

Class 13: Sequence Assembly (2)

Held: Thursday, 8 October 2009

Summary: We continue our exploration of how DNA sequence databases are created.

Related Pages:

- EBoard.

Notes:

- Upcoming work: Read Chapter 6 for Tuesday, read Kellis et al. 2003 for Thursday.
- Things that depress Sam: So few students at George Drake's history of the two Grinnells.
- Bio and CS talks at noon tomorrow!
- Class will end early today (2:25).
- Interestingly, one of today's topic (P vs. NP) was a subject of an article in yesterday's *New York Times*.

Overview:

- Sequencing DNA, Continued
- CS Detour: TSP and NP-Completeness
- Reassembling Shotgun Sequences
- Testing Assembly Algorithms

Sequencing DNA

- Problem: How do you use a technique for sequencing short segments and turn it into a technique for sequencing long segments?
 - Solutions may require both biology and computer science
- Approach one (primarily biological): Sequential sequencing:
 - Since you can use a primer to determine where sequencing starts (or so we hope), you sequence a little, make a primer, sequence a little more, make a primer, and so on and so forth.
 - Should be very accurate (unless, of course, your primer is wrong or binds at the wrong place)
 - Takes a lot of time to do
- Approach two (primarily computational): Shotgun sequencing
 - Blast your sequence into lots and lots of small sequences.
 - Sequence each of those small sequences
 - Use cool algorithms to glue them back together.
- Why choose one over the other?

CS Detour: Assessing the Complexity of Algorithms

- The book mentions that alignment is similar in difficulty to the Traveling Salescritter Problem (TSP).
 - Why do they say this?
- Computer scientists like to analyze the efficiency by which problems can be solved.
- We often look at how much work an algorithm we develop requires. For example, the N-W algorithm requires some constant times $N \times M$ steps, where N is the length of one sequence and M is the length of the other.
- We also like to provide *bounds* on the solution time of algorithms. For example, it would be impossible to write an algorithm that aligns the two sequences that takes less than $N+M$ steps, since we presumably have to look at each element of each sequence at least once.

The Traveling Salescritter Problem

- TSP is a relatively straightforward problem:
 - You have a list of cities and distances between them
 - You want to find the shortest path that visits every city (and doesn't visit any city twice)
- There's an obvious solution:
 - List every possible order of visiting the cities
 - For each order, find the distance
 - Choose the smallest
- Unfortunately, there are a *lot* of ways to order N different cities (N factorial, to be precise).
- Suppose there are 20 different cities
 - $20! = 2,432,902,008,176,640,000$
 - Suppose we could do a trillion operations a second (a lot more than most current computers).
 - We'd still have about 2 million seconds worth of operations to do.
 - About 33,000 minutes
 - About 555 hours
 - You can do the rest
- Interestingly, there is no known algorithm that is significantly better.
- And we've been working on TSP for more than thirty years.
- There are known fast algorithms that approximate the best solution.

NP Completeness

- A number of years ago, some computer scientists looked at a group of problems like TSP and developed a useful theory.
- There are some problems we know that have relatively efficient algorithms. Approximate string alignment is one of these. We call this set of problems 'P' (for 'solvable in polynomial time')
- There are some problems in which we can check a potential solution of a problem quickly. We call these 'NP'.
 - A variant of TSP is in this form: 'Is there a path through the cities of distance less than D ?'
 - The 'NP' problems are 'nondeterministic polynomial'.
- There are a subset of problems in NP that are provably as hard as any other problem in NP. We call

these 'NP Complete'.

- There are problems that we know to be at least as hard as the problems in NP. We call these NP hard.
- Many problems in NP are also in P. (That is, we can check an answer quickly and we can find an answer quickly.)
 - But there are many problems for which we have yet to find a fast algorithm, even if we can check solutions quickly.
 - Open question: Are there problems in NP for which we can never find a fast solution? (Most computer scientists say 'Yes.')
- When we say a problem is 'NP complete' or 'NP hard' we mean:
 - There's no known fast (or even reasonably fast) solution.
 - We can prove that this problem is as difficult as many other problems for which there is no known solution.
 - We therefore think it's unlikely that there will every be a fast exact solution.
 - All in all, it's bad news.

Reassembling Shotgun Sequences

- Expected Problems
 - A lot of data: Computationally expensive
 - The algorithm the book gives appears exponential in the number of sequences
 - Don't know which strand each sequence comes from
 - Doesn't do well with long sections of repeats (but, then, neither does the previous method)
 - Since there is no clear biological basis to the assembly, how do we know that the result is accurate?
- Note: The task of taking a group of strings and finding a string that contains all of them is called the shortest superstring problem
 - It is known to be NP complete [Maier and Storer, 1977, cited in Gary and Johnson 1979]
 - No provenly good approximation is known either
 - Gap between CS and Bio: Whether or not the shortest superstring problem or an approximation was solved, these techniques may not give the correct sequence.
- But, hey let's try
- Basic operation: Finding the best alignment of two segments (contigs)
 - Slightly different metric for best than we've used previously, since it's okay for ends to stick out (these would normally be considered insertions and deletions).
 - How would you upate our algorithms to handle this issue?
- Once we can align two sequences (or know the value of their best alignment), what do we do next?
- The greedy algorithm:
 - As long as contigs with high-value alignments remain
 - Find the value of the alignment of all pairs of contigs.
 - Choose the one with the greatest value.
 - Merge those two contigs into a single contig

Testing Assembly Algorithms

- How do we know whether an assembly algorithm works? We test it.
 - Computational approach: Start with a known sequence, simulate breaking it up into pieces, assemble, see if you get the the original sequence.
 - Another computational approach: Think hard about kinds of sequences in which the algorithm will have difficulty, and then do the same thing.
 - A possible biological approach: Sequence using another technique, break up using the shotgun approach, and see if the result is the known sequence.
-

Copyright © 2009 Vida Praitis and Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.