 3), (1, 3, 4), (2, 3, 4),$
  $(1, 2, 3, 4) \}$

Example: Extracting the Solution (2)

	N_0	N_1	N_2	N_3	N_4
initial	(1,3), (2,3), (3,4), (1,2,3), (2,3,4), (1,2,3,4)	empty	empty	empty	empty
i = 0 separate on 1	(2,3), (3,4), (2,3,4)	(1,3), (1,2,3), (1,3,4), (1,2,3,4)	empty	empty	empty
i = 1 separate on 2	(3,4)	(1,3), (1,3,4) (2,3), (2,3,4)	(1,2,3), (1,2,3,4)	empty	empty
i = 2 separate on 3	empty	(3,4)	(1,3), (1,3,4), (2,3), (2,3,4)	(1,2,3), (1,2,3,4)	empty
i = 3 separate on 4	empty	empty	(1,3), (2,3), (3,4)	(1,2,3), (1,3,4), (2,3,4)	(1,2,3,4)

Final Considerations

- The operations outlined above can be used to program more practical solutions to other programs.
- One such area is in cryptography, where it is postulated that a DNA system such as the one outlined is capable of breaking the common DES (Data Encryption Standard) used in many cryptosystem.
- Using a (579, 56) library, with 20 oligonucleotide length memory strands, and an overall memory strand of 11,580 nucleotides, it is estimated that one could break the DES with about 4 months of laboratory work.

References

- Paun, G., Rozenberg, G., and Salomaa, A., *DNA Computing*, Springer,1998.
- Garret Suen, *Beginning of Molecular Computing*, CPSC 601.73 (W2002) presentation.