Algorithms in Bioinformatics

More greedy algs:
Assembly

Recap

- HW due next Tuesday

Notes:

Problem 16:

Give an example

Not +2 (shift) to everything
{0, 7, 10} {0, 3, 10}

Problem 3:

Input: $D[1..n]$, $l1$, $l2$, g, d

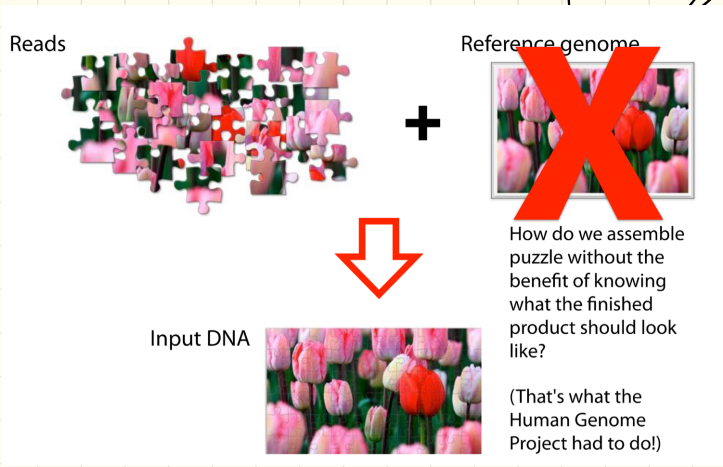
Psendocode:

think of Python

Today: more greed

New problem:

Assembly (+ Shortest Common Superseqence)



Why? "Shotgun" sequences copies DNA:

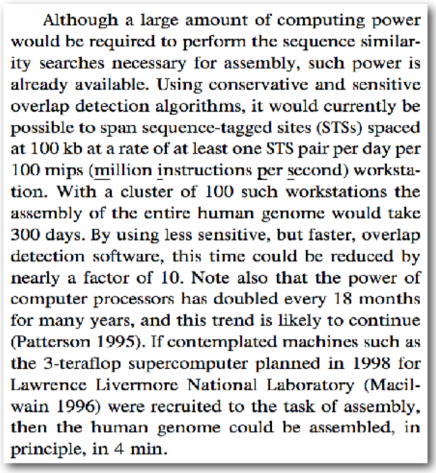
Input: GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT

Copy: GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT

And then fragments it:

Fragment: GGCGTCTA TATCTCGG CTCTAGGCCCTC ATTTTTT
GGC GTCTATAT CTCGGCTCTAGGCCCTCA TTTTTT
GGCGTC TATATCT CGGCTCTAGGCCCT CATTTTTT
GGCGTCTAT ATCTCGGCTCTAG GCCCTCA TTTTTT

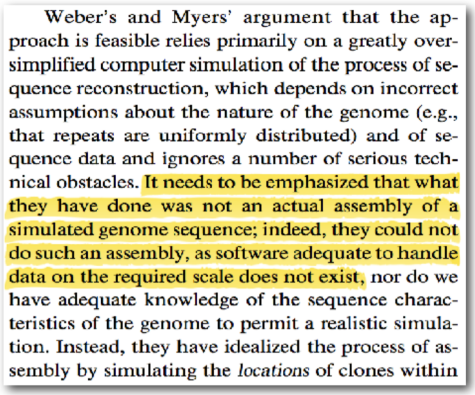
The debate:



Although a large amount of computing power would be required to perform the sequence similarity searches necessary for assembly, such power is already available. Using conservative and sensitive overlap detection algorithms, it would currently be possible to span sequence-tagged sites (STSs) spaced at 100 kb at a rate of at least one STS pair per day per 100 mips (million instructions per second) workstation. With a cluster of 100 such workstations the assembly of the entire human genome would take 300 days. By using less sensitive, but faster, overlap detection software, this time could be reduced by nearly a factor of 10. Note also that the power of computer processors has doubled every 18 months for many years, and this trend is likely to continue (Patterson 1995). If contemplated machines such as the 3-teraflop supercomputer planned in 1998 for Lawrence Livermore National Laboratory (Macilwain 1996) were recruited to the task of assembly, then the human genome could be assembled, in principle, in 4 min.

Weber, James L., and Eugene W. Myers. "Human whole-genome shotgun sequencing." *Genome Research* 7.5 (1997): 401-409.

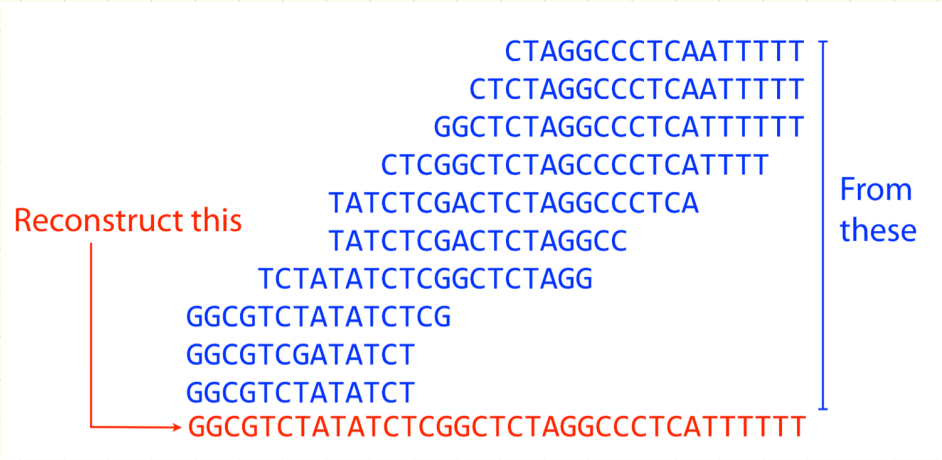
(Even more...)



Weber's and Myers' argument that the approach is feasible relies primarily on a greatly oversimplified computer simulation of the process of sequence reconstruction, which depends on incorrect assumptions about the nature of the genome (e.g., that repeats are uniformly distributed) and of sequence data and ignores a number of serious technical obstacles. It needs to be emphasized that what they have done was not an actual assembly of a simulated genome sequence; indeed, they could not do such an assembly, as software adequate to handle data on the required scale does not exist, nor do we have adequate knowledge of the sequence characteristics of the genome to permit a realistic simulation. Instead, they have idealized the process of assembly by simulating the *locations* of clones within

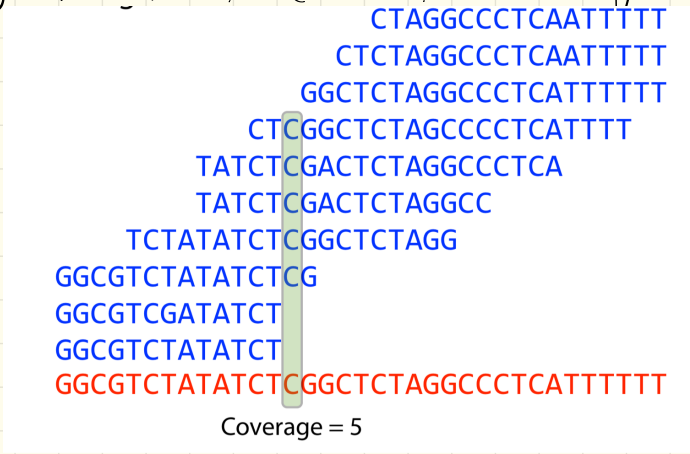
Green, Philip. "Against a whole-genome shotgun." *Genome Research* 7.5 (1997): 410-417.

The problem:

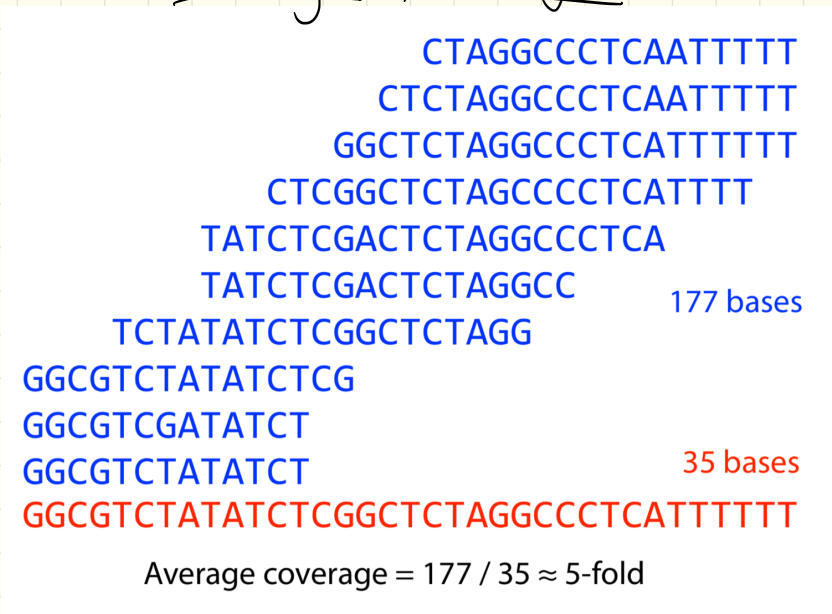


Well, it's actually worse:

Key term: Coverage, or # of reads that contain a position



Usually, we mean really mean average coverage:



Basic principle:

- if a prefix (mostly) matches a suffix, it is likely they came from overlapping reads

Reads:

```
TCTATATCTCGGCTCTAGG
|||||||  |||||||
 TATCTCGACTCTAGGCC
```

Possible correct answer

```
TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
 TATCTCGACTCTAGGCC
```

Reasons for differences:

(1) Sequencing error

```
TCTATATCTCGGCTCTAGG
|||||||  |||||||
 TATCTCGACTCTAGGCC
```

use d_H

(2) Differences between inherited copies of a chromosome (we are diploids)

Assumption:

We'll assume we can (efficiently) tell if one string is a suffix/prefix of another.

(Come back next week.)

(also really if nearly a prefix or suffix)

Formal Problem: SCS

Given set of strings S, find $SCS(S)$: shortest string containing the strings in S as substrings

S: BAA AAB BBA ABA ABB BBB AAA BAB

Concat(S): BAAAABBBAABAABBBBBAAABAB

24

$SCS(S)$: AAABBBABAA

10

Note: Without shortest - easy!

A first try: be really greedy!

Pick an ordering & build superstring:

order 1: AAA AAB ABA ABB BAA BAB BBA BBB

AAABABBAABABBABBB

17

Try again:

order 2: AAA AAB ABA BAB ABB BBB BAA BBA

AAABABBBAABBA ← superstring 2

13

Problem: order matters!

try them all

↳ $n!$

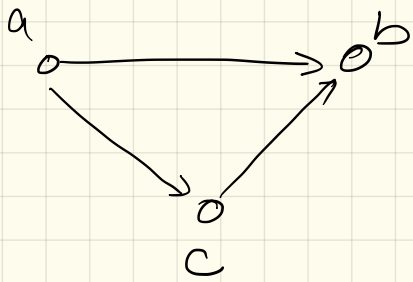
Graph theory:

We'll build a directed graph:

$G = (V, E)$ ← ordered pair of sets

V = set of vertices

E = ordered pair



$V = \{a, b, c\}$

$E = \{(a,b), (cb), (ac)\}$

Dfn: source, sink

Next: SCS build a graph

Each node is a read

$V = \{$ strings $\}$

CTCGGCTCTAGCCCCTCATTTT

V ① V

Draw edge A -> B when suffix of A overlaps prefix of B

CTCGGCTCTAGCCCCTCATTTT

GGCTCTAGGCCCTCATTTTTT

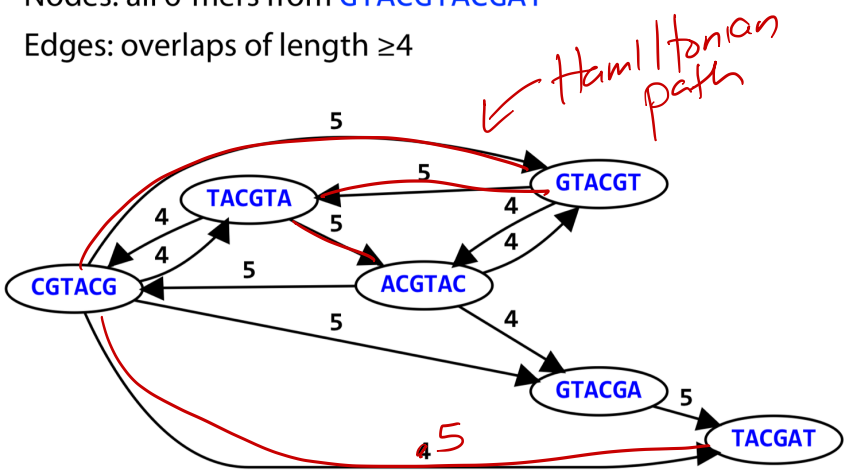
V_1: list of adj vertices

weight edge by length of overlap

Example

Nodes: all 6-mers from GTACGTACGAT

Edges: overlaps of length ≥4



Common superstring:

GTACGTACGAT

The "best path" in this graph is the one we want:

- visit every node
- maximize overlap.

Unfortunately, this is the traveling salesman problem — so NP-Hard!

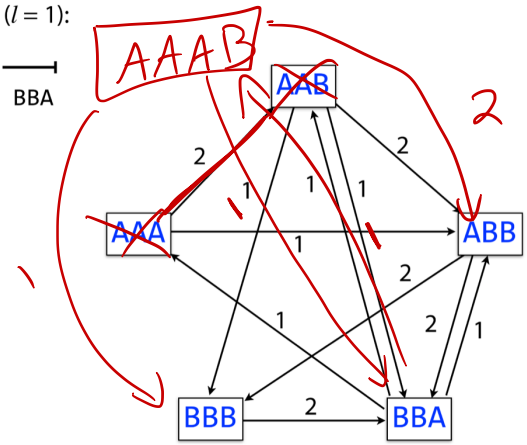
Instead:

Greedy-SCS: in each round, merge pair of strings with maximal overlap. Stop when there's 1 string left. l = minimum overlap.

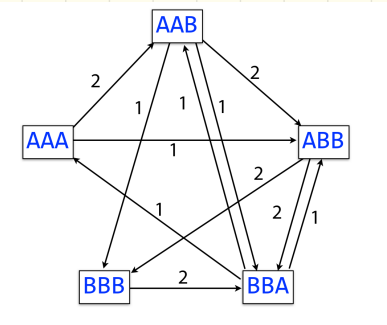
Algorithm in action (l = 1):

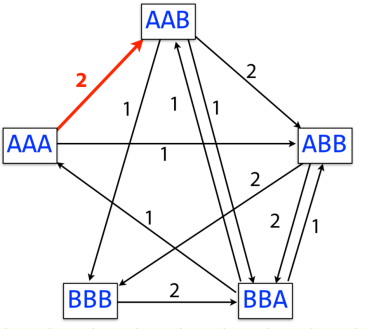
Input strings

AAA AAB ABB BBB BBA

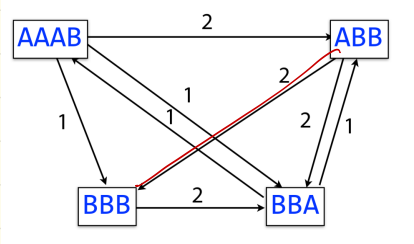


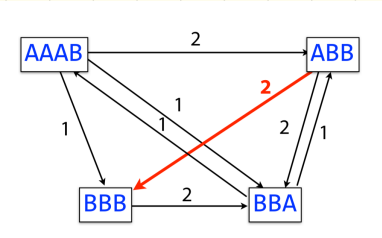
In action:



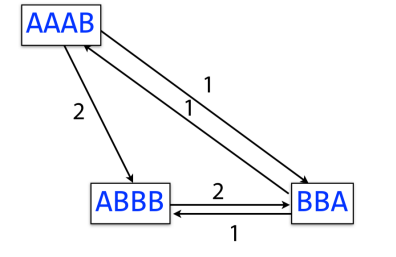


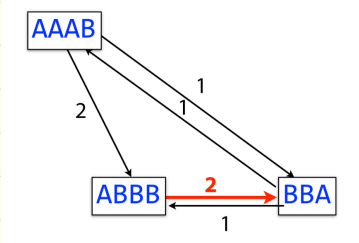
collapse



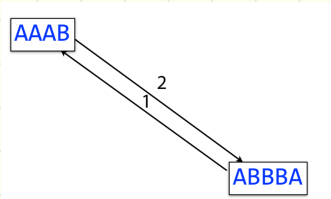


collapse





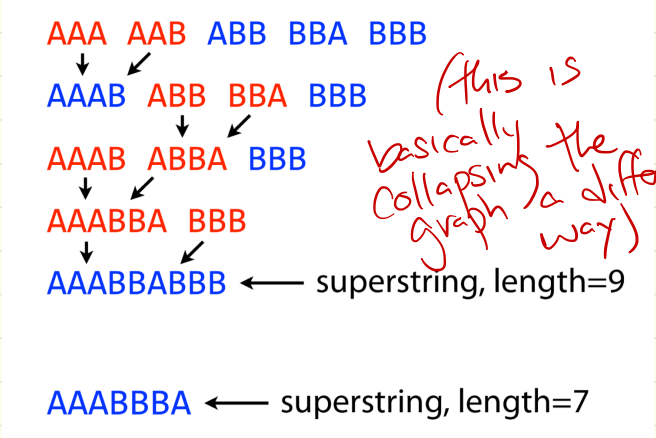
collapse



final:

AAABBBA ← superstring, length=7

Problem: Greed (usually) doesn't win!



Approximation

However, this does give a ~2.5-approximation

length of greedy ≤ ~2.5 (length of OPT)

In particular, known issue

Greedy-SCS assembling all substrings of length 6 from:
a_long_long_long_time. $l = 3$.

6 characters

```
ng_lon _long_ a_long long_l ong_ti ong_lo long_t g_long g_time ng_tim
ng_time ng_lon _long_ a_long long_l ong_ti ong_lo long_t g_long
ng_time g_long_ ng_lon a_long long_l ong_ti ong_lo long_t
ng_time long_ti g_long_ ng_lon a_long long_l ong_lo
ng_time ong_lon long_ti g_long_ a_long long_l
ong_lon long_time g_long_ a_long long_l
long_lon long_time g_long_ a_long
long_lon g_long_time a_long
long_long_time a_long
a_long_long_time
```

Foiled by repeat!

To fix: longer reads!

length 8

```
long_lon ng_long_ _long_lo g_long_t ong_long g_long_l ong_time a_long_l _long_ti long_tim
long_time long_lon ng_long_ _long_lo g_long_t ong_long g_long_l a_long_l _long_ti
_long_time long_lon ng_long_ _long_lo g_long_t ong_long g_long_l a_long_l
_long_time a_long_lo long_lon ng_long_ g_long_t ong_long g_long_l
_long_time ong_long_ a_long_lo long_lon g_long_t g_long_l
g_long_time ong_long_ a_long_lo long_lon g_long_l
g_long_time ong_long_ a_long_lon g_long_l
g_long_time ong_long_l a_long_lon
g_long_time a_long_long_l
a_long_long_long_time
a_long_long_long_time
```

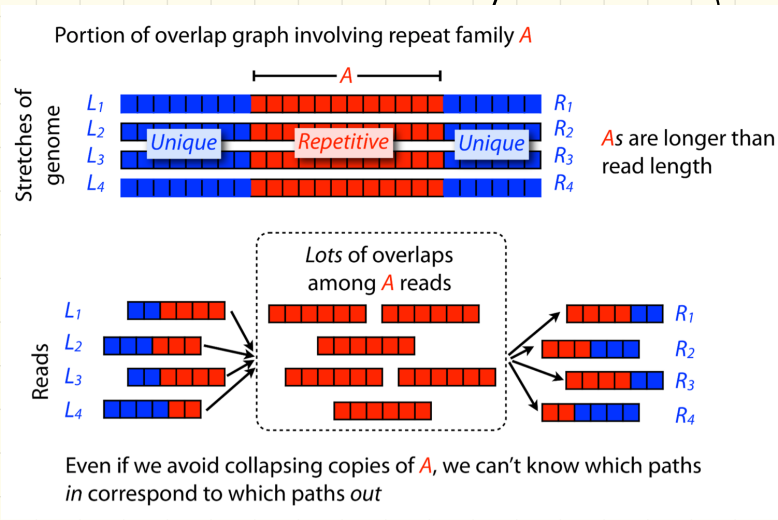
Repeats

These often foil assembly - certainly SCS, b/c of "shortest"

Need longer reads

↳ catches the repeat

But: algorithms that don't pay attention to repeats will always collapse them



Problem:

Human genome is 50% repetition!

So: SCS is flawed.

- Not tractible
- Collapses repeats

So: More to come!

(This was more about greedy strategies.)