Introduction to Computational Biology
Homework 1 Solution

Problem 1: Biological properties of recombination
We have discussed the uniform one point recombination model in class and argued that it satisfies only the first two of the following properties:

- Mendel's first law: there is a 50% chance for a gene to come from either chromosomes.
- The probability of recombination is higher for distant genes.
- Mendel's second law: genes are asymptotically independent i.e. the probability that a recombination occurs between two genes at a large distance is equal to $p_1 \cdot q_2 + p_2 \cdot q_1$, where p_i is the probability of the first gene coming from chromosome i, and q_i is defined similarly for the second gene. That's a $1/2$ according to the first property.

Let's try to satisfy all three. Consider the following model: At each position $1 \ldots n$ along the chromosome, there is a probability p of crossing over to the other chromosome (and hence a probability $1-p$ of staying on the same chromosome). In other terms, this model assumes that the frequency of recombination is uniform along the chromosome (although in reality some sites are hot spots or cold spots for recombination).

(a) What is the probability that a given gene comes from chromosome 1 and how does it depend on p? Explain your answer.

Solution: We will assume that the recombination process is equally likely to start with any chromosome, $Pr[start\ X_1] = Pr[start\ X_2] = \frac{1}{2}$. Under this assumption, the recombination process is very symmetric. Consider a path (a pattern of jumps between the two chromosomes) by which a gene comes from chromosome 1. This path has a certain probability. There is a symmetric path with the same probability by which the same gene comes from chromosome 2.

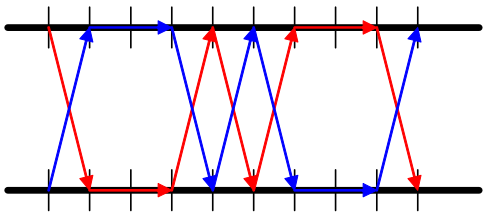

Figure 1: symmetric paths

Therefore, for every possible way of obtaining a gene from chromosome 1, there is a symmetric way of obtaining the same gene from chromosome 2. Hence, the

probability that a gene comes from either chromosomes is $\frac{1}{2}$. This is independent from the jumping parameter p.

(b) Derive an expression for the probability of recombination (or a way to compute it) between two genes at a distance d as a function of d and p.

Solution: Two genes at a distance d will recombine if there is an odd number of jumps between them. Therefore, the probability of recombination between two genes at distnace d is

$$p_d = \sum_{k \ odd} \left(\begin{array}{c} d \\ k \end{array} \right) p^k (1-p)^{d-k}$$

Another way to look at it is by regarding the jumping process as a Markov chain with two states X_1 and X_2 and the following transitional probability matrix

$$P = \left[\begin{array}{cc} 1-p & p \\ p & 1-p \end{array} \right]$$

Then the probability of being in state X_2 after d steps given that we start at X_1 is P_{12}^d. Similarly, the probability of being in state X_1 after d steps given that we start at X_2 is P_{21}^d. The probability of starting at X_1 is $\frac{1}{2}$ (see part (a), it is the same as the probability of a gene coming from chromosome 1). Therefore, $p_d = \frac{1}{2}P_{12}^d + \frac{1}{2}P_{21}^d = P_{12}^d$, because P is symmetric ($P_{12} = P_{21}$).

(c) What values of p satisfy the three biological properties listed above?

Solution: We can see that for $p < \frac{1}{2}$, all three properties will be satisfied since the first property is satisfied regardless of what p is (see part (a)), and the probability of recombination increases with d until it hits $\frac{1}{2}$ for large d where the genes act as if they are independent.

Theoretically speaking, for $p < 1$, we expect that the probability of recombination P_{12}^d will converge to $\frac{1}{2}$. This is a property of Markov chains, where P^d converges to the steady state probabilities of the states X_1 and X_2, $\lim_{d\to\infty} P^d = \left[\begin{array}{cc} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{array} \right]$.

The convergence will not occur when $p = 1$, because in this case we will have a periodic Markov chain with period 2 (a state can only be revisited at even intervals of steps). Intuitively, $p = 1$ means that we always jump, so the probability of recombination is 0 for even d and 1 for odd d (no convergence).

It would be interesting to explain why $p < \frac{1}{2}$ satisfies the second property (i.e. probability of recombination increases with distance), whereas $p \geq \frac{1}{2}$ does not (oscillates). The key is in the evaluation of the expression:

$$p_d = \sum_{k \ odd} \left(\begin{array}{c} d \\ k \end{array} \right) p^k (1-p)^{d-k}$$

Instead of explicitly evaluating the above expression, we can express p_d in terms of p_{d-1}. Note that

$$p_d = p(1 - p_{d-1}) + (1-p)p_{d-1}$$

Therefore, $p_d = p + (1-2p)p_{d-1}$. From this recurrence, and the fact that $p_1 = p$, we can compute

$$p_d = \frac{1 - (1-2p)^d}{2}$$

When $p < 1/2$, $(1-2p)$ is positive and, therefore, p_d increases to eventually $1/2$. When $p > 1/2$ (but less than 1), $(1-2p)$ is negative and, therefore, $(1-2p)^d$ alternates between positive and negative depending of whether d is even or odd respectively, but p_d will also converge to $1/2$.

Problem 2: DNA coverage
Many DNA sequencing techniques rely on cutting the DNA into many overlapping fragments. This exercise should help you understand how many fragments are needed. Assume we have n fragments of length l each and a DNA sample of total length T. Assume further that the position of a fragments along the DNA is uniformly random.

(a) Show that the expected number of bases that are **not** covered by fragments is approximately $Te^{-nl/T}$. *Hint*: Let X_i be an indicator random variable that base i is not covered, i.e. $X_i = 1$ iff base i is not covered. Compute $E[X_i]$ and then use linearity of expectation.

Solution: $E[X_i] = P(X_i = 1)$. The probability that a fragment covers base i is approximately l/T (if the base is at the extremity this is not true). Therefore, $P(X_i = 1) = (1 - l/T)^n = (1 - nl/Tn)^n \approx e^{-nl/T}$. Using linearity of expectation, we get $Te^{-nl/T}$.
(b) Let $n = \alpha T/l$. How big should α be?

Solution: The above becomes $Te^{-\alpha}$. So if we want an ϵ fraction not to be covered, we need to choose α such that $\alpha = \log 1/\epsilon$. This is how much redundancy we will need.
(c) Given a fragment, what is expected number of overlapping fragments? What is the probability distribution for that number (this is needed to detect a possible repeat).

Solution: A fragment overlaps a given fragment with probability approximately $2l/T$. So the expected number of fragments that overlaps a given one is $2nl/T = 2\alpha$. The probability distribution is the binomial, but if $n \gg 2l/T$, it can be approximated by a Poisson with parameter 2α.

Problem 3: Shortest Covering String
Recall the shortest covering string problem we described in class. The goal is to find a shortest string over the alphabet of probes that covers all the fragments. A string S is said to cover a fragment f if S has a substring that contains the exact set of probes in f (order and multiplicity are ignored).

Example:

$f_1 : \{A, B\}$, $f_2 : \{A, C\}$

The string $ABAC$ is a covering string for f_1 and f_2. However, this string is not the shortest possible. Since the order of probes is not important, BAC, for instance, is also a covering string. In BAC the substring BA contains the probes $\{A, B\}$ of f_1 and the substring AC contains the probes $\{A, C\}$ of f_2. In BAC both f_1 and f_2 are covered by substrings (BA and AC respectively) that do not contain probe repetitions.

Construct an example where, in the shortest covering string, one fragment must be covered by a substring that contains a probe repetition. This is not trivial!
Solution: Consider the following instance of the problem:

$C_1 : \{A, B\}$
$C_2 : \{A, C\}$
$C_3 : \{A, D\}$
$C_4 : \{A, E\}$
$C_5 : \{A, B, C, D, E\}$

The thing to note about this instance is that a shortest covering string for C_1, C_2, C_3, and C_4 is also a covering string for C_5. Therefore, we will focus on the shortest covering string for the first four probes only.

The shortest covering string must have at least one occurrence of each of B, C, D, and E.

Moreover, the shortest covering string must have two occurrences of A that are not the start or the end of the string. Proof: By contradiction. Assume that the shortest covering string has at most one A that is not the start or end of the string. Therefore, the shortest covering string has at most three As, two of them are at the extremities of the string. We need to cover $C_1 : \{A, B\}$, $C_2 : \{A, C\}$, $C_3 : \{A, D\}$, and $C_4 : \{A, E\}$. Therefore, the middle A must cover two clones, and each A at one extremity must cover one clone. Without loss of generality, the shortest covering string looks like $AB...CAD...EA$. But then we can obtain a shorter covering string by droping the last A and moving E to the beginning: $EAB...CAD...$.

Therefore, the shortest covering string must have a length of at least 6 with

at least two As being not at the start or end of the string. As a result, any shortest covering string must have the form $-A--A-$, where the four $-$ are filled arbitrarily with B, C, D, and E, because this form has exactly length 6.

Therefore, C_5 will be covered by a substring (the whole string in this case) that contains a repetition of A.

Problem 4: Shortest Superstring
Construct a shortest superstring for all the binary strings of length 4, i.e. 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111. *Hint*: you will know it is shortest if, starting with some pattern of 4 bits, it requires only 1 bit to cover an additional pattern.

Solution: A possible shortest superstring of length 19: 0100001110110010101. The length 19 is also a lower bound on any superstring for this problem, because the best thing we can do is start with one string of 4 characters, and then add 1 character for each additional string for the remaining 15 strings.

In general, for any k, the lower bound is $k + 2^k - 1$. But the question is: Can we always construct a superstring of length $k + 2^k - 1$ for all binary strings of length k? The answer is yes...

Problem 5: Circular DNA alignment
Consider two circular DNAs x and y of length m and n respectively. We are after the optimal global alignment of x and y. This can be obtained as follows: Consider a circular shift of x, $x_i...x_m x_1...x_{i-1}$ for some $1 \leq i \leq m$. Consider a circular shift of y, $y_j...y_n y_1...y_{j-1}$ for some $1 \leq j \leq n$. Find their optimal global alignment, and repeat for every possible pair of circular shifts of x and y. Finally pick the highest scoring alignment. Since there are m circular shifts of x and n circular shifts of y, the above algorithm will take $O(m^2n^2)$ time.

(a) Design an $O(mn^2)$ time algorithm that will find the optimal global alignment of two circular DNAs x and y.

Solution: Consider an optimal circular alignment of x and y. If we cut the circular alignment at x_1, we obtain an alignment of $x_1...x_m$ with some circular shift of y that has the same score as the score of the optimal circular alignment. Therefore, to find the optimal circular alignment it is enough to compute the optimal scores for the alignments of $x_1...x_m$ with all circular shifts of y and return the alignment corresponding to the best score. So we compute n dynamic programming tables A_i, each of which takes $O(mn)$ time. Then we pick the highest score among the $A_i(m, n)$ in $O(n)$ time. The total time so far is $O(mn^2)$. We need an additional $O(m + n)$ to obtain the alignment itself, but this does not change the asymptotic bound of $O(mn^2)$.

(b) [optional] Can you obtain the optimal global alignments for all pairs of circular shifts (i.e. mn) of x and y in $O(mn^2 + nm^2)$?

Solution: We can build n tables A_1, ..., A_n for aligning $x_1...x_m$ with the n circular shifts of $y_1...y_n$. We do the same for the reverse of x and y, i.e. we build n tables A'_1, ..., A'_n for aligning $x_m...x_1$ with the n circular shifts of $y_n...y_1$. This so far takes $O(mn^2)$ time.

$A_k(i, j)$ is the optimal score of aligning $x_1...x_i$ with the j^{th} prefix of the k^{th} circular shift of y. Similarly, $A'_k(i, j)$ is the optimal score of aligning $x_m...x_i$ with the j^{th} suffix of the k^{th} circular shift of y.

Now observe that an alignment of a circular shift of x, $x_i...x_m x_1...x_{i-1}$ with a circular shift of y, $y_j...y_{j-1}$ corresponds to an alignment of $x_i...x_m$ with some suffix $y_j...y_k$ of the k^{th} circular shift of y followed by an alignment of $x_1...x_{i-1}$ with some prefix $y_{k+1}...y_{j-1}$ of the $(k + 1)^{st}$ ciruclar shift of y. We can try all the n possibilities for k and pick the best one, each of which takes $O(1)$ time given the information we pre-computed above.

Therefore, after the pre-computation step, the score of each of the mn alignments can be computed in $O(n)$ time, and each alignment can be obtain in $O(m + n)$ time (by obtaining the actual alignments for each of the two highest scores of the sub-alignments), resulting in $O(m + n)$ time for each of the mn alignments. The total running time of the algorithm is therefore: $O(mn^2) + mn.O(m + n) = O(mn^2 + n^2m)$