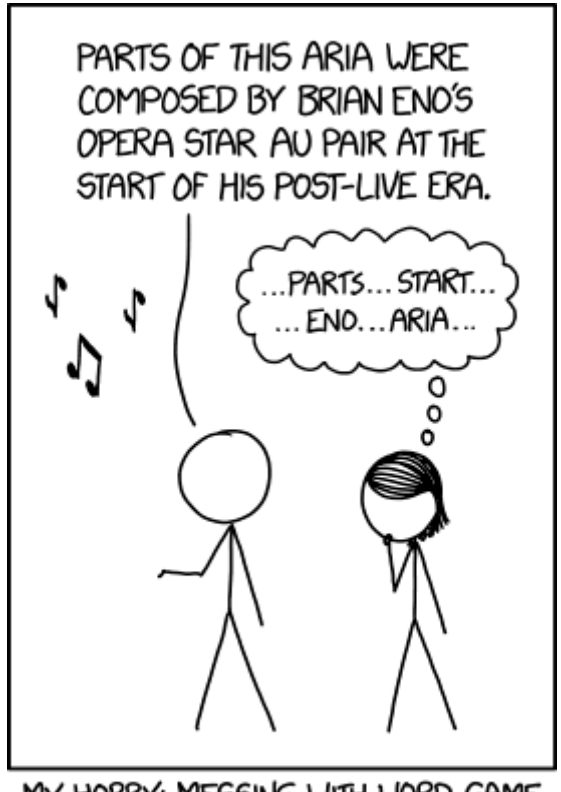
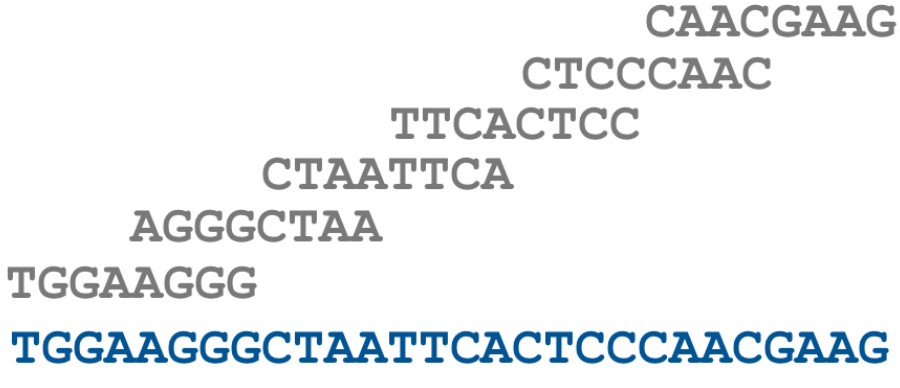
MIMM4750G
de novo assembly



What is *de novo* assembly?

- Combine short reads that seem to overlap so they form a longer sequence.
- A *contig* is a contiguous (uninterrupted) run of nucleotides that is formed from the assembly of short reads.



Pros and cons

- Short-read *mapping* is generally faster and easier than *de novo* assembly, but needs a good reference.
- Mapping better suited for variant detection.
- *de novo* assembly is better suited for discovering new genomes, where no suitable reference exists.
- Hybrid methods use both *de novo* assembly and mapping to re-assemble local contigs.

de novo assembly for pathogens

Percentage of unique reads as a function of read length for (a) λ-phage and (b) E.coli K12.

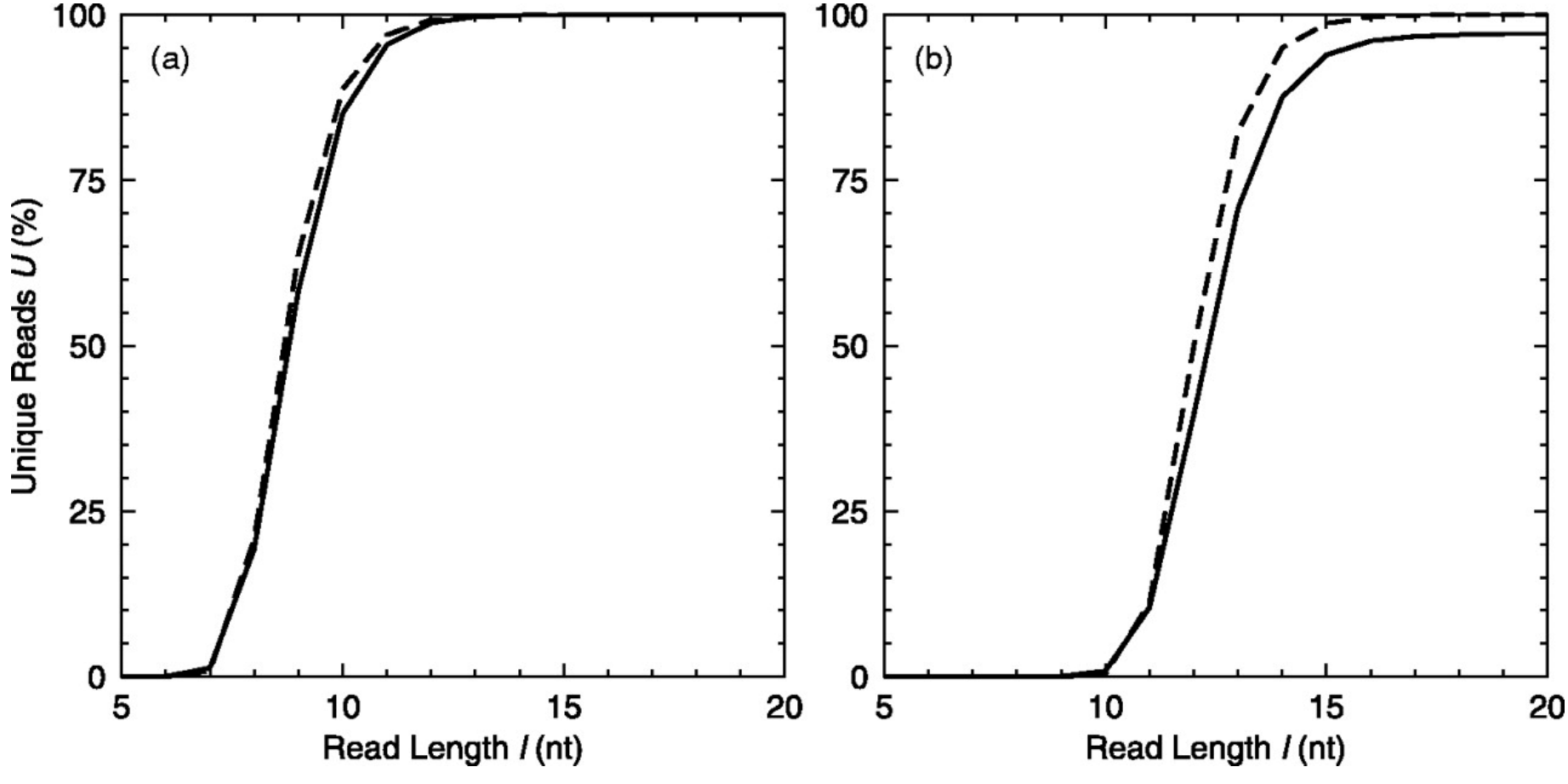


Figure from N Whiteford *et al.* (2005) Nucleic Acids Research 33: e171.

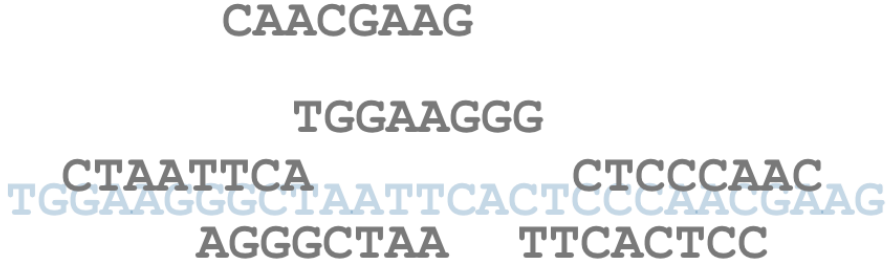
Finding overlaps

- This has been compared to opening a box of ten million puzzle pieces and finding which pieces match together...
- except that the edges always line up, and some of the pieces are repeated many times, and other pieces belong to different sets...

Image source: http://homeli.co.uk/1000-colours-rainbow-cmyk-gamut-jigsaw-puzzle-by-clemens-habicht/

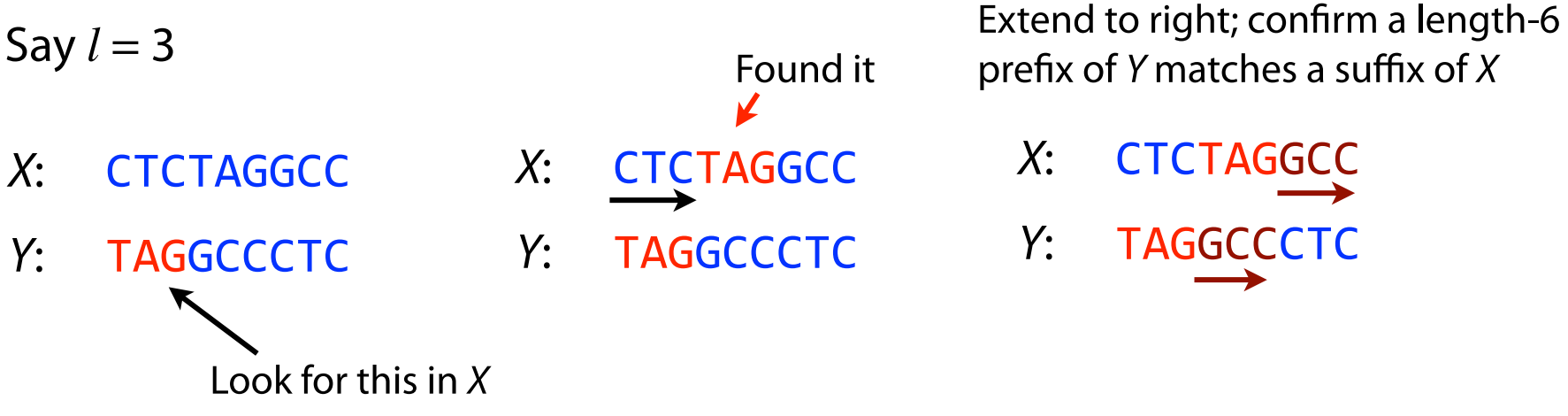
Finding overlaps

- This requires that we compare every pair of pieces!
- Quadratic complexity ($O(N^2)$) with the number of reads, which is already a huge number.
- Made even more difficult if we want to tolerate *inexact* matches!



Substrings

- Immediately looking for the largest matching sub-string between two strings is time-consuming.
- Instead, we can check if a short *prefix* of one string occurs somewhere in the second string.
- Requiring the suffix to match



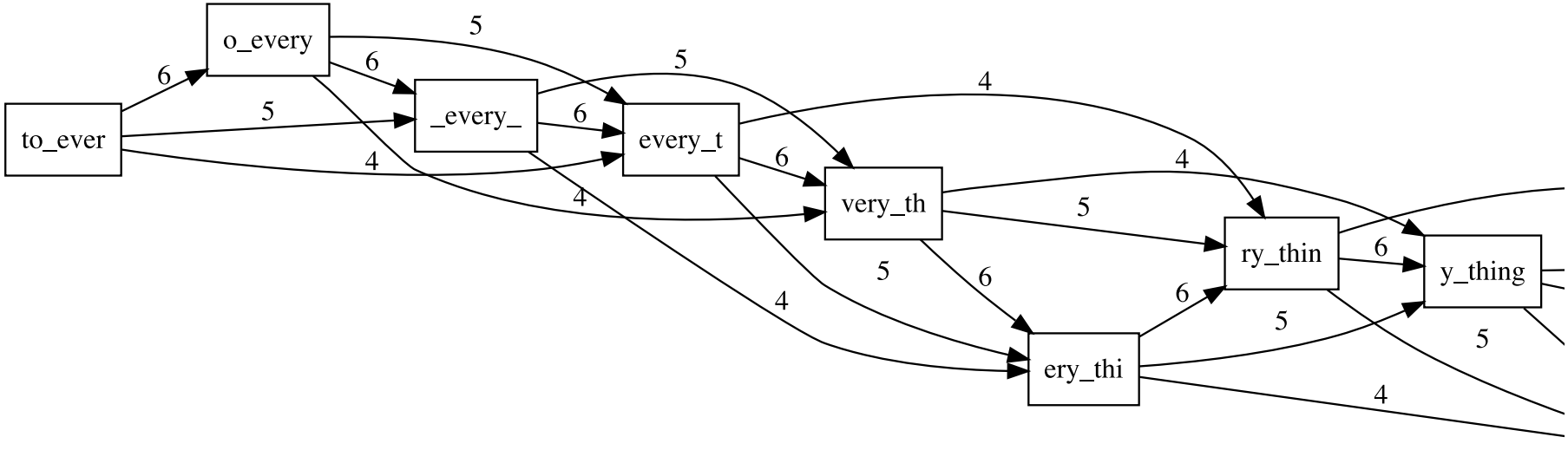
Overlap graph

- The presence/absence of an overlap between any two reads can be represented as a *graph* (or network)
- To illustrate, Ben Langmead (developer of Bowtie) generates an overlap graph of the string:

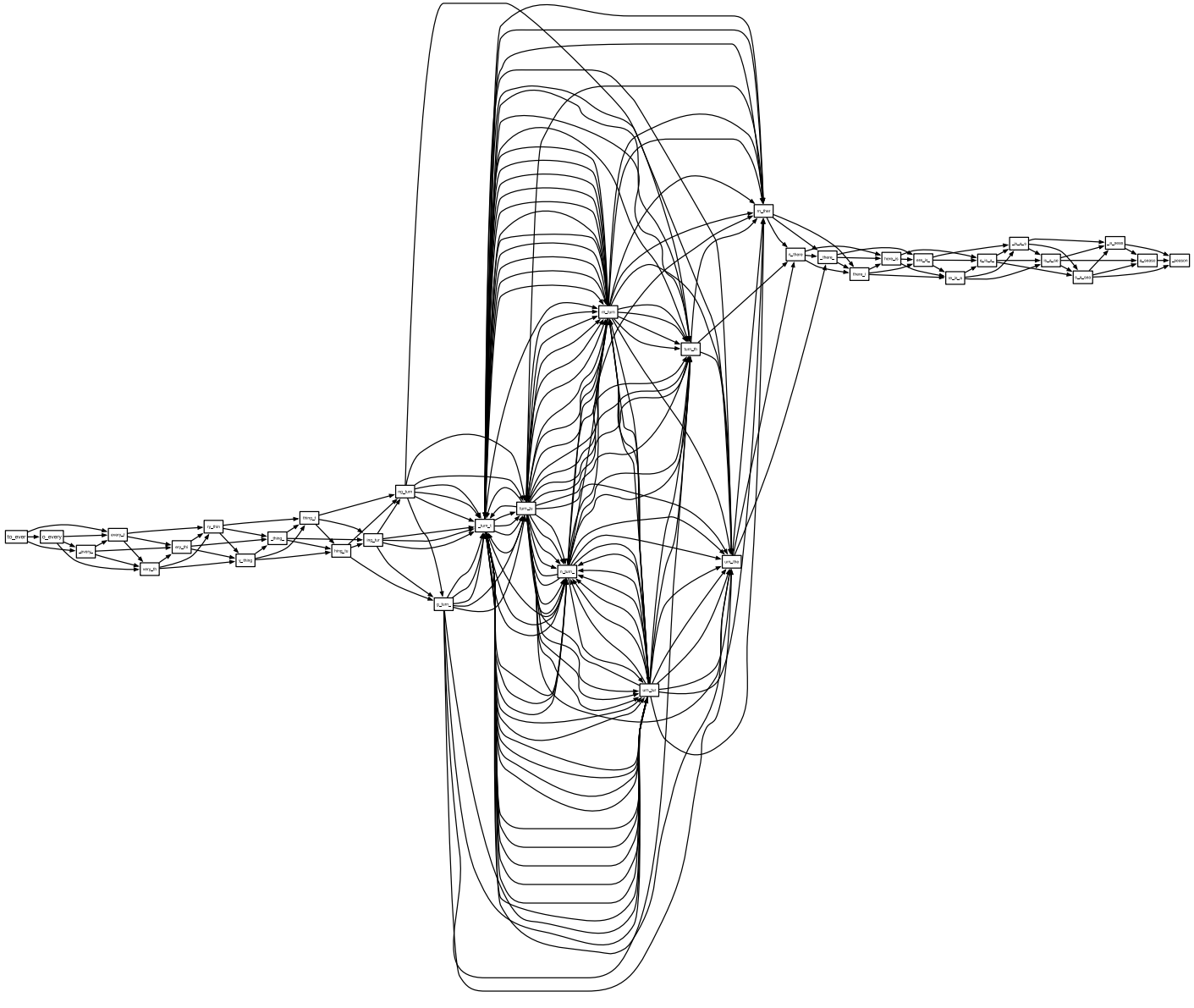
```
to_every_thing_turn_turn_turn_there_is_a_season
```

- This example sets the prefix search length to $l = 3$.
- For example, `o_every` has a 3-prefix `o_e` that appears at position 1 of `to_ever`. The longest match is length 6: `o_ever`

The start of the graph is not too difficult to read:

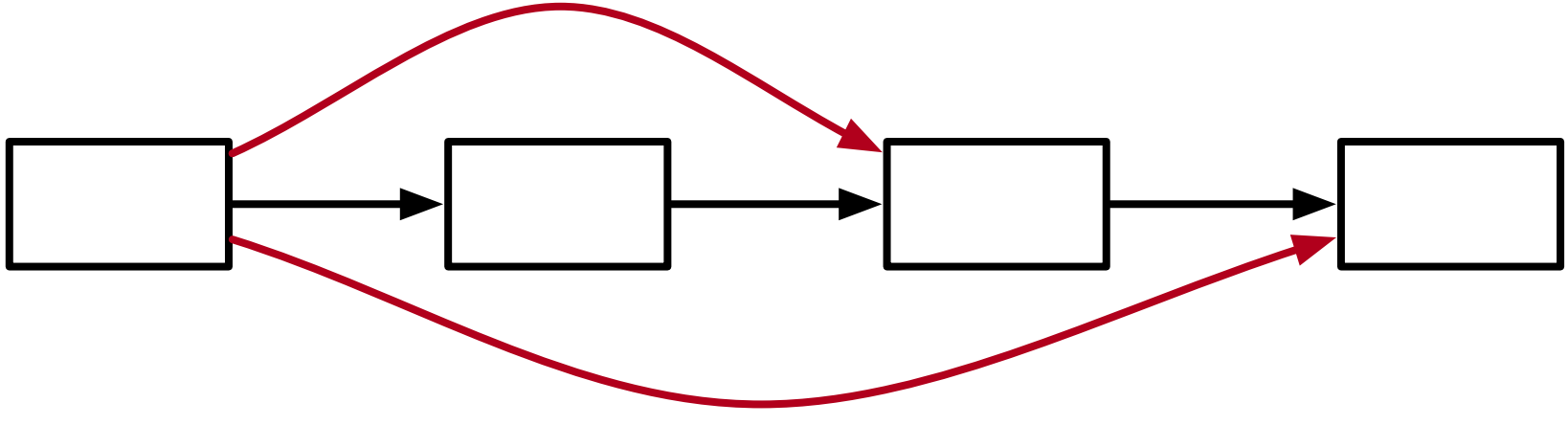


but the whole thing is pretty gnarly!

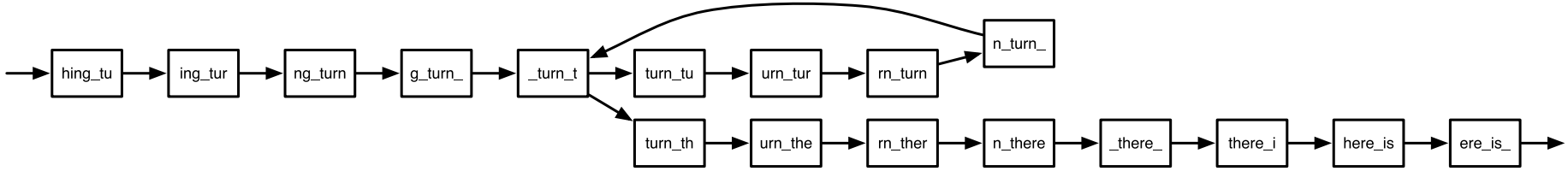


Simplifying the graph

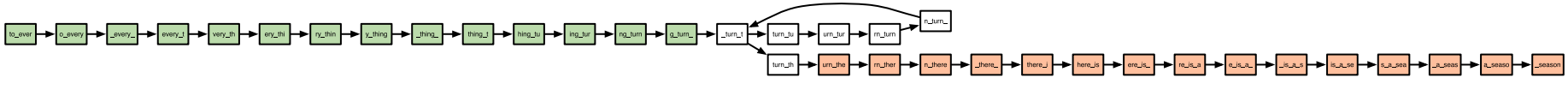
- We can remove *transitive* edges.
- A relation R is *transitive* if $(a\,R\,b) \cap (b\,R\,c) \implies a\,R\,c$ for all possible values of a, b, c.



This simplifies things a LOT!



- but the repeat could also go on forever ("turns, turns, turns, turns, turns, turns...")
- The intact chains are turned into contigs:



de Bruijn graphs

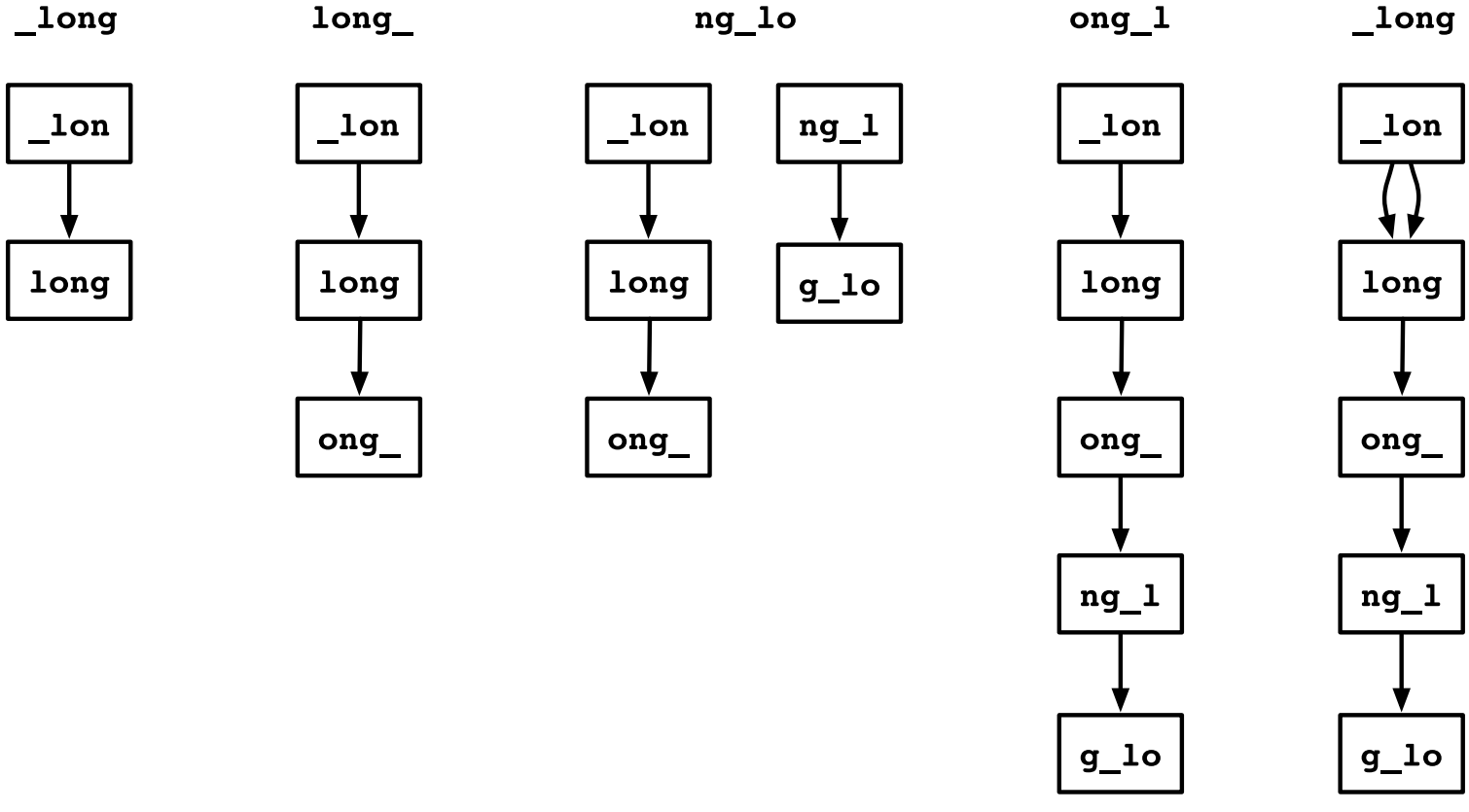
- Assume every possible *k*-mer in the genome is sequenced exactly once
- A de Bruijn graph connects "words" that differ by a single letter.
- For each *k*-mer, split it into two (*k*-1)-mers.
- For example, `puppy` becomes `pupp` and `uppy`.

Langmead's example

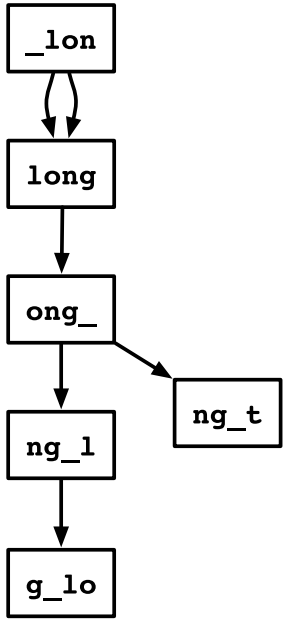
- Get all 5-mers from the string `a_long_long_long_time`.
- Scramble the 5-mers so they are incorporated in random order.
- Some Python:

```
>>> s = "a_long_long_long_time"
>>> kmers = [s[i:(i+5)] for i in range(0, len(s)-5)]
>>> kmers
['a_lon', '_long', 'long_', 'ong_l', 'ng_lo', 'g_lon', '_long', 'long_',
>>> import random
>>> random.shuffle(kmers)
>>> kmers
['_long', 'long_', 'ng_lo', 'ong_l', '_long', 'ong_t', 'ng_lo', '_long',
```

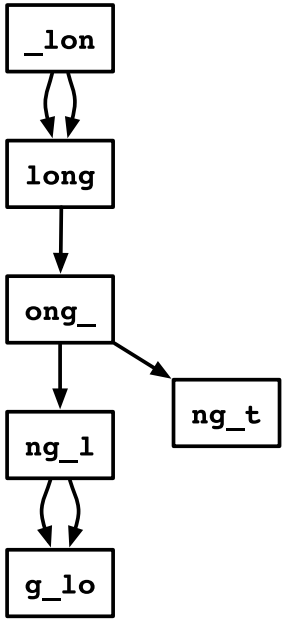
- Each pair of k-1 words are automatically connected by an edge
- Draw an additional edge whenever we encounter the same pair.



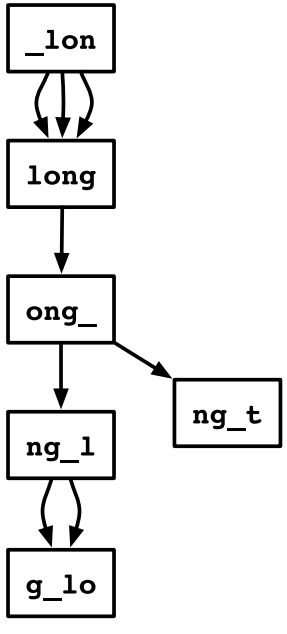
ong_t



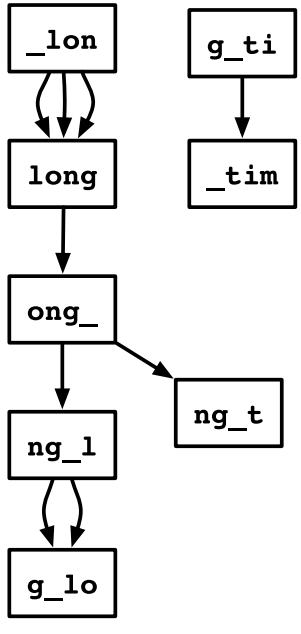
ng_lo



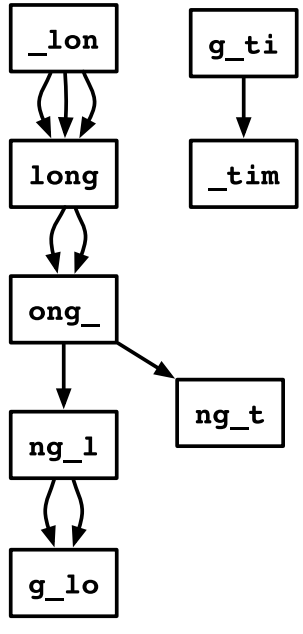
_long

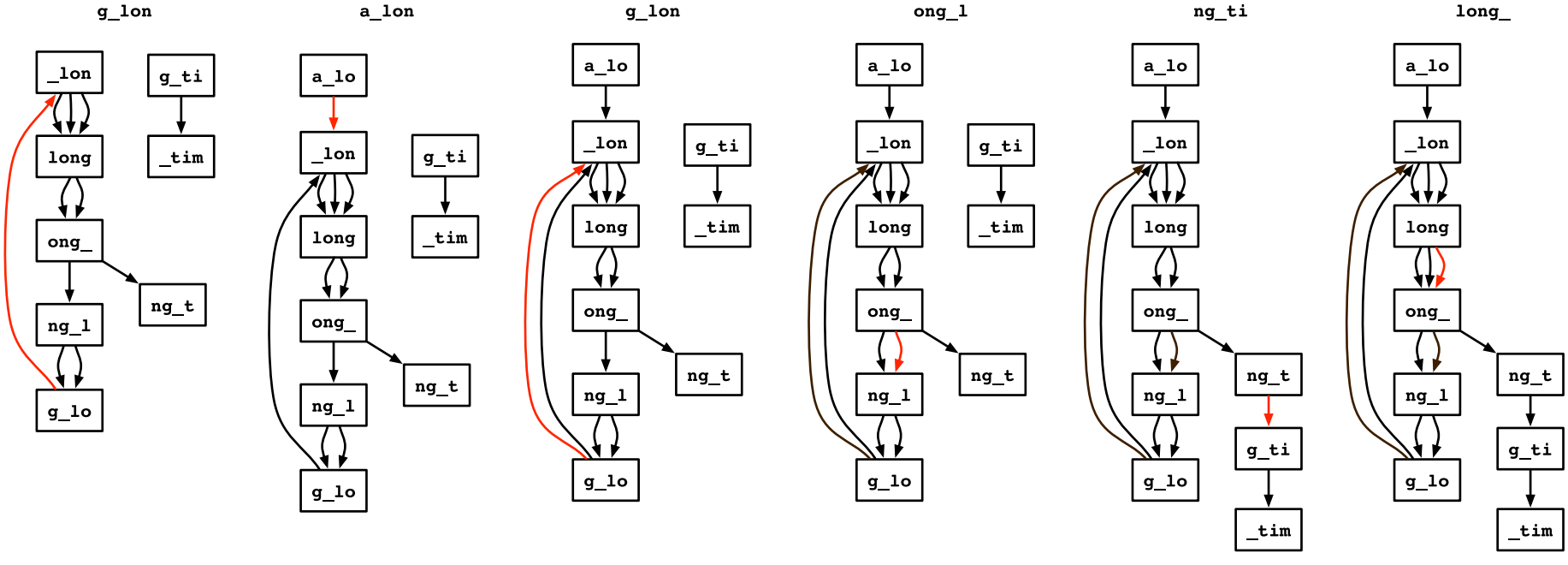


g_tim



long_




g_lon
_lon
g_ti
long
_tim
ong_
ng_t
ng_l
g_lo
a_lon
a_lo
_lon
g_ti
long
_tim
ong_
ng_t
ng_l
g_lo
g_lon
a_lo
_lon
g_ti
long
_tim
ong_
ng_t
ng_l
g_lo
ong_l
a_lo
_lon
g_ti
long
_tim
ong_
ng_t
ng_l
g_lo
ng_ti
a_lo
_lon
long
ong_
ng_t
ng_l
g_lo
g_ti
_tim
long_
a_lo
_lon
long
ong_
ng_t
ng_l
g_lo
g_ti
_tim

Pros and cons of de Bruijn graphs

- If we start at the node without incoming edges, and follow every edge *exactly once*, we can reconstruct the original string!
- No mucking about with removing transitive edges.
- Repeats are still a problem.
- We were assuming that every k-mer appears exactly once - extra copies break the "walk" property of deBruijn graphs.

Software

- SPAdes - uses de Bruijn graphs, designed for bacterial genomes
- Velvet - uses de Bruijn graphs, popular, Linux only.
- ABySS - Canadian! uses a "Bloom filter" (advancement on de Bruijn graphs)
- Ray - Also Canadian (Quebec)! yes, still de Bruijn graphs.

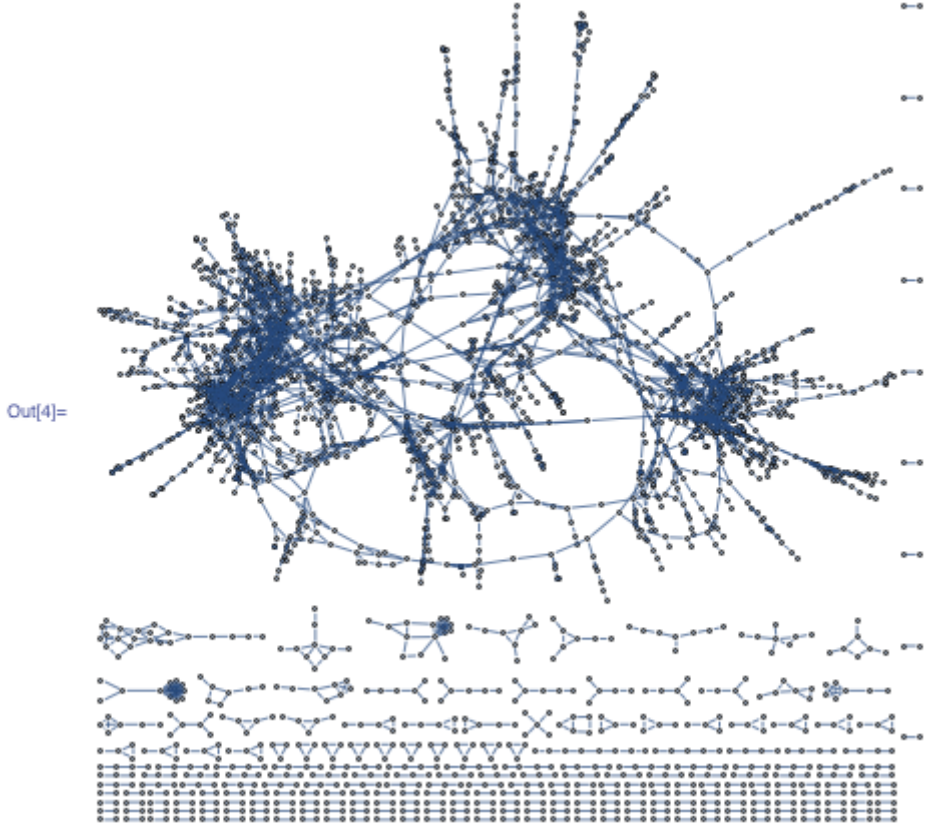


Image from https://blog.wolfram.com/2012/01/11/the-longest-word-ladder-puzzle-ever/

Further readings

- De novo assembly of short sequence reads by K Paszkiewicz and DJ Studholme (2010), Briefings in Bioinformatics 11
- Ben Langmead's lecture materials