Lecture 13

Mapping, II

13.1. Practical STS Mapping

13.1.1. A Model of STS Mapping with Errors

The previous mapping lecture demonstrated how, in the absence of errors, STS mapping could be solved efficiently via a reduction to a well-known abstract problem called the consecutive-ones problem. The solution made use of a data structure called a PQ-tree which eloquently represents all possible orderings of the STS sites that are consistent with the data. The flaw in this approach, however, is that it is not tolerant of the experimental error found in real instances of the STS mapping problem.

This section presents a different model of the STS mapping problem which includes several types of common errors. Unfortunately, the problem becomes NP-hard under this model, but it is possible to approximate a solution with various optimization methods. In this model of the problem, just as before, we consider each clone to be a 0-1 vector in which each 1 represents the measured presence of a particular probe on the clone and each 0 represents its absence. Once again, we can generate a matrix in which the columns represent the clones, and try to rearrange the order of the rows so that all columns satisfy the consecutive-ones property. However, we also allow the data to contain three types of errors, which may cause some columns to violate the consecutive-ones property:

**False Positive** A false positive is a 1 in a particular clone that should be a 0. This is generally due to experimental error.

**False Negative** A false negative is 0 in a particular clone that should be a 1. This can be due to experimental error, or due to a *deletion*, in which a DNA fragment is deleted from the clone during replication by the host organism.

**Chimeric Clone** This is a clone that is composed of DNA fragments from two distinct regions from the genome. Although such an error may appear as a number of false positives or negatives, it is generated by a distinct biological event and in general it appears as a clone in which there are two separate sections that are contiguous in the target DNA.
Figure 13.1: An example of STS data from a particular region. Note that clones 3 and 5 are chimeric, and thus are split into two segments.

Figure 13.2: Data derived from Figure 13.1. (A) The initial data; the probes are in no particular order. (B) The probes in their proper order. The “x” represents a false positive, and the two asterisks show false negatives. Clones 3 and 5 are chimeric, and hence their columns contain two groups of consecutive ones.

Figure 13.1 gives a graphical depiction of an area with 6 probes and 5 clones. Figure 13.2 contains two tables; the first is a set of data derived from the situation in the picture including two false negatives and a false positive, and the second shows the correct arrangement of the probes.

Suppose for now that we have a particular solution in mind, and we want to know how likely it is. We assume that we have ordered the probes and designated false positives/negatives and chimeric clones in such a way as to make the order valid. Suppose also that along with the data, we are given the probability $\epsilon$ that a given 0 is a false negative, the probability $\delta$ that a given 1 is a false positive, and the probability $\gamma$ that a given clone is chimeric. We claim that we can easily express the “cost” of the solution as a negative log likelihood function of $\epsilon$, $\delta$, and $\gamma$. Furthermore, we claim that given a particular ordering of the probes and the cost function, we can easily find the least costly labeling of false positives, negatives, and chimeric clones via dynamic programming. The key to the simplicity of the dynamic programming algorithm lies in the fact that the columns can be treated independently once the clone ordering is given. The dynamic programming algorithm allows us to think of our cost function $c$ as simply a mapping of a probe ordering to a cost. Hence, we only need to find an ordering $\pi$ satisfying $\min_{\pi} c(\pi)$. This requires us to choose a search technique to find such a $\pi$. 
13.1.2. Introduction to Local Search

Suppose that we are given a permutation \( \pi \) of probes. We define the *neighborhood* of \( \pi \) to be a set of permutations obtained by applying a single “step” to \( \pi \). The definition of a “step” is up to the creator of the search algorithm; in the case of a permutation it may be defined as reversing some block in the permutation, swapping a pair of elements, etc. The only necessary property is that, given a valid starting permutation for the search, it is possible to reach any other valid permutation via a series of steps. It is obvious that using these definitions, we can think of the search space as a graph in which there is an edge between each permutation and all of its neighbors.

The simplest type of search is local, greedy search, in which we start with a permutation \( \pi \). At each step of the search, we choose the neighbor \( \pi' \) of \( \pi \) such that \( c(\pi') \) is lower than the cost of any other neighbor and is also strictly smaller (or larger, if a maximum rather than minimum is desired) than any previously-encountered cost. We stop when no such \( \pi' \) exists. This approach is known as hill climbing (when searching for a maximum) or descent (when searching for a minimum). In the case of a complicated function, this type of search is very likely to find a local extremal point, so one common variation is to repeat the procedure with a number of starting positions and take the best result.

Another variation on greedy search is simulated annealing, named after a physical process in metallurgy. In this procedure, there is a monotonically decreasing function \( T(n) \) called the *cooling schedule* which represents the “temperature” after \( n \) steps. This function is up to the creator of the search, and the quality of the function can dramatically affect the quality of the results. Choosing a good cooling function for a particular application is more of an art than a science. The search proceeds by repeating the following step a number of times, until either a maximum number of steps is reached or the process converges:

- Let the current step be \( n \) and the current permutation be \( \pi \).
- Randomly choose a neighbor \( \pi' \) of \( \pi \).
- If \( c(\pi') < c(\pi) \) then move to \( \pi' \); otherwise, move to \( \pi' \) with probability \( e^{-\frac{c(\pi') - c(\pi)}{T(n)}} \).
- Increment \( n \), thus reducing \( T \), and repeat.

See [1] for more details of this approach to STS mapping, and [3] for another approach.

13.2. Radiation Hybrid Mapping

Radiation hybrid mapping is an alternate technique for STS mapping in which human cells are exposed to gamma radiation, thus randomly cleaving the target DNA into many small sections. The cells are then fused with hamster cells, and a random subset of the fragments is incorporated into the genome of each hamster cell. A hybrid cell line is then grown from each hamster cell, and members of each line are tested for presence of each STS. This technique has been used to create a map of the human genome with more than 12,000 STS’s.

We can model the data from this experiment with a matrix in which the entry in row \( i \) and column \( j \) is a + if STS \( j \) has been detected in hybrid \( i \), or a − otherwise. We define an *obligate break* as a transition
Figure 13.3: An example of radiation hybrid mapping. The two tables underneath the map show two orderings of the probes and the corresponding number of obligate breaks.

from + to − or vice-versa in a row of the matrix. Figure 13.3 depicts a map constructed via radiation hybrid mapping and also the number of obligate breaks resulting from placing the probes in different orders.

One traditional method of approaching the problem is to attempt to find an ordering that minimizes the total number of obligate breaks. This approach makes the assumption that the map that minimizes the number of fragments for each hybrid is probably correct.

The criterion of minimizing the number of obligate breaks suggests a distance metric between sites, in which the distance from site A to site B is defined as the number of hybrids in which either A or B appears, but not both. Ben-Dor and Chor [2] then use this data to try to find a map that minimizes the sum of all the distances between consecutive sites on the map; such a map minimizes the number of obligate breaks. This corresponds to the very well-known traveling salesman problem, which can be stated abstractly as follows: given a weighted graph find a cycle (or, equivalently in terms of difficulty, a path) of minimum weight that visits all of the vertices exactly once. The stereotypical explanation is that the vertices represent cities, the edges represent flights between cities with associated costs, and a salesman wishes to visit each of the cities exactly once and as cheaply as possible. In our model, the sites correspond to the vertices, there is an edge between every pair of sites, and the weights correspond to the distances above. This problem is NP-hard, but since the problem has been heavily studied there are heuristics that do very well in practice.

Slonim et al. [4] adopt a more complex model which incorporates missing data, false positives and negatives, retention probabilities of fragments, and break probabilities. Their approach relies on a method for estimating the likelihood of an ordering of any subset of the probes. From this, they construct a scaffold, which is a map of a subset of all of the probes that is believed to be very likely to be correct. Finally, they try to position the more ambiguous probes in relation to the scaffold.

To evaluate a particular ordering of probes 1, 2, …, n, they construct a hidden Markov model in which there are two states, $R_t$ and $L_t$, for each probe $t$. The former represents the probability that $t$ is retained, while the latter represents the probability that $t$ is lost. The model also relies on the following probabilities: $p$, the probability that a fragment is retained; $\Theta_t$, the probability of a break between site $t$ and site $t + 1$; $\alpha$, the chance of a false negative; $\gamma$, the chance of a false positive; and finally, $\beta$, the chance of incomplete
or inconclusive data. Figure 13.4 shows a depiction of the HMM, and Figure 13.5 shows the transition and output probabilities. The HMM can be used to estimate the probability of the data given the ordering; it is simply the product of the probability of each hybrid as computed by the HMM given the ordering.

Using the HMM, they can construct a set of betweenness assertions for the data. A betweenness assertion has the form \( \langle i, k, j \rangle \) and corresponds to the belief that site \( k \) lies between sites \( i \) and \( j \). They make the assertion \( \langle i, k, j \rangle \) if the probability of the ordering \( i, k, j \) as computed by the HMM is much greater than that of \( k, i, j \) or \( i, j, k \). The scaffold problem is then to construct an ordering of the sites that satisfies the maximum number of betweenness assertions. The problem, as usual, is NP-hard, but can be approximated by heuristics or integer programming. Note also that a similar approach relying on betweenness is used by Christof et al. in their end-probe approach to STS mapping [3]. Once the scaffolding is in place, each of the remaining sites is considered in turn, and placed in its most likely location relative to the scaffolding sites as determined by the HMM. Note, however, that the non-scaffold sites are not ordered relative to each other.

References

