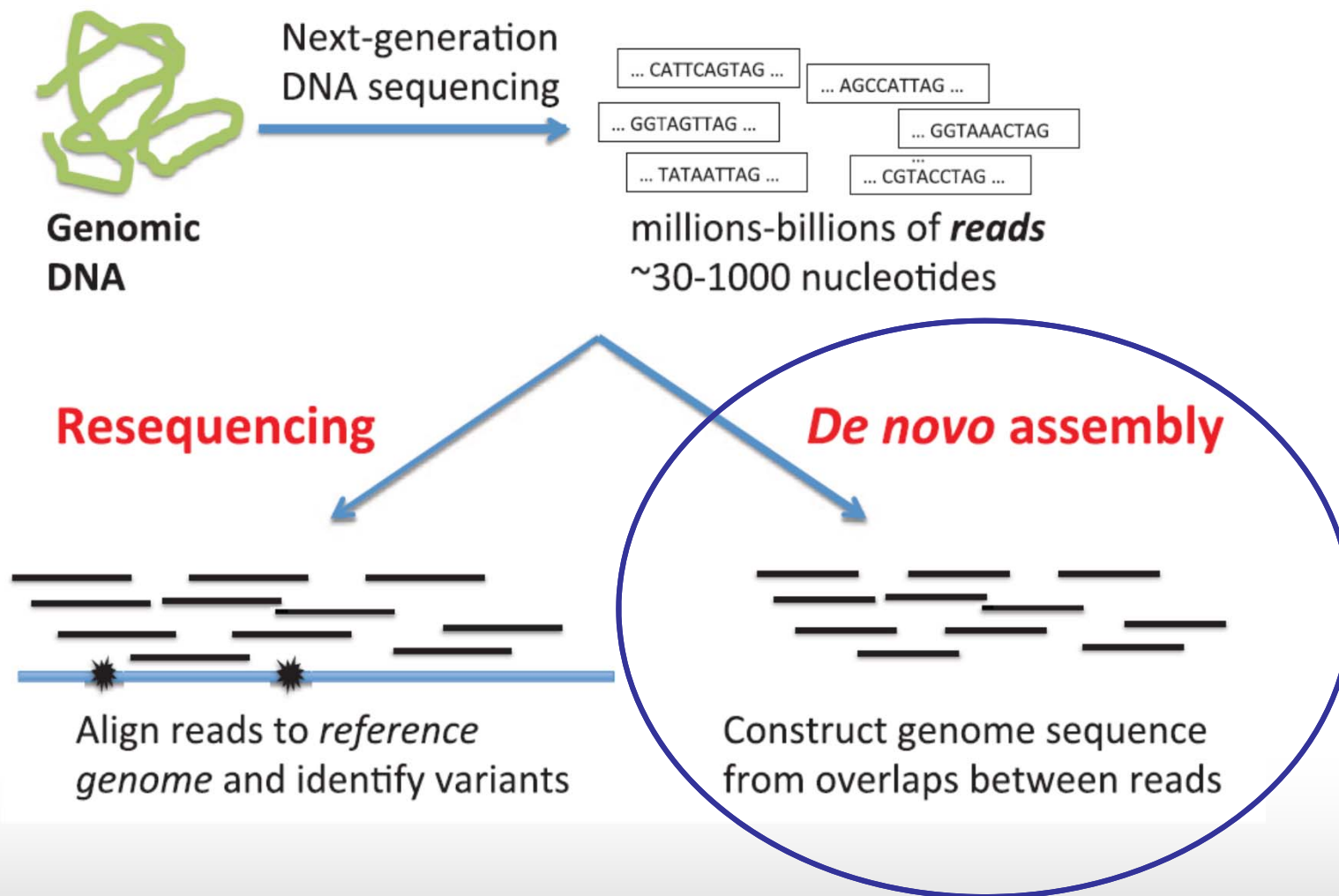




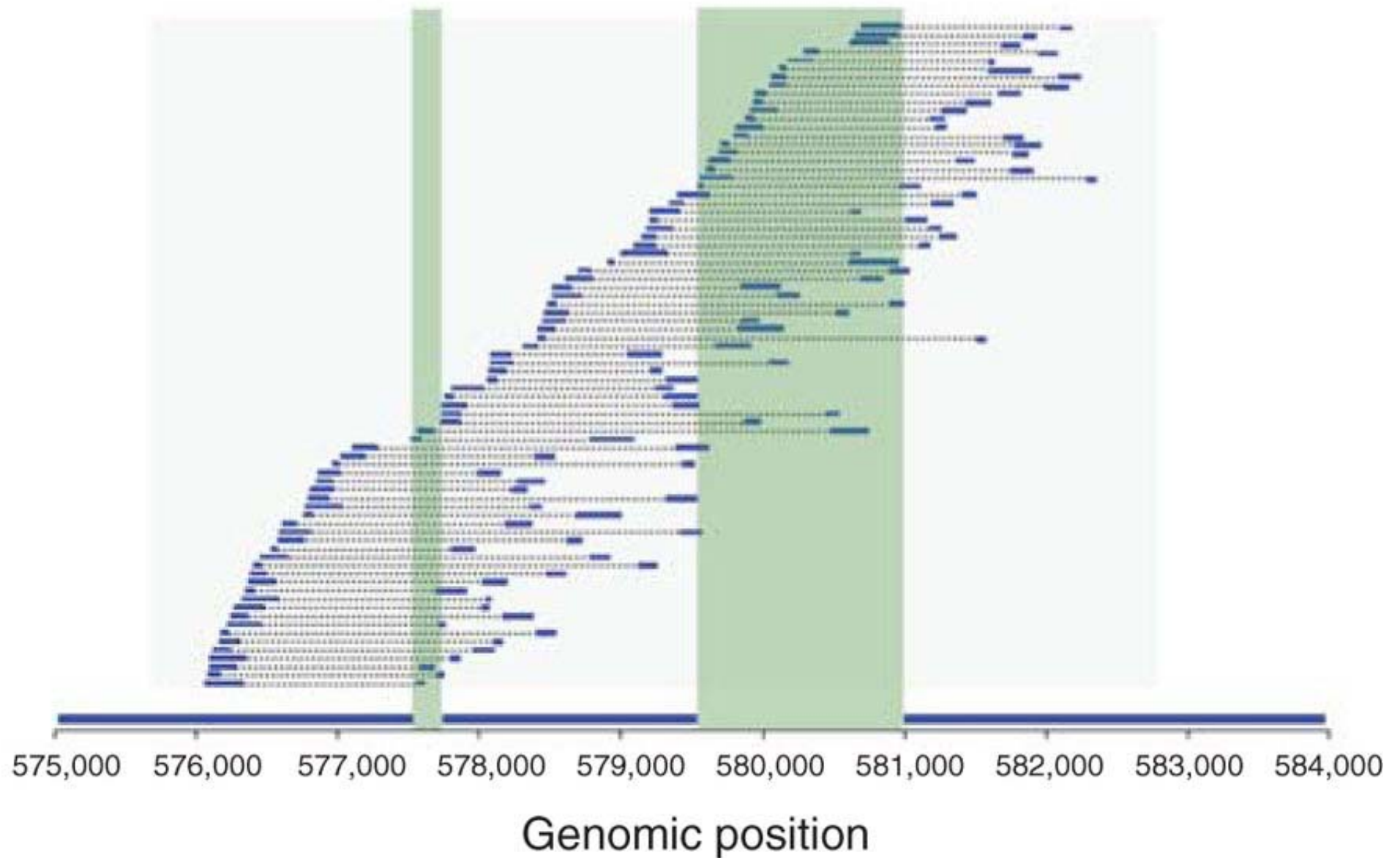
生物信息学

第十章 基因组组装

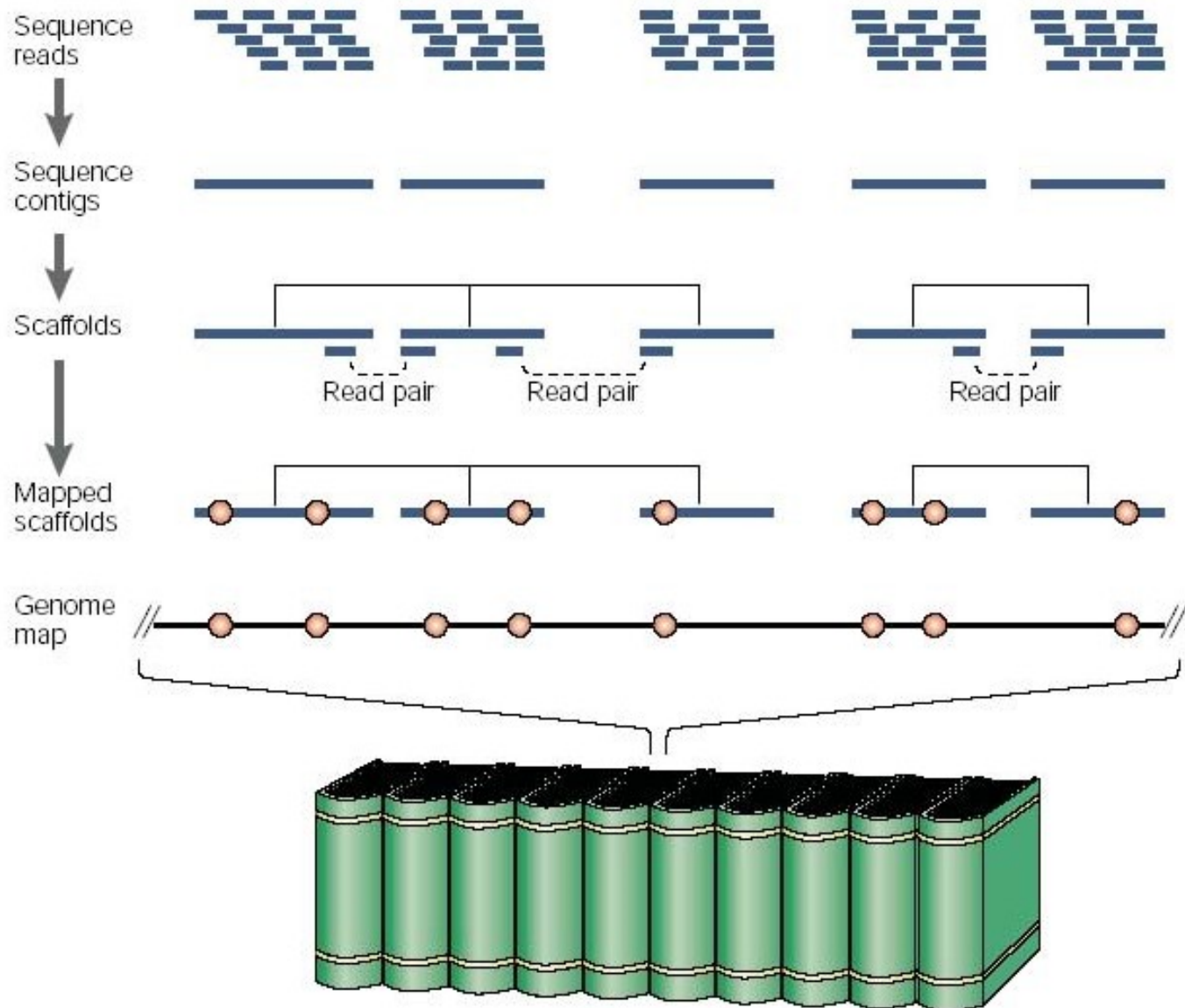
全基因组测序



读段如何组装成基因组？



从头测序的基因组组装流程



Contig:
重叠群

Scaffold:
框架

组装



- ❑ 全基因组测序：DNA复制和片段化
- ❑ “鸟枪法”：全基因组**随机**片段化

输入：GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT

复制：GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT

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

组装



- 测序：如果基因组上每个位置都有许多读段覆盖
- 问题：读段在基因组上哪个位置？

重建

CTAGGCCCTCAATTTTT
CTCTAGGCCCTCAATTTTT
GGCTCTAGGCCCTCATTTTTT
CTCGGCTCTAGCCCCTCATT
TATCTCGACTCTAGGCCCTCA
TATCTCGACTCTAGGCC
TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCGATATCT GGCGTCTATATCT

测序读段

GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT

组装



□ 覆盖率 (*Coverage*, *C*):

- ✿ 一般指平均覆盖率
- ✿ 基因组上每一个位置平均覆盖读段的条数

```
CTAGGCCCTCAATTTT
CTCTAGGCCCTCAATTTT
GGCTCTAGGCCCTCATTTTT
CTCGGCTCTAGCCCCTCATTTT
TATCTCGACTCTAGGCCCTCA
TATCTCGACTCTAGGCC
TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCGATATCT
GGCGTCTATATCT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
```

177个碱基

35个碱基

$$\text{平均覆盖率} = 177 / 35 \approx 7x$$

组装



□ 也可以表示特定位置的覆盖率

```
CTAGGCCCTCAATTTT
CTCTAGGCCCTCAATTTT
GGCTCTAGGCCCTCATTTT
CTCGGCTCTAGCCCCTCATTTT
TATCTCGACTCTAGGCCCTCA
TATCTCGACTCTAGGCC
TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCG
GGCGTCGATATCT
GGCGTCTATATCT
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTT
```

该位置覆盖率 = 6

组装：基本原则



- 一条读段的后端与另一条读段的前端越相似，则

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

- 则两条读段越有可能在基因组上重叠

```
TATCTCGACTCTAGGCC
TCTATATCTCGGCTCTAGG
GGCGTCTATATCTCGGCTCTAGGCCCTCATTTTTT
```

组装



- 如果两条读段的确来自基因组上重叠的位置，为何会不同？

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

- ✿ 测序错误

- ✿ 不同染色体拷贝上的差异：例如人是双倍体，来自父亲和母亲的拷贝可能有差异

来自母亲的读段

TATCTCGACTCTAGGCC

来自父亲的读段

TCTATATCTCGGCTCTAGG

母亲的参考序列

TCTATATCTCGACTCTAGGCC

父亲的参考序列

TCTATATCTCGGCTCTAGGCC

Genotype &
SNP calling



两大类组装算法

❑ OLC: Overlap-Layout-Consensus

- ✿ String Graph Assemblers
- ✿ 根据读段直接构建重叠图
- ✿ 去除冗余读段，寻找组装的路径
- ✿ 工具：SGA, Fermi

❑ DBG: De Bruijn graph

- ✿ 根据读段构建 k -mer图，丢弃原始读段
- ✿ 寻找组装的路径
- ✿ 工具：Velvet, ABySS, SOAPdenovo2

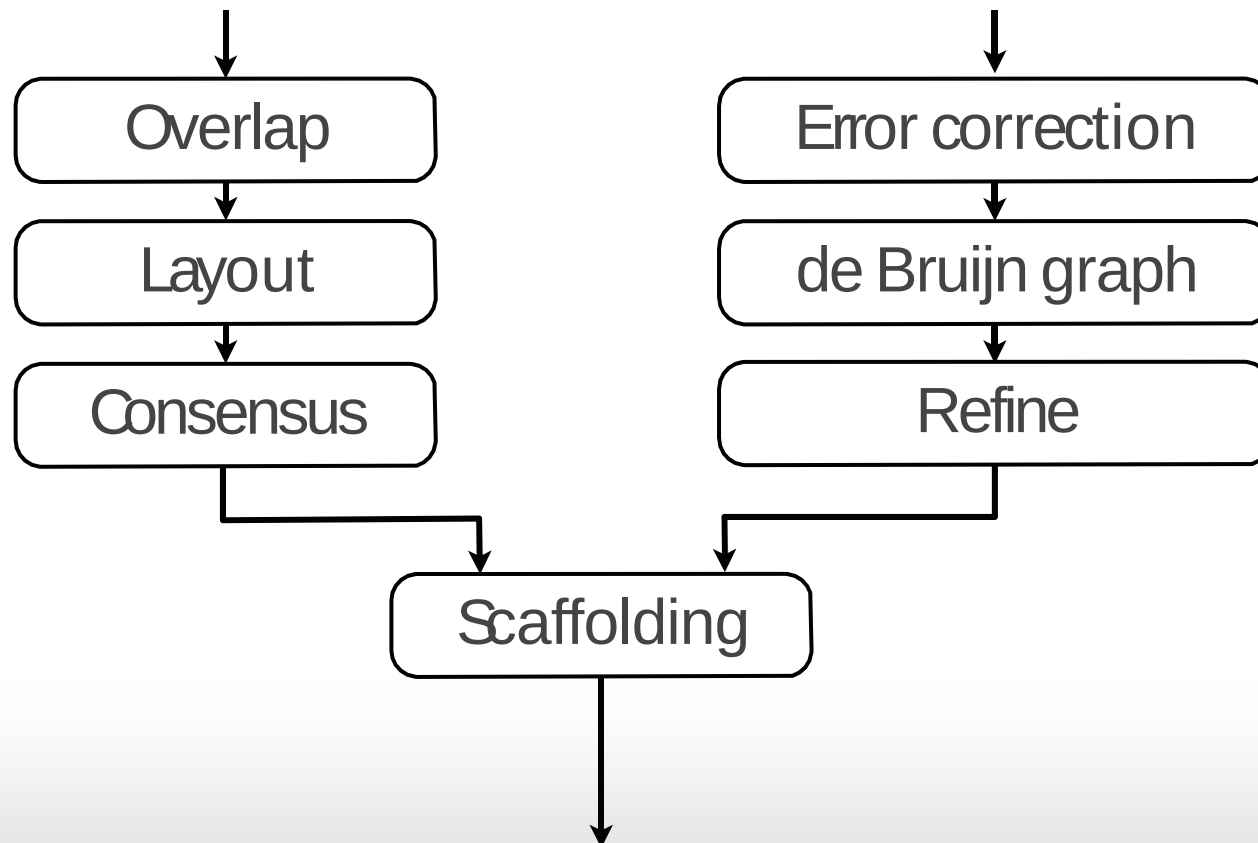
❑ 重复片段（Repeat）的影响

- ✿ 破坏基因组组装：Contig
- ✿ 两类算法不考虑重复片段

两大类组装算法



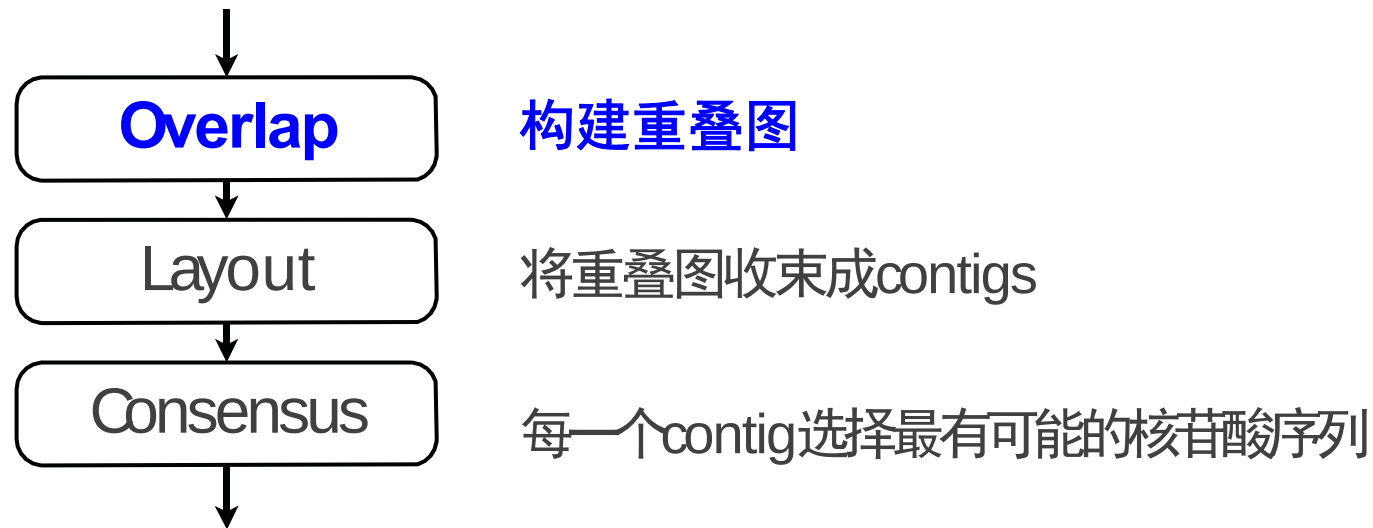
- ❑ OLC: Overlap-Layout-Consensus
- ❑ DBG: De Bruijn graph



Overlap-Layout-Consensus



❑ OLC: Overlap-Layout-Consensus



Overlap



□ 发现所有重叠:

- ✿ 建立一个有向图 (directed graph)
- ✿ 结点 (vertex, node) 为读段
- ✿ 有向边 (directed edge) 连接结点

CTCGGCTCTAGCCCCTCATTTT
||||||| |||||||||
GGCTCTAGGCCCTCATTTTTT

一条读段的后端与另
一条读段的前端相似

- CTAGGCCCTCAATTTTT
- GCGTCTATATCT
- CTCTAGGCCCTCAATTTTT
- TCTATATCTCGGCTCTAGG
- GGCTCTAGGCCCTCATTTTT
- CTCGGCTCTAGCCCCTCATTTT
- TATCTCGACTCTAGGCCCTCA
- GCGTCGATATCT
- TATCTCGACTCTAGGCC
- GCGTCTATATCTCG



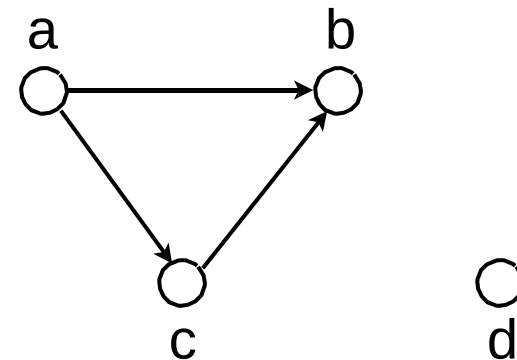
有向图

□ 有向图 $G(V, E)$

- ✿ 一系列结点 V
- ✿ 一系列有向边 E

□ 有向边：有序的结点对

- ✿ (source, sink)
- ✿ 节点：圆圈
- ✿ 边：连接两个圆圈的线



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

$E = \{(a, b), (a, c), (c, b)\}$

发起

接收

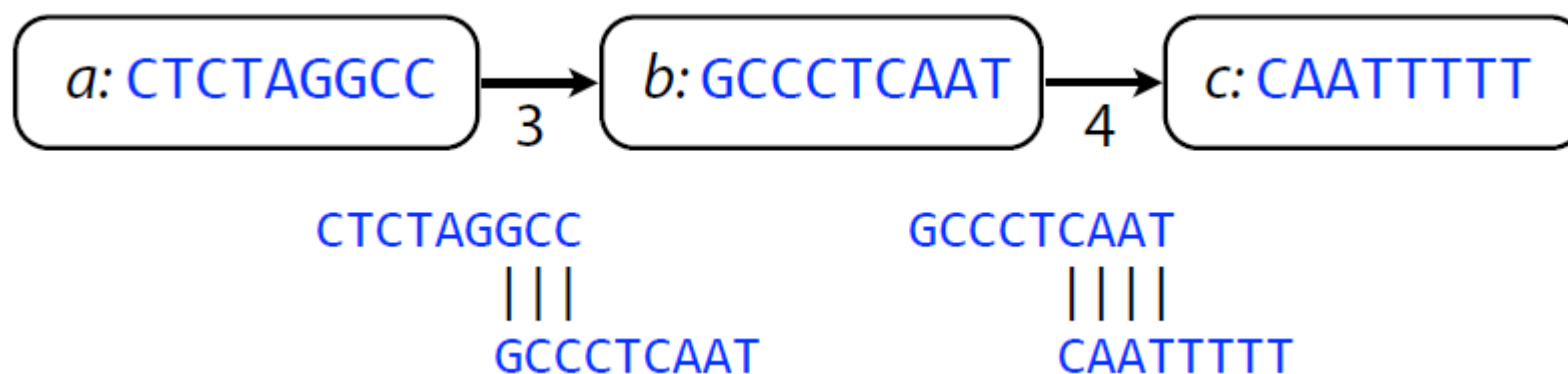
重叠图



- 例如，后缀/前缀至少3个字符重叠
- 结点为读段，有向边为发起的后缀和接收的前缀重叠数

结点 $\{ a: \text{CTCTAGGCC}, b: \text{GCCCTCAAT}, c: \text{CAATTTT} \}$

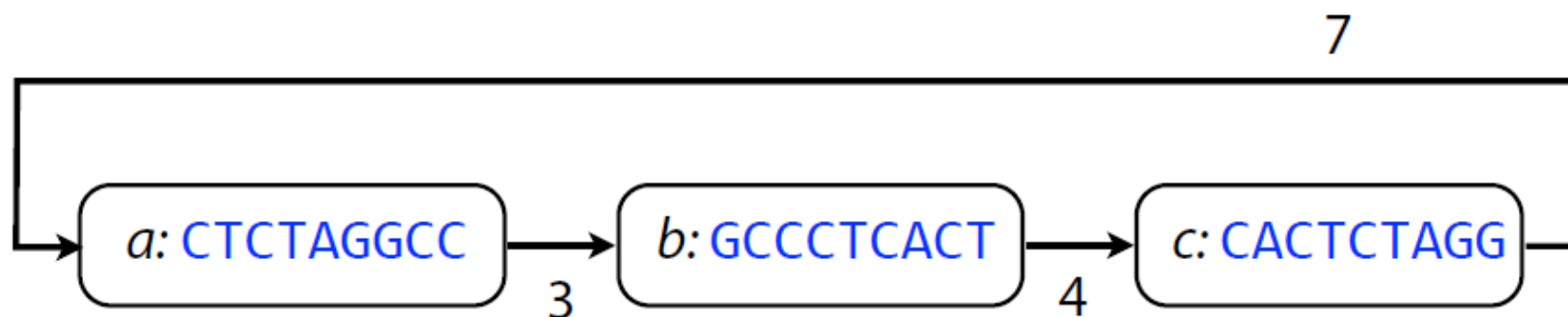
有向边 $\{ (a, b), (b, c) \}$



重叠图

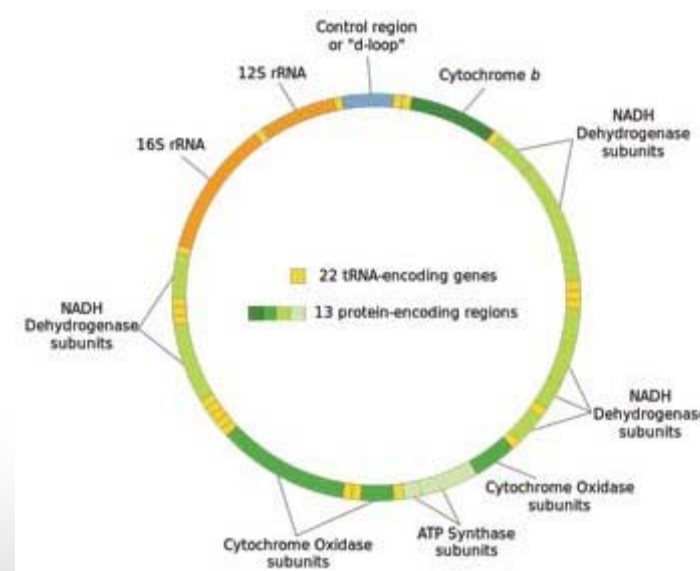


□ 重叠图可以是环形，起点和终点是同一个结点



□ 环形DNA链

- ✿ 细菌基因组一般为环形
- ✿ 线粒体DNA为环形
- ✿ DNA的重复序列





重叠搜索

□ 如何建立重叠图？

- ✿ 两条读段 X, Y ，长度为 k
- ✿ X 后缀精确匹配 Y 前缀的长度 $\geq l$

□ 思路：

- ✿ 利用FM索引，在 Y 里搜索长度为 l 的 X 后缀
- ✿ 若匹配则向左延伸，确认 Y 的前缀是否都可以匹配

例如， $l=3$

从左向右比对

X: CTCTAGGCC

Y: TAGGCCCTC

X: CTCTAGGCC

Y: TAGGCCCTC

发现匹配

向左延伸

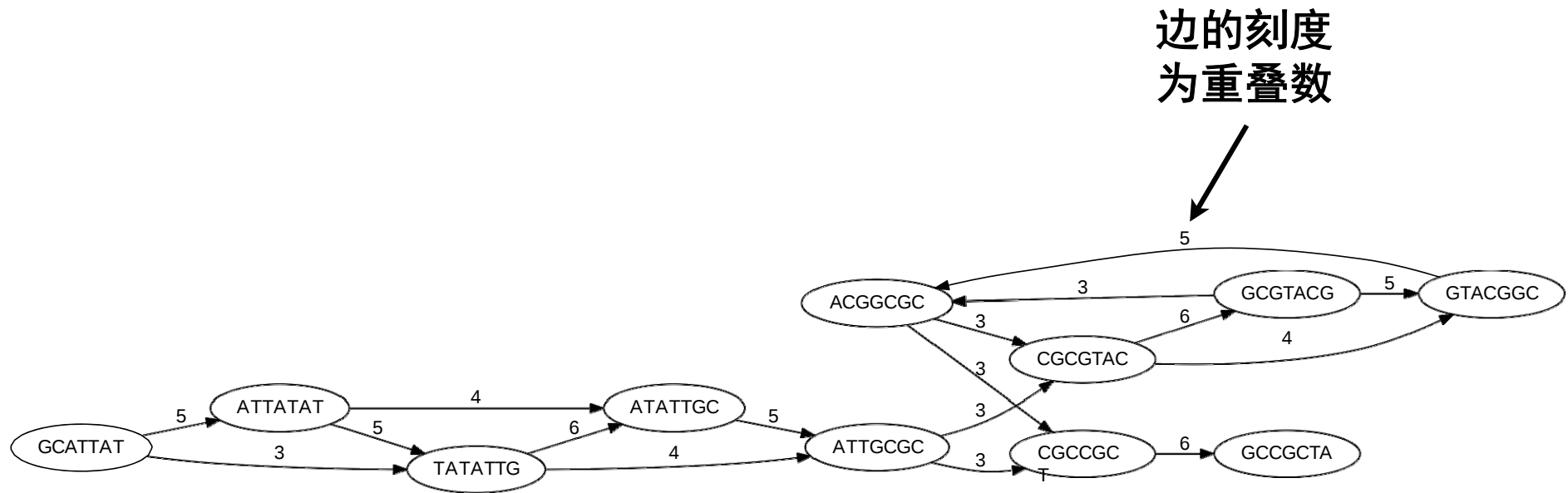
X: CTCTAGGCC

Y: TAGGCCCTC

重叠搜索



□ 重叠图 $l = 3$



原始序列: GCATTATATATTGCGCGTACGGCGCCGCTACA

Overlap-Layout-Consensus



Shortest common superstring



□ 如何求解SCS(S)?

- ✿ 考虑边的权重 = -重叠长度

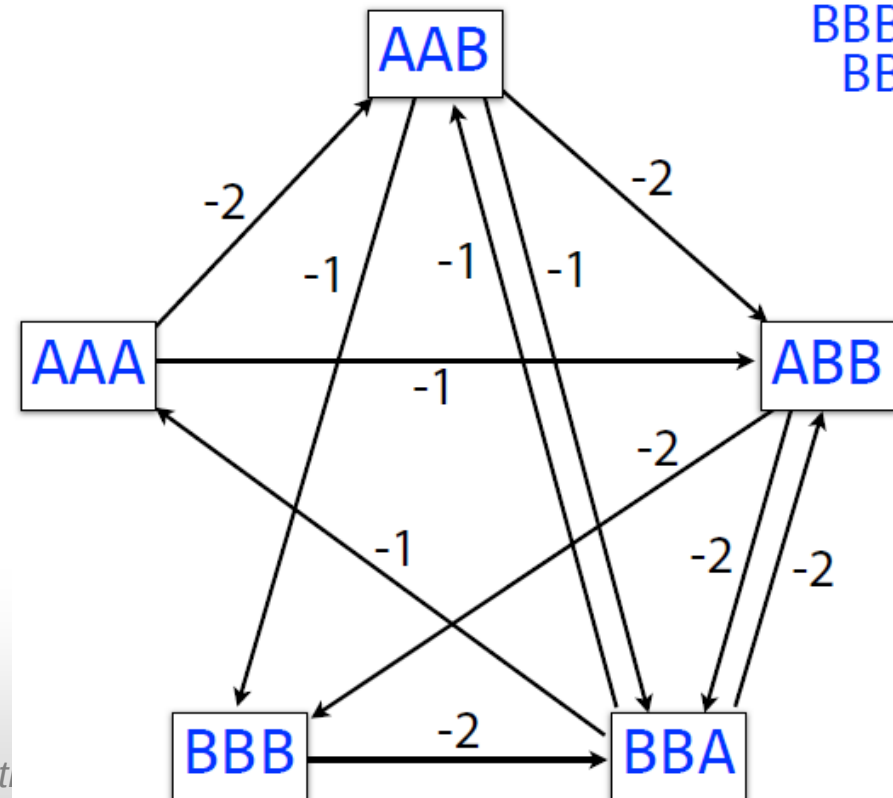
□ SCS:

- ✿ 走代价最小的路径
- ✿ 访问每个结点仅一次
- ✿ Traveling Salesman Problem (TSP)
- ✿ NP-hard

S: AAA AAB ABB BBB BBA

SCS(S): AAABBBBA

AAA
AAB
ABB
BBB
BBA



Shortest common superstring



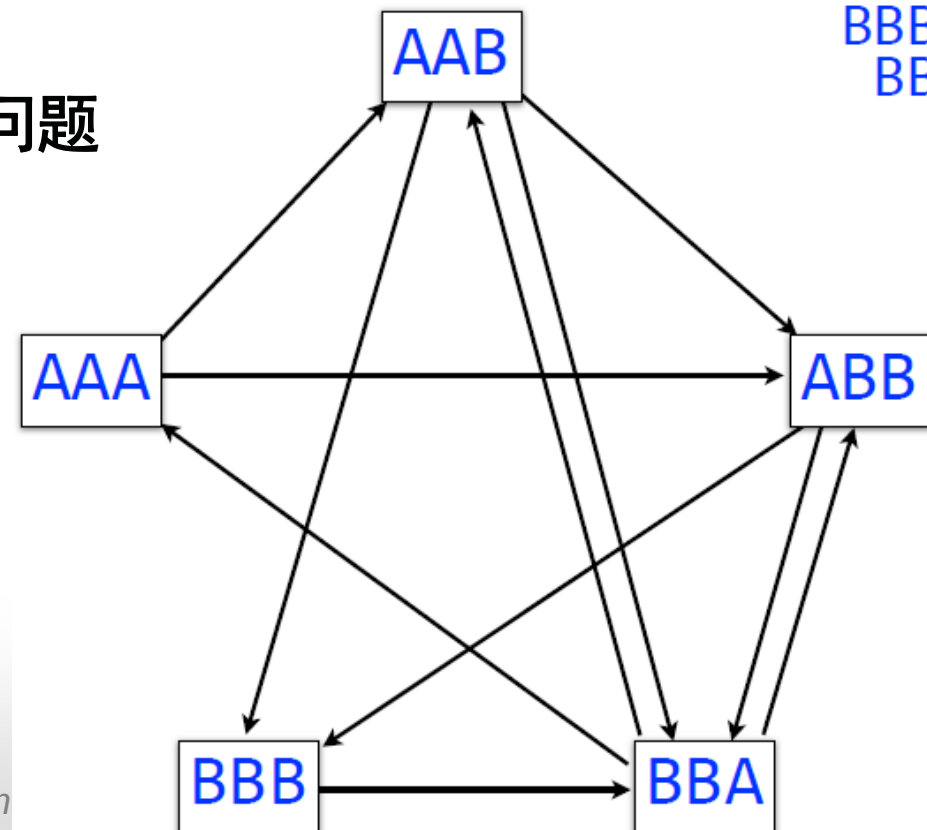
- 不考虑边的权重
- Hamiltonian Path problem: NP-complete

- ✿ NP完全问题
- ✿ 既是NP问题又是NP-hard问题
- ✿ 需要启发式算法

S: AAA AAB ABB BBB BBA

SCS(S): AAABBBBA

AAA
AAB
ABB
BBB
BBA



SCS: 贪心算法



❑ 选择最长的重叠，合并发起和接收：不保证最优

Greedy-SCS algorithm ($l = 1$):

┌────────── 输入读段 ─────────┐
ABA ABB AAA **AAB** BBB BBA BAB **BAA**
2 BAAB ABA **ABB** AAA BBB BBA **BAB**
2 BABB **BAAB** ABA AAA BBB **BBA**
2 **BBAAB** BABB ABA AAA **BBB**
2 **BBBAAB** BABB **ABA** AAA
2 **BBBAABA** BABB AAA
2 **BABBBAABA** AAA
1 BABBBAAABAAA
BABBBAAABAAA
└─ Superstring ─┘

红色读段合并后进入下一轮

贪心算法:
BABBBAAABAAA

真实的SCS:
AAABBBABAA

每一行合并一次
第一列：合并字符的长度

SCS: 贪心算法



□ Greedy-SCS算法 ($k = 6, l = 3$):

✿ `a_long_long_long_time`

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

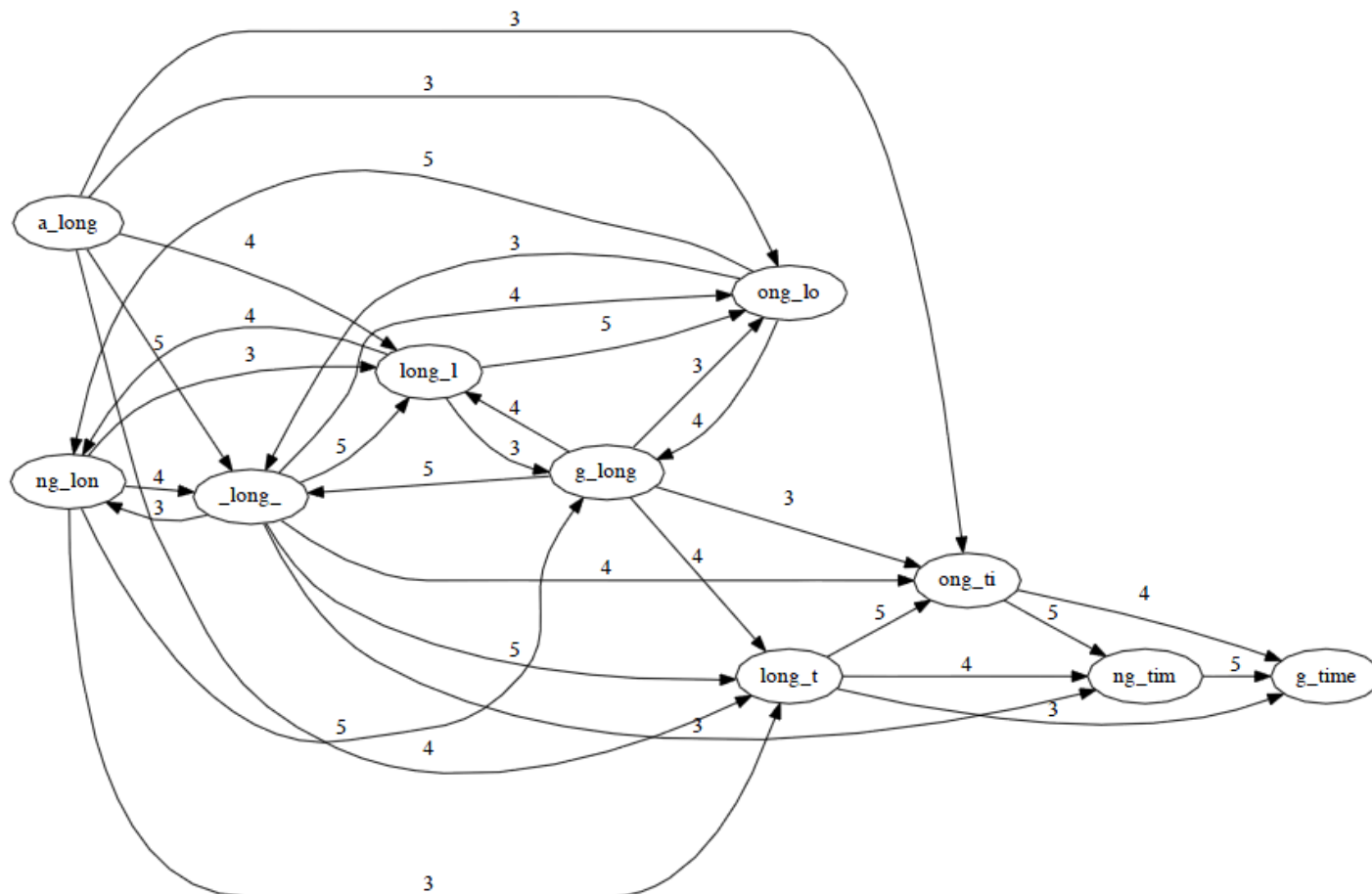
组装结果: `a_long_long_time?` 丢失: `_long`

SCS: 贪心算法



□ Greedy-SCS算法 ($k = 6, l = 3$):

✿ a_long_long_long_time

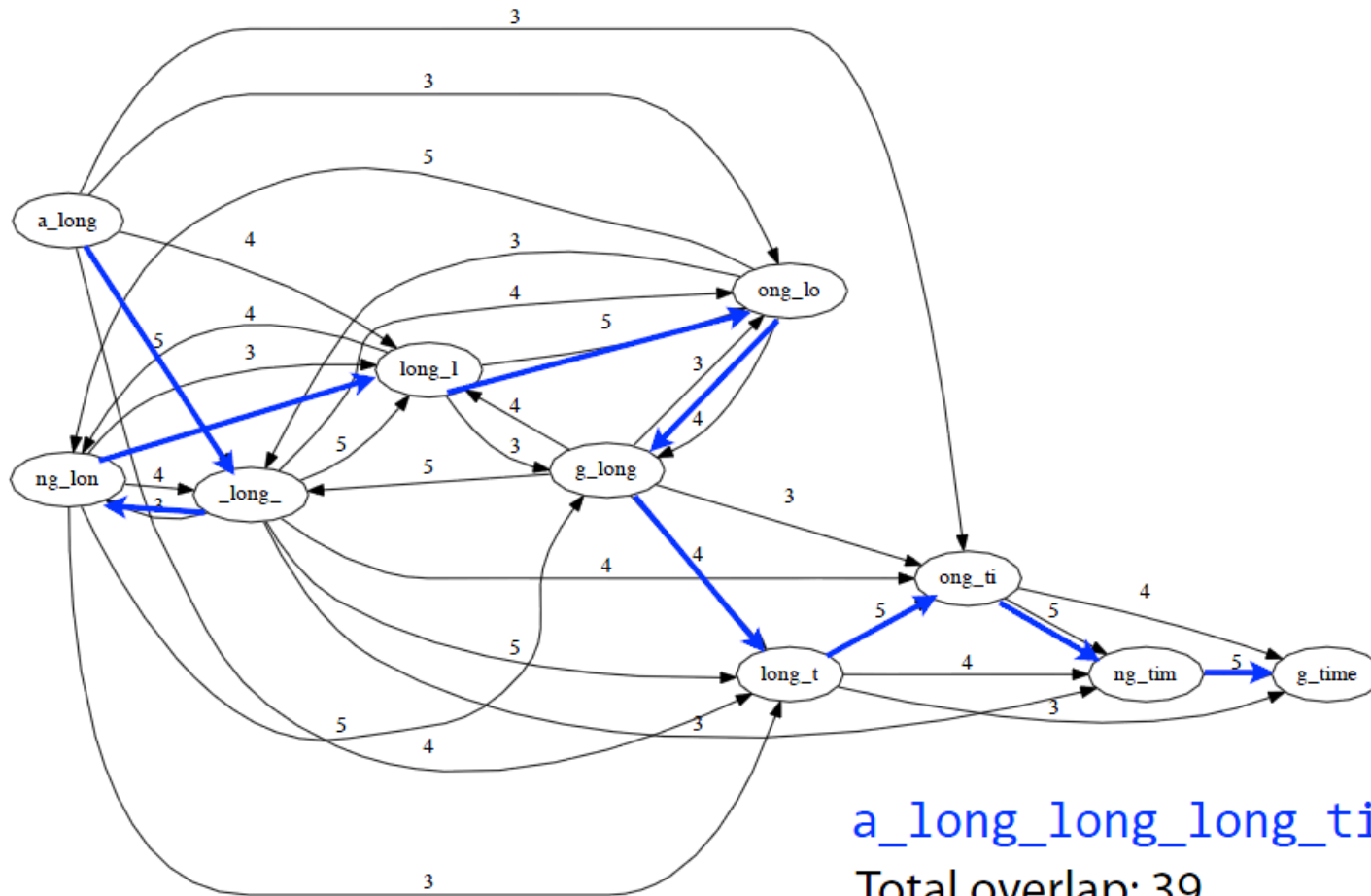


SCS: 贪心算法



□ Greedy-SCS算法 ($k = 6, l = 3$):

✿ `a_long_long_long_time`

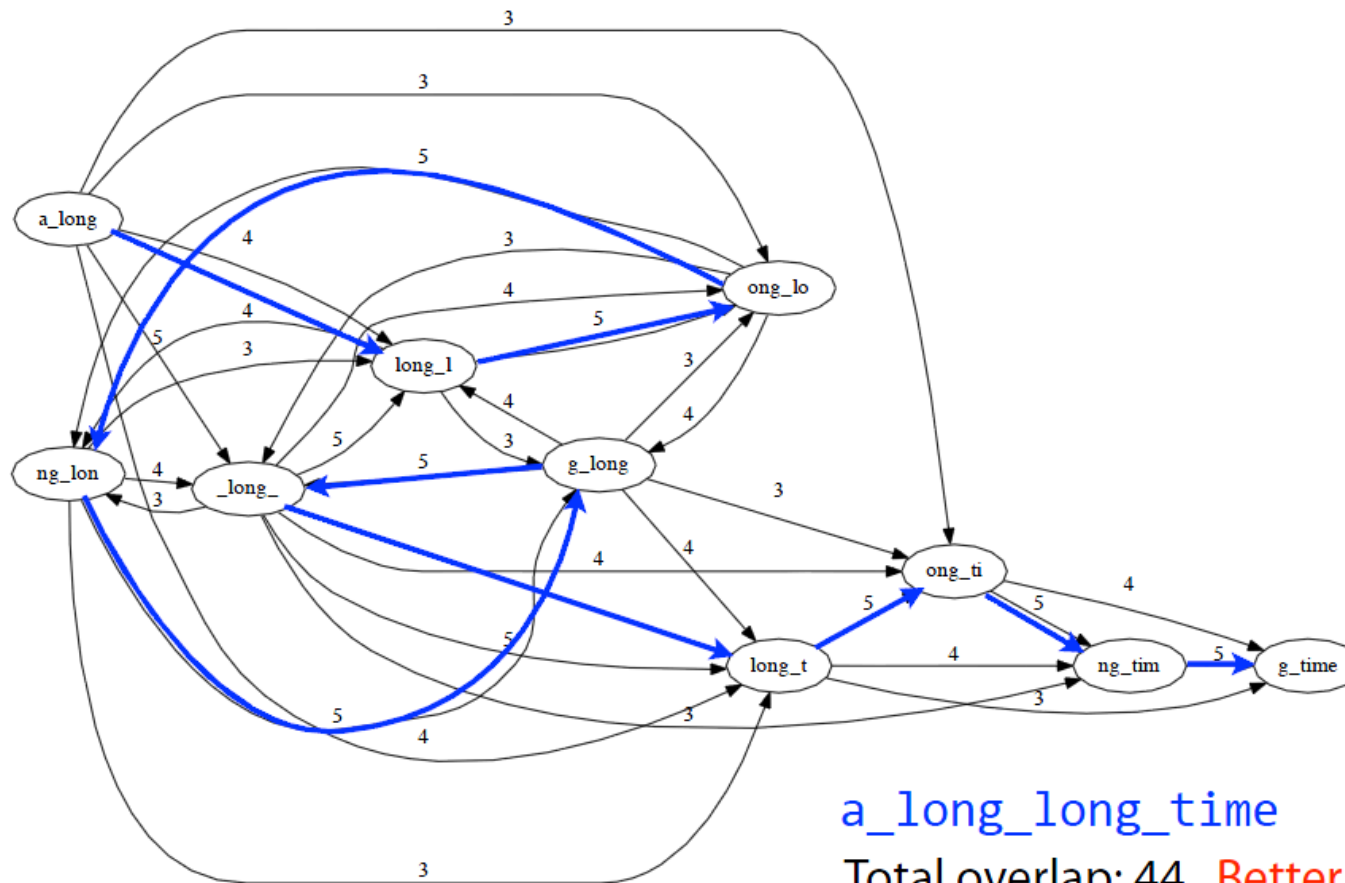


SCS: 贪心算法



□ Greedy-SCS算法 ($k = 6, l = 3$):

✿ `a_long_long_long_time`



`a_long_long_time`

Total overlap: 44 Better than the correct path!

SCS: 贪心算法



□ Greedy-SCS算法 ($k = 8, l = 3$):

✿ `a_long_long_long_time`

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

组装结果: `a_long_long_long_time`

SCS: 贪心算法



□ $k = 8$ 的读段可以正确识别三个 `_long`

✿ 一个读段横跨三个 `_long`

`a_long_long_long_time`

`g_long_l`



□ 重复序列：破坏组装

✿ 读段长度增加：解决重复序列的问题

✿ 算法不能解决重复序列

`a_long_long_long_time`

collapse

`a_long_long_time`

人类基因组~50%为重复序列

Bioinformatics, 2020, HUST

重复序列：破坏组装



□ Greedy-SCS算法，不同的 k, l 值

输入 `it_was_the_best_of_times_it_was_the_worst_of_times`

输出	l, k	output
	3, 5	<code>the_worst_of_times_it_was_the_best_o</code>
	3, 7	<code>s_the_worst_of_times_it_was_the_best_of_t</code>
	3, 10	<code>_was_the_best_of_times_it_was_the_worst_of_tim</code>
	3, 13	<code>it_was_the_best_of_times_it_was_the_worst_of_times</code>

重复序列： `it_was_the_` 长度为11

重复序列：破坏组装



□ Greedy-SCS算法，不同的 k, l 值

输入 `swinging_and_the_ringing_of_the_bells_bells_bells_bells_bells`

输出 l, k output

3, 7	<code>swinging_and_the_ringing_of_the_bells_bells</code>
3, 13	<code>swinging_and_the_ringing_of_the_bells_bells_bells</code>
3, 19	<code>swinging_and_the_ringing_of_the_bells_bells_bells_bells_b</code>
3, 25	<code>swinging_and_the_ringing_of_the_bells_bells_bells_bells_bells</code>

子串越长越有可能正确识别重复序列

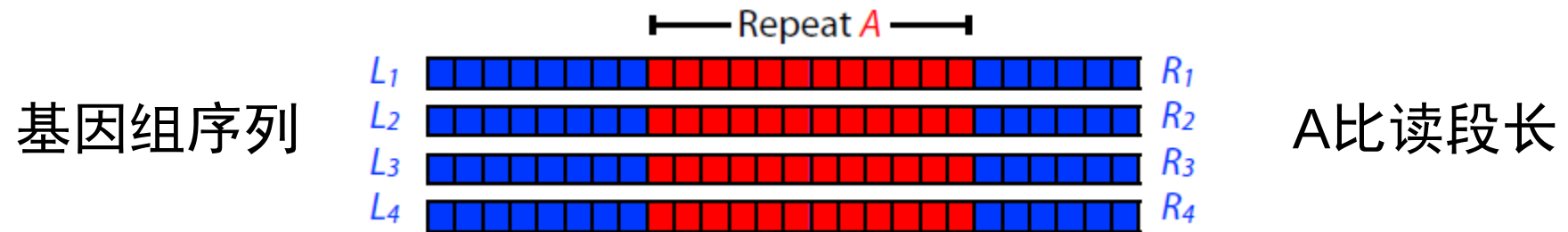
$k = 25$

<code>e_bells_bells_bells_bells</code>
<code>_bells_bells_bells_bells_</code>
<code>bells_bells_bells_bells_b</code>

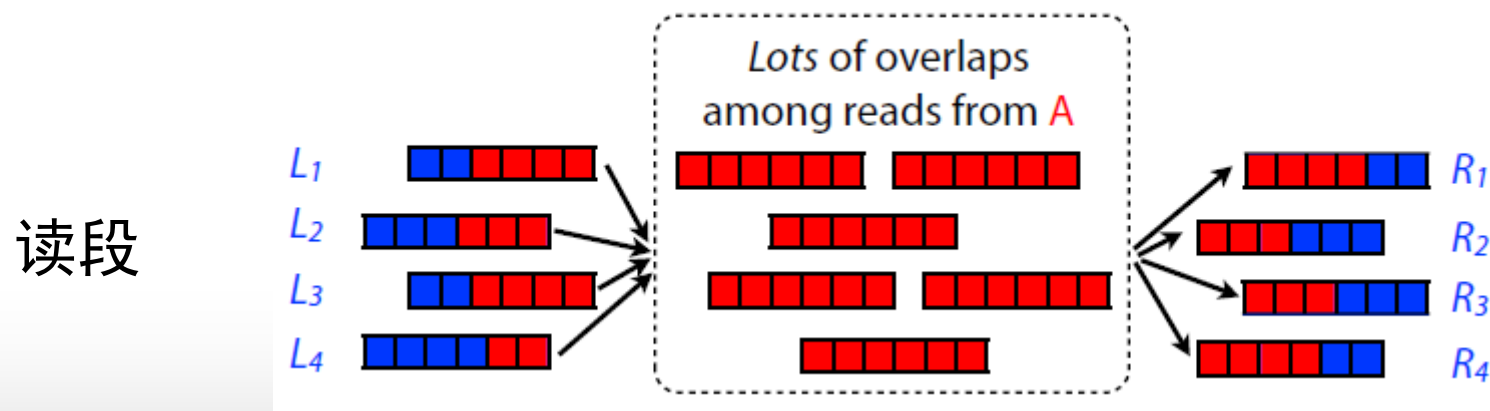
重复序列



□ 关于重复序列A的重叠图



□ 不能确定重复序列的路径



Layout

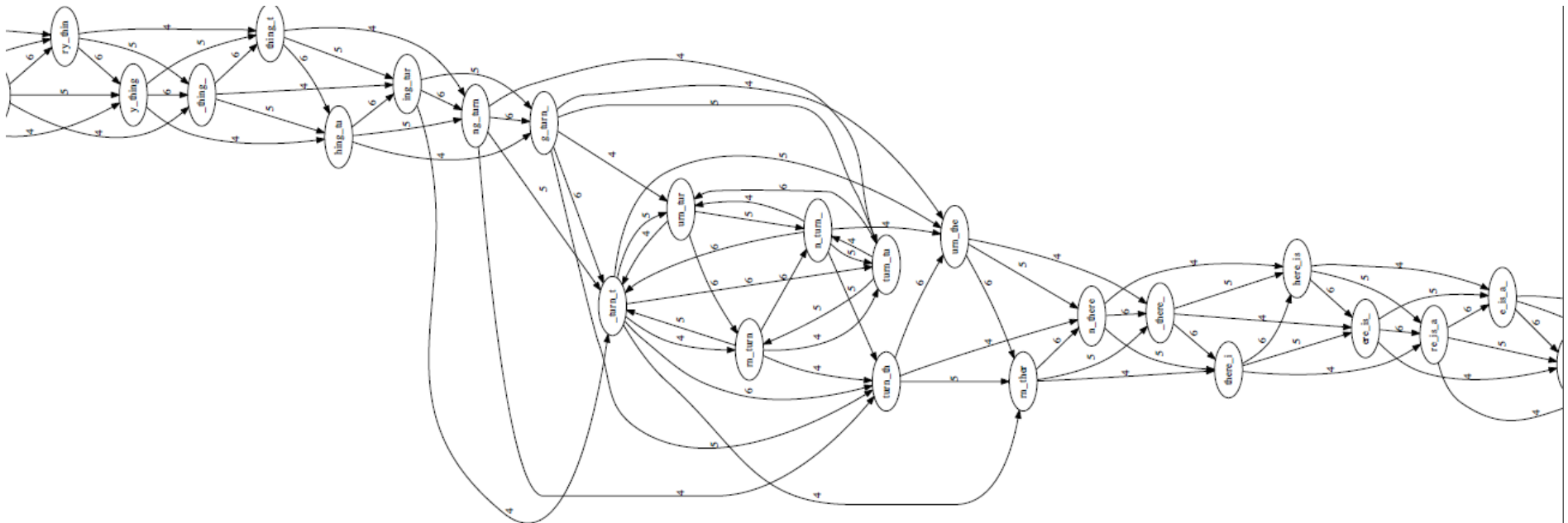


❑ Greedy-SCS算法的缺陷

- ❁ 不能发现最优的SCS
- ❁ 存在重复序列时得到错误的路径

❑ 若不考虑重复序列，如何从重叠图获得Contigs?

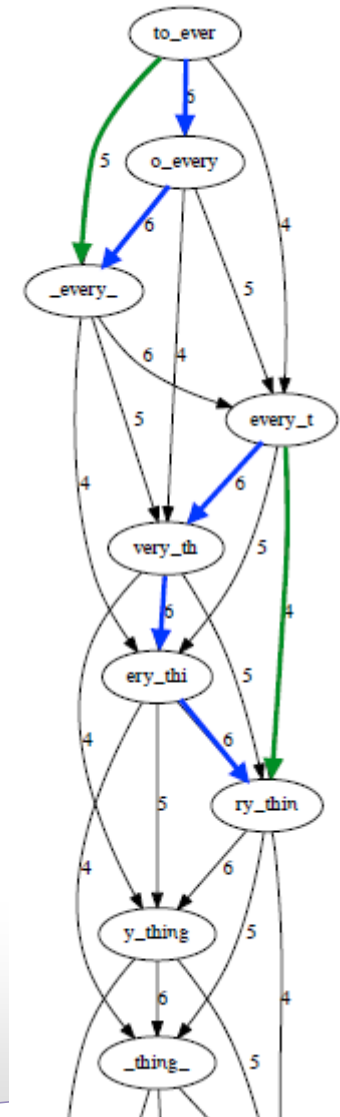
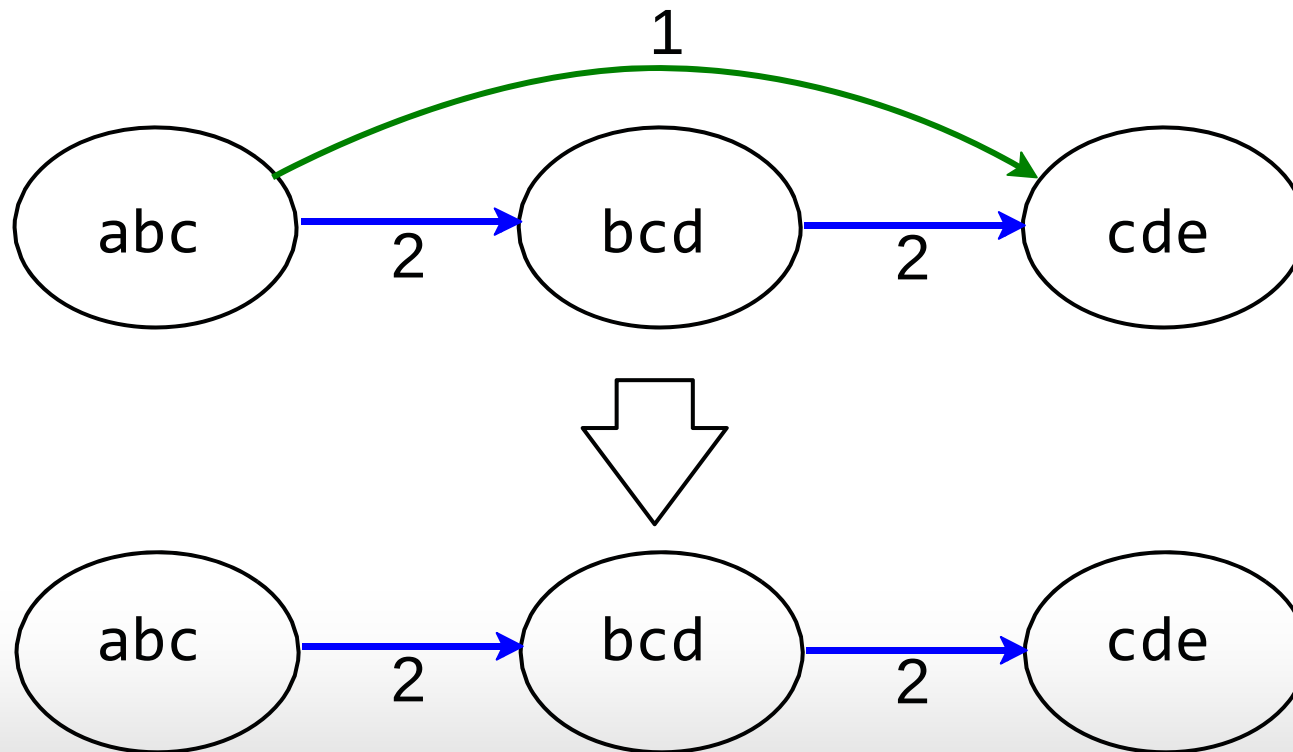
重叠图 `to_everything_turn_turn_turn_there_is_a_season`
 $l=4, k=7$



Layout



- ❑ 去掉可间接推断的（transitively-inferrible）边
- ❑ 例如绿色的连接可以由蓝色的间接获得

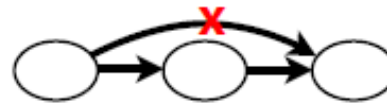


A diagram showing a sequence of three nodes (ovals) connected by arrows. The first node points to the second, and the second points to the third. A red 'X' is placed over the second node, indicating a deletion or error.

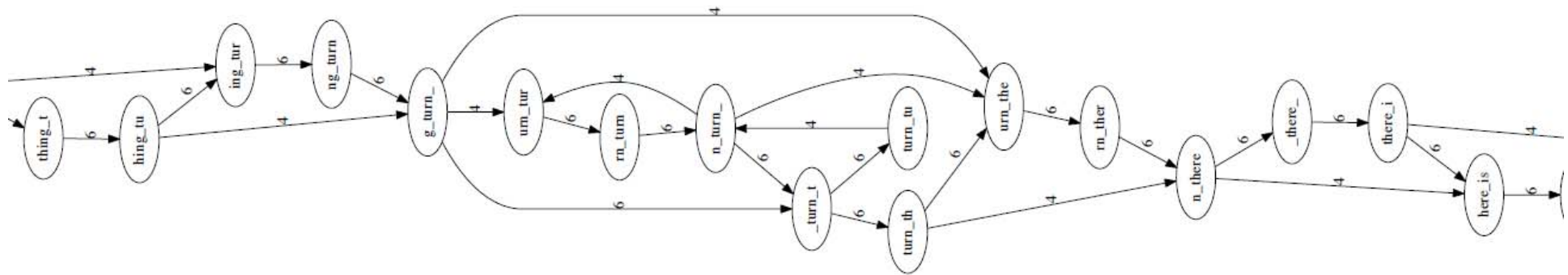
Layout



□ 继续去掉可间接推断的 边



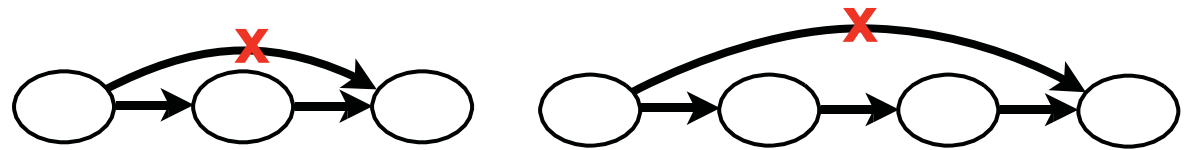
获得:



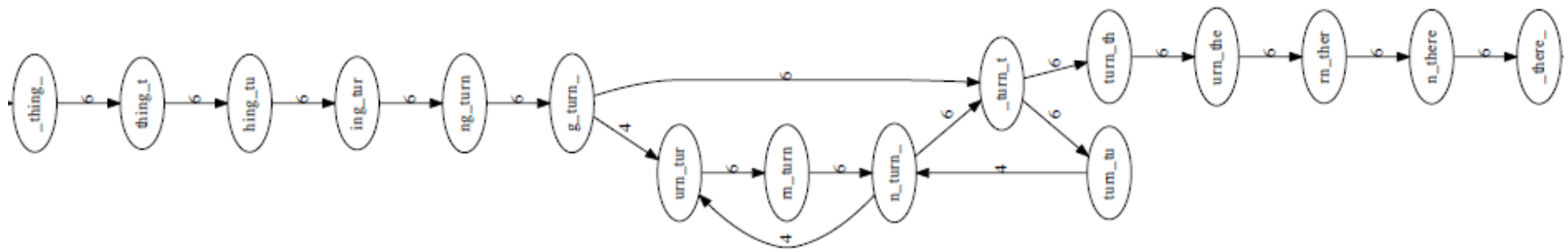
Layout



□ 去掉所有可间接推断的边



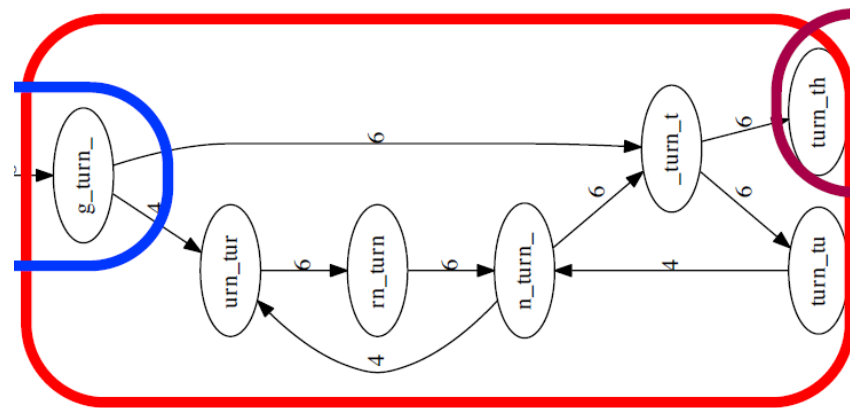
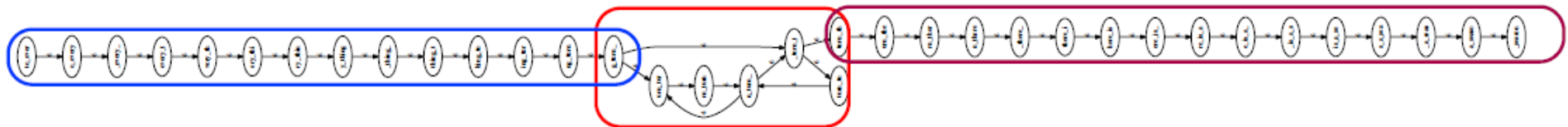
最终结果:



Layout



□ Contig: 没有分支



Contig 1

to_every_thing_turn_

Contig 2

turn_there_is_a_season

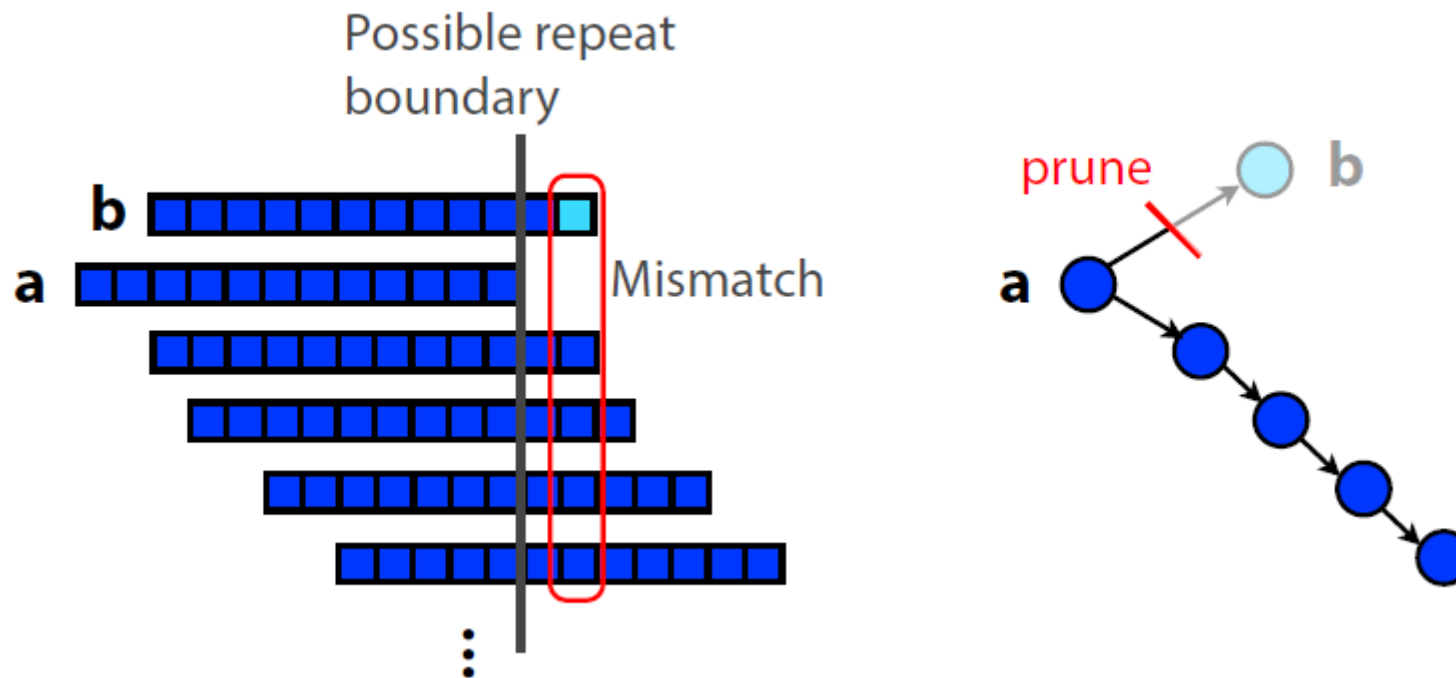


无法解决的重复序列

Layout



- 现实中，layout还需要处理有错误的子图，例如，测序错误



测序错误导致错配。通过b末端的路径引起错配，因此推断为测序错误，并删除相应路径

Overlap-Layout-Consensus



Consensus



□ 确定每个位置的碱基，需要考虑：

- ✿ 测序错误？
- ✿ 倍性 (ploidy)？

□ 例如，若真实的genotype为AA

- ✿ 测序错误高
- ✿ 测序深度低：每个位置仅有6条读段

```
TAGATTACACAGATTACTGA TTGATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGCGTAACTA
TAG TTACACAGATTATTGACTTCATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA
```

比对Contigs

↓ ↓ ↓ ↓ ↓

```
TAGATTACACAGATTACTGACTTGATGGCGTAA CTA
```

Majority vote :
确定consensus

Overlap-Layout-Consensus



❑ OLC算法的主要缺点：

- ⚙️ 重叠图构建的速度慢
- ⚙️ NGS测序数据规模大：人类基因组30X， $\sim 9 \times 10^{10}$ 碱基



SGA - String Graph Assembler



❑ 2015年发布, <https://github.com/jts/sga>

GitHub [Explore](#) [Features](#) [Enterprise](#) [Blog](#) [Sign up](#) [Sign in](#)

jts / sga [Watch](#) 35 [Star](#) 114 [Fork](#) 57

de novo sequence assembler using string graphs <http://genome.cshlp.org/content/22/3/549>

[1,641 commits](#) [17 branches](#) [38 releases](#) [17 contributors](#)

[branch: master](#) [sga / +](#)

Undo unwanted changed wrt master branch before merge

jts authored on 30 Jan latest commit af92f7d652

[src](#) Undo unwanted changed wrt master branch before merge 2 months ago

[README.md](#) add a hotlink to src/README a year ago

[index.html](#) Fixed formatting 5 years ago

[README.md](#)

SGA - String Graph Assembler

SGA is a de novo genome assembler based on the concept of string graphs. The major goal of SGA is to be very memory efficient, which is achieved by using a compressed representation of DNA sequence.

[Code](#) [Issues](#) 14 [Pull requests](#) 0 [Wiki](#) [Pulse](#) [Graphs](#)

HTTPS clone URL
<https://github.cc>
You can clone with [HTTPS](#) or [Subversion](#).
[Clone in Desktop](#) [Download ZIP](#)

Fermi



- ❑ 李恒，2012年发布
- ❑ 全基因组的从头组装
- ❑ SNP和INDEL calling
- ❑ <https://github.com/lh3/fermi>

GitHub This repository Search Explore Features Enterprise Blog Sign up Sign in

lh3 / fermi Watch 13 Star 45 Fork 11

A WGS de novo assembler based on the FMD-index for large genomes

755 commits 15 branches 26 releases 1 contributor

branch: master fermi / +

r751: change the version string
lh3 authored on 6 Jul 2013 latest commit 01cfc0750e

misc	no default filename for ZFile()	3 years ago
.gitignore	updated run-fermi.pl	3 years ago
Makefile	r735: speed up solid k-mer collection	3 years ago
Makefile.am	Makefile.am	2 years ago
README.md	r744: release fermi-1.1	3 years ago

Code Issues 3 Pull requests 0 Pulse Graphs

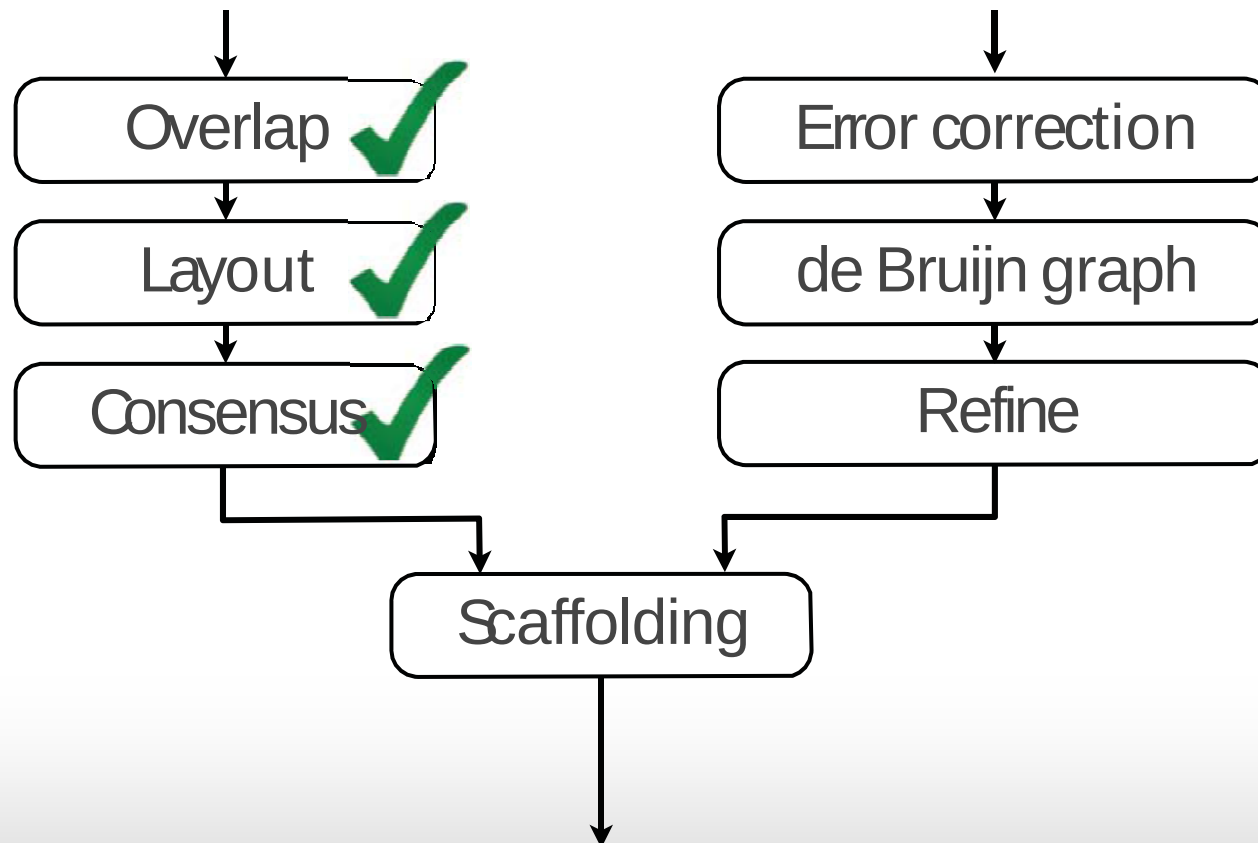
HTTPS clone URL
<https://github.cc>

You can clone with HTTPS or Subversion.

两大类组装算法



- ❑ OLC: Overlap-Layout-Consensus
- ❑ **DBG: De Bruijn graph**



De Bruijn graph



❑ “Seven Bridges of Königsberg”: 哥尼斯堡七桥问题

✿ Hamiltonian cycle: 每个点遍历一次

✿ Eulerian cycle: 每条边遍历一次

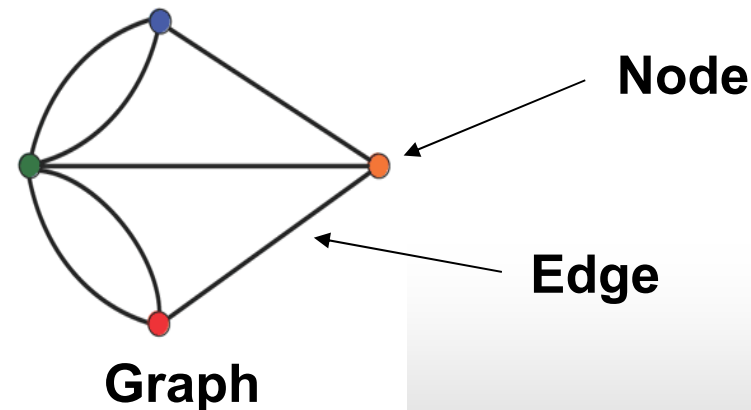
❑ 点 (Node): 读段或 k -mer

❑ 边 (Edge): 两点之间的关联

a



b



Velvet



- ❑ 第一个使用DBG算法的基因组组装工具
 - ⚙ 小的或中等规模的基因组
- ❑ <http://www.ebi.ac.uk/~zerbino/velvet/>



Velvet

Sequence assembler for very short reads

- [Current version: 1.2.10](#)
- [Manual](#) and [extension for Columbus](#) in pdf format
- Public [Git](#) URL: `git clone git://github.com/dzerbino/velvet.git`
- For up-to-date info, you can consult and/or subscribe to the [mailing list](#).
- For transcriptomic assembly Velvet is extended by [Oases](#).

ABYSS



❑ 实现DBG的并行化：大基因组

❑ <http://www.bcgsc.ca/platform/bioinfo/software/abyss>

Platforms

Bioinformatics

Bioinformatics Licenses

GSC Software Centre

PASsIT

Adapter Trimming for Small RNA Sequencing

Sealer

LINKS

Konnector

Spark

TASR

XpressAlign: FPGA Short Read Aligner

Barnacle

Satellog

ABYSS

Assembly By Short Sequences - a de novo, parallel, paired-end sequence assembler

Project Description



ABYSS is a *de novo*, parallel, paired-end sequence assembler that is designed for short reads. The single-processor version is useful for assembling genomes up to 100 Mbases in size. The parallel version is implemented using MPI and is capable of assembling larger genomes.

To assemble transcriptome data, see [Trans-ABYSS](#).

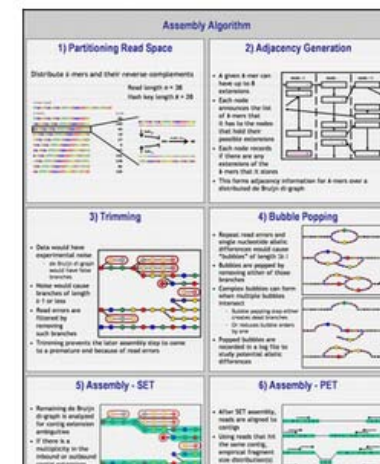
Publications

- ABYSS: A parallel assembler for short read sequence data. Simpson JT, Wong K, Jackman SD, Schein JE, Jones SJ, Birol I. Genome Research, 2009-June. ([Genome Research](#), [PubMed](#))
- De novo Transcriptome Assembly with ABYSS. İnanç Birol, Shaun D Jackman, Cydney Nielsen, Jenny Q Qian, Richard Varhol, Greg Stazyk, Ryan D Morin, Yongjun Zhao, Martin Hirst, Jacqueline E Schein, Doug E Horsman, Joseph M

Project Resources

- [Releases](#)
- [Documentation](#)
- [Issue tracker](#)
- [Support](#)
- [Contact address](#)

Project owner: [Anthony Raymond](#)



SOAPdenovo & SOAPdenovo2



- ❑ 华大基因：大基因组
- ❑ <http://soap.genomics.org.cn/soapdenovo.html>

**Short Oligonucleotide
Analysis Package**

[SOAP-ICLU](#) [SOAP-HLA](#) [SOAP-popIndel](#) [SOAPdenovo-Trans](#) [SOAPdenovo2](#) [About](#)

[Home](#) [SOAPIndel](#) [SOAPsnv](#) [SOAPsv](#) [SOAPfusion](#) [SOAP3-dp](#) [SOAPfuse](#) [SOAPaligner](#) [SOAPsplice](#) [SOAPsnp](#)

SOAPdenovo

- [Introduction](#) »
- [System requirements](#) »
- [Download](#) »
- [Installation](#) »
- [Command Line Options](#) »
- [FAQ](#) »

Introduction

SOAPdenovo is a novel short-read assembly method that can build a de novo draft assembly for the human-sized genomes. The program is specially designed to assemble Illumina GA short reads. It creates new opportunities for building reference sequences and carrying out accurate analyses of unexplored genomes in a cost effective way. Now the new version is available. SOAPdenovo2, which has the advantage of a new algorithm design that reduces memory consumption in graph construction, resolves more repeat regions in contig assembly, increases coverage and length in scaffold construction, improves gap closing, and optimizes for large genome.

System requirements

SOAPdenovo aims for large plant and animal genomes, although it also works well on bacteria and fungi genomes. It runs on 64-bit Linux system with a minimum of 5G physical memory. For big genomes like human, about 150 GB memory would be required.

De Bruijn graph



- ❑ 设计理念类似SCS，各有优缺点
- ❑ k -mer

读段S: GGCGATTCATCG
S的一个4-mer: ATTC
S的所有3-mer: GGC
GCG
CGA
GAT
ATT
TTC
TCA
CAT
ATC
TCG

- ❑ $k-1$ -mer: 长度为 $k-1$ 的子串

De Bruijn graph



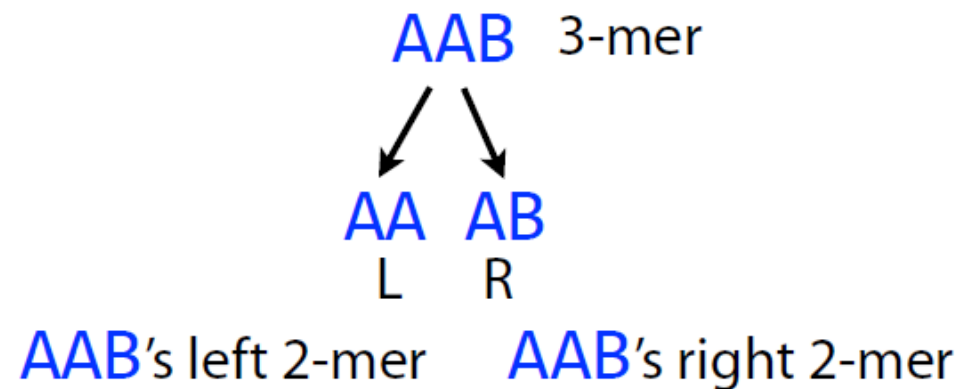
- 给定一系列来源于参考基因组的读段，获得 k -mer, $k=3$

AAA, AAB, ABB, BBB, BBA

- AAB是一个 k -mer ($k=3$)

- ✿ AA是左 $k-1$ -mer

- ✿ AB是右 $k-1$ -mer



De Bruijn graph



- 将每一个长度为3的字串切分成长度为2的、两个字串重叠的子串

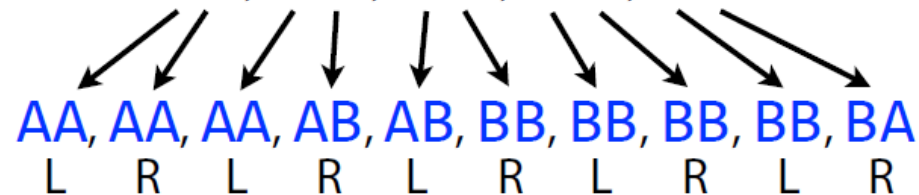
读段S:

AAABBBBA

S的所有3-mer:

AAA, AAB, ABB, BBB, BBA

所有左/右2-mer:

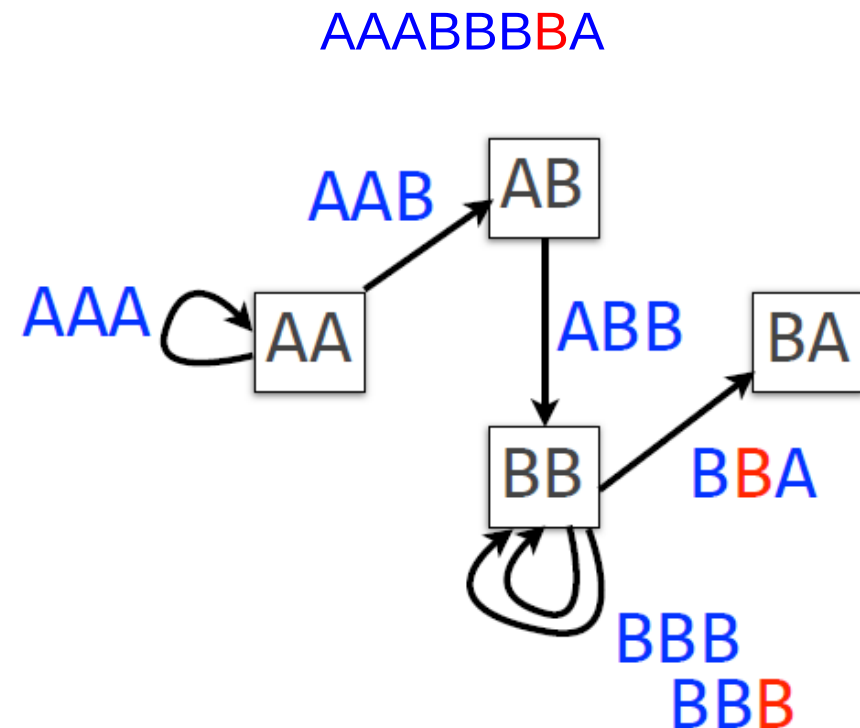
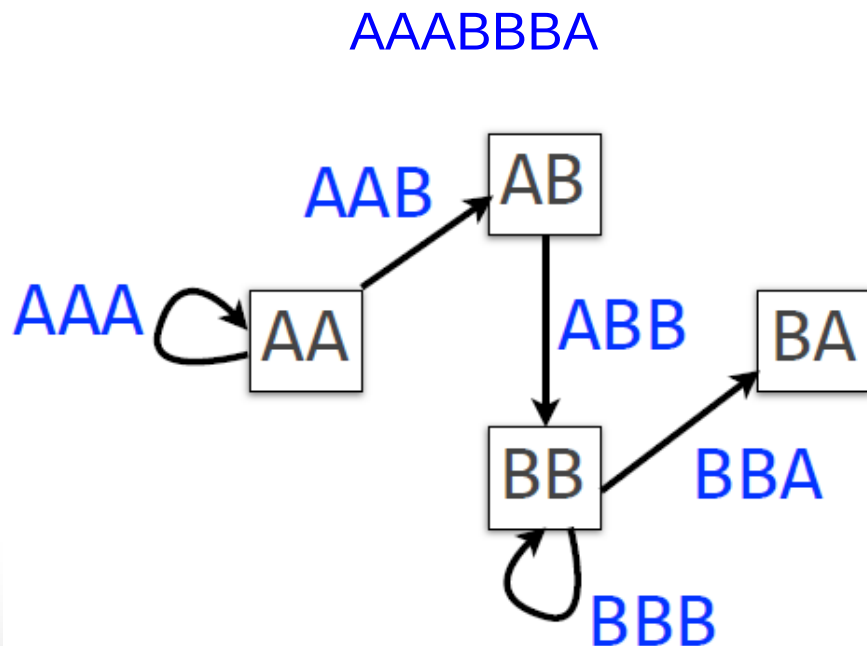


- 令2-mer为新图的结点
- 根据左2-mer ->右2-mer的重叠方向确定有向边

De Bruijn graph



- 图中每一条边代表长度为3的字串，等于k-mer
- 若在读段中再加一个B: AAABBBBA
- 重建De Bruijn graph: 多边 (Multiedge)



Eulerian walk



□ De Bruijn graph: 有向多重图 (Directed multigraph)

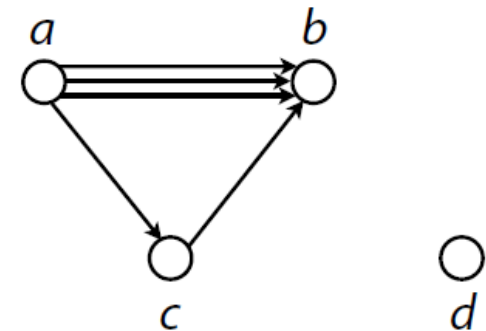
- ✿ 一系列顶点/结点 V ，多重有向边 E
- ✿ 结点入度 (indegree) = 进入的边
- ✿ 结点出度 (outdegree) = 出去的边
- ✿ 平衡 (balanced) 结点: 入度=出度
- ✿ 半平衡 (semi-balanced) 结点: 入度与出度差1
- ✿ 连接图: 所有结点都可通过其他结点访问

□ 欧拉行走

- ✿ 遍历每条边仅一次

□ 欧拉图

- ✿ 仅有两个半平衡结点
- ✿ 其余都是平衡结点



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

$$E = \{(a, b), (a, b), (a, b), (a, c), (c, b)\}$$

└── Repeated ─┘

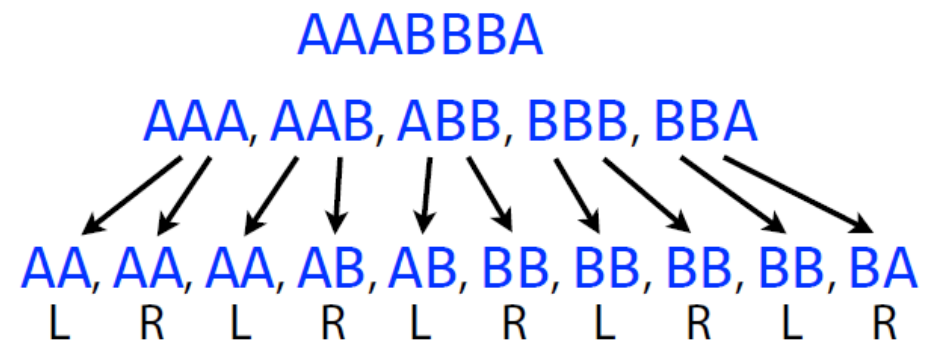
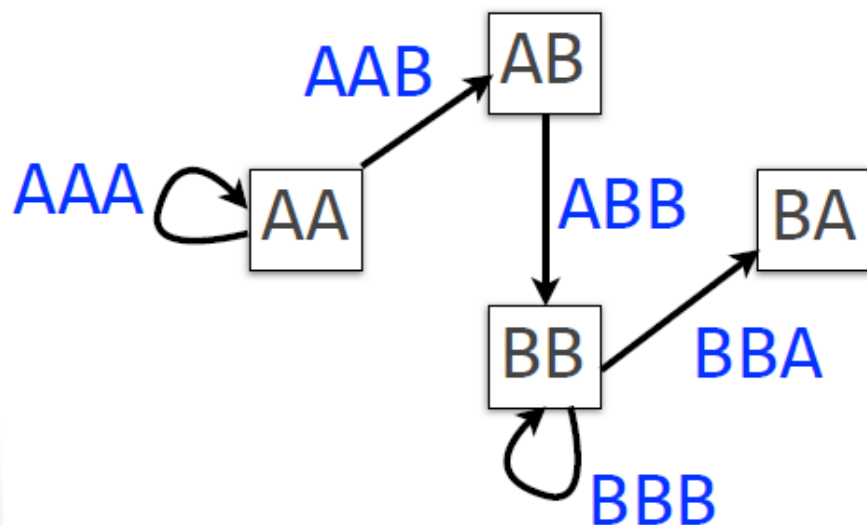
De Bruijn graph



□ 欧拉图：读段**AAABBBBA**生成的De Bruijn graph

⚙ 所有结点都可访问：**AA** → **AA** → **AB** → **BB** → **BB** → **BA**

⚙ **AA**和**AB**为半平衡结点，**AB**和**BB**为平衡结点



De Bruijn graph

- ❑ 基因组De Bruijn graph的构建流程
- ❑ 假设没有测序错误

读段长度 $k=5$: 5

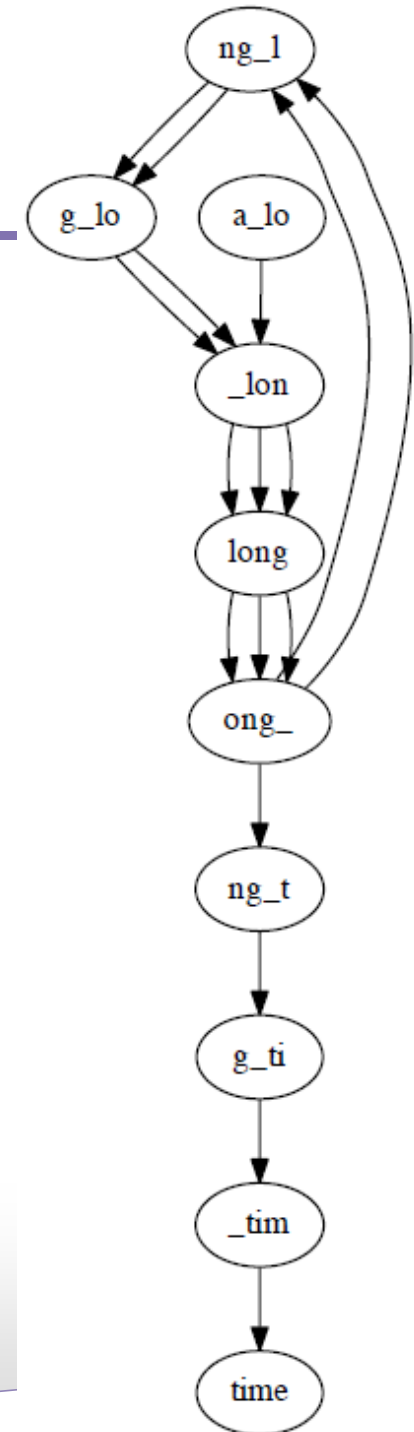
参考序列:

a_long_long_long_time

将 k -mer切分成
左/右 $k-1$ -mer:

long_
long ong_

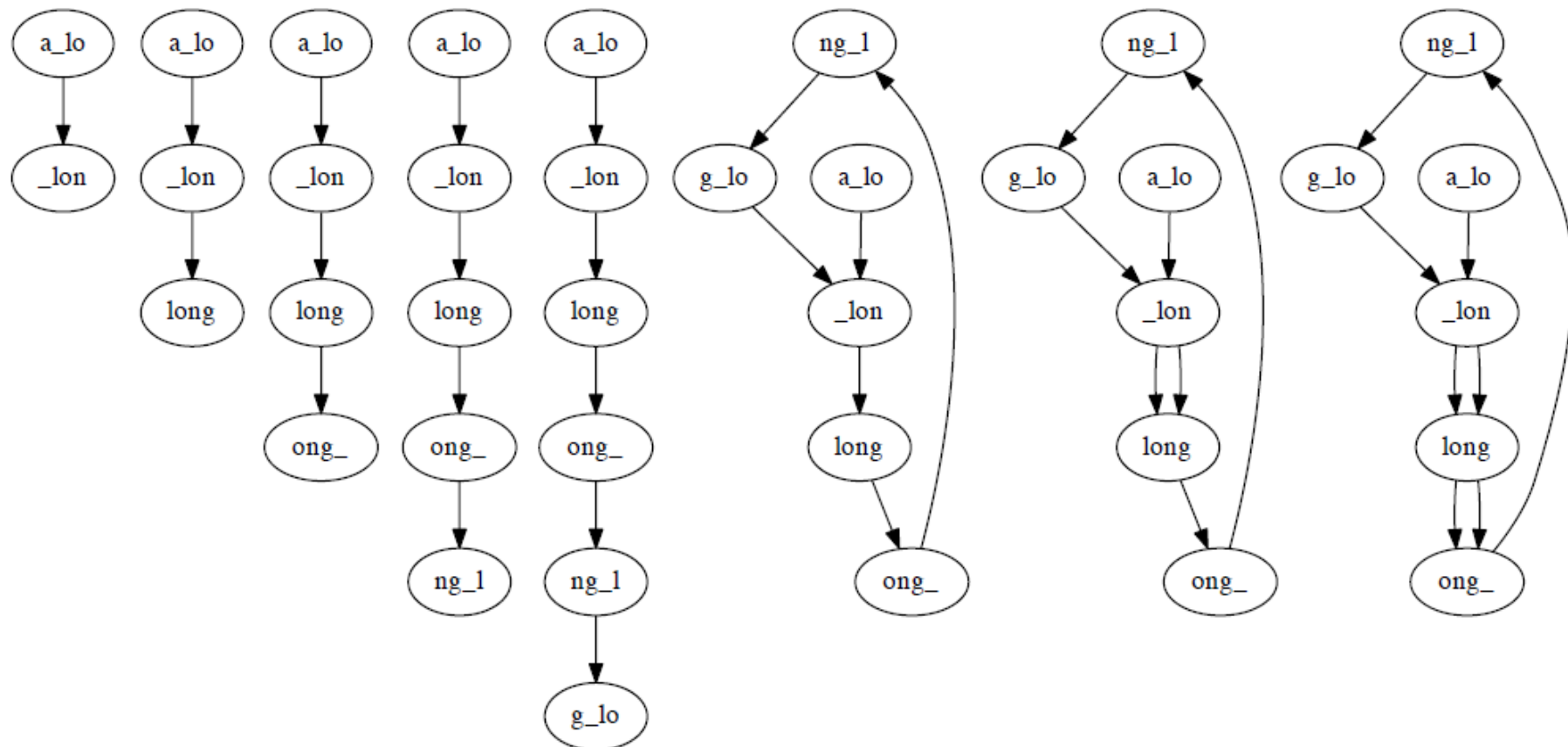
- ❑ 将新的 $k-1$ -mer作为结点加入图中
- ❑ 根据左 $k-1$ -mer $\rightarrow$ 右 $k-1$ -mer重叠加边



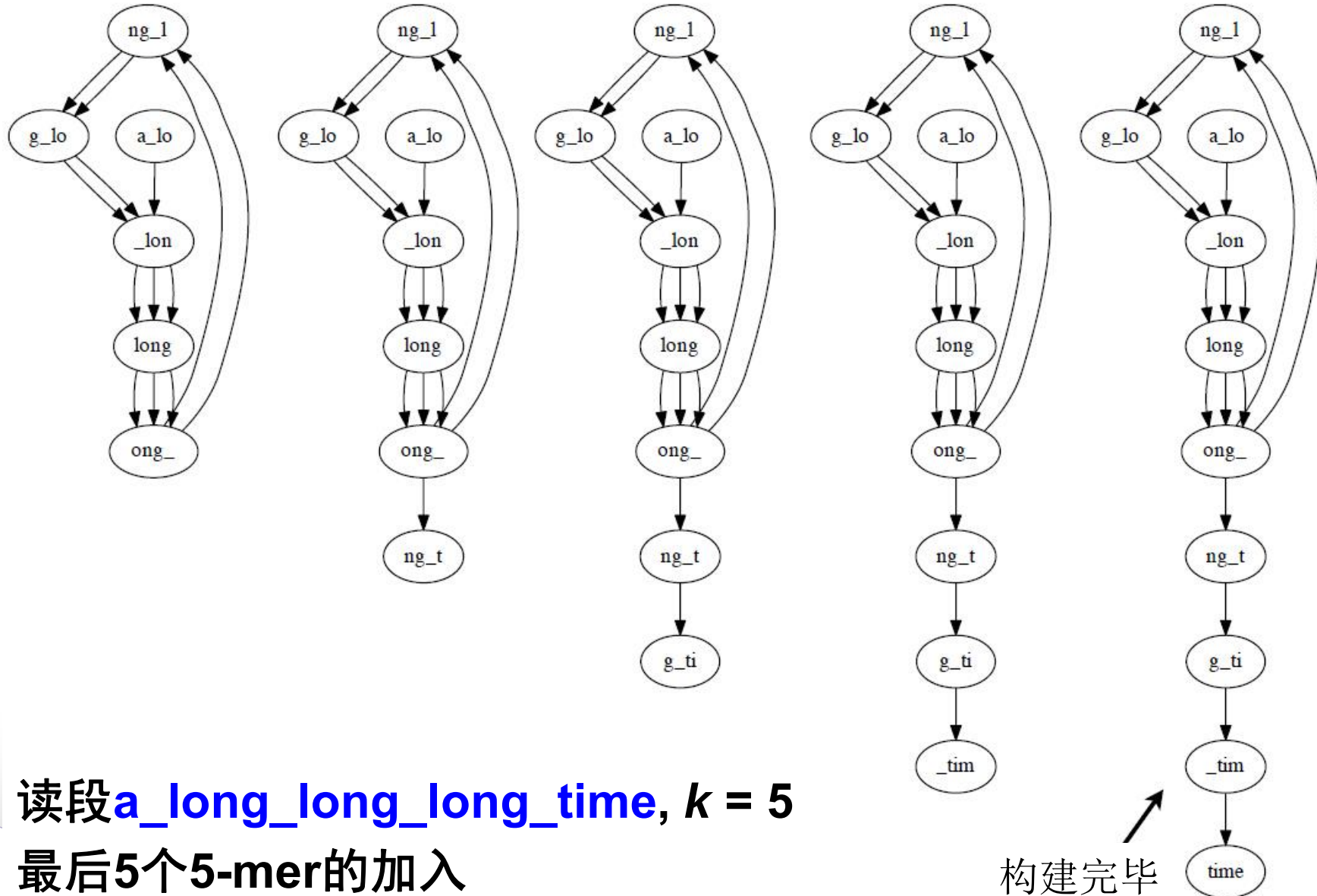
De Bruijn graph



- ❑ 读段 **a_long_long_long_time**, $k = 5$
- ❑ 8个5-mer的加入



De Bruijn graph



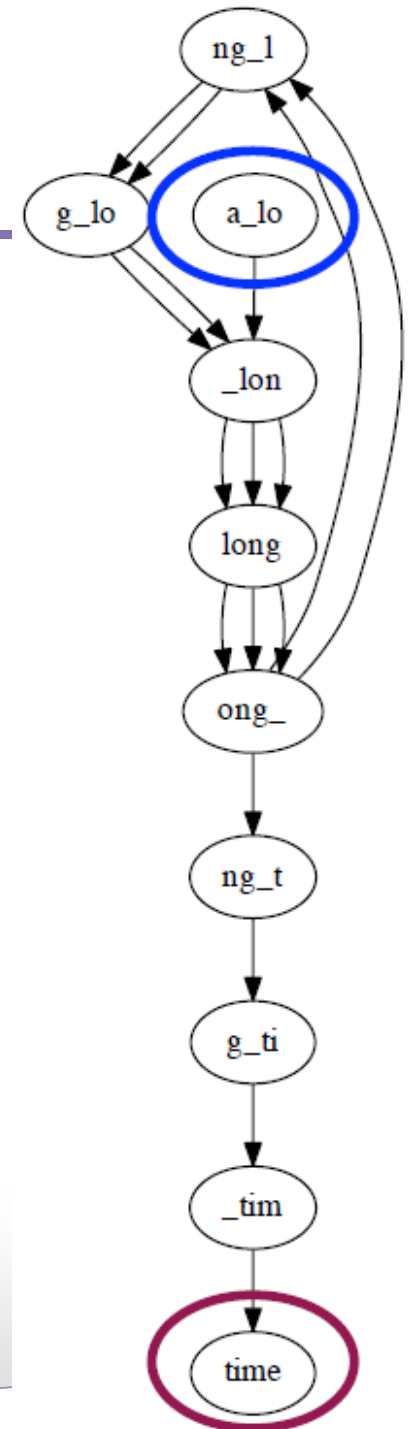
❑ 读段 `a_long_long_long_time`, $k = 5$

❑ 最后5个5-mer的加入

构建完毕

De Bruijn graph

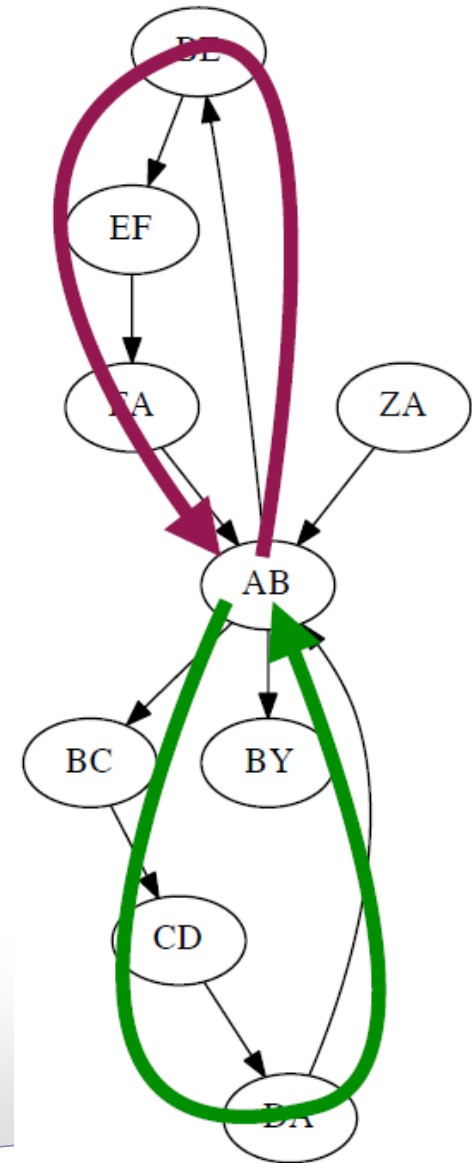
- 若是完美测序，该流程产生欧拉图
- 半平衡结点
 - ✿ 左端 $k-1$ -mer结点，出度比入度多1
 - ✿ 右端 $k-1$ -mer结点，入度比出度多1
- 其他结点为平衡结点
- 因此，欧拉行走可以实现
- 问题：是否总是如此？



De Bruijn graph



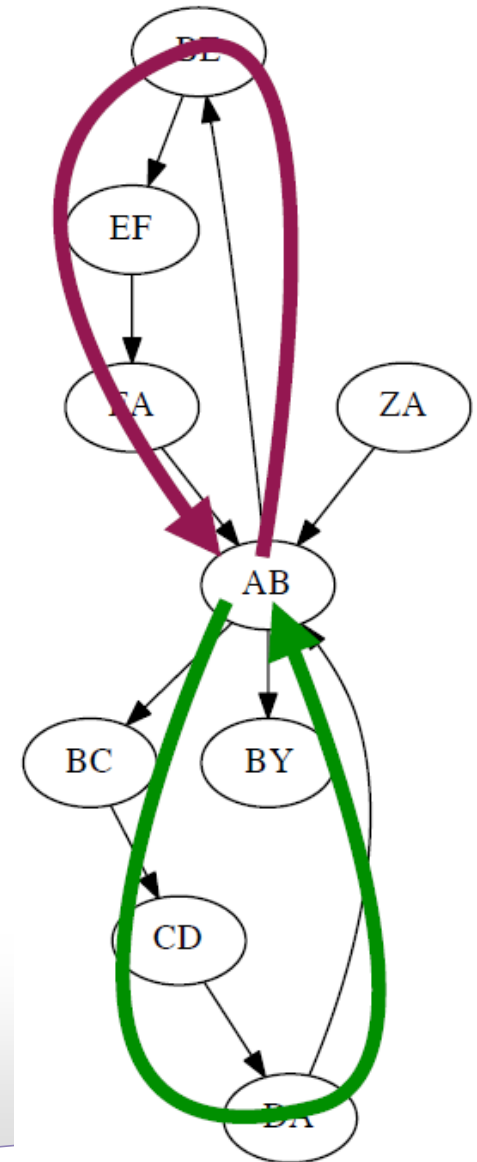
- 可能有多个欧拉行走的路线，仅有一条符合原始的参考序列
- 例如，DBG图：ZABCDABEFABY, $k = 3$
- 可选的欧拉行走路线
 - ✿ ZA → AB → BE → EF → FA → AB → BC → CD → DA → AB → BY
 - ✿ ZA → AB → BC → CD → DA → AB → BE → EF → FA → AB → BY
- 结点AB
 - ✿ 两条不连接的有向环
 - ✿ AB是重复序列: ZABCDABEFABY





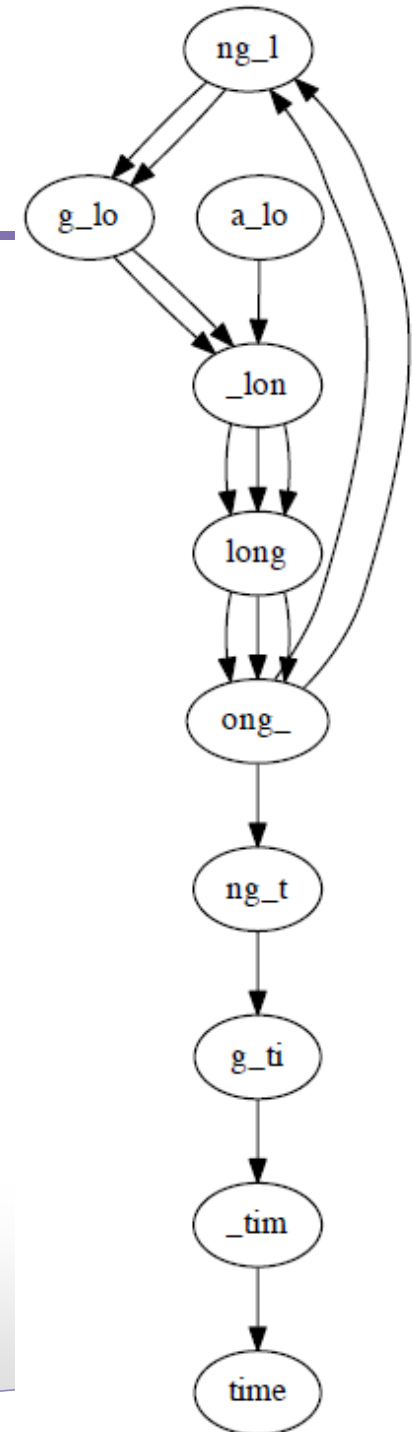
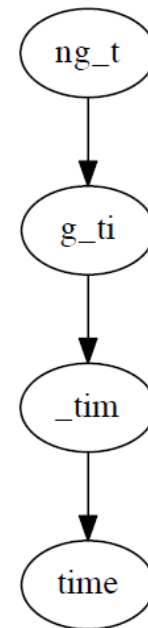
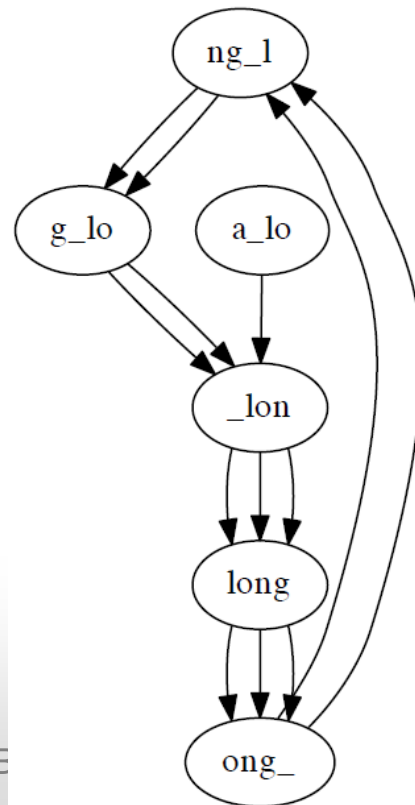
重复序列的影响

- ❑ 可能有多个欧拉行走的路线，仅有一条符合原始的参考序列
- ❑ 例如，DBG图：ZABCDABEFABY, $k = 3$
- ❑ 可选的欧拉行走路线
 - ✿ ZA → AB → BE → EF → FA → AB → BC → CD → DA → AB → BY
 - ✿ ZA → AB → BC → CD → DA → AB → BE → EF → FA → AB → BY
- ❑ 结点AB
 - ✿ 两条不连接的有向环
 - ✿ AB是重复序列: ZABCDABEFABY
- ❑ DBG不能解决所有问题
 - ✿ 存在重复序列



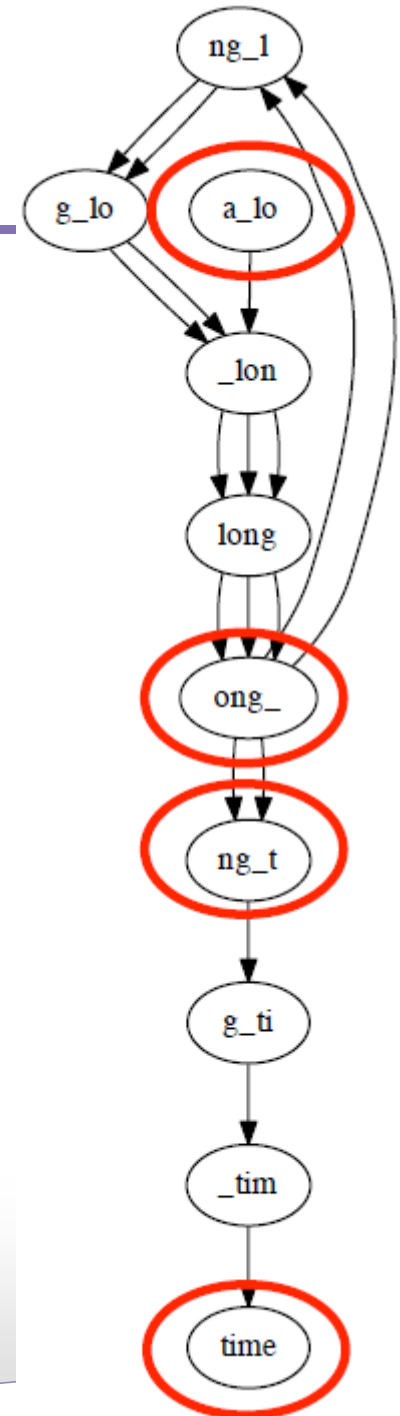
测序深度的影响

- ❑ 读段a_long_long_long_time, $k = 5$
- ❑ 若ong_t没有测到
 - ✿ 两个欧拉图
 - ✿ 每个图都可以欧拉遍历
 - ✿ 但整张图不是欧拉图



测序深度的影响

- ❑ 读段a_long_long_long_time, $k = 5$
- ❑ 若ong_t测到2次
 - ⚙ 全图有4个半平衡结点
 - ⚙ 非欧拉图，不能作欧拉行走



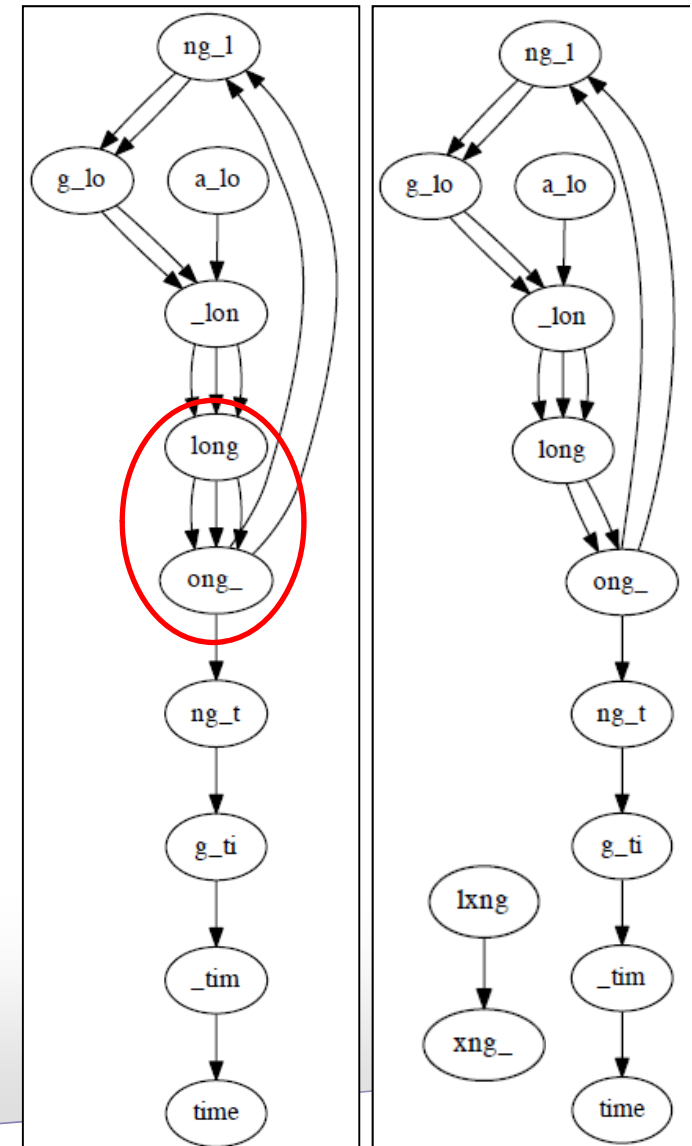
测序错误的影响



- ❑ 测序错误和染色体之间的差异有影响
- ❑ 读段a_long_long_long_time, $k = 5$
- ❑ 若其中一个long_测为lxng_
 - ⚙ 现图非欧拉图
 - ⚙ 主要部分有四个半平衡结点

原图

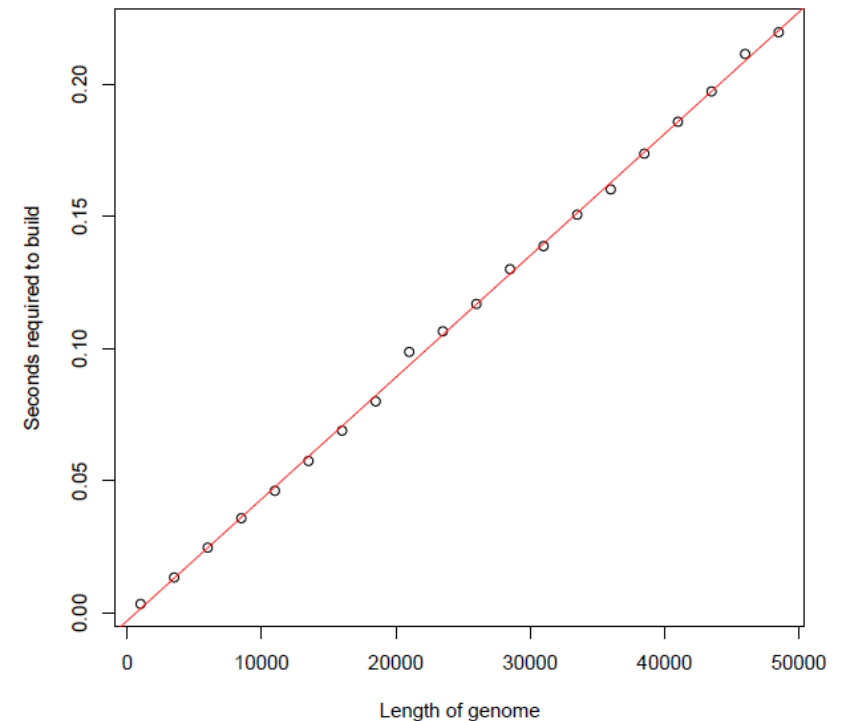
现图



De Bruijn graph



- ❑ DBG构建的时间复杂度
- ❑ 单个 k -mer
 - ✿ 加上1条边和最多2个结点
 - ✿ 时间复杂度 $O(1)$
- ❑ 利用哈希表编码结点和边
 - ✿ 建立结点 $O(1)$
 - ✿ 搜索/加入键值= $O(1)$
- ❑ N 个 k -mer
 - ✿ 时间复杂度 $O(N)$

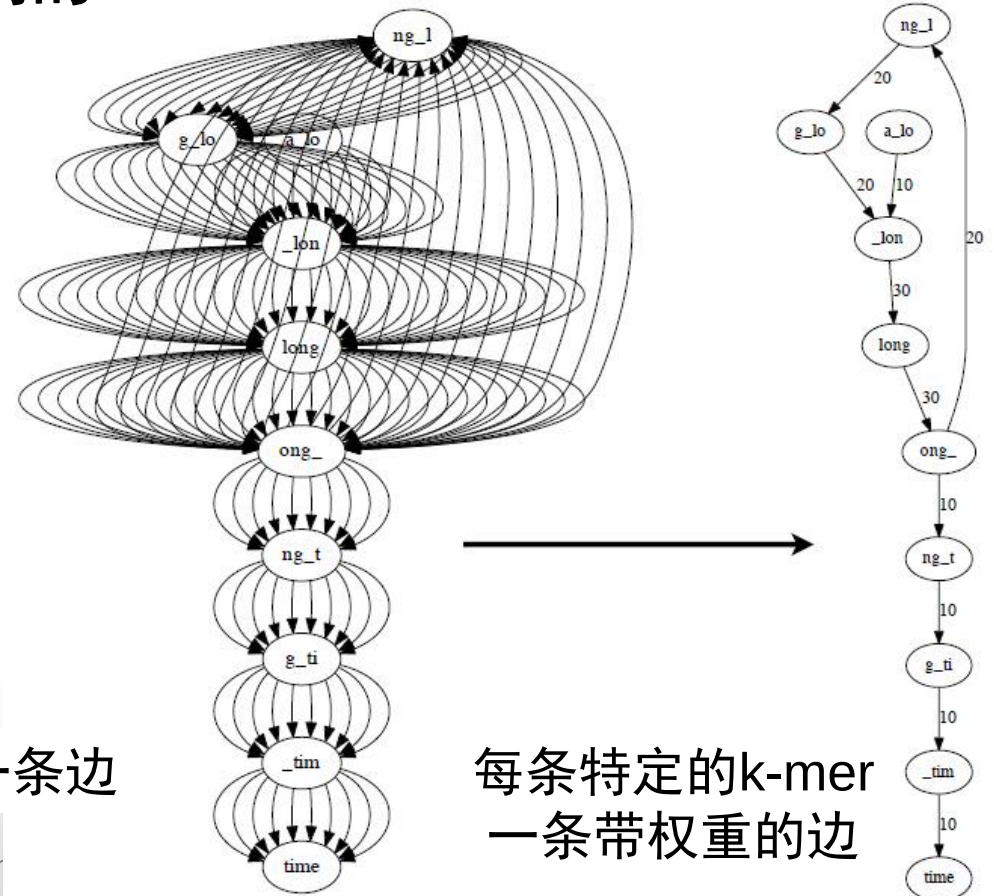


- ❑ 噬菌体基因组的DBG构建, $\sim 50,000\text{bp}$, $k=14$

De Bruijn graph



- ❑ 现实中一般测序深度为~ 30 - 50X
- ❑ 相同边出现多次：边的权重
- ❑ 利用边的权重代表每条不同的k-mer



一条k-mer一条边

每条特定的k-mer
一条带权重的边

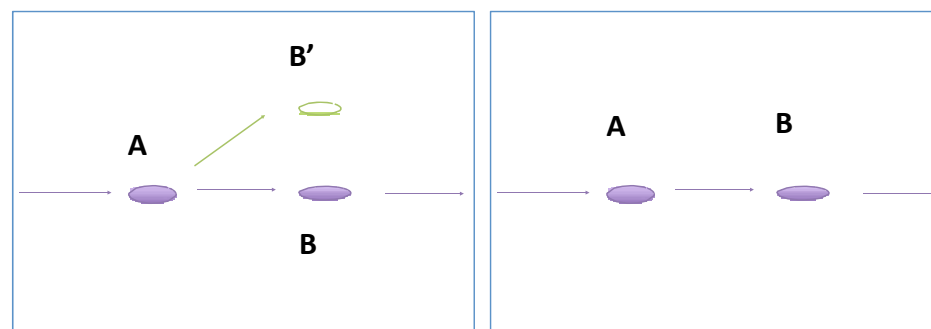
Error correction



□ 基于图的拓扑结构

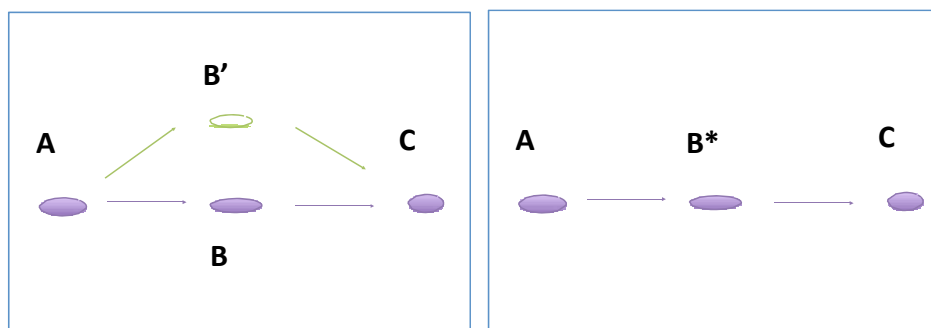
□ 读段尾端

- ✿ 去掉不能延伸的结点



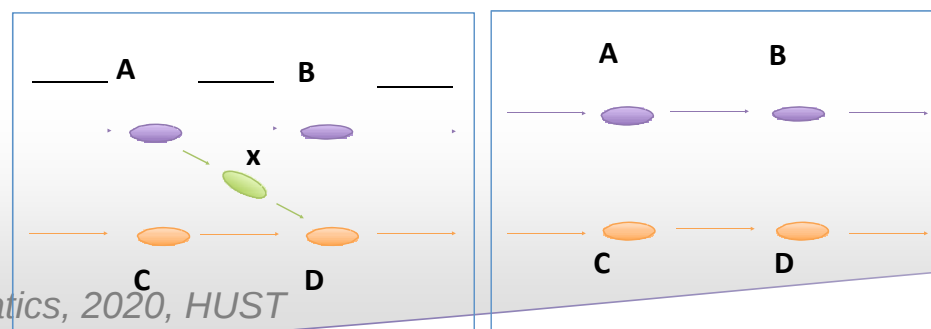
□ 读段中间

- ✿ 序列比对, 合并



□ 嵌合边

- ✿ 去掉覆盖率低的边





DBG算法的缺点

- ❑ 读段需要被切分成更短的 k -mer
- ❑ 不能解决重复序列的问题
- ❑ 存在测序错误时很难解决
- ❑ 丢失序列一致性：有些路径与输入读段不一致
- ❑ 各有优缺点：OLC → DBG

OLC vs. DBG



- ❑ **N50**: 将Contig从长至短排序，加和长度达到总长度一半时，最后一个Contig长度即为 N50
- ❑ **DBG更快，OLC需要内存小**

	<i>t</i> =75	<i>k</i> =61	<i>k</i> =67	<i>k</i> =59
Table 1. Assembly statistics for <i>C. elegans</i> data set				
	SGA	Velvet	ABYSS	SOAPdenovo
Scaffold N50 size	26.3 kbp	31.3 kbp	23.8 kbp	31.1 kbp
Aligned contig N50 size	16.8 kbp	13.6 kbp	18.4 kbp	16.0 kbp
Mean aligned contig size	4.9 kbp	5.3 kbp	6.0 kbp	5.6 kbp
Sum aligned contig size	96.8 Mbp	95.2 Mbp	98.3 Mbp	95.4 Mbp
Reference bases covered	96.2 Mbp	94.8 Mbp	95.9 Mbp	95.1 Mbp
Reference bases covered by contigs ≥ 1 kb	93.0 Mbp	92.1 Mbp	93.9 Mbp	92.3 Mbp
Mismatch rate at all assembled bases	1 per 21,545 bp	1 per 8786 bp	1 per 5577 bp	1 per 26,585 bp
Mismatch rate at bases covered by all assemblies	1 per 82,573 bp	1 per 18,012 bp	1 per 8209 bp	1 per 81,025 bp
Contigs with split/bad alignment (sum size)	458 (4.4 Mbp)	787 (7.2 Mbp)	638 (9.1 Mbp)	483 (4.4 Mbp)
Total CPU time	41 h	2 h	5 h	13 h
Max memory usage	4.5 GB	23.0 GB	14.1 GB	38.8 GB

100 MBase genome, 33.8M read pairs, 100bp reads each end, 250bp insert size

性能比较



- 性能大致相当
- 参考序列：50 bp ~ 5000 bp, 10,000条读段

