Computational Biology
Lecture 13: Physical mapping by hybridization
Saad Mneimneh

As mentioned before, we have two approaches for physical mapping: Restriction mapping and mapping by hybridization. We covered restriction mapping previously through the two problems of double digest and partial digest. We now look at mapping by hybridization.

While restriction mapping involves the mapping of restriction sites (precise short sequences) of a cutting enzyme based on the lengths of the restriction fragments, mapping by hybridization involves the mapping of clones (fragments) based on hybridization data with probes. The mapping of clones in this case is not exact; we try to find a likely overlap of the clones.

Mapping by hybridization - a quick jump into abstraction

Mapping by hybridization involves a number of biological experiments known as hybridization experiments. Several copies of the DNA are cut into overlapping fragments, called clones in this context. A set of probes is constructed, and for each clone, the set of probes that hybridize with it is determined. This hybridization data can be represented as a matrix D, where $D_{ij} = 1$ iff probe j hybridizes with clone i. Given D, the problem is to find a possible overlap of the clones on the original DNA.

The matrix D does not constitute enough information for determining the overlap. Many overlaps can be consistent with the experimental data. For instance, we could assume that clones do not actually overlap and occur consecutively along the DNA. To make the problem sound, i.e. to be able to favor one overlap over another, we need to take into consideration some information about the clones and how the probes where generated. In other words, we need to model the clones and probes.

A famous model is the Lander-Waterman model:

- The clones have equal length and each clone start at a position that is uniformly distributed and independent from all other clones
- Probes (represented as points) are placed along the DNA according to mutually independent Poisson processes

Given an instance D obtained from the model above, the problem is to find the most likely overlap of clones that is consitent with D.

Alizadeh et al. 1995 showed that the problem of obtaining this maximum likelihood can be approximated as the problem of minimizing the number of occurrnences of probes needed to explain D: the Shortest Covering String SCS problem.

Shortest Covering String SCS: A clone c is covered by a string s if s has a substring containing only the probes that hybridize with c (order and multiplicity are ignored). The shortest covering string is the shortest string that covers all the clones.

We have two variations for the SCS problem:

- SCS with Non unique probes (NP-hard)
- SCS with unique probes (polynomial)

SCS with non unique probes

If probes are non unique, then a probe can occur in more than one place along the DNA. Since a probe needs not be unique on the DNA, these non unique probes can afford to be relatively short, and hence are easy to generate biologically. As we have seen before with restriction mapping, what might seem easy to accomplish biologically might be very hard to solve computationally, and vice versa. We will encounter this situation here again: Finding the shortest covering string is NP-hard when probes are non unique. However, we will see later that if we can ensure that probes are unique (every probe occurs once along the DNA, harder biologically), then the shortest covering string can be obtained in polynomial time.

In this section we will focus on a heuristic algorithm for SCS with non unique probes since the problem is NP-hard. An important discovery is the following:

Given a permutation of the clones π, finding the shortest string s_π that covers the clones in the order (of their left end points) given by π has a polynomial time algorithm. For simplicity, we say that s_π covers π. Of course, this is not necessarily the shortest covering string as defined above.

For example, given the permutation $(1, 3, 2)$ of clones $C_1 = \{A, B, C\}$, $C_2 = \{B, C, D\}$, and $C_3 = \{C, D, E\}$, the shortest string that covers this permutation is $s = ABCEDBC$. Note that this is longer than the shortest covering string $s' = ABCDE$ which covers the permutation $(1, 2, 3)$.

The shortest covering string can be alternatively defined as the shortest string in $\{s_\pi \mid \pi \textit{ is a permutation}\}$. Explicitely trying all permutations will give an exponential time algorithm because we have $n!$ possible permutations.

Using the polynomial time algorithm for finding s_π for a given π, we develop a heuristic algorithm that does not necessarily return the shortest covering: we start with an arbitrary permutation π and compute s_π in polynomial time. Then we change π slightly to obtain a number of neighboring permutations (polynomially many of course), and for each such permutation π', we compute $s_{\pi'}$ in polynomial time. If $s_{\pi'}$ is shorter than s_π for some neighbor π' of π, then we set $\pi = \pi'$ and repeat the process. Otherwise, we stop. It is obvious that after each iteration, we get a shorter covering string. This will therefore stop after at most a polynomial number of iterations (the length of the original string s_π is polynomial in the size of the problem). The final string is not necessarily to shortest covering string, but it represents a local minimum along a path of neighboring permutations. Below we show the algorithm:

start with an arbitrary permutation of clones π
$length \leftarrow \infty$
compute s_π in polynomial time
while $|s_\pi| < length$
    $length \leftarrow |s|$
    compute a set of neighbors of π (polynomially many)
    for each neighbor π' of π
        compute $s_{\pi'}$ in polynomial time
        if $|s_{\pi'}| < |s_\pi|$
            then $\pi \leftarrow \pi'$
return s_π

It remains to show how to compute s_π for a given π in polynomial time. For developing the algorithm we will adopt the following three assumptions:

- No clone properly contains another (motivated by the Lander-Waterman model where all clones have the same length).
- The DNA is completely covered by clones, i.e. we have no gaps (motivated by good coverage of the DNA, the expected fraction of uncovered DNA is $e^{-nl/T}$ where n is the number of fragments and l is their length, and T is the length of the DNA).
- Without loss of generality, $\pi = (1, 2, ..., n)$, and hence we consider overlaps in which C_1, C_2, ..., C_n appear in order (of their left end points).

Given the three assumptions above, an overlap of clones C_1, ..., C_n defines intervals marked by their end points. In other words, every interval is a maximal interval that does not contain a clone's end point (left or right). This is illustrated in the figure below.

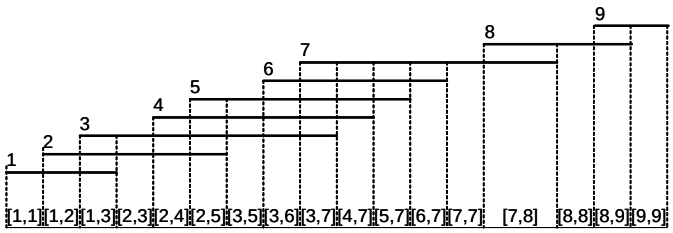


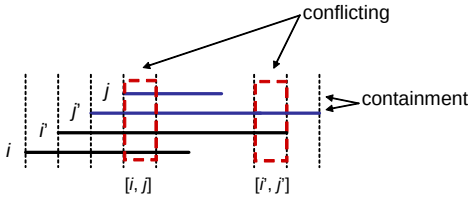
Figure 1: Intervals defined by an overlap

Each interval $[i, j]$ means that clones C_i to C_j overlap. We will add another assumption to our list above:

Error Free Hybridization: If a probe $p \in [i, j]$, then p hybridizes with all clones from C_i to C_j. A probe is therefore considered as a point in an interval of overlaps (i.e. we cannot detect an overlap smaller than the length of a probe).

We will define two intervals $[i, j]$ and $[i', j']$ to be conflicting iff $i < i' \leq j' < j$. Then the intervals defined by an overlap of C_1, ..., C_n are conflict free.

Proof: If not, then some clone will contain another. Assume $[i, j]$ and $[i', j']$ are conflicting, e.g. $i < i' \leq j' < j$. The left end points of the intervals are sorted (clones appear in order). Without loss of generality, the following figure depicts the situation of conflict, where clone j is contained in clone j' (and possibly others) (i' and j' could be the same).



The conflict-free property implies that intervals defined by an overlap of C_1, ..., C_n can be sorted by both their left end points and their right end points (how? first sort the intervals by the left end points, then within each set of intervals with the same left end point, sort the intervals by their right end points). In fact, conflict free and the ability to be sorted by both end points are equivalent.

Now we define an equivalence between an overlap of C_1, ..., C_n and a set conflict free intervals.

- Every overlap of C_1, ..., C_n defines a conflict free set of intervals.
- For every conflict free set of intervals I, there is a conflict free set of intervals I', $I \subseteq I'$, that defines an overlap of C_1, ..., C_n.

The first part of the equivalence is trivial and we just proved it. To prove the second part, we explicitly construct an overlap with a confclit free set of intervals that contain all intervals in I. First we add $[1, 1]$ and $[n, n]$ to the set I (if needed). Note that I is still conflict free because no interval can contain $[1, 1]$ or $[n, n]$. Then we sort all intervals in I by both their left end points and their right end points. Finally we fill gaps by adding new intervals to I. For instance, to fill the gap between $[i, j]$ and $[k, l]$ we first increment j until it reaches l, then increment i until it reaches $k - 1$, e.g.

$$[\mathbf{i}, \mathbf{j}],\ [i, j+1],\ ...,\ [i, l],\ [i+1, l],\ ...,\ [k-1, l],\ [\mathbf{k}, \mathbf{l}]$$

This is equivalent to saying that clones $j + 1$ to l start in successive intervals, then clones i to $k - 1$ ends in successive intervals, thus going from $[i, j]$ to $[k, l]$ with conflict free intervals.

The equivalence between a valid overlap (i.e. C_1 to C_n) and the existence of conflit free intervals is an imporant one that we will use in finding the shortest string that covers C_1 to C_n in special cases, then generalize for all cases.

Consider the hybridization matrix D. Consider all intervals $[i, j]$ such that there is a column with a a maximal run of ones from C_i to C_j. In other words, there is a probe p that hybridizes with C_i, ..., C_j but not with C_{i-1} and C_{j+1} (we can assume C_0 and C_{n+1} are dummy clones with no hybridizations). We say that $[i, j]$ is an interval of p. If these intervals are conflict free, then they can define an overlap of C_1, ..., C_n by our equivalence above. Therefore, if we sort these intervals (by their left end points and their right end points) and place each probe in its intervals, we end up with a string of probes that covers C_1 to C_n in order. Figure 2 shows an example.

The conflict free intervals of D in Figure 2 are: $[1, 2]$, $[6, 8]$, $[2, 4]$, $[7, 8]$, $[1, 1]$, $[3, 5]$, $[7, 9]$, $[2, 3]$, $[3, 6]$, $[3, 8]$. Sorting them and placing each probe in its intervals give the following:

$[1,1]$	$[1,2]$	$[2,3]$	$[2,4]$	$[3,5]$	$[3,6]$	$[3,8]$	$[6,8]$	$[7,8]$	$[7,9]$
C	A	E	B	CD	F	G	A	B	D

The string $CAEBCDFGABD$ is a string that covers C_1, ..., C_n in order. The question is now the following: is it the shortest such string? The answer to this question is yes. We can easily show that in any string covering C_1, ..., C_n, each probe has to appear a number of times euqal to the number of its intervals.

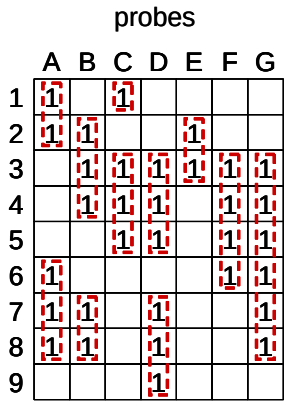
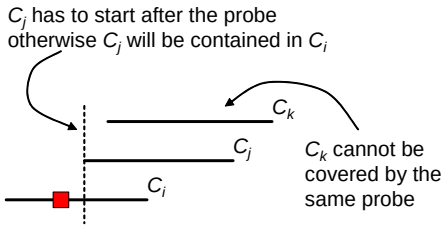


Figure 2: Conflict free intervals of D

Proof: This is a direct consequence of the following fact: Let $i < j < k$, then any probe common to C_i and C_k but not C_j has to appear twice, once for C_i and another time for C_k; otherwise, C_j would be contained in C_i, as illustrated below.



Therefore, what we obtain in the special case of conflict free intervals of D is shortest because each probe appears exactly a number of times equal to the number of its intervals.

We sould like to generalize this idea to all cases. What if the intervals of D are not conflit free? We will see that the problem of finding the shortest string that covers C_1, ..., C_n reduces to sub-dividing the set of intervals into a conflict free set with a minimum number of intervals, and then obtain the shortest covering string as explained above.

The see why the above is true, first note that a set of t conflict free intervals in D defines a covering string of length t (we just proved this above). Moreover, a covering string of length t defines a set of t conflict free intervals in D (not necessarily maximal runs of 1's anymore): (1) obtain an overlap of C_1, ..., C_n given by the covering string, (2) the overlap defines a set of conflict free intervals, (3), each occurrence of a probe p falls within an interval (i, j), (4) an interval $[i, j]$ containing a probe p must be a run of 1's (not necessarily maximal) in column p of matrix D (error-free assumption, p hybridizes with $C_i, ..., C_j$, (5) therefore we find t conflict free intervals in D. Therefore, the shortest covering string corresponds to the minimum number of conflict free intervals.

The problem is how to sub-divide the intervlas in a way to obtain a minimum number of conflict free intervals? Here's the strategy that we are going to use:

- Let $[i', j'] \subset_c [i, j]$ mean that $i < i' \leq j' < j$.
- Sub-divide an interval $[i, j]$ into the minimum number of intervals needed to remove any conflict with $[i', j']$ where $[i', j'] \subset_c [i, j]$.
- The sub-intervals of $[i, j]$ must not create additional conflicts (this is key)
- Then we repeat this for every interval and hence we compare an interval $[i, j]$ only with the original set of intervals

The most crucial aspect of the above algorithm is for the new sub-divisions not to create additional conflicts. Although this seems unreasonable to assume, we will see how to ensure this feature. But first, lets see how we can sub-divide $[i, j]$ into the minimum number of intervals to remove conflicts with $[i', j']$ where $[i', j'] \subset_c [i, j]$.

Consider the minimal j' such that $[i', j'] \subset_c [i, j]$. Then it is clear that a cut of $[i, j]$ in the region $[i, j']$ is unavoidable, since otherwise $[i', j'] \subset_c [i, j]$ remains. The best thing therefore is to delay this cut as much as possible, hence cutting $[i, j]$ at j'. This is illustrated in the Figure 3 below.

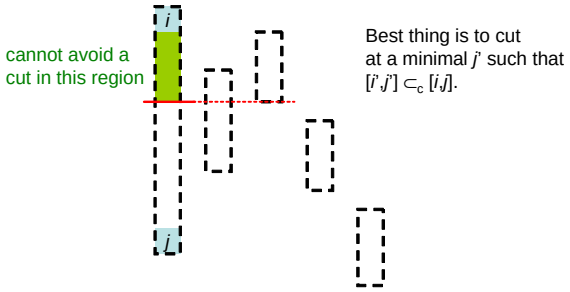


Figure 3: Unavoidable cut

Inductive reasoning and the above idea suggest the following algorithm for sub-dividing the interval $[i, j]$:

- Find a maximum number of original intervals $[i_1, j_1]$, $[i_2, j_2]$, ..., $[i_t, j_t]$ such that:
  - $[i_x, j_x] \subset_c [i, j]$
  - j_1 is minimum,
  - $i_2 > j_1 + 1$, $i_3 > j_2 + 1$, ..., $i_t > j_{t-1} + 1$
- Cut $[i, j]$ at j_1, j_2, ..., j_t producing $t + 1$ sub-intervals

In the algorithm above, each cut is unavoidable and taken as late as possible; therefore, this is the minimum number of sub-divisions to remove all conflicts of $[i, j]$ with $[i', j']$, $[i', j'] \subset_c [i, j]$.

Now we come back to the the concern of not creating new conflicts. It seems that this is quite impossible because it is very conceivable that the small sub-divisions will be in conflict with other intervals. Although this is true, any new conflict is part of a previously existing conflict that must be avoided anyway. The following claim (almost) proves that there is no need to worry about the sub-divisions.

Assume two sub-divisions of the original intervals $[i, j]$ and $[k, l]$ are in conflict. Then either $[i, j]$ or $[k, l]$ did not resolve all conflicts with original intervals.

Proof (almost): By picture. The middle interval (original) must exist for the first sub-division to occur (by our algorithm above). Therefore, if the first sub-division is in conflict with the second sub-division, so is the middle original interval.

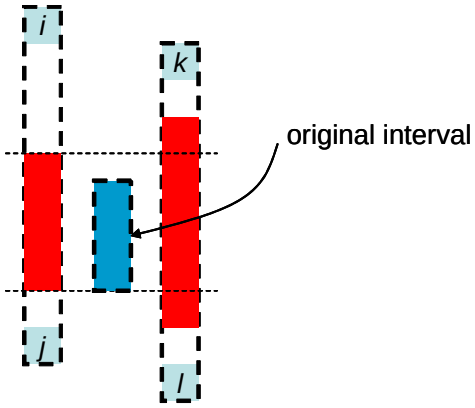


Figure 4: Almost a proof

We say that the above is almost a proof because the argument does not hold for the last sub-division of an interval. If in the figure above, the sub-division is the last sub-division of $[i, j]$, then it is the remainder of $[i, j]$ resulting from the last cut, and was not produced as a consequence of the existence of an original interval. In this case, the sub-division could in deed be in conflict with others. However, this can be resolved.

Fixing the last sub-division:

Consider the interval $[a, b]$ such that $[a, b] \subset_c [i, j]$ and a is maximum such that $a \geq i_t$ (it could be $[i_t, j_t]$ itself). Then fix the last sub-division to be $[a, j]$. Then we are back to the same argument because if $[a, j]$ is in conflict with some sub-division, so is the origial interval $[a, b]$. Note that this process does not change the number of sub-divisions and by the choice of $[a, b]$ does not create any additional conflicts. This is illustrated in Figure 5 below:

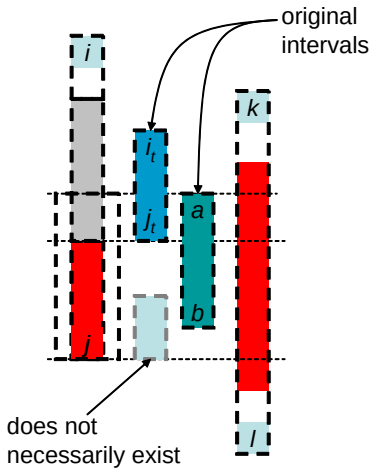


Figure 5: Fixing the last sub-division

References

Pevzner P., Computational Molecular Biology, Chapter 3.