Introduction to computational and systems biology

- Lecture 6: Algorithmic sequence assembly

- Ion Petre
- Department of Mathematics and Statistics
- University of Turku
- Fall 2019

Turku Centre *for* Computer Science

Sequencing

- Learn the exact sequence of nucleotides of a DNA molecule
- The most popular method of sequencing (for short molecules): the *Sanger method*
- Main tool: nucleotide analogues – nucleotides chemically altered so that no other nucleotide can attach to their 3' end: ddA, ddC, ddG, ddT (dideoxynucleotides)

Sequencing – Sanger method

- Problem: sequence a single stranded molecule alpha
- Extend it to 3' by gamma: let beta be the new molecule
- Prepare 4 tubes (A,C,G,T) containing:
  - beta molecules
  - Primers gamma'
  - Nucleotides
  - Tube X contains a *limited* amount of nucleotide analogues ddX, for all X∈{A,C,T,G}

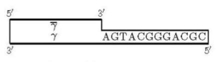


Figure 1.34: β' molecule

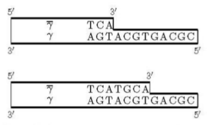


Figure 1.36: Incomplete molecules in Tube A

Sequencing

- Gel electrophoresis using 4 wells - one for each tube
- Read the bands (the primers were marked)
- Obtain the sequence
- Limitation: only fragments of up to 1000 bp can be sequenced with this method
- Idea:
  - Physical maps: large pieces of DNA are ordered according to their position in the genome (no sequencing here !)
  - Each fragment is cut in small pieces, each of them sequenced and then reassembled
  - **The critical difference between various methods is how to divide the problem into smaller tasks**

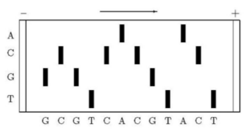


Figure 1.37: Sequencing ladder

Sequencing – the shotgun approach

- The *target sequence*: too long to sequence directly
  - We know approximately the length of the target (usually within 10%)
- Take a large number of copies of the target and cut them individually so that individual fragments overlap
  - We have some single stranded *fragments*
  - We do not know from which strand they are, their position, or if they contain errors
- *Shotgun approach*:
  - Obtain a large number of fragments
  - Sequence them
  - Reconstruct the target based on the fragment overlap: obtain a DNA sequence so that all fragments "fit" in one of the strands of the sequence
- *Other models exist, e.g., directed sequencing, sequencing by hybridization and DNA chips*

Sequencing – some numbers

- Typical data
  - Target sequences of 30 000 to 100 000 bp
  - Fragments of 200 to 700 bases
  - Total number of fragments: 500 to 2000
  - The length of the target sequence is known within 10%
- Deduce the sequence of the target DNA molecule based on the fragment overlapping
- The problem: *Fragment Assembly* (or *Sequence Assembly*)

Sequence assembly – ideal case

- Example – four fragments (TACCGT , ACCGT, CGTGC, TTAC) of a target of about 10 bp long
- One possible way to assemble the set:

```
-  T  A  C  C  G  T  -  -
-  -  A  C  C  G  T  -  -
-  -  -  -  C  G  T  G  C
T  T  A  C  -  -  -  -  -
--------------------------
T  T  A  C  C  G  T  G  C
```

- Guidance: the length of the target and the fragment overlap
- Positioning the fragments we get a *layout*
- The sequence bellow the line is the *consensus sequence*

Sequence assembly - complications

- Errors
  - *Base call errors*: base substitutions, insertions, deletions
    - Rates usually vary from 1% to 5%, they tend to concentrate to the 3' end
    - Solution: The sequence below the line: consensus sequence – majority vote on columns

Example

- Fragments: TACCGT, ACCGT, CGTGC, TTAC coming from the sequence TTACCGTGC
- Lab data: TGCCGT, ACCGT, CGTGC, TTAC
  - substitution error in the second position of the first fragment

```
-  T  G  C  C  G  T  -  -
-  -  A  C  C  G  T  -  -
-  -  -  -  C  G  T  G  C
T  T  A  C  -  -  -  -  -
--------------------------
T  T  A  C  C  G  T  G  C
```

- Correct consensus because of majority vote

Sequence assembly - complications

- Errors
  - *Chimeric fragments*:
    - two disjoint fragments are joined together
    - fragments that have nothing to do with the target (contamination by the host of the clone)
    - Solution: recognize and eliminate them in a preprocessing stage

Example

- Fragments: TACCGT, ACCGT, CGTGC, TTAC coming from the sequence TTACCGTGC
- Lab data: TACGT, ACCGT, CGTGC, TTAC
  - deletion of the fourth base in the first fragment

```
-  T  A  C  -  G  T  -  -
-  -  A  C  C  G  T  -  -
-  -  -  -  C  G  T  G  C
T  T  A  C  -  -  -  -  -
--------------------------
T  T  A  C  C  G  T  G  C
```

- Correct consensus

Sequence assembly - complications

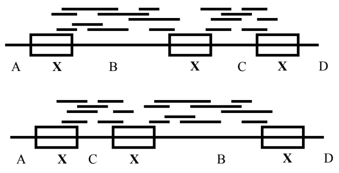
- Unknown orientation: we do not know from which strand of the target comes each fragment
  - We know however the 5'-3' sequence of each fragment

```
Input                          Answer
  5'-CACGT                            →     CACGT---------------
  5'-ACGT                             →     -ACGT---------------
  5'-ACTACG                           ←     --CGTAGT----------
  5'-GTACT                            ←     -------AGTAC------
  5'-ACTGA                            →     ------------ACTGA
  5'-CTGA                             →     --------------CTGA
                                          ______________________
                                            CACGTAGTACTGA
```

- This can be dealt with without considering all 2^n possible orientations for n strings (a modification of the basic algorithm)

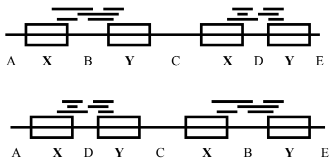
Sequence assembly - complications

- Repeated regions in the target sequence: regions that appear several times in the target
  - Direct repeats (below a repeat of the form XXX)
  - Inverted repeats



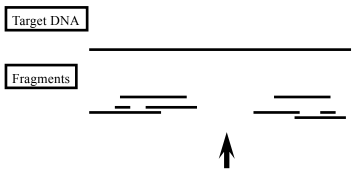
Sequence assembly - complications

- Repeated regions in the target sequence: regions that appear several times in the target
  - Direct repeats (below a repeat of the form XYXY)
  - Inverted repeats



Sequence assembly - complications

- Lack of coverage
  - Coverage at position j of the target is the number of fragments covering the position
  - Some parts of the target may be insufficiently covered: lack of information
  - Solutions: more samples of that area, direct sequencing (or *walking*) of that area



Shortest common superstring

- Formalize the shotgun sequencing: *shortest common superstring* (SCS)
- *SCS*: given a set F of strings, find a shortest possible string S such that for every u in F, u is a substring of S
- Example: ACT,CTA,AGT → ACTAGT
- NP-complete problem
  - There is no unique solution in general
  - We describe here how to find an approximation of the solution: Greedy algorithm

SCS – Greedy approximation

- Idea
  - Overlap detection
  - Substring layout
  - Deciding the consensus
- Outline of the algorithm
  - Select the two most overlapping strings in F
  - Replace them in F with their shortest superstring
  - Continue with the new set of strings
  - Stop when only one string remains in F

SCS – Greedy algorithm

- Example 1: **TCAGT, CATCAG, GTG, GCA**
  - The two most overlapping strings are CATCAG and TCAGT: replace them with their shortest common superstring: CATCAGT
    - The new set of strings: **CATCAGT, GTG, GCA**
  - Both GTG and GCA have 2 letter overlap with the first string: choose either of them, say GTG
    - The new set of strings: **CATCAGTG, GCA**
  - Final solution: GCATCAGTG
  - This is indeed the optimal solution – the shortest common superstring of the original set of strings

SCS – Greedy algorithm

- Example 2: **GCC, ATGC, TGCAT**
  - The two most overlapping strings are ATGC and TGCAT: replace them with their shortest common superstring: ATGCAT
  - The new set of strings: **GCC, ATGCAT**
  - They have no overlap
    - Reported common superstrings of those two strings: **GCCATGCAT** or **ATGCATGCC**
    - Length of the Greedy solution: **9**
  - The optimal solution to the original problem: **TGCATGCC**
    - Length of the optimal solution: **8**

The Greedy algorithm – optimality

- It can be proved that the Greedy algorithm gives a 2.75-approximation of the solution
  - The Greedy solution is a string of length at most 2.75 times the optimal length
  - There are other algorithms giving 2.5-approximations
- *Conjecture*: it is in fact a 2-approximation of the optimal solution
  - Noticed in practice that the Greedy solution is never more than twice the optimal length
  - Proof?

Example: sometimes longer solutions may be better

- One finds the optimal solution for an assembly problem

```
AGTATTGGCAATC------AATCGATG---------------
---------------------------------ATGCAACCT-------
------TTGGCAATCACT------------------CCTTTTGG
______________________________________________
AGTATTGGCAATCACT AATCGATGCAACCTTTTGG
```

  - Solution of length 36, generated by the Greedy algorithm
- The following longer solution is more acceptable than the previous shorter one because it gives improved linkage

```
AGTATTGGCAATC-----------CCTTTTGG-----------
--------------AATCGATG----------TTGGCAATCACT
----------------------ATGCAACCT------------------
______________________________________________
AGTATTGGCAATCGATGCAACCTTTTGGCAATCACT
```

  - Solution of length 37

Shortest common superstring – modeling errors

- The SCS model does not take into account the experimental errors
- Errors can be introduced in the model in many ways; an example:
- *Sequence reconstruction problem*: given a set of strings F and an error rate $0 \le e < 1$, find a shortest string S such that for all $f \in F$, there is a substring *a* of S that approximates either *f* or its inverse *f'* within error level *e*:

$$\min\{ d(a,f), d(a,f') \} \le e|f|,$$

where d is the *edit distance*.

- Edit distance d(a,b): the number of insertions, deletions, and letter changes needed to transform string a into string b

Sequence reconstruction problem - example

- Consider the fragments ATCCT, CGAGT, TCT
- Shortest common superstring (no errors): ATCCTCGTGTCT (length 12)
- If an error threshold of 0.2 is accepted then we get the solution ATCCTGTCT (length 9)

```
A  T  C  C  T
      C  G  A  G  T
                  T  C  T
__________________________
A  T  C  G  T  G  T  C  T
```

- Proof: for each string u, find a substring f of the solution such that $\min\{d(a,f),d(a,f')\} \le e|f|$,
  - u=ATCCT matches "well" with f=ATCGT: d(u,f)=1 and |f|=5: $1 \le 0.2 \times 5$
  - Similarly for u=CGAGT and f= CGTGT
  - u=TCT and f=TCT match "perfectly"
  - Note: TCT does not match "well enough" with TGT: the distance is 1 and |f|=3, but $1 > 0.2 \times 3$ (because of the small error threshold, small strings must match perfectly)

Another approach

SEQUENCING BY HYBRIDIZATION

Sequencing by hybridization (SBH)

- Hybridization: alternative approach to sequencing
- *Idea*:
  - As in shotgun, obtain a high number of overlapping clones from the target DNA sequence
  - Having a set of probes, determine which ones hybridize to the clone and based on this info, sequence the clone
  - Q: Which probes should one consider?
- Tool: DNA arrays (a.k.a. DNA chips)
  - Recall. DNA array: thousands of short DNA fragments attached to a surface (e.g., all possible DNA fragments of length 10: 1Mb array)

Sequencing by hybridization

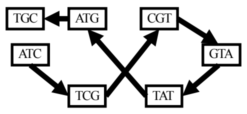
- Build the DNA array, e.g., attach all possible probes of length *l* (say *l*=10) to a surface
- Apply a solution containing the unknown DNA fragment
  - The DNA fragment has been (e.g., fluorescently) labeled
- The DNA fragment hybridizes with those probes that are complementary to some of its substrings of length *l*
- Read the results
- Apply a combinatorial algorithm to reconstruct the sequence of the DNA fragment from the set of its substrings of length *l*

SBH – algorithmic solutions

- The computational problem
  - One idea: this is a particular case of the shortest common superstring problem with all strings having the same length
    - Solve the SCS, e.g., find a Greedy approximation
  - Another idea: this is the Eulerian path problem
    - Knowing already the first *n-1* nucleotides of the sequence $a_1a_2…a_{n-1}$, the *n*-th nucleotide is such that the probe $a_2a_3…a_n$ (if there was such a probe) has hybridized to the clone
    - This leads to a linear algorithm (if we do not consider the errors)

SBH – algorithmic solutions

- SBH and the Eulerian path problem:
  - The DNA array contains all sequences of length l
  - Build a graph with nodes all sequences of length l-1
  - For each sequence u with positive answer build an edge from node *pref*(u,l-1) to node *suff*(u,l-1). In other words, if u= $a_1a_2…a_l$, then draw an edge from node $a_1a_2…a_{l-1}$ to node $a_2…a_{l-1}a_l$ and label the edge with u
  - Example: l=4 yields the fragments ATCG, ATGC, CGTA, GTAT, TATG, TCGT
    - Build the graph below
    - Q: How do we find the solution?
    - Solution: ATCGTATGC

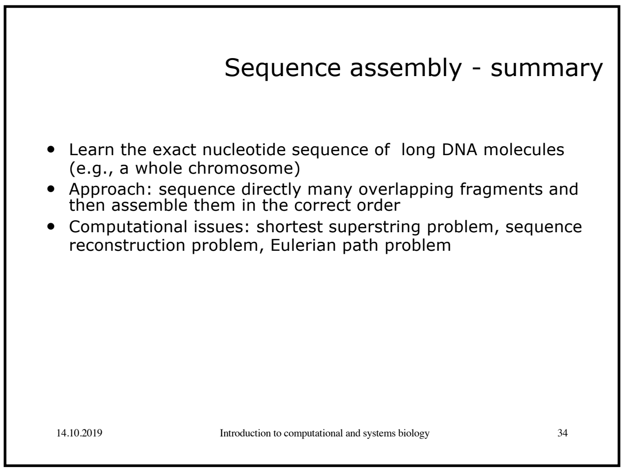


Sequencing by hybridization – a solution

- SBH and the Eulerian path problem
  - SBH reduces to finding an *Eulerian path* in this graph
  - *Eulerian path*: a path that visits *all edges* of the graph
  - Eulerian path problem: for a given directed multigraph, decide if an Eulerian path exists (and find it if so)
  - Solution for the Eulerian path problem: linear algorithm (the naive one!)
    - For directed graphs: for any node, there is an equal number of arrows entering the node and arrows exiting the node
    - For undirected graphs: all nodes must have an even number of edges
  - Compare with *Hamiltonian path*: a path that visits *all nodes* of the graph
  - Finding a Hamiltonian path: NP-complete problem

SBH - considerations

- If the length *l* of the probes is too small, then the sequence is impossible to reconstruct (too many possible solutions)
- If the length *l* is too large, it is technically impossible to build "complete" DNA arrays with all possible strings of length *l*
  - Idea here: design other types of DNA arrays, different than the ones with all strings of length *l*
- If the graph is not Eulerian, then there are errors in the data
  - Eliminate some of the arrows so that the graph becomes Eulerian



Sequence assembly - summary

- Learn the exact nucleotide sequence of long DNA molecules (e.g., a whole chromosome)
- Approach: sequence directly many overlapping fragments and then assemble them in the correct order
- Computational issues: shortest superstring problem, sequence reconstruction problem, Eulerian path problem