哈尔滨工业大学计算学部

读书/论文笔记

课程名称：生物信息学

课程类型：选修

项目名称：基因组序列拼接 Velvet

班级：2103601

学号：2021112845

姓名：张智雄

设计成绩	报告成绩	指导老师
		刘博

一、 论文的主要研究问题描述

这篇论文的主要研究问题是开发一种新的算法，用于处理高通量测序技术产生的非常短的DNA序列读段（25-50 bp）。这些短读段对于传统的基因组序列拼接方法来说是一个挑战，因为它们通常不足以跨越基因组中的重复区域，导致难以生成较长的连续序列（contigs）。为了解决这个问题，作者提出了一套名为“Velvet”的算法，专门针对 de novo 短读段拼接，使用 de Bruijn 图作为核心数据结构。

在传统的重叠布局一致性（OLC）方法中，每个读段被视为图中的一个节点，而检测到的重叠则作为节点间的弧。然而，由于短读段产生的大量节点和弧，使得 OLC 方法在处理短读段时变得非常低效，且容易在重复区域产生歧义。相比之下，de Bruijn 图通过 k-mers（k 个核苷酸的序列）来表示数据，每个 k-mer 作为一个节点，节点通过有向弧相连，表示 k-mers 的顺序。这种方法在处理高冗余数据时更为高效，因为重复的序列在图中只出现一次，并且可以通过明确的链接来识别不同的起始和终止点。

Velvet 算法包括两个主要的步骤：错误校正和重复解决。首先，错误校正算法通过合并序列来纠正测序过程中产生的误差。接着，重复解决算法通过分离共享局部重叠的路径来解决重复区域的问题。此外，作者还开发了一个名为“Breadcrumb”的模块，利用配对末端读段信息来解决重复区域，提高拼接的连贯性。

为了评估 Velvet 算法的性能，作者在模拟数据和真实数据上进行了测试。模拟数据包括大肠杆菌、酿酒酵母、秀丽隐杆线虫和人类基因组的 5 Mb 区域，使用 35 bp 长的读段进行测试。实验数据显示，Velvet 能够在短读段数据上生成较长的 contigs，并且在引入 1%的错误率和模拟二倍体装配时，表现依然稳定。此外，作者还将 Velvet 与其他几种短读段拼接程序进行了比较，包括 SSAKE 和 VCAKE，结果表明 Velvet 在内存使用、速度和生成的 contigs 大小方面具有优势。

最后，作者讨论了 Velvet 算法的计算复杂性和扩展性问题。指出图的构建是算法中最耗时和内存的部分，并且提出了可能的解决方案，如使用持久化数据结构来处理全基因组装配。作者认为，尽管基因组序列拼接问题尚未完全解决，但 Velvet 算法已经能够将高覆盖率的短读段转换成合理长度的 contigs，并且通过结合配对读段信息，可以解决大部分重复区域，为基因组序列的 de novo 拼接提供了一个有效的工具。

二、 论文的主要方法

2.1 de Bruijn 图的构建

de Bruijn 图的构建是 Velvet 算法中的关键步骤，它为后续的序列拼接提供了基础的数据结构。以下是 de Bruijn 图构建过程的详细描述：k-mer 是长度为 k 的 DNA 序列片段。在构建 de Bruijn 图之前，首先需要确定 k-mer 的长度。这个长度受限于读段的长度，并允许有一定的重叠，以确保能够检测到读段之间的重叠区域。将所有的读段根据 k-mer 长度进行

哈希处理。对于每个 k-mer，记录它首次出现的读段的 ID 以及在该读段中的位置。同时，每个 k-mer 及其反转互补序列都会被记录，以确保图中能够表示正反两条链上的信息。在 de Bruijn 图中，每个节点代表一个 k-mer 序列。如果 k 是奇数，那么每个 k-mer 及其反转互补序列将被视为不同的节点。每个节点与其反转互补序列的节点配对，形成一个“块”。节点之间的连接（有向弧）表示 k-mers 之间的顺序关系。如果一个 k-mer 的最后一个 k-1 个核苷酸与另一个 k-mer 的第一个 k-1 个核苷酸相同，那么这两个节点之间就可以建立一个有向弧。通过哈希表中的信息，将每个读段重写为一系列原始 k-mers 的组合，以及它们与之前哈希的读段的重叠。这种新的读段表示方式被称为“路线图”。记录每个读段的原始 k-mers 被后续读段重叠的情况。每次读段与另一个读段开始或结束重叠时，该读段的原始 k-mers 序列就被切断。使用路线图，从图中的一个节点开始，沿着有向弧逐步构建路径，直到覆盖所有读段。每条路径都代表了一个可能的 DNA 序列。在图构建完成后，通常会简化它以减少内存使用和计算时间。简化的过程涉及合并只有单个出弧和入弧的节点，将链状的块合并成单个块。在进行错误校正之前，需要确保图中的每个节点都正确地表示了读段中的 k-mers 序列，并且所有的连接都反映了 k-mers 之间的正确顺序。通过上述步骤，de Bruijn 图被构建完成，为下一步的错误校正和序列拼接打下了基础。Velvet 算法利用这种图结构有效地处理了短读段数据，尤其是在处理重复序列和错误校正时，de Bruijn 图的优势更为明显。

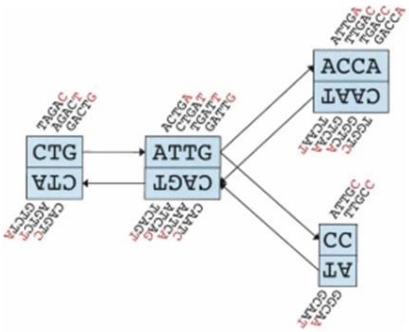

Figure 1. Schematic representation of our implementation of the de Bruijn graph. Each node, represented by a single rectangle, represents a series of overlapping *k*-mers (in this case, $k = 5$), listed directly *above* or *below*. (Red) The last nucleotide of each *k*-mer. The sequence of those final nucleotides, copied in large letters in the rectangle, is the sequence of the node. The twin node, directly attached to the node, either *below* or *above*, represents the reverse series of reverse complement *k*-mers. Arcs are represented as arrows *between* nodes. The last *k*-mer of an arc's origin overlaps with the first of its destination. Each arc has a symmetric arc. Note that the two nodes on the *left* could be merged into one without loss of information, because they form a chain.

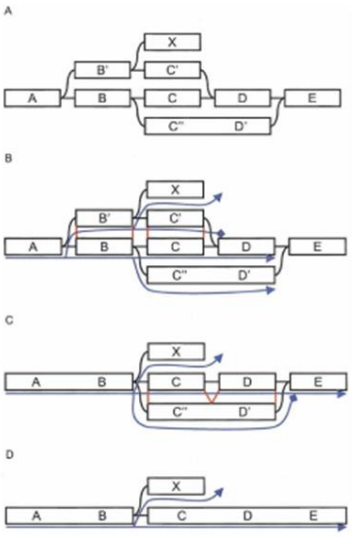

Figure 2. Example of Tour Bus error correction. (*A*) Original graph. (*B*) The search starts from A and spreads toward the *right*. The progression of the *top* path (through B′ and C′) is stopped because D was previously visited. The nucleotide sequences corresponding to the alternate paths B′C′ and BC are extracted from the graph, aligned, and compared. (*C*) The two paths are judged similar, so the longer one, B′C′, is merged into the shorter one, BC. The merging is directed by the alignment of the consensus sequences, indicated in red lines in *B*. Note that node X, which was connected to node B′, is now connected to node B. The search progresses, and the *bottom* path (through C′ and D′) arrives second in E. Once again, the corresponding paths, C′D′ and CD are compared. (*D*) CD and C′D′ are judged similar enough. The longer path is merged into the shorter one.

2.2 图的简化

在 de Bruijn 图中，每个节点代表一个 k-mer 序列，而节点之间的连接（有向弧）表示 k-mers 之间的顺序关系。当一个读段结束时，它在图中形成一个“链”，这是一系列线性连

接的块。这些链状结构增加了内存的使用量，并且延长了计算时间，因为每个块都需要单独处理。如果一个节点 A 只有一个出弧，且指向的节点 B 也只有一个入弧，那么这两个节点（以及它们的双胞胎节点）可以合并成单个节点，而不会丢失任何信息。这个过程是迭代进行的，图中的链状结构被逐步合并，直到无法进一步简化为止。在合并节点时，相关的信息（如读段 ID、覆盖度和序列信息）被适当地转移到合并后的节点上。随着节点的合并，图中的路径也会相应地调整，以确保读段在图中的正确映射。简化后的图更加紧凑，减少了内存占用和处理时间，同时保留了足够的信息用于后续的错误校正和序列拼接步骤。

通过减少节点和弧的数量，简化操作减少了内存的使用，这对于处理大规模基因组数据集尤其重要。简化图减少了算法在后续步骤中需要处理的数据量，从而加快了处理速度。尽管进行了简化，但图中的序列信息和覆盖度信息仍然被保留，这对于维持拼接结果的准确性至关重要。在某些情况下，过度简化可能会导致丢失对拼接有用的信息，尤其是在基因组中存在复杂重复区域时。简化过程可能受到 k-mer 长度等参数的影响，需要仔细调整以避免损失关键信息。

2.3 错误校正

在测序过程中可能会产生不同类型的错误，包括替换、插入和缺失。Velvet 算法主要关注替换错误，因为这是短读段测序技术中最常见的错误类型。在进行错误校正之前，首先使用选定的 k-mer 长度构建 de Bruijn 图。图中的每个节点代表一个 k-mer，节点之间的连接（弧）表示 k-mers 的顺序关系。Velvet 的错误校正包括三个主要步骤：去除尖端（tips）、去除气泡（bubbles）和处理错误连接：

- **去除尖端：**尖端是图中未连接的末端链，可能是由于测序错误或真实的序列变异造成的。Velvet 通过设置长度阈值（通常是 2k）来识别和去除这些尖端，以避免移除真实的序列。
- **去除气泡：**气泡是图中的简单循环，可能由测序错误或真实的序列多态性造成。Velvet 使用“Tour Bus 算法”来识别和合并相似的路径，从而去除气泡。
- **处理错误连接：**错误连接是图中不形成循环的异常连接，可能是由测序错误或克隆错误造成的。Velvet 通过设置覆盖度阈值来识别和移除这些错误连接。
- **Tour Bus 算法：**该算法通过广度优先搜索（BFS）遍历图，并在遇到重复节点时回溯，以找到两个相似路径的共同祖先。然后，算法会比较这两条路径的序列，并在它们足够相似时合并它们。
- **覆盖度阈值：**在 Tour Bus 算法之后，Velvet 使用用户定义的覆盖度阈值来移除剩余的低覆盖度节点，这些节点可能是由于错误造成的。

通过校正测序过程中产生的错误，Velvet 能够提高最终拼接序列的准确性。Tour Bus 算法能够有效处理图中的气泡结构，即使在存在序列多态性的情况下也能进行校正。错误校正步骤能够适应不同的测序错误率和基因组复杂性。错误校正的性能在很大程度上依赖于参数（如 k-mer 长度和覆盖度阈值）的选择，这可能需要用户的经验和优化。错误校正步骤可能

难以区分真实的生物学变异（如 SNPs）和测序错误，这可能导致一些真实的序列变异被错误地校正。错误校正步骤，特别是 Tour Bus 算法，可能需要较多的计算资源，尤其是在处理大规模基因组数据集时。

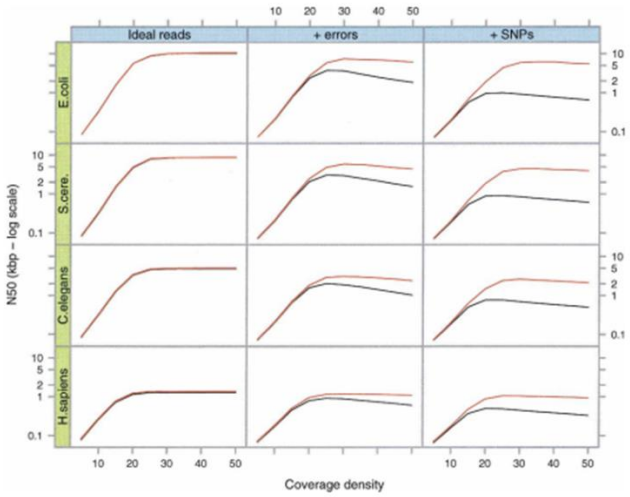

Figure 3. Simulations of Tour Bus. The genome of *E. coli* and 5-Mb samples of DNA from three other species (*S. cerevisiae, C. elegans, and H. sapiens*, respectively) were used to generate 35-bp read sets of varying read depths (*X*-axis of each plot). We measured the contig length N50 (*Y*-axis, log scale) after tip-clipping (black curve) then after the subsequent bubble smoothing (red curve). In the first column are the results for perfect, error-free reads. In the second column, we inserted errors in the reads at a rate of 1%. In the third column, we generated a slightly variant genome from the original by inserting random SNPs at a rate of 1 in 500. The reads were then generated with errors from both variants, thus simulating a diploid assembly.

Table 1. Efficiency of the Velvet error-correction pipeline on the BAC data set

Step	No. of nodes	N50 (bp)	Maximum length (bp)	Coverage (percent >50 bp)	Coverage (percent >100 bp)
Initial	1,353,791	5	7	0	0
Simplified	945,377	5	80	4.3	0.2
Tips clipped	4898	714	5037	93.5	78.7
Tour Bus	1147	1784	7038	93.4	90.1
Coverage cutoff	685	1958	7038	92.0	90.0
Ideal	620	2130	9045	93.7	91.9

Each line in this table represents a different stage in Velvet. The initial graph was built directly from the BAC reads. The second was the result of node concatenation. The next three graphs were the result of the three consecutive steps of error correction: tip clipping, Tour Bus, and coverage cutoff. The last graph was obtained by building the graph of the reference sequence then submitting it to Tour Bus, to simulate an error-free and gap-free assembly.

Table 2. Efficiency of the Velvet error-correction pipeline on the *Streptococcus* data set

Step	No. of nodes	N50 (bp)	Maximum length (bp)	Coverage (percent >50 bp)	Coverage (percent >100 bp)
Initial	3,621,167	16	16	0	0
Simplified	2,222,845	16	44	0.1	0
Tips clipped	15,267	2195	7949	96.2	95.4
Tour Bus	3303	4334	17,811	96.8	96.4
Coverage cutoff	1496	8564	29,856	96.9	96.5
Ideal	1305	9609	29,856	97.0	96.8

2.4 Breadcrumb 模块

Breadcrumb 模块的主要目的是利用配对末端（Paired-end）信息来解决 de Bruijn 图中由于重复序列造成的复杂结构，从而提高基因组序列的拼接质量。

Breadcrumb 模块首先定义一个长度阈值，该阈值通常比大多数插入片段（inserts）的长度要长。所有长度超过这个阈值的节点被称为“长节点”（long nodes）。使用配对末端信息，Breadcrumb 模块尝试将长节点两两配对。即使没有关于配对的唯一性信息，该步骤也尝试找到可以连接的节点对。对于每个长节点，Breadcrumb 标记所有包含该节点配对末端读段的节点。如果存在一个对应的长节点，它会标记所有与该对应节点相连的节点。Breadcrumb 尝试从每个被标记的节点出发，探索可能的路径，直到遇到没有标记的节点或存在多个可能的路径为止。如果一个长节点通过少于 5 个配对末端读段与多个长节点相连，这些连接被认为是不可靠的，并将被丢弃。Breadcrumb 在探索路径时考虑到了错误的发生，它使用 Tour Bus 算法中的错误校正信息，忽略被认为是不可靠的读段。Breadcrumb 能够通过配对末端信息将两个长节点之间的序列连接起来，形成所谓的序列连接的超连续序列（Sequence Connected Super Contigs，SCSCs）。这些 SCSCs 提供了比传统间隙超连续序列（gapped supercontigs）更多的信息。

Breadcrumb 能够有效地处理基因组中的重复区域，提高了拼接的连贯性和准确性。通过使用配对末端读段，Breadcrumb 能够跨越 de Bruijn 图中的复杂结构，连接分离的序列。模块设计中包含了对测序错误的容忍，使得即使在存在错误的情况下也能进行有效的拼接。Breadcrumb 的性能可能受到插入片段长度分布和覆盖度等参数的影响，需要适当的调整。

对于非常复杂或高度重复的区域，即使使用 Breadcrumb 模块，也可能难以完全解决拼接问题。处理大量的配对末端信息和探索可能的路径可能会消耗较多的计算资源。

三、 论文的主要实验结果

3.1 模拟数据测试

作者使用模拟数据对 Velvet 算法进行了测试，选择了四种不同物种的基因组：大肠杆菌（E. coli）、酿酒酵母（S. cerevisiae）、秀丽隐杆线虫（C. elegans）和人类（H. sapiens）。这些测试包括了不同覆盖度（5x 到 50x）下，35 bp 长度的读段，并且考虑了 1%的突变错误率。实验结果表明，在没有错误的情况下，随着覆盖度的增加，N50 值开始迅速增长，并在达到一定水平后趋于稳定，这个稳定值与基因组的复杂性和重复性有关。当读段中引入 1%的错误率时，Velvet 算法仍然能够有效地进行拼接，尽管最大 N50 值有所下降。这显示了 Velvet 在错误存在的情况下仍保持了较好的性能。在模拟二倍体装配的情况下，即使在参考基因组中随机加入 SNPs，Velvet 算法的性能也未受到显著影响，这表明算法对基因组的自然变异具有一定的容忍度。

3.2 真实数据测试

使用 Solexa 测序技术获得的真实人类 BAC 克隆数据进行了测试。该 BAC 克隆的长度为 173,428 bp，平均覆盖度为 970x。通过 Velvet 算法进行错误校正后，使用 Tour Bus 算法合并了相似的路径，有效地处理了重复序列。实验结果显示，与使用已知完成序列构建的理想 de Bruijn 图相比，Velvet 生成的图在结构上保持了高度的一致性，中位节点长度和最大节点长度都与完成的 BAC 图相当。

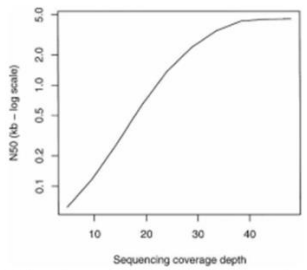

Figure 4. Effect of coverage on contig length with experimental *Streptococcus* data.

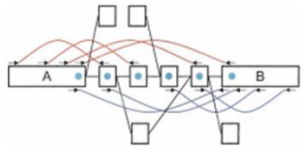

Figure 5. Breadcrumb algorithm. Two long contigs produced after error correction, A and B, are joined by several paired reads (red and blue arcs). The path between the two can be broken up because of a repeat internal to the connecting sequence, because of an overlap with a distinct part of the genome, or because of some unresolved errors. The small square nodes represent either nodes of the path between A and B, or other nodes of the graph connected to the former. Finding the exact path in the graph from A to B is not straightforward because of all the alternate paths that need to be explored. However, if we mark all the nodes that are paired up to either A or B (with a blue circle), we can define a subgraph much simpler to explore. Ideally, only a linear path connects both nodes.

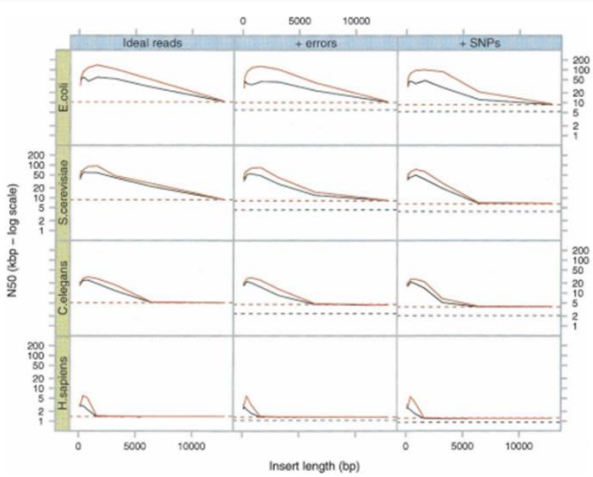

Figure 6. Breadcrumb performance on simulated data sets. As in Figure 3, we sampled 5-Mb DNA sequences from four different species (*E. coli*, *S. cerevisiae*, *C. elegans*, and *H. sapiens*, respectively) and generated 50× read sets. The horizontal lines represent the N50 reached at the end of Tour Bus (see Fig. 3) (broken black line) and after applying a 4× coverage cutoff (broken red line). Note how the difference in N50 between the graph of perfect reads and that of erroneous reads is significantly reduced by this last cutoff. (Black curves) The results after the basic Breadcrumb algorithm; (red curves) the results after super-contigging.

在 Streptococcus suis P1/7 的 2 Mb 基因组上进行的测试中，Velvet 算法处理了 270 万个 36 bp 的读段，平均覆盖度为 48x。测试结果没有产生错误的装配，基因组的 96.5%被覆盖，且序列一致性达到了 99.996%。

3.3 覆盖度对 N50 的影响

通过构建不同覆盖度的子集读段的 de Bruijn 图，研究了覆盖度对 N50 值的影响。实验结果表明，随着覆盖度的增加，N50 值呈指数增长，但最终会趋于一个平台期，这个平台期可能与基因组的重复结构有关。

3.4 Breadcrumb 模块测试

Breadcrumb 模块是为了解决 de Bruijn 图中的重复区域而设计的。通过使用模拟数据集进行测试，作者评估了 Breadcrumb 模块的性能。实验结果显示，当插入序列（insert length）足够长以跨越图中的障碍时，N50 值会增加。然而，如果插入序列过长，可能会导致 scaffolding（支架）问题，从而降低装配的质量。Breadcrumb 模块在处理重复区域时，通过配对末端信息来连接长节点，有效地提高了装配的连贯性。尽管如此，一些复杂或稀疏的重复结构仍然对装配构成了挑战。

四、 论文方法的优缺点分析

Velvet 算法特别设计用于处理新一代测序技术产生的非常短的读段（25-50 bp），适用于高覆盖率的数据集。通过使用 de Bruijn 图，Velvet 能够以紧凑的形式表示基因组数据，这使得即使是大规模的数据集也能够被有效管理。Velvet 算法包括一个错误校正步骤，可以识别并校正测序过程中产生的错误，提高了拼接的准确性。通过 Breadcrumb 模块，Velvet 能够利用配对末端信息来解决基因组中的重复区域，增强了拼接的连贯性。Velvet 在模拟数据和真实数据上进行了测试，证明了其方法的有效性和鲁棒性。Velvet 的代码是开源的，并且可以在 GNU 公共许可下免费使用，这增加了其在研究社区中的可访问性。与其他短读段拼接程序相比，Velvet 在内存使用、速度和生成的 contigs 大小方面表现出优势。

Velvet 的性能对参数 k（k-mer 长度）非常敏感，需要用户进行适当的调整，这可能需要一定的实验和优化。尽管 Velvet 在处理短读段方面表现出色，但在处理大规模基因组数据时，图构建阶段可能会消耗大量内存和计算资源。于非常复杂或长串联重复的区域，即使是 Breadcrumb 模块也可能无法完全解决，这可能限制了算法在某些复杂基因组中的应用。错误校正步骤可能无法完全区分真实的生物学变异（如 SNPs）和测序错误，这可能需要后续的后处理步骤来区分。随着基因组数据量的增加，Velvet 算法需要进一步优化以处理更大的数据集，尤其是在内存和计算效率方面。在某些特定的应用案例中，如非常长的重复序列或结构变异丰富的基因组，Velvet 可能需要与其他技术或算法结合使用以获得最佳结果。由于基因组学是一个快速发展的领域，新的测序技术和算法不断涌现，Velvet 算法可能需要不断的更新和改进以保持其竞争力。